

Gröbner Bases, Macaulay Matrices and Dubé's Degree Bounds

Alexander Maletzky*

October 13, 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

1	Introduction	4
1.1	Future Work	4
2	Degree Sections of Power-Products	4
3	Utility Definitions and Lemmas about Degree Bounds for Gröbner Bases	11
4	Computing Gröbner Bases by Triangularizing Macaulay Matrices	18
4.1	Gröbner Bases	18
4.2	Bounds	19
5	Integer Binomial Coefficients	23
5.1	Inequalities	23
5.2	Backward Difference Operator	26

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

6	Integer Polynomial Functions	28
6.1	Definition and Basic Properties	28
6.2	Closure Properties	29
7	Monomial Modules	33
7.1	Sets of Monomials	33
7.2	Modules	33
7.3	Reduction	35
7.4	Gröbner Bases	38
8	Preliminaries	45
8.1	Sets	45
8.2	Sums	47
8.3	<i>count-list</i>	47
8.4	<i>listset</i>	47
9	Direct Decompositions and Hilbert Functions	53
9.1	Direct Decompositions	53
9.2	Direct Decompositions and Vector Spaces	65
9.3	Homogeneous Sets of Polynomials with Fixed Degree	71
9.4	Interpreting Polynomial Rings as Vector Spaces over the Co- efficient Field	76
9.5	(Projective) Hilbert Function	79
10	Cone Decompositions	84
10.1	More Properties of Reduced Gröbner Bases	84
10.2	Quotient Ideals	85
10.3	Direct Decompositions of Polynomial Rings	85
10.4	Basic Cone Decompositions	93
10.5	Splitting w.r.t. Ideals	118
10.6	Function <i>split</i>	130
10.7	Splitting Ideals	151
10.8	Exact Cone Decompositions	156
10.9	Functions <i>shift</i> and <i>exact</i>	170
11	Dubé's Degree-Bound for Homogeneous Gröbner Bases	195
11.1	Hilbert Function and Hilbert Polynomial	196
11.2	Dubé's Bound	206
12	Sample Computations of Gröbner Bases via Macaulay Ma- trices	232
12.1	Combining <i>Groebner-Macaulay</i> , <i>Groebner-Macaulay</i> and <i>Groeb- ner-Macaulay.Dube-Bound</i>	233
12.2	Preparations	233

12.2.1	Connection between $(\text{'}x \Rightarrow_0 \text{'}a) \Rightarrow_0 \text{'}b$ and $(\text{'}x, \text{'}a) \textit{pp}$ $\Rightarrow_0 \text{'}b$	235
12.2.2	Locale <i>pp-powerprod</i>	237
12.3	Computations	247

1 Introduction

The formalization consists of two main parts:

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

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

1.1 Future Work

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

2 Degree Sections of Power-Products

theory *Degree-Section*

imports *Polynomials.MPoly-PM*

begin

definition *deg-sect* :: ‘*x set* \Rightarrow *nat* \Rightarrow (*x::countable* \Rightarrow_0 *nat*) *set*
where *deg-sect* *X d* = $.[X] \cap \{t. \text{deg-pm } t = d\}$

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

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

lemma *deg-sectD*:

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

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

using *assms* **by** (*simp-all add: deg-sect-def*)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

lemma *deg-sect-Suc*:
 $\text{deg-sect } X \ (\text{Suc } d) = (\bigcup x \in X. (+) (\text{Poly-Mapping.single } x \ 1) \ \{\dots \text{deg-sect } X \ d\}) \ (\text{is } ?A = ?B)$
proof (rule set-eqI)
 fix t

show $t \in ?A \longleftrightarrow t \in ?B$
proof
 assume $t \in ?A$
 hence $t \in .[X]$ and $\text{deg-pm } t = \text{Suc } d$ **by** (rule *deg-sectD*) +
 from *this*(2) **have** $\text{keys } t \neq \{\}$ **by** *auto*
 then **obtain** x **where** $x \in \text{keys } t$ **by** *blast*
 hence $1 \leq \text{lookup } t \ x$ **by** (simp *add: in-keys-iff*)
 from $\langle t \in .[X] \rangle$ **have** $\text{keys } t \subseteq X$ **by** (rule *PPsD*)
 with $\langle x \in \text{keys } t \rangle$ **have** $x \in X$..
 let $?s = \text{Poly-Mapping.single } x \ (1::\text{nat})$
 from $\langle 1 \leq \text{lookup } t \ x \rangle$ **have** $?s \text{ adds } t$
 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 $t - ?s \in \text{deg-sect } X \ d$
proof (rule *deg-sectI*)
 from $\langle t \in .[X] \rangle$ **show** $t - ?s \in .[X]$ **by** (rule *PPs-closed-minus*)
next
 from *deg-pm-plus*[of $?s \ t - ?s$] **have** $\text{deg-pm } t = \text{Suc } (\text{deg-pm } (t - ?s))$
 by (simp only: *t deg-pm-single*)
 thus $\text{deg-pm } (t - ?s) = d$ **by** (simp *add: \langle deg-pm t = Suc d \rangle*)
qed
 hence $?s + (t - ?s) \in (+) \ ?s \ ' \ \text{deg-sect } X \ d$ **by** (rule *imageI*)
 hence $t \in (+) \ ?s \ ' \ \text{deg-sect } X \ d$ **by** (simp only: *t*)
 with $\langle x \in X \rangle$ **show** $t \in ?B$..
next
 assume $t \in ?B$
 then **obtain** x **where** $x \in X$ and $t \in (+) \ (\text{Poly-Mapping.single } x \ 1) \ ' \ \text{deg-sect } X \ d$..
 from *this*(2) **obtain** s **where** $s: s \in \text{deg-sect } X \ d$
 and $t: t = \text{Poly-Mapping.single } x \ 1 + s$ (**is** $t = ?s + s$) ..
show $t \in ?A$
proof (rule *deg-sectI*)
 from $\langle x \in X \rangle$ **have** $?s \in .[X]$ **by** (rule *PPs-closed-single*)
 moreover from s **have** $s \in .[X]$ **by** (rule *deg-sectD*)
 ultimately **show** $t \in .[X]$ **unfolding** t **by** (rule *PPs-closed-plus*)
next
 from s **have** $\text{deg-pm } s = d$ **by** (rule *deg-sectD*)
 thus $\text{deg-pm } t = \text{Suc } d$ **by** (simp *add: t deg-pm-single deg-pm-plus*)
qed
qed
qed

lemma *deg-sect-insert*:
 $\text{deg-sect } (\text{insert } x \ X) \ d = (\bigcup d0 \leq d. (+) \ (\text{Poly-Mapping.single } x \ (d - d0)) \ ' \ \text{deg-sect } X \ d0)$
 (**is** $?A = ?B$)
proof (rule *set-eqI*)
fix t
show $t \in ?A \longleftrightarrow t \in ?B$

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

lemma *deg-le-sect-Suc*: $\text{deg-le-sect } X (\text{Suc } d) = \text{deg-le-sect } X \text{ } d \cup \text{deg-sect } X (\text{Suc } d)$

by (simp add: *deg-le-sect-def atMost-Suc Un-commute*)

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

$\text{deg-le-sect } X (\text{Suc } d) = \text{insert } 0 (\bigcup_{x \in X. (+) (\text{Poly-Mapping.single } x \text{ } 1))$ ‘
 $\text{deg-le-sect } X \text{ } d)$

(is $?A = ?B$)

proof –

have $\text{eq1}: \{\text{Suc } 0 .. \text{Suc } d\} = \text{Suc } \{..d\}$ by (simp add: *image-Suc-atMost*)

have $\text{insert } 0 \{1 .. \text{Suc } d\} = \{.. \text{Suc } d\}$ by *fastforce*

hence $?A = (\bigcup_{d0 \in \text{insert } 0 \{1 .. \text{Suc } d\}. \text{deg-sect } X \text{ } d0})$ by (simp add: *deg-le-sect-def*)

also have $\dots = \text{insert } 0 (\bigcup_{d0 \leq d. \text{deg-sect } X (\text{Suc } d0))}$ by (simp add: *eq1*)

also have ... = *insert* 0 ($\bigcup d0 \leq d. (\bigcup x \in X. (+) (Poly-Mapping.single\ x\ 1) \text{ ‘}$
deg-sect *X* *d0*))
by (*simp only: deg-sect-Suc*)
also have ... = *insert* 0 ($\bigcup x \in X. (+) (Poly-Mapping.single\ x\ 1) \text{ ‘}$ ($\bigcup d0 \leq d.$
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* \Rightarrow *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 $p \in P[X]$ **and** *poly-deg p* $\leq d$
shows *keys p* \subseteq *deg-le-sect X d*
proof
fix *t*
assume $t \in \text{keys } p$
also from *assms*(1) **have** ... $\subseteq .[X]$ **by** (*rule PolysD*)
finally have $t \in .[X]$.
from $\langle t \in \text{keys } p \rangle$ **have** *deg-pm t* \leq *poly-deg p* **by** (*rule poly-deg-max-keys*)
from *this* *assms*(2) **have** *deg-pm t* $\leq d$ **by** (*rule le-trans*)
with $\langle t \in .[X] \rangle$ **show** $t \in \text{deg-le-sect } X\ d$ **by** (*rule deg-le-sectI*)
qed

lemma *binomial-symmetric-plus*: $(n + k) \text{ choose } n = (n + k) \text{ choose } k$
by (*metis add-diff-cancel-left' binomial-symmetric le-add1*)

lemma *card-deg-sect*:
assumes *finite X* **and** $X \neq \{\}$
shows *card* (*deg-sect X d*) = $(d + (\text{card } X - 1)) \text{ choose } (\text{card } X - 1)$
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  $0 < \text{card } X$  by (simp add: card-gt-0-iff)
    let ?f =  $\lambda d0. \text{Poly-Mapping.single } x (d - d0)$ 
    from False have eq2:  $\text{card} (\text{deg-sect } X \ d0) = d0 + (\text{card } X - 1)$  choose (card
X - 1) for d0
      by (rule insert.hyps)
    have finite {..d} by simp
    moreover from insert.hyps(1) have  $\forall d0 \in \{..d\}. \text{finite } ((+) \text{ (?f } d0) \text{ ' deg-sect$ 
X d0)
      by (simp add: finite-deg-sect)
    moreover have  $\forall d0 \in \{..d\}. \forall d1 \in \{..d\}. d0 \neq d1 \longrightarrow$ 
 $((+) \text{ (?f } d0) \text{ ' deg-sect } X \ d0) \cap ((+) \text{ (?f } d1) \text{ ' deg-sect } X \ d1)$ 
= {}
    proof (intro ballI impI, rule ccontr)
      fix d1 d2 :: nat
      assume d1  $\neq$  d2
      assume  $((+) \text{ (?f } d1) \text{ ' deg-sect } X \ d1) \cap ((+) \text{ (?f } d2) \text{ ' deg-sect } X \ d2) \neq \{\}$ 
      then obtain t where  $t \in ((+) \text{ (?f } d1) \text{ ' deg-sect } X \ d1) \cap ((+) \text{ (?f } d2) \text{ '$ 
deg-sect X d2)
        by blast
      hence t1:  $t \in (+) \text{ (?f } d1) \text{ ' deg-sect } X \ d1$  and t2:  $t \in (+) \text{ (?f } d2) \text{ ' deg-sect$ 
X d2 by simp-all
      from t1 obtain s1 where  $s1 \in \text{deg-sect } X \ d1$  and  $s1: t = \text{?f } d1 + s1$  ..
      from this(1) have  $s1 \in .[X]$  by (rule deg-sectD)
      hence  $\text{keys } s1 \subseteq X$  by (rule PPsD)
      with insert.hyps(2) have eq3:  $\text{lookup } s1 \ x = 0$  by (auto simp: in-keys-iff)
      from t2 obtain s2 where  $s2 \in \text{deg-sect } X \ d2$  and  $s2: t = \text{?f } d2 + s2$  ..
      from this(1) have  $s2 \in .[X]$  by (rule deg-sectD)
      hence  $\text{keys } s2 \subseteq X$  by (rule PPsD)
      with insert.hyps(2) have eq4:  $\text{lookup } s2 \ x = 0$  by (auto simp: in-keys-iff)
      from s2 have  $\text{lookup } (\text{?f } d1 + s1) \ x = \text{lookup } (\text{?f } d2 + s2) \ x$  by (simp only:
s1)
      hence  $d - d1 = d - d2$  by (simp add: lookup-add eq3 eq4)
      moreover assume  $d1 \in \{..d\}$  and  $d2 \in \{..d\}$ 
      ultimately have  $d1 = d2$  by simp
      with  $\langle d1 \neq d2 \rangle$  show False ..
    qed
  ultimately have  $\text{card} (\text{deg-sect } (\text{insert } x \ X) \ d) =$ 
 $(\sum d0 \leq d. \text{card } ((+) \text{ (monomial } (d - d0) \ x) \text{ ' deg-sect } X \ d0))$ 
    unfolding deg-sect-insert by (rule card-UN-disjoint)
  also from refl have  $\dots = (\sum d0 \leq d. \text{card } (\text{deg-sect } X \ d0))$ 
  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 $... = (\sum d0 \leq d. (card X - 1) + d0 choose (card X - 1))$ **by** (simp
only: eq2 add.commute)
also have $... = (\sum d0 \leq d. (card X - 1) + d0 choose d0)$ **by** (simp *only:*
binomial-symmetric-plus)
also have $... = Suc ((card X - 1) + d) choose d$ **by** (rule *sum-choose-lower*)
also from $\langle 0 < card X \rangle$ **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 $\langle 0 < card X \rangle$ **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** $deg-le-sect X d \cap deg-sect X (Suc d) = \{ \}$

```

    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  $\wedge$  ( $\bigcup g:G$ . indets g)  $\subseteq$ 
    ( $\bigcup f:F$ . 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  $\implies \exists F' q$ . 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  $\implies \exists F' q$ . 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:  $\bigwedge g. g \in G \implies \exists F' q. \text{finite } F' \wedge F' \subseteq F \wedge g = (\sum_{f \in F'} q f * f) \wedge$ 
  ( $\forall f \in F'. \text{poly-deg } (q f * f) \leq b$ )
  by (auto simp: is-GB-cofactor-bound-def)
  from this(1, 2, 3) show ?thesis
proof
  fix g
  assume g  $\in$  G
  show  $\exists F' q. \text{finite } F' \wedge F' \subseteq F \wedge g = (\sum_{f \in F'} q f * f) \wedge$ 
  ( $\forall f. \text{indets } (q f) \subseteq ?X \wedge \text{poly-deg } (q f * f) \leq b \wedge (f \notin F' \longrightarrow q f =$ 
0))
  proof (cases g = 0)
    case True
    define q where q = ( $\lambda f. (('x \Rightarrow_0 \text{nat}) \Rightarrow_0 'b. 0 :: ('x \Rightarrow_0 \text{nat}) \Rightarrow_0 'b)$ )
    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  $\langle g \in G \rangle$  have  $\exists F' q. \text{finite } F' \wedge F' \subseteq F \wedge g = (\sum_{f \in F'} q f * f) \wedge$ 
    ( $\forall f \in F'. \text{poly-deg } (q f * f) \leq b$ )
    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:  $\bigwedge f. f \in F' \implies \text{poly-deg } (q0 f * f) \leq b$  by blast
    define sub where sub = ( $\lambda x. (('x. \text{if } x \in ?X \text{ then } \text{monomial } (1 :: 'b) (\text{Poly-Mapping.single } x (1 :: \text{nat}))$ 
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  $\langle g \in G \rangle$  have x  $\in \bigcup$  (indets ' G) by blast
      also have ...  $\subseteq$  ?X by fact
      finally obtain f where f  $\in$  F' and x  $\in$  indets f ..
      assume  $\forall f \in F'. x \notin \text{indets } f$ 
      hence x  $\notin$  indets f using  $\langle f \in F' \rangle$  ..
      thus monomial 1 (monomial (Suc 0) x) = 1 using  $\langle x \in \text{indets } f \rangle$  ..
    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  $\forall f \in F'. x \notin \text{indets } f$ 
      moreover from that(1)  $\langle F' \subseteq F \rangle$  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 = (λf. 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
        qed
      qed

```

```

      qed
      also from  $\langle f \in F' \rangle$  have  $\dots \leq b$  by (rule q0)
      finally show  $\text{poly-deg } (q f * f) \leq b$  .
    qed
  next
    case False
    thus ?thesis by (simp add: q-def)
  qed
  thus  $\text{indets } (q f) \subseteq ?X$  and  $\text{poly-deg } (q f * f) \leq b$  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  $F :: (('x \Rightarrow_0 \text{nat}) \Rightarrow_0 'b::\text{field}) \text{ set}$ 
  assumes is-GB-cofactor-bound  $F$   $b$  and  $F \subseteq P[X]$ 
  obtains  $G$  where  $\text{punit.is-Groebner-basis } G$  and  $\text{ideal } G = \text{ideal } F$  and  $G \subseteq P[X]$ 
  and  $\bigwedge g. g \in G \implies \exists F' q. \text{finite } F' \wedge F' \subseteq F \wedge g = (\sum_{f \in F'} q f * f) \wedge$ 
     $(\forall f. q f \in P[X] \wedge \text{poly-deg } (q f * f) \leq b \wedge (f \notin F' \longrightarrow q f$ 
     $= 0))$ 
  proof -
    let  $?X = \bigcup (\text{indets } ` F)$ 
    have  $?X \subseteq X$ 
    proof
      fix  $x$ 
      assume  $x \in ?X$ 
      then obtain  $f$  where  $f \in F$  and  $x \in \text{indets } f$  ..
      from this(1) assms(2) have  $f \in P[X]$  ..
      hence  $\text{indets } f \subseteq X$  by (rule PolysD)
      with  $\langle x \in \text{indets } f \rangle$  show  $x \in X$  ..
    qed
    from assms(1) obtain  $G$  where  $\text{punit.is-Groebner-basis } G$  and  $\text{ideal } G = \text{ideal } F$ 
    and 1:  $\bigcup (\text{indets } ` G) \subseteq ?X$ 
    and 2:  $\bigwedge g. g \in G \implies \exists F' q. \text{finite } F' \wedge F' \subseteq F \wedge g = (\sum_{f \in F'} q f * f) \wedge$ 
       $(\forall f. \text{indets } (q f) \subseteq ?X \wedge \text{poly-deg } (q f * f) \leq b \wedge (f \notin F' \longrightarrow q f = 0))$ 
    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 $\text{indets } g \subseteq \bigcup (\text{indets } 'G) \text{ by } \textit{blast}$
 also have $\dots \subseteq ?X \text{ by } \textit{fact}$
 also have $\dots \subseteq X \text{ by } \textit{fact}$
 finally show $g \in P[X] \text{ by } (\textit{rule PolysI-alt})$
 qed
 next
 fix g
 assume $g \in G$
 hence $\exists F' q. \text{finite } F' \wedge F' \subseteq F \wedge g = (\sum f \in F'. q f * f) \wedge$
 $(\forall f. \text{indets } (q f) \subseteq ?X \wedge \text{poly-deg } (q f * f) \leq b \wedge (f \notin F' \longrightarrow q f$
 $= 0))$
 by $(\textit{rule } 2)$
 then obtain $F' q$ where $\text{finite } F'$ and $F' \subseteq F$ and $g = (\sum f \in F'. q f * f)$
 and $\bigwedge f. \text{indets } (q f) \subseteq ?X$ and $\bigwedge f. \text{poly-deg } (q f * f) \leq b$ and $\bigwedge f. f \notin F'$
 $\implies q f = 0$
 by \textit{blast}
 show $\exists F' q. \text{finite } F' \wedge F' \subseteq F \wedge g = (\sum f \in F'. q f * f) \wedge$
 $(\forall f. q f \in P[X] \wedge \text{poly-deg } (q f * f) \leq b \wedge (f \notin F' \longrightarrow q f = 0))$
 proof $(\textit{intro exI allI conjI impI})$
 fix f
 from $\langle \text{indets } (q f) \subseteq ?X \rangle \langle ?X \subseteq X \rangle$ have $\text{indets } (q f) \subseteq X \text{ by } (\textit{rule}$
 $\textit{subset-trans})$
 thus $q f \in P[X] \text{ by } (\textit{rule PolysI-alt})$
 qed $\textit{fact+}$
 qed
 qed

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

fixes $F :: (('x \Rightarrow_0 \text{nat}) \Rightarrow_0 'b::\text{field}) \text{ set}$
 assumes *is-GB-cofactor-bound* F b and *finite* F and $F \subseteq P[X]$
 obtains G where *punit.is-Groebner-basis* G and *ideal* $G = \text{ideal } F$ and $G \subseteq P[X]$
 and $\bigwedge g. g \in G \implies \exists q. g = (\sum f \in F. q f * f) \wedge (\forall f. q f \in P[X] \wedge \text{poly-deg}$
 $(q f * f) \leq b)$
 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: $\bigwedge g. g \in G \implies \exists F' q. \text{finite } F' \wedge F' \subseteq F \wedge g = (\sum f \in F'. q f * f) \wedge$
 $(\forall f. q f \in P[X] \wedge \text{poly-deg } (q f * f) \leq b \wedge (f \notin F' \longrightarrow q f$
 $= 0))$
 by $(\textit{rule is-GB-cofactor-boundE-Polys}) \textit{blast}$
 from *this*(1, 2, 3) show *?thesis*
 proof
 fix g
 assume $g \in G$
 hence $\exists F' q. \text{finite } F' \wedge F' \subseteq F \wedge g = (\sum f \in F'. q f * f) \wedge$
 $(\forall f. q f \in P[X] \wedge \text{poly-deg } (q f * f) \leq b \wedge (f \notin F' \longrightarrow q f$
 $= 0))$
 by $(\textit{rule } 1)$

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 $\bigwedge f. \text{poly-deg } (q f * f) \leq b$ and $2: \bigwedge f. f \notin F' \implies$
 $q f = 0$ by *blast*
 show $\exists q. g = (\sum_{f \in F} q f * f) \wedge (\forall f. q f \in P[X] \wedge \text{poly-deg } (q f * f) \leq b)$
 proof (intro exI conjI impI allI)
 from *assms*(2) $\langle F' \subseteq F \rangle$ 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* $\{0\} = \text{ideal } F$ by (*metis ideal.span-empty ideal-eq-zero-iff*)
 qed *simp-all*

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

lemma *is-hom-GB-boundD*:
 $\text{is-hom-GB-bound } F b \implies (\bigwedge f. f \in F \implies \text{homogeneous } f) \implies g \in \text{punit.reduced-GB } F \implies \text{poly-deg } g \leq b$
 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* $\text{None } \text{'extend-indets' } F$) b
 shows *is-GB-cofactor-bound* $F b$
 proof –
 let $?F = \text{homogenize } \text{None } \text{'extend-indets' } F$
 define Y where $Y = \bigcup (\text{indets } \text{' } F)$
 define G where $G = \text{restrict-indets } \text{' } (\text{extended-ord.punit.reduced-GB } ?F)$
 have $Y \subseteq X$
 proof


```

fix x
assume x ∈ Y
then obtain f where f ∈ F and x ∈ indets f unfolding Y-def ..
from this(1) assms(2) have f ∈ P[X] ..
hence indets f ⊆ X by (rule PolysD)
with ⟨x ∈ indets f⟩ show x ∈ X ..
qed
hence finite Y using assms(1) by (rule finite-subset)
moreover have F ⊆ P[Y] by (auto simp: Y-def Polys-alt)
ultimately have punit.is-Groebner-basis G and ideal G = ideal F and G ⊆
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 ⊆ P[Y]⟩ show ⋃ (indets ‘ G) ⊆ ⋃ (indets ‘ F) by (auto simp: Y-def
Polys-alt)
next
fix g
assume g ∈ G
then obtain g' where g': g' ∈ extended-ord.punit.reduced-GB ?F
  and g: g = restrict-indets g' unfolding G-def ..
have f ∈ ?F ⟹ homogeneous f for f by (auto simp: homogeneous-homogenize)
with assms(3) have poly-deg g' ≤ b using g' by (rule extended-ord.is-hom-GB-boundD)
from g' have g' ∈ 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 ⊆ P[insert None (Some ‘ Y)]
proof
fix f0
assume f0 ∈ ?F
then obtain f where f ∈ F and f0: f0 = homogenize None (extend-indets
f) by blast
  from this(1) ⟨F ⊆ P[Y]⟩ have f ∈ P[Y] ..
  hence extend-indets f ∈ P[Some ‘ Y] by (auto simp: indets-extend-indets
Polys-alt)
  thus f0 ∈ P[insert None (Some ‘ Y)] unfolding f0 by (rule homoge-
nize-in-Polys)
qed
qed
finally have g' ∈ ideal ?F .
with ⟨⋀f. f ∈ ?F ⟹ homogeneous f⟩ obtain F0 q where finite F0 and F0
⊆ ?F
  and g': g' = (∑ f ∈ F0. q f * f) and deg-le: ⋀f. poly-deg (q f * f) ≤ poly-deg
g'
  by (rule homogeneous-idealE) blast+
  from this(2) obtain F' where F' ⊆ 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  $\exists F' q. \text{finite } F' \wedge F' \subseteq F \wedge g = (\sum_{f \in F'} q f * f) \wedge (\forall f \in F'. \text{poly-deg } (q f * f) \leq b)$ 
    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  $g = (\sum_{f \in F'} (\text{restrict-indets} \circ q \circ \text{homogenize None} \circ \text{extend-indets}) f * f)$ 
      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  $\text{poly-deg } ((\text{restrict-indets} \circ q \circ \text{homogenize None} \circ \text{extend-indets}) f * f)$ 
      =
         $\text{poly-deg } (\text{restrict-indets } (q (\text{homogenize None } (\text{extend-indets } f)) * \text{homogenize None } (\text{extend-indets } f)))$ 
      by (simp add: restrict-indets-times)
      also have  $\dots \leq \text{poly-deg } (q (\text{homogenize None } (\text{extend-indets } f)) * \text{homogenize None } (\text{extend-indets } f))$ 
      by (rule poly-deg-restrict-indets-le)
      also have  $\dots \leq \text{poly-deg } g'$  by (rule deg-le)
      also have  $\dots \leq b$  by fact
      finally show  $\text{poly-deg } ((\text{restrict-indets} \circ q \circ \text{homogenize None} \circ \text{extend-indets}) f * f) \leq b$  .
    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  $\text{pmdl } (\text{set } ps) = \text{pmdl } G$  and  $G \subseteq \text{phull } (\text{set } ps)$ 
shows is-Groebner-basis  $(\text{set } (\text{Macaulay-list } ps))$ 
proof (simp only: GB-alt-3-finite[OF finite-set] pmdl-Macaulay-list, intro ballI impI)
  fix  $f$ 
  assume  $f \in \text{pmdl } (\text{set } ps)$ 
  also from  $\text{assms}(2)$  have  $\dots = \text{pmdl } G$  .
  finally have  $f \in \text{pmdl } G$  .
  assume  $f \neq 0$ 
  with  $\text{assms}(1)$   $\langle f \in \text{pmdl } G \rangle$  obtain  $g$  where  $g \in G$  and  $g \neq 0$  and  $\text{lt } g \text{ adds}_t$ 
 $\text{lt } f$ 
    by (rule GB-adds-lt)
    from  $\text{assms}(3)$   $\langle g \in G \rangle$  have  $g \in \text{phull } (\text{set } ps)$  ..
    from this  $\langle g \neq 0 \rangle$  obtain  $g'$  where  $g' \in \text{set } (\text{Macaulay-list } ps)$  and  $g' \neq 0$  and
 $\text{lt } g = \text{lt } g'$ 
    by (rule Macaulay-list-lt)
    show  $\exists g \in \text{set } (\text{Macaulay-list } ps). g \neq 0 \wedge \text{lt } g \text{ adds}_t \text{ lt } f$ 
    proof (rule, rule)
      from  $\langle \text{lt } g \text{ adds}_t \text{ lt } f \rangle$  show  $\text{lt } g' \text{ adds}_t \text{ lt } f$  by (simp only: lt g = lt g')
    qed fact+
qed

```

4.2 Bounds

```

context pm-powerprod
begin

```

```

context
  fixes  $X :: 'x \text{ set}$ 
  assumes  $\text{fin-}X$ : finite  $X$ 
begin

```

```

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

```

lemma *set-deg-shifts*:

```

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

```

proof –

```

from  $\text{fin-}X$  have finite  $(\text{deg-le-sect } X \text{ d0})$  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*:

```

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

```

```

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

lemma deg-shifts-superset: set fs  $\subseteq$  set (deg-shifts d fs)
proof -
  have set fs = ( $\bigcup f \in \text{set fs. } \{ \text{punit.monom-mult } 1 \ 0 \ f \}$ ) by simp
  also have  $\dots \subseteq$  set (deg-shifts d fs) unfolding set-deg-shifts using subset-refl
  proof (rule UN-mono)
    fix f
    assume f  $\in$  set fs
    have punit.monom-mult 1 0 f  $\in$  ( $\lambda t. \text{punit.monom-mult } 1 \ t \ f$ ) ‘ deg-le-sect X
    (d - poly-deg f)
    using zero-in-deg-le-sect by (rule imageI)
    thus  $\{ \text{punit.monom-mult } 1 \ 0 \ f \} \subseteq$  ( $\lambda t. \text{punit.monom-mult } 1 \ t \ f$ ) ‘ deg-le-sect
    X (d - poly-deg f)
    by simp
  qed
  finally show ?thesis .
qed

lemma deg-shifts-mono:
  assumes set fs  $\subseteq$  set gs
  shows set (deg-shifts d fs)  $\subseteq$  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))  $\subseteq$  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)  $\subseteq$  ideal (set (deg-shifts d fs))
      by (rule ideal.span-mono)
  qed

lemma thm-2-3-6:
  assumes set fs  $\subseteq P[X]$  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  $\subseteq P[X]$ 
    and 1:  $\bigwedge g. g \in G \implies \exists q. g = (\sum f \in \text{set fs. } q \ f * f) \wedge (\forall f. q \ f \in P[X] \wedge$ 
    poly-deg (q f * f)  $\leq b$ )
    by (rule is-GB-cofactor-boundE-finite-Polys) blast
  from this(1) show ?thesis
  proof (rule punit.Macaulay-list-is-GB)
    show G  $\subseteq$  phull (set (deg-shifts b fs)) (is -  $\subseteq$  ?H)
  proof
    fix g

```

```

assume  $g \in G$ 
hence  $\exists q. g = (\sum_{f \in \text{set } fs} q f * f) \wedge (\forall f. q f \in P[X] \wedge \text{poly-deg } (q f * f)$ 
 $\leq b)$  by (rule 1)
then obtain  $q$  where  $g: g = (\sum_{f \in \text{set } fs} q f * f)$  and  $\bigwedge f. q f \in P[X]$ 
and  $\bigwedge f. \text{poly-deg } (q f * f) \leq b$  by blast
show  $g \in ?H$  unfolding  $g$ 
proof (rule phull.span-sum)
  fix  $f$ 
  assume  $f \in \text{set } fs$ 
  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  $\langle \text{poly-deg } (q f * f) \leq b \rangle$  have  $\text{poly-deg } (q f) \leq b - \text{poly-deg } f$ 
by (simp add: poly-deg-times)
        with  $\langle q f \in P[X] \rangle$  have  $\text{keys } (q f) \subseteq \text{deg-le-sect } X \ (b - \text{poly-deg } f)$ 
by (rule keys-subset-deg-le-sectI)
        with finite-deg-le-sect[OF fin-X]
        have  $q f * f = (\sum_{t \in \text{deg-le-sect } X \ (b - \text{poly-deg } f)} \text{punit.monom-mult}$ 
(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  $\dots = (\sum_{t \in \text{deg-le-sect } X \ (b - \text{poly-deg } f)} \text{punit.monom-mult } 1 \ t \ f)$ 
by (simp add: punit.monom-mult-assoc punit.map-scale-eq-monom-mult)
        also have  $\dots = (\sum_{t \in \text{deg-le-sect } X \ (b - \text{poly-deg } f)} ((\lambda f0. (\text{lookup } (q f) (\text{punit.lp } f0 - \text{punit.lp } f)) \cdot f0) \circ$ 
 $(\lambda t. \text{punit.monom-mult } 1 \ t \ f)) \ t)$ 
using refl by (rule sum.cong) (simp add: punit.lt-monom-mult[OF  $\langle 1 \neq 0 \rangle \langle f \neq 0 \rangle$ ])
        also have  $\dots = (\sum_{f0 \in \text{set } (\text{deg-shifts } b \ [f])} (\text{lookup } (q f) (\text{punit.lp } f0 - \text{punit.lp } f)) \cdot f0)$ 
unfolding set-deg-shifts-singleton
proof (intro sum.reindex[symmetric] inj-onI)
  fix  $s \ t$ 
  assume  $\text{punit.monom-mult } 1 \ s \ f = \text{punit.monom-mult } 1 \ t \ f$ 
  thus  $s = t$  using  $\langle 1 \neq 0 \rangle \langle f \neq 0 \rangle$  by (rule punit.monom-mult-inj-2)
qed
finally have  $q f * f \in \text{phull } (\text{set } (\text{deg-shifts } b \ [f]))$ 
by (simp add: phull.sum-in-spanI)
also have  $\dots \subseteq ?H$  by (rule phull.span-mono, rule deg-shifts-mono, simp
add:  $\langle f \in \text{set } fs \rangle$ )
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  $1 \in \text{ideal } (\text{set fs}) \iff 1 \in \text{set } (\text{punit.Macaulay-list } (\text{deg-shifts } b \text{ fs}))$  (is
    ?L  $\iff$  ?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  $\langle ?L \rangle$  have  $1 \in \text{ideal } ?G$  by (simp add: punit.pmdl-Macaulay-list[simplified])
    moreover have  $1 \neq (0::\text{'a}) \Rightarrow_0 'a$  by simp
    ultimately obtain g where  $g \in ?G$  and  $g \neq 0$  and punit.lt g adds punit.lt
      (1::\text{'a})  $\Rightarrow_0 'a$ 
    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  $\langle g \in ?G \rangle \langle g \neq 0 \rangle$  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  $\neg t \preceq \text{punit.lt } g$  by (simp add: lp-g)
      with punit.lt-max-keys have  $t \notin \text{keys } g$  by blast
      with False show ?thesis by (simp add: lookup-one in-keys-iff)
    qed
  qed
  qed
  with  $\langle g \in ?G \rangle$  show  $1 \in ?G$  by simp
next
  assume ?R
  also have  $\dots \subseteq \text{phull } (\text{set } (\text{punit.Macaulay-list } (\text{deg-shifts } b \text{ fs})))$ 
    by (rule phull.span-superset)
  also have  $\dots = \text{phull } (\text{set } (\text{deg-shifts } b \text{ fs}))$  by (fact punit.phull-Macaulay-list)
  also have  $\dots \subseteq \text{ideal } (\text{set } (\text{deg-shifts } b \text{ fs}))$  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 \text{ideal } (set fs) \iff 1 \in \text{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  $\langle \text{Sign.add-const-constraint } (@\{const\text{-name } gbinomial\}, SOME @\{typ 'a::\{semidom-divide, semiring-char\} \Rightarrow nat \Rightarrow 'a\}) \rangle$ 

```

5.1 Inequalities

```

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

lemma binomial-plus-le:
  assumes  $0 < k$ 
  shows  $(m \text{ choose } k) + (n \text{ choose } k) \leq (m + n) \text{ choose } k$ 
proof -
  define k0 where  $k0 = k - 1$ 
  with assms have  $k: k = \text{Suc } k0$  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:  $2 * ((n + i) \text{ choose } k) \leq (n \text{ choose } k) + ((n + 2 * i) \text{ choose } k)$ 
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  $2 * (n + \text{Suc } i \text{ choose } \text{Suc } k0) = 2 * (n + i \text{ choose } k0) + 2 * (n + i \text{ choose } \text{Suc } k0)$ 
    by simp
    also have  $\dots \leq ((n + 2 * i \text{ choose } k0) + (\text{Suc } (n + 2 * i) \text{ choose } k0)) + ((n \text{ choose } \text{Suc } k0) + (n + 2 * i \text{ choose } \text{Suc } k0))$ 
    proof (rule add-mono)
      have  $n + i \text{ choose } k0 \leq n + 2 * i \text{ choose } k0$ 
      by (rule binomial-mono) simp
      moreover have  $n + 2 * i \text{ choose } k0 \leq \text{Suc } (n + 2 * i) \text{ choose } k0$ 
      by (rule binomial-mono) simp
      ultimately show  $2 * (n + i \text{ choose } k0) \leq (n + 2 * i \text{ choose } k0) + (\text{Suc } (n + 2 * i) \text{ choose } k0)$ 
      by simp
    qed (fact Suc)
    also have  $\dots = (n \text{ choose } \text{Suc } k0) + (n + 2 * \text{Suc } i \text{ choose } \text{Suc } k0)$  by simp
    finally show ?case .
  qed
qed

lemma gbinomial-int-mono:
  assumes  $0 \leq x$  and  $x \leq (y::int)$ 
  shows  $x \text{ choose } k \leq y \text{ choose } k$ 
proof -
  from assms have  $\text{nat } x \leq \text{nat } y$  by simp
  hence  $\text{nat } x \text{ choose } k \leq \text{nat } y \text{ choose } k$  by (rule binomial-mono)

```


hence $\text{int } (\text{nat } x \text{ choose } k) \leq \text{int } (\text{nat } y \text{ choose } k)$ **by** (*simp only: zle-int*)
 hence $\text{int } (\text{nat } x) \text{ gchoose } k \leq \text{int } (\text{nat } y) \text{ gchoose } k$ **by** (*simp only: int-binomial*)
 with *assms* **show** *?thesis* **by** *simp*
qed

lemma *gbinomial-int-plus-le*:

assumes $0 < k$ and $0 \leq x$ and $0 \leq (y::\text{int})$
 shows $(x \text{ gchoose } k) + (y \text{ gchoose } k) \leq (x + y) \text{ gchoose } k$
proof –
 from *assms*(1) **have** $(\text{nat } x \text{ choose } k) + (\text{nat } y \text{ choose } k) \leq \text{nat } x + \text{nat } y \text{ choose } k$
 by (*rule binomial-plus-le*)
 hence $\text{int } ((\text{nat } x \text{ choose } k) + (\text{nat } y \text{ choose } k)) \leq \text{int } (\text{nat } x + \text{nat } y \text{ choose } k)$
 by (*simp only: zle-int*)
 hence $(\text{int } (\text{nat } x) \text{ gchoose } k) + (\text{int } (\text{nat } y) \text{ gchoose } k) \leq \text{int } (\text{nat } x) + \text{int } (\text{nat } y) \text{ gchoose } k$
 by (*simp only: int-plus int-binomial*)
 with *assms*(2, 3) **show** *?thesis* **by** *simp*
qed

lemma *binomial-int-ineq-1*:

assumes $0 \leq x$ and $0 \leq (y::\text{int})$
 shows $2 * (x + y \text{ gchoose } k) \leq (x \text{ gchoose } k) + ((x + 2 * y) \text{ gchoose } k)$
proof –
 from *binomial-ineq-1* [of *nat x nat y k*]
 have $\text{int } (2 * (\text{nat } x + \text{nat } y \text{ choose } k)) \leq \text{int } ((\text{nat } x \text{ choose } k) + (\text{nat } x + 2 * \text{nat } y \text{ choose } k))$
 by (*simp only: zle-int*)
 hence $2 * (\text{int } (\text{nat } x) + \text{int } (\text{nat } y) \text{ gchoose } k) \leq (\text{int } (\text{nat } x) \text{ gchoose } k) + (\text{int } (\text{nat } x) + 2 * \text{int } (\text{nat } y) \text{ gchoose } k)$
 by (*simp only: int-binomial int-plus int-ops(7)*) *simp*
 with *assms* **show** *?thesis* **by** *simp*
qed

corollary *binomial-int-ineq-2*:

assumes $0 \leq y$ and $y \leq (x::\text{int})$
 shows $2 * (x \text{ gchoose } k) \leq (x - y \text{ gchoose } k) + (x + y \text{ gchoose } k)$
proof –
 from *assms*(2) **have** $0 \leq x - y$ **by** *simp*
 hence $2 * ((x - y) + y \text{ gchoose } k) \leq (x - y \text{ gchoose } k) + ((x - y + 2 * y) \text{ gchoose } k)$
 using *assms*(1) **by** (*rule binomial-int-ineq-1*)
 thus *?thesis* **by** *smt*
qed

corollary *binomial-int-ineq-3*:

assumes $0 \leq y$ and $y \leq 2 * (x::\text{int})$
 shows $2 * (x \text{ gchoose } k) \leq (y \text{ gchoose } k) + (2 * x - y \text{ gchoose } k)$
proof (*cases* $y \leq x$)

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

5.2 Backward Difference Operator

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

lemma *bw-diff-const* [*simp*]: $\text{bw-diff } (\lambda_. c) = (\lambda_. 0)$
 by (*rule ext*) (*simp add: bw-diff-def*)

lemma *bw-diff-id* [*simp*]: $\text{bw-diff } (\lambda x. x) = (\lambda_. 1)$
 by (*rule ext*) (*simp add: bw-diff-def*)

lemma *bw-diff-plus* [*simp*]: $\text{bw-diff } (\lambda x. f \ x + g \ x) = (\lambda x. \text{bw-diff } f \ x + \text{bw-diff } g \ x)$
 by (*rule ext*) (*simp add: bw-diff-def*)

lemma *bw-diff-uminus* [*simp*]: $\text{bw-diff } (\lambda x. - f \ x) = (\lambda x. - \text{bw-diff } f \ x)$
 by (*rule ext*) (*simp add: bw-diff-def*)

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

lemma *bw-diff-const-pow*: $(\text{bw-diff } \sim^k) (\lambda_. c) = (\text{if } k = 0 \text{ then } \lambda_. c \text{ else } (\lambda_. 0))$
 by (*induct k, simp-all*)

lemma *bw-diff-id-pow*:
 $(\text{bw-diff } \sim^k) (\lambda x. x) = (\text{if } k = 0 \text{ then } (\lambda x. x) \text{ else if } k = 1 \text{ then } (\lambda_. 1) \text{ else } (\lambda_. 0))$
 by (*induct k, simp-all*)

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

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

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

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

lemma *bw-diff-sum-pow* [*simp*]:
 $(bw\text{-diff} \sim k) (\lambda x. (\sum i \in I. f i x)) = (\lambda x. (\sum i \in I. (bw\text{-diff} \sim k) (f i) x))$
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 $\dots = (x + n - 1 gchoose k - 1) + ((x + n - 1) gchoose (Suc (k - 1)))$
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*:
 $(bw\text{-diff} \sim l) (\lambda x::int. (x + n) gchoose k) =$
 $(if l \leq k then (\lambda x. (x + n - int l) gchoose (k - l)) else (\lambda x. 0))$
proof -
have $*$: $l0 \leq k \implies (bw\text{-diff} \sim l0) (\lambda x::int. (x + n) gchoose k) = (\lambda x. (x + n - int l0) gchoose (k - l0))$
for $l0$
proof (*induct l0*)
case 0
show *?case* **by** *simp*
next
case ($Suc l0$)
from *Suc.prem*s **have** $0 < k - l0$ **and** $l0 \leq k$ **by** *simp-all*
from *this*(2) **have** $eq: (bw\text{-diff} \sim l0) (\lambda x. x + n gchoose k) = (\lambda x. x + n - int l0 gchoose (k - l0))$
by (*rule Suc.hyps*)
have $(bw\text{-diff} \sim Suc l0) (\lambda x. x + n gchoose k) = bw\text{-diff} (\lambda x. x + (n - int l0) gchoose (k - l0))$

```

    by (simp add: eq algebra-simps)
  also from ⟨0 < k - l0⟩ have ... = (λx. (x + (n - int l0) - 1) gchoose (k -
l0 - 1))
    by (rule bw-diff-gbinomial)
  also have ... = (λx. 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 ≤ k
  hence (l - k) + k = l and l - k ≠ 0 by simp-all
  hence (bw-diff ^^ l) (λx. x + n gchoose k) = (bw-diff ^^ ((l - k) + k)) (λx.
x + n gchoose k)
    by (simp only:)
  also have ... = (bw-diff ^^ (l - k)) (λ-. 1) by (simp add: * funpow-add)
  also from ⟨l - k ≠ 0⟩ have ... = (λ-. 0) by (simp add: bw-diff-const-pow)
  finally show (bw-diff ^^ l) (λx. 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** $q: \bigwedge a. \text{rat-of-int } (g \ a) = \text{poly } q \ (\text{rat-of-int } a)$
by (*rule poly-funE*, *blast*)
have $p = q$
proof (*rule ccontr*)
let $?A = \{a. \text{poly } p \ (\text{rat-of-int } a) = \text{poly } q \ (\text{rat-of-int } a)\}$
assume $p \neq q$
hence $p - q \neq 0$ **by** *simp*
hence *fin*: *finite* $\{x. \text{poly } (p - q) \ x = 0\}$ **by** (*rule poly-roots-finite*)
have *rat-of-int* ‘ $?A \subseteq \{x. \text{poly } (p - q) \ x = 0\}$ **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** $\bigwedge a. b \leq a \implies f \ a = g \ a$
shows $f = g$
using *assms*(1, 2)
proof (*rule poly-fun-eqI*)
have $\{b..\} \subseteq \{a. f \ a = g \ a\}$ **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** $\bigwedge a. b < a \implies f \ a = g \ a$
shows $f = g$
using *assms*(1, 2)
proof (*rule poly-fun-eqI*)
have $\{b<..\} \subseteq \{a. f \ a = g \ a\}$ **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* $(\lambda-. c)$
by (*rule poly-funI*[**where** $p = [: \text{rat-of-int } c:]$]) *simp*

lemma *poly-fun-id* [*simp*]: *poly-fun* $(\lambda x. x)$ *poly-fun id*

proof –

show *poly-fun* $(\lambda x. x)$ **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** $p: \bigwedge a. \text{rat-of-int } (f a) = \text{poly } p (\text{rat-of-int } a)$
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 $(\lambda x. - f x) \longleftrightarrow \text{poly-fun } f \text{ poly-fun } (- f) \longleftrightarrow \text{poly-fun } f$
proof –
show *poly-fun* $(\lambda x. - f x) \longleftrightarrow \text{poly-fun } f$
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* $(- f) \longleftrightarrow \text{poly-fun } f$ **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** $p: \bigwedge a. \text{rat-of-int } (f a) = \text{poly } p (\text{rat-of-int } a)$
by (*rule poly-funE*, *blast*)
from *assms*(2) **obtain** *q* **where** $q: \bigwedge a. \text{rat-of-int } (g a) = \text{poly } q (\text{rat-of-int } a)$
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** $p: \bigwedge a. \text{rat-of-int } (f a) = \text{poly } p (\text{rat-of-int } a)$
by (*rule poly-funE*, *blast*)
from *assms*(2) **obtain** *q* **where** $q: \bigwedge a. \text{rat-of-int } (g a) = \text{poly } q (\text{rat-of-int } a)$
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** $p: \bigwedge a. \text{rat-of-int } (f a) = \text{poly } p (\text{rat-of-int } a)$
by (*rule poly-funE*, *blast*)

from *assms*(2) **obtain** *q* **where** $q: \bigwedge a. \text{rat-of-int } (g \ a) = \text{poly } q \ (\text{rat-of-int } a)$
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** $\bigwedge a. c \text{ dvd } f \ a$
shows *poly-fun* $(\lambda x. f \ x \ \text{div } c)$
proof –
from *assms*(1) **obtain** *p* **where** $p: \bigwedge a. \text{rat-of-int } (f \ a) = \text{poly } p \ (\text{rat-of-int } a)$
by (*rule poly-funE*, *blast*)
let $?p = p * [:1 \ / \ \text{rat-of-int } c:]$
show *?thesis*
proof (*rule poly-funI*)
fix *a*
have $c \text{ dvd } f \ a$ **by** *fact*
hence $\text{rat-of-int } (f \ a \ \text{div } c) = \text{rat-of-int } (f \ a) \ / \ \text{rat-of-int } c$ **by** *auto*
also have $\dots = \text{poly } ?p \ (\text{rat-of-int } a)$ **by** (*simp add: p*)
finally show $\text{rat-of-int } (f \ a \ \text{div } c) = \text{poly } ?p \ (\text{rat-of-int } a)$.
qed
qed

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

lemma *poly-fun-comp*:
assumes *poly-fun f* **and** *poly-fun g*
shows *poly-fun* $(\lambda x. f \ (g \ x))$ **and** *poly-fun* $(f \circ g)$
proof –
from *assms*(1) **obtain** *p* **where** $p: \bigwedge a. \text{rat-of-int } (f \ a) = \text{poly } p \ (\text{rat-of-int } a)$
by (*rule poly-funE*, *blast*)
from *assms*(2) **obtain** *q* **where** $q: \bigwedge a. \text{rat-of-int } (g \ a) = \text{poly } q \ (\text{rat-of-int } a)$
by (*rule poly-funE*, *blast*)
show *poly-fun* $(\lambda x. f \ (g \ x))$ **by** (*rule poly-funI*[**where** $p=p \circ_p q$]) (*simp add: p q poly-pcompose*)
thus *poly-fun* $(f \circ g)$ **by** (*simp only: comp-def*)
qed

lemma *poly-fun-sum* [*simp*]: $(\bigwedge i. i \in I \implies \text{poly-fun } (f \ i)) \implies \text{poly-fun } (\lambda x. \sum_{i \in I. f \ i \ x})$
proof (*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 ∈ insert i I by simp
  hence poly-fun (f i) by (rule insert.prem)
  moreover have poly-fun (λx. ∑ i∈I. f i x)
  proof (rule insert.hyps)
    fix j
    assume j ∈ I
    hence j ∈ insert i I by simp
    thus poly-fun (f j) by (rule insert.prem)
  qed
  ultimately have poly-fun (λx. f i x + (∑ i∈I. f i x)) by (rule poly-fun-plus)
  with insert.hyps(1, 2) show ?case by simp
qed

lemma poly-fun-prod [simp]: (∧ i. i ∈ I ⇒ poly-fun (f i)) ⇒ poly-fun (λx.
(∏ 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 ∈ insert i I by simp
  hence poly-fun (f i) by (rule insert.prem)
  moreover have poly-fun (λx. ∏ i∈I. f i x)
  proof (rule insert.hyps)
    fix j
    assume j ∈ I
    hence j ∈ insert i I by simp
    thus poly-fun (f j) by (rule insert.prem)
  qed
  ultimately have poly-fun (λx. f i x * (∏ i∈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 ⇒ poly-fun (λx. pochhammer (f
x) k)
  by (simp add: pochhammer-prod)

lemma poly-fun-gbinomial [simp]: poly-fun f ⇒ poly-fun (λx. 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  $\longleftrightarrow (\forall p \in A. \text{is-monomial } p)$ 
```

```
lemma is-monomial-setI: ( $\bigwedge p. p \in A \implies \text{is-monomial } p$ )  $\implies$  is-monomial-set A
  by (simp add: is-monomial-set-def)
```

```
lemma is-monomial-setD: is-monomial-set A  $\implies p \in A \implies \text{is-monomial } p$ 
  by (simp add: is-monomial-set-def)
```

```
lemma is-monomial-set-subset: is-monomial-set B  $\implies A \subseteq B \implies$  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  $(c * d) v = \text{monom-mult } c \ t \ b$  by (simp add: b monom-mult-monomial)
    also from step.hyps(3) have  $\dots \in \text{pmdl } B$  by (rule monom-mult-in-pmdl)
    finally show monomial  $(c * d) v \in \text{pmdl } B$  .
  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  $p \in \text{pmdl } B$  and  $v \in \text{keys } (p::\Rightarrow_0 'b::\text{field})$ 
  shows monomial  $c \ v \in \text{pmdl } B$ 
proof -
  from assms(1, 2) have monomial  $(\text{lookup } p \ v) \ v \in \text{pmdl } B$  by (rule monomial-pmdl)
  hence monom-mult  $(c / \text{lookup } p \ v) \ 0 \ (\text{monomial } (\text{lookup } p \ v) \ v) \in \text{pmdl } B$ 
    by (rule pmdl-closed-monom-mult)
  with assms(3) show ?thesis by (simp add: monom-mult-monomial plus-zero in-keys-iff)
qed

```

end

context *ordered-term*
begin

lemma *keys-monomial-pmdl*:

```

  assumes is-monomial-set  $F$  and  $p \in \text{pmdl } F$  and  $t \in \text{keys } p$ 
  obtains  $f$  where  $f \in F$  and  $f \neq 0$  and  $lt \ f \ \text{adds}_t \ t$ 
  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  $\text{keys } f0 = \{lt \ f0\}$  and  $f0 \neq 0$  by (rule keys-monomial, rule monomial-not-0)
  from Poly-Mapping.keys-add step(6) have  $t \in \text{keys } p \cup \text{keys } (\text{monom-mult } c \ s \ f0)$  ..
  thus ?case
proof
  assume  $t \in \text{keys } p$ 
  from step(2)[OF - this] obtain  $f$  where  $f \in F$  and  $f \neq 0$  and  $lt \ f \ \text{adds}_t \ t$  by blast
  thus ?thesis by (rule step(5))
next
  assume  $t \in \text{keys } (\text{monom-mult } c \ s \ f0)$ 
  with keys-monom-mult-subset have  $t \in (\oplus) \ s \ \text{'keys } f0$  ..
  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 $\langle f0 \in F \rangle \langle f0 \neq 0 \rangle$ 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 $b \in B$ and $b \neq 0$ and $red\ \{b\}\ p\ q$
 proof -
 from assms obtain $b\ t$ where $b \in B$ and $red_single\ p\ q\ b\ t$ by (rule red-setE)
 from this(2) have $b \neq 0$ 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 $cp0: lookup\ p\ (s \oplus lt\ r) \neq 0$ and $q-def0: q = p - monom_mult\ (lookup\ p\ (s \oplus lt\ r) / lc\ r)\ s\ r$
 unfolding red-single-def by simp-all
 from assms(1) obtain $c\ t$ where $c \neq 0$ 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 $u = s \oplus t$
 from $q-def0$ have $q = p - monom_mult\ (lookup\ p\ u / c)\ s\ r$ unfolding u-def ltr lcr .
 also have $\dots = p - monomial\ ((lookup\ p\ u / c) * c)\ u$ unfolding u-def r-def monom-mult-monomial ..
 finally have $q-def: q = p - monomial\ (lookup\ p\ u)\ u$ using $\langle c \neq 0 \rangle$ by simp
 from $cp0$ have $lookup\ p\ u \neq 0$ unfolding u-def ltr .
 hence $u \in keys\ p$ by (simp add: in-keys-iff)

have $keys\ q = keys\ p - \{u\}$ unfolding q-def

proof (rule, rule)

fix x

assume $x \in keys\ (p - monomial\ (lookup\ p\ u)\ u)$

hence $lookup\ (p - monomial\ (lookup\ p\ u)\ u)\ x \neq 0$ by (simp add: in-keys-iff)

hence $a: lookup\ p\ x - lookup\ (monomial\ (lookup\ p\ u)\ u)\ x \neq 0$ unfolding lookup-minus .

hence $x \neq u$ unfolding lookup-single by auto

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

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

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

corollary $\text{is-red-monomial-monomial-setD}$:
 assumes $\text{is-monomial } p$ **and** $\text{is-monomial-set } B$ **and** $\text{is-red } B \ p$
 shows $\text{red } B \ p \ 0$
proof –
 from $\text{assms}(3)$ **obtain** q **where** $\text{red } B \ p \ q$ **by** (rule is-redE)
 moreover from $\text{assms}(1, 2)$ this **have** $q = 0$ **by** $(\text{rule red-monomial-monomial-setD})$
 ultimately show $?thesis$ **by** simp
qed

corollary $\text{is-red-monomial-monomial-set-in-pmdl}$:
 $\text{is-monomial } p \implies \text{is-monomial-set } B \implies \text{is-red } B \ p \implies p \in \text{pmdl } B$
by $(\text{intro red-rtrancp-0-in-pmdl r-into-rtrancp is-red-monomial-monomial-setD})$

corollary *red-rtranc1-monomial-monomial-set-cases*:
assumes *is-monomial* p **and** *is-monomial-set* B **and** $(\text{red } B)^{**} p \ q$
obtains $q = p \mid q = 0$
using *assms*(3)
proof (*induct* q *arbitrary*: *thesis rule*: *rtranc1p-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** $\text{red } B \ p \ z$ **by** *simp*
 with *assms*(1, 2) **have** $z = 0$ **by** (*rule red-monomial-monomial-setD*)
 thus $?thesis$ **by** (*rule step.prem*s)
 next
 assume $y = 0$
 from *step.hyps*(2) **have** $\text{is-red } B \ 0$ **unfolding** $\langle y = 0 \rangle$ **by** (*rule is-redI*)
 with *irred-0* **show** $?thesis$..
qed
qed

lemma *is-red-monomial-lt*:
assumes $0 \notin B$
shows $\text{is-red } (\text{monomial } (1::'b::\text{field}) \ 'lt \ 'B) = \text{is-red } B$
proof
 fix p
 let $?B = \text{monomial } (1::'b) \ 'lt \ 'B$
 show $\text{is-red } ?B \ p \longleftrightarrow \text{is-red } B \ p$
 proof
 assume $\text{is-red } ?B \ p$
 then obtain $f \ v$ **where** $f \in ?B$ **and** $v \in \text{keys } p$ **and** $\text{adds: } lt \ f \ \text{adds}_t \ v$ **by** (*rule is-red-addsE*)
 from *this*(1) **have** $lt \ f \in lt \ 'B$ **by** (*rule imageI*)
 also have $\dots = lt \ 'B$ **by** (*fact image-lt-monomial-lt*)
 finally obtain b **where** $b \in B$ **and** $eq: lt \ f = lt \ b$..
 note *this*(1)
 moreover from *this assms* **have** $b \neq 0$ **by** *blast*
 moreover note $\langle v \in \text{keys } p \rangle$
 moreover from *adds* **have** $lt \ b \ \text{adds}_t \ v$ **by** (*simp only: eq*)
 ultimately show $\text{is-red } B \ p$ **by** (*rule is-red-addsI*)
 next
 assume $\text{is-red } B \ p$
 then obtain $b \ v$ **where** $b \in B$ **and** $v \in \text{keys } p$ **and** $\text{adds: } lt \ b \ \text{adds}_t \ v$ **by** (*rule is-red-addsE*)
 from *this*(1) **have** $lt \ b \in lt \ 'B$ **by** (*rule imageI*)
 also from *image-lt-monomial-lt* **have** $\dots = lt \ 'B$ **by** (*rule sym*)
 finally obtain f **where** $f \in ?B$ **and** $eq: lt \ b = lt \ f$..
 note *this*(1)

```

    moreover from this have  $f \neq 0$  by (auto simp: monomial-0-iff)
    moreover note  $\langle v \in \text{keys } p \rangle$ 
    moreover from adds have  $\text{lt } f \text{ adds}_t v$  by (simp only: eq)
    ultimately show  $\text{is-red } ?B \ p$  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  $f \in \text{pmdl } G$ 
  thus  $(\text{red } G)^{**} f = 0$ 
  proof (induct  $f$  rule: poly-mapping-plus-induct)
    case 1
    show ?case ..
  next
    case (2  $f \ c \ t$ )
    let  $?f = \text{monomial } c \ t + f$ 
    from 2(1) have  $t \in \text{keys } (\text{monomial } c \ t)$  by simp
    from this 2(2) have  $t \in \text{keys } ?f$  by (rule in-keys-plusI1)
    with assms  $\langle ?f \in \text{pmdl } G \rangle$  obtain  $g$  where  $g \in G$  and  $g \neq 0$  and  $\text{lt } g \text{ adds}_t t$ 
    by (rule keys-monomial-pmdl)
    from this(1) have  $\text{red } G \ ?f \ f$ 
    proof (rule red-setI)
      from  $\langle \text{lt } g \text{ adds}_t t \rangle$  have  $\text{component-of-term } (\text{lt } g) = \text{component-of-term } t$ 
    and  $\text{lp } g \text{ adds } \text{pp-of-term } t$ 
      by (simp-all add: adds-term-def)
      from this have  $\text{eq: } (\text{pp-of-term } t - \text{lp } g) \oplus \text{lt } g = t$ 
      by (simp add: adds-minus splus-def term-of-pair-pair)
      moreover from 2(2) have  $\text{lookup } ?f \ t = c$  by (simp add: lookup-add
in-keys-iff)
      ultimately show red-single  $(\text{monomial } c \ t + f) \ f \ g \ (\text{pp-of-term } t - \text{lp } g)$ 
    proof (simp add: red-single-def  $\langle g \neq 0 \rangle \langle t \in \text{keys } ?f \rangle$  2(1))
      from  $\langle g \neq 0 \rangle$  have  $\text{lc } g \neq 0$  by (rule lc-not-0)
      hence  $\text{monomial } c \ t = \text{monom-mult } (c / \text{lc } g) \ (\text{pp-of-term } t - \text{lp } g)$ 
      (monomial  $(\text{lc } g) \ (\text{lt } g)$ )
      by (simp add: monom-mult-monomial eq)
      moreover from assms  $\langle g \in G \rangle$  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::(- ⇒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' addst 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 addst 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' addst lt f}
    from dgrad fin-comps have almost-full-on (addst) ?A by (rule Dickson-term)
    moreover have transp-on ?A (addst) by (auto intro: transp-onI dest: adds-term-trans)
    ultimately have wfp-on (strict (addst)) ?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
  qed

```

finally have $lt\ f \in lt\ 'F'$.
 with *adds-term-refl* show *?thesis* by *fastforce*
 qed
 moreover have $lt\ ' ?Q \subseteq ?A$
 proof
 fix s
 assume $s \in lt\ ' ?Q$
 then obtain q where $q \in ?Q$ and $s: s = lt\ q$..
 from *this(1)* have $q \in F'$ by *simp*
 hence $q \in F - \{0\}$ using $\langle F' \subseteq F - \{0\} \rangle$..
 hence $q \in F$ and $q \neq 0$ by *simp-all*
 from *this(1)* *F-sub* have $q \in dgrad\text{-}p\text{-}set\ d\ m$..
 from $\langle q \neq 0 \rangle$ have $lt\ q \in keys\ q$ by (rule *lt-in-keys*)
 hence *pp-of-term* $(lt\ q) \in pp\text{-}of\text{-}term\ ' keys\ q$ by (rule *imageI*)
 also from $\langle q \in dgrad\text{-}p\text{-}set\ d\ m \rangle$ have $\dots \subseteq dgrad\text{-}set\ d\ m$ by (simp *add:*
dgrad-p-set-def)
 finally have $1: pp\text{-}of\text{-}term\ s \in dgrad\text{-}set\ d\ m$ by (simp only: *s*)
 from $\langle lt\ q \in keys\ q \rangle \langle q \in F \rangle$ have $lt\ q \in Keys\ F$ by (rule *in-KeysI*)
 hence *component-of-term* $s \in ?K$ unfolding *s* by (rule *imageI*)
 with 1 show $s \in ?A$ by *simp*
 qed
 ultimately obtain t where $t \in lt\ ' ?Q$ and $t\text{-}min: \bigwedge s. strict\ (adds_t)\ s\ t \implies$
 $s \notin lt\ ' ?Q$
 by (rule *wfp-onE-min*) *blast*
 from *this(1)* obtain g where $g \in ?Q$ and $t: t = lt\ g$..
 from *this(1)* have $g \in F'$ and *adds:* $lt\ g\ adds_t\ lt\ f$ by *simp-all*
 show *?thesis*
 proof (rule *that*)
 {
 fix f'
 assume $f' \in F'$
 assume $lt\ f'\ adds_t\ lt\ g$
 hence $lt\ f'\ adds_t\ lt\ f$ using *adds* by (rule *adds-term-trans*)
 with $\langle f' \in F' \rangle$ have $f' \in ?Q$ by *simp*
 hence $lt\ f' \in lt\ ' ?Q$ by (rule *imageI*)
 with *t-min* have $\neg strict\ (adds_t)\ (lt\ f')\ (lt\ g)$ unfolding *t* by *blast*
 with $\langle lt\ f'\ adds_t\ lt\ g \rangle$ have $lt\ g\ adds_t\ lt\ f'$ by *blast*
 with $\langle lt\ f'\ adds_t\ lt\ g \rangle$ have $lt\ f' = lt\ g$ by (rule *adds-term-antisym*)
 with $\langle inj\text{-}on\ lt\ F' \rangle$ have $f' = g$ using $\langle f' \in F' \rangle \langle g \in F' \rangle$ by (rule *inj-onD*)
 }
 with $\langle g \in F' \rangle$ show $g \in G$ by (simp *add:* *G-def*)
 qed *fact*
 qed
 have *2:* *is-red* $G\ q$ if $q \in pmdl\ F$ and $q \neq 0$ for q
 proof -
 from *that(2)* have $keys\ q \neq \{\}$ by *simp*
 then obtain t where $t \in keys\ q$ by *blast*
 with *F-monom* *that(1)* obtain f where $f \in F$ and $f \neq 0$ 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 $\langle g \in G \rangle \langle G \subseteq F - \{0\} \rangle$ have $g \in F - \{0\}$..
 hence $g \neq 0$ by simp
 with $\langle g \in G \rangle$ show ?thesis using $\langle t \in keys\ q \rangle$ ** by (rule is-red-addsI)
 qed
 from $\langle G \subseteq F - \{0\} \rangle$ 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 $\langle G \subseteq F \rangle$ 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 $\langle pmdl\ G \subseteq pmdl\ F \rangle$..
 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 $\langle G \subseteq F \rangle$..
 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 $\langle t \in keys\ g \rangle$ have $t = lt\ g$ by (simp add: keys- g)
 with $\langle g \in G \rangle \langle g' \in F' \rangle$ adds have $g' = g$ by (simp add: G -def)
 with $\langle g' \neq g \rangle$ 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) $\langle G \subseteq F - \{0\} \rangle$ 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 $f \in \text{pmdl } F$ **by** (rule *pmdl.span-base*)
note *dgrad*
moreover from $\langle G \subseteq F \rangle$ *F-sub* **have** $G \subseteq \text{dgrad-p-set } d \ m$ **by** (rule *sub-set-trans*)
moreover note $\langle \text{pmdl } G \subseteq \text{pmdl } F \rangle$ 2 $\langle f \in \text{pmdl } F \rangle$
moreover from $\langle f \in F \rangle$ *F-sub* **have** $f \in \text{dgrad-p-set } d \ m$..
ultimately have $(\text{red } G)^{**} f \ 0$ **by** (rule *is-red-implies-0-red-dgrad-p-set*)
thus $f \in \text{pmdl } G$ **by** (rule *red-rtrancplp-0-in-pmdl*)
qed
qed fact
ultimately have *reduced-GB* $F = \text{monic } 'G$ **by** (rule *reduced-GB-unique-dgrad-p-set*)
also from $\langle G \subseteq F \rangle$ **have** $\dots \subseteq \text{monic } 'F$ **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 $g \in \text{reduced-GB } F$
also have $\dots \subseteq \text{monic } 'F$ **by** (fact *reduced-GB-subset-monic-dgrad-p-set*)
finally obtain f **where** $f \in F$ **and** $g: g = \text{monic } f$..
from *F-monom* *this(1)* **have** *is-monomial* f **by** (rule *is-monomial-setD*)
hence $\text{card } (\text{keys } f) = 1$ **by** (simp only: *is-monomial-def*)
hence $f \neq 0$ **by** *auto*
hence $\text{lc } f \neq 0$ **by** (rule *lc-not-0*)
hence $1 / \text{lc } f \neq 0$ **by** *simp*
hence $\text{keys } g = (\oplus) \ 0 \ ' \ \text{keys } f$ **by** (simp add: *keys-monom-mult monic-def g*)
also from *refl* **have** $\dots = (\lambda x. x) \ ' \ \text{keys } f$ **by** (rule *image-cong*) (simp only: *splus-zero*)
finally show *is-monomial* g **using** $\langle \text{card } (\text{keys } f) = 1 \rangle$ **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* $'S \subseteq \text{dgrad-set } d \ m$
shows *is-red* (*reduced-GB* (monomial 1 $'S$)) = *is-red* (monomial (1::'b::field) $'S$)

proof

fix p
let $?F = \text{monomial } (1::'b) \ 'S$
from *assms(1)* **have** 1: *finite* (component-of-term $' \text{Keys } ?F$) **by** (simp add: *Keys-def*)
moreover from *assms(2)* **have** 2: $?F \subseteq \text{dgrad-p-set } d \ m$ **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 \subseteq *monic* ‘ ?F **by** (rule *reduced-GB-subset-monic-dgrad-p-set*)
also have $\dots =$?F **by** (auto simp: *monic-def intro!: image-eqI*)
finally have \exists : *reduced-GB* ?F \subseteq ?F .
show *is-red* (*reduced-GB* ?F) $p \longleftrightarrow$ *is-red* ?F p
proof
 assume *is-red* (*reduced-GB* ?F) p
 thus *is-red* ?F p **using** \exists **by** (rule *is-red-subset*)
next
 assume *is-red* ?F p
 then obtain $f\ v$ **where** $f \in$?F **and** $v \in$ *keys* p **and** $f \neq 0$ **and** *adds1*: *lt* f *adds_t* v
 by (rule *is-red-addsE*)
 from *this*(1) **have** $f \in$ *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 $\langle f \in$ *pmdl* ?F \rangle *dgrad* 1 2 **have** $f \in$ *pmdl* (*reduced-GB* ?F)
 by (simp only: *reduced-GB-pmdl-dgrad-p-set*)
 ultimately obtain g **where** $g \in$ *reduced-GB* ?F **and** $g \neq 0$ **and** *lt* g *adds_t* *lt* f
 using $\langle f \neq 0 \rangle$ **by** (rule *GB-adds-lt*)
 from *this*(3) *adds1* **have** *lt* g *adds_t* v **by** (rule *adds-term-trans*)
 with $\langle g \in$ *reduced-GB* ?F \rangle $\langle g \neq 0 \rangle$ $\langle v \in$ *keys* $p \rangle$ **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** $G \subseteq$ *dgrad-p-set* $d\ m$ **and** $0 \notin G$
 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 \subseteq *Keys* G **by** (auto intro: *lt-in-keys in-KeysI*)
 hence *component-of-term* ‘ ?S \subseteq *component-of-term* ‘ *Keys* G
 and *: *pp-of-term* ‘ ?S \subseteq *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 \subseteq *dgrad-set* $d\ m$
 proof (rule *subset-trans*)
 from *assms*(2) **show** *pp-of-term* ‘ *Keys* $G \subseteq$ *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** $\dots =$ *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  $F \subseteq \text{dgrad-p-set } d \ m$ 
  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  $f: f = \text{monomial } 1 \ (\text{lt } g)$  by blast
      assume is-red (?F - {f}) f
      then obtain f' v where  $f' \in ?F - \{f\}$  and  $v \in \text{keys } f$  and  $f' \neq 0$  and
adds1: lt f' addst v
      by (rule is-red-addsE)
      from this(1) have  $f' \in ?F$  and  $f' \neq f$  by simp-all
      from this(1) obtain g' where  $g' \in ?G$  and  $f': f' = \text{monomial } 1 \ (\text{lt } g')$  by
blast
      from  $\langle v \in \text{keys } f \rangle$  have  $v: v = \text{lt } g$  by (simp add: f)
      from  $\langle g \in ?G \rangle$  have  $\neg \text{is-red } (?G - \{g\}) \ g$  by (rule is-auto-reducedD)
      moreover have is-red (?G - {g}) g
      proof (rule is-red-addsI)
        from  $\langle g' \in ?G \rangle \langle f' \neq f \rangle$  show  $g' \in ?G - \{g\}$  by (auto simp: f f')
      next
        from  $\langle g' \in ?G \rangle \langle 0 \notin ?G \rangle$  show  $g' \neq 0$  by blast
      next
        from  $\langle g \in ?G \rangle \langle 0 \notin ?G \rangle$  have  $g \neq 0$  by blast
        thus lt g  $\in \text{keys } g$  by (rule lt-in-keys)
      next
        from adds1 show adds2: lt g' addst 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 ∪ {c}) = card (insert c B) by simp
  also have ... = Suc (card (B - {c}))
    by (rule card.insert-remove, rule finite-subset, fact ⟨B ⊆ A⟩, fact)
  finally show card (B ∪ {c}) = Suc n unfolding ⟨B - {c} = B⟩ card .
next
show B ∪ {c} ⊆ A unfolding Un-subset-iff
proof (intro conjI, fact)
  from ⟨c ∈ ?C⟩ show {c} ⊆ A by auto
qed
qed
qed

```

```

lemma card-2-E-1:
  assumes card A = 2 and x ∈ A
  obtains y where x ≠ y and A = {x, y}
proof -
  have A - {x} ≠ {}
  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 ∈ A - {x} by auto
  hence y ∈ A and x ≠ 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 ∈ A⟩ ⟨y ∈ A⟩ show {x, y} ⊆ A by simp
    next
      from ⟨x ≠ y⟩ show card A ≤ 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 ≠ y and A = {x, y}
proof -
  from assms have A ≠ {} by auto
  then obtain x where x ∈ A by blast
  with assms obtain y where x ≠ y and A = {x, y} by (rule card-2-E-1)
  thus ?thesis ..

```

qed

8.2 Sums

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

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

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

8.3 count-list

lemma *count-list-gr-1-E*:
assumes $1 < \text{count-list } xs \, x$
obtains $i \, j$ **where** $i < j$ **and** $j < \text{length } xs$ **and** $xs ! i = x$ **and** $xs ! j = x$
proof –
from *assms* **have** $\text{count-list } xs \, x \neq 0$ **by** *simp*
hence $x \in \text{set } xs$ **by** (*simp only: count-list-0-iff not-not*)
then obtain $ys \, zs$ **where** $xs = ys @ x \# zs$ **and** $x \notin \text{set } ys$ **by** (*meson split-list-first*)
hence $\text{count-list } xs \, x = \text{Suc } (\text{count-list } zs \, x)$ **by** (*simp*)
with *assms* **have** $\text{count-list } zs \, x \neq 0$ **by** *simp*
hence $x \in \text{set } zs$ **by** (*simp only: count-list-0-iff not-not*)
then obtain j **where** $j < \text{length } zs$ **and** $x = zs ! j$ **by** (*metis in-set-conv-nth*)
show *?thesis*
proof
show $\text{length } ys < \text{length } ys + \text{Suc } j$ **by** *simp*
next
from $\langle j < \text{length } zs \rangle$ **show** $\text{length } ys + \text{Suc } j < \text{length } xs$ **by** (*simp add: xs*)
next
show $xs ! \text{length } ys = x$ **by** (*simp add: xs*)
next
show $xs ! (\text{length } ys + \text{Suc } j) = x$
by (*simp only: xs \langle x = zs ! j \rangle nth-append-length-plus nth-Cons-Suc*)
qed
qed

8.4 listset

lemma *listset-Cons*: $\text{listset } (x \# xs) = (\bigcup_{y \in x.} (\#) \, y \, ' \text{listset } xs)$
by (*auto simp: set-Cons-def*)

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

```

by (simp add: set-Cons-def)

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

lemma listsetI:
   $\text{length } ys = \text{length } xs \implies (\bigwedge i. i < \text{length } xs \implies ys ! i \in xs ! i) \implies ys \in \text{listset } xs$ 
  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  $ys \in \text{listset } xs$ 
  shows  $\text{length } ys = \text{length } xs$  and  $\bigwedge i. i < \text{length } xs \implies ys ! i \in xs ! i$ 
proof -
  from assms have  $\text{length } ys = \text{length } xs \wedge (\forall i < \text{length } xs. ys ! i \in xs ! i)$ 
  proof (induct xs arbitrary: ys)
    case Nil
    thus ?case by simp
  next
    case (Cons x xs)
    from Cons.premis obtain  $y \ ys'$  where  $y \in x$  and  $ys' \in \text{listset } xs$  and  $ys = y \# ys'$ 
    by (rule listset-ConsE)
    from this(2) have  $\text{length } ys' = \text{length } xs \wedge (\forall i < \text{length } xs. ys' ! i \in xs ! i)$  by
    (rule Cons.hyps)
    hence 1:  $\text{length } ys' = \text{length } xs$  and 2:  $\bigwedge i. i < \text{length } xs \implies ys' ! i \in xs ! i$ 
  by simp-all
  show ?case
  proof (intro conjI allI impI)
    fix i
    assume  $i < \text{length } (x \# xs)$ 
    show  $ys ! i \in (x \# xs) ! i$ 
    proof (cases i)
      case 0
      with  $\langle y \in x \rangle$  show ?thesis by (simp add: ys)
    next
      case (Suc j)
      with  $\langle i < \text{length } (x \# xs) \rangle$  have  $j < \text{length } xs$  by simp
      hence  $ys' ! j \in xs ! j$  by (rule 2)
      thus ?thesis by (simp add: ys  $\langle i = \text{Suc } j \rangle$ )
    qed
  qed (simp add: ys 1)
qed
thus  $\text{length } ys = \text{length } xs$  and  $\bigwedge i. i < \text{length } xs \implies ys ! i \in xs ! i$  by simp-all
qed

```



```

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

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

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

lemma listset-doubletonE:
  assumes  $ys \in \text{listset } [A, B]$ 
  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:
   $ys1 \in \text{listset } xs1 \implies ys2 \in \text{listset } xs2 \implies ys = ys1 @ ys2 \implies ys \in \text{listset } (xs1 @ xs2)$ 
  by (induct xs1 arbitrary: ys ys1 ys2)
  (simp, auto simp del: listset.simps elim!: listset-ConsE intro!: listset-ConsI)

lemma listset-appendE:
  assumes  $ys \in \text{listset } (xs1 @ xs2)$ 
  obtains  $ys1\ ys2$  where  $ys1 \in \text{listset } xs1$  and  $ys2 \in \text{listset } xs2$  and  $ys = ys1 @ ys2$ 
  using assms
proof (induct xs1 arbitrary: thesis ys)
  case Nil
  have  $[] \in \text{listset } []$  by simp
  moreover from Nil(2) have  $ys \in \text{listset } xs2$  by simp
  ultimately show ?case by (rule Nil) simp
next
  case (Cons x xs1)
  from Cons.prem1 have  $ys \in \text{listset } (x \# (xs1 @ xs2))$  by simp
  then obtain  $y\ ys'$  where  $y \in x$  and  $ys' \in \text{listset } (xs1 @ xs2)$  and  $ys = y \# ys'$ 
  by (rule listset-ConsE)
  from - this(2) obtain  $ys1\ ys2$  where  $ys1: ys1 \in \text{listset } xs1$  and  $ys2 \in \text{listset } xs2$ 
  and  $ys': ys' = ys1 @ ys2$  by (rule Cons.hyps)
  show ?case
  proof (rule Cons.prem1)
    from  $\langle y \in x \rangle$  ys1 refl show  $y \# ys1 \in \text{listset } (x \# xs1)$  by (rule listset-ConsI)
  next
    show  $ys = (y \# ys1) @ ys2$  by (simp add: ys ys')
  qed fact

```

qed

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

lemma *listset-map-imageE*:
assumes $ys \in \text{listset } (\text{map } ((\cdot) f) \text{ } xs)$
obtains $ys' \text{ where } ys' \in \text{listset } xs \text{ and } ys = \text{map } f \text{ } ys'$
using *assms*
proof (*induct xs arbitrary: thesis ys*)
case *Nil*
from *Nil(2)* **have** $ys = \text{map } f \text{ } []$ **by** *simp*
with - **show** *?case* **by** (*rule Nil*) *simp*
next
case (*Cons x xs*)
from *Cons.prem(2)* **have** $ys \in \text{listset } (f \text{ } x \# \text{map } ((\cdot) f) \text{ } xs)$ **by** *simp*
then obtain $y \text{ } ys' \text{ where } y \in f \text{ } x \text{ and } ys' \in \text{listset } (\text{map } ((\cdot) f) \text{ } xs)$ **and** $ys: ys = y \# ys'$
by (*rule listset-ConsE*)
from - *this(2)* **obtain** $ys1 \text{ where } ys1: ys1 \in \text{listset } xs \text{ and } ys': ys' = \text{map } f \text{ } ys1$
by (*rule Cons.hyps*)
from $\langle y \in f \text{ } x \rangle$ **obtain** $y1 \text{ where } y1 \in x \text{ and } y: y = f \text{ } y1 \text{ ..}$
show *?case*
proof (*rule Cons.prem(1)*)
from $\langle y1 \in x \rangle$ *ys1 refl* **show** $y1 \# ys1 \in \text{listset } (x \# xs)$ **by** (*rule listset-ConsI*)
qed (*simp add: ys ys' y*)
qed

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

fix i
assume $i < \text{length } xs'$

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

lemma *listset-closed-map*:

assumes $ys \in \text{listset } xs$ **and** $\bigwedge x y. x \in \text{set } xs \implies y \in x \implies f y \in x$
 shows $\text{map } f \, ys \in \text{listset } xs$
 using *assms*
proof (*induct xs arbitrary: ys*)
 case *Nil*
 from *Nil(1)* show *?case* **by** *simp*
next
 case (*Cons x xs*)
 from *Cons.prem(1)* obtain $y \, ys'$ where $y \in x$ **and** $ys' \in \text{listset } xs$ **and** $ys: ys$
 = $y \# ys'$
 by (*rule listset-ConsE*)
 show *?case*
proof (*rule listset-ConsI*)
 from $\langle y \in x \rangle$ show $f y \in x$ **by** (*rule Cons.prem*) *simp*
next
 show $\text{map } f \, ys' \in \text{listset } xs$
proof (*rule Cons.hyps*)
 fix $x0 \, y0$
 assume $x0 \in \text{set } xs$
 hence $x0 \in \text{set } (x \# xs)$ **by** *simp*
 moreover assume $y0 \in x0$
 ultimately show $f y0 \in x0$ **by** (*rule Cons.prem*)
qed fact
qed (*simp add: ys*)
qed

lemma *listset-closed-map2*:

assumes $ys1 \in \text{listset } xs$ **and** $ys2 \in \text{listset } xs$
 and $\bigwedge x y1 y2. x \in \text{set } xs \implies y1 \in x \implies y2 \in x \implies f y1 y2 \in x$
 shows $\text{map2 } f \, ys1 \, ys2 \in \text{listset } xs$

```

using assms
proof (induct xs arbitrary: ys1 ys2)
  case Nil
  from Nil(1) show ?case by simp
next
  case (Cons x xs)
  from Cons.prem1 obtain y1 ys1' where y1 ∈ x and ys1' ∈ listset xs and
ys1: ys1 = y1 # ys1'
  by (rule listset-ConsE)
  from Cons.prem2 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.prem1) 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.prem2)
  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** $\bigwedge i. i < \text{length } ys \implies xs ! i \subseteq ys ! i$
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 $0 < \text{length } (y \# ys)$ **by** *simp*
hence $(x \# xs) ! 0 \subseteq (y \# ys) ! 0$ **by** (*rule Cons.prem1*)
hence $x \subseteq y$ **by** *simp*
with *⟨z ∈ x⟩* **have** $z \in y$..

```

moreover from  $zs$  have  $zs \in \text{listset } ys$ 
proof
  show  $\text{listset } xs \subseteq \text{listset } ys$ 
  proof (rule Cons.hyps)
    fix  $i$ 
    assume  $i < \text{length } ys$ 
    hence  $\text{Suc } i < \text{length } (y \# ys)$  by simp
    hence  $(x \# xs) ! \text{Suc } i \subseteq (y \# ys) ! \text{Suc } i$  by (rule Cons.prem)
    thus  $xs ! i \subseteq ys ! i$  by simp
  qed
qed
ultimately show  $zs' \in \text{listset } (y \# ys)$  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  $\Rightarrow$  'a::comm-monoid-add set list  $\Rightarrow$  bool
  where direct-decomp  $A$   $ss \iff \text{bij-betw sum-list (listset } ss) A$ 

```

```

lemma direct-decompI:
   $\text{inj-on sum-list (listset } ss) \implies \text{sum-list ' listset } ss = A \implies \text{direct-decomp } A ss$ 
  by (simp add: direct-decomp-def bij-betw-def)

```

```

lemma direct-decompI-alt:
   $(\bigwedge qs. qs \in \text{listset } ss \implies \text{sum-list } qs \in A) \implies (\bigwedge a. a \in A \implies \exists ! qs \in \text{listset } ss. a$ 
   $= \text{sum-list } qs) \implies$ 
   $\text{direct-decomp } A ss$ 
  by (auto simp: direct-decomp-def intro!: bij-betwI') blast

```

```

lemma direct-decompD:
  assumes direct-decomp  $A$   $ss$ 
  shows  $qs \in \text{listset } ss \implies \text{sum-list } qs \in A$  and  $\text{inj-on sum-list (listset } ss)$ 

```

```

    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 ∈ A
  obtains qs where qs ∈ 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 ⇒ qs ∈ listset ss ⇒ qs' ∈ listset ss ⇒ sum-list qs =
  sum-list qs' ⇒
    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 ∈ listset [A]
  then obtain q where q ∈ A and qs = [q] by (rule listset-singletonE)
  thus sum-list qs ∈ A by simp
next
  fix a
  assume a ∈ A
  show ∃!qs∈listset [A]. a = sum-list qs
proof (intro ex1I conjI allI impI)
  from ⟨a ∈ A⟩ refl show [a] ∈ listset [A] by (rule listset-singletonI)
next
  fix qs
  assume qs ∈ listset [A] ∧ a = sum-list qs
  hence a: a = sum-list qs and qs ∈ 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 {..

```

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

lemma *direct-decomp-perm:*

assumes *direct-decomp* $A \text{ } ss1$ **and** $\text{mset } ss1 = \text{mset } ss2$

shows *direct-decomp* $A \text{ } ss2$

proof –

from *assms*(2) **have** $\text{len-ss1: } \text{length } ss1 = \text{length } ss2$

using *mset-eq-length* **by** *blast*

from *assms*(2) **obtain** f **where** $\langle f \text{ permutes } \{..\text{length } ss2\} \rangle$

$\langle \text{permute-list } f \text{ } ss2 = ss1 \rangle$

by (rule *mset-eq-permutation*)

then have $f\text{-bij: } \text{bij-betw } f \{..\text{length } ss2\} \{..\text{length } ss1\}$

and $f: \bigwedge i. i < \text{length } ss2 \implies ss1 ! i = ss2 ! f i$

by (*auto simp add: permutes-imp-bij permute-list-nth*)

define g **where** $g = \text{inv-into } \{..\text{length } ss2\} f$

from $f\text{-bij}$ **have** $g\text{-bij: } \text{bij-betw } g \{..\text{length } ss1\} \{..\text{length } ss2\}$

unfolding $g\text{-def len-ss1}$ **by** (rule *bij-betw-inv-into*)

have $f\text{-g: } f (g i) = i \text{ if } i < \text{length } ss1$ **for** i

proof –

from *that f-bij* **have** $i \in f \text{ } \{..\text{length } ss2\}$ **by** (*simp add: bij-betw-def len-ss1*)

thus $?thesis$ **by** (*simp only: f-inv-into-f g-def*)

qed

have $g\text{-f: } g (f i) = i \text{ if } i < \text{length } ss2$ **for** i

proof –

from $f\text{-bij}$ **have** $\text{inj-on } f \{..\text{length } ss2\}$ **by** (*simp only: bij-betw-def*)

moreover from *that* **have** $i \in \{..\text{length } ss2\}$ **by** *simp*

ultimately show $?thesis$ **by** (*simp add: g-def*)

qed

have $g: ss2 ! i = ss1 ! g i \text{ if } i < \text{length } ss1$ **for** i

proof –

from *that* **have** $i \in \{..\text{length } ss2\}$ **by** (*simp add: len-ss1*)

hence $g i \in g \text{ } \{..\text{length } ss2\}$ **by** (rule *imageI*)

also from $g\text{-bij}$ **have** $\dots = \{..\text{length } ss2\}$ **by** (*simp only: len-ss1 bij-betw-def*)

finally have $g i < \text{length } ss2$ **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 ∈ listset ss2
  then obtain qs1 where qs1-in: qs1 ∈ listset ss1 and len-qs1: length qs1 =
length qs2
    and *:  $\bigwedge i. i < \text{length } qs2 \implies qs1 ! i = qs2 ! f i$  using f-bij f by (rule
listset-permE) blast+
    from ⟨qs2 ∈ listset ss2⟩ have length qs2 = length ss2 by (rule listsetD)
    with f-bij have bij-betw f {.. $\text{length } qs1$ } {.. $\text{length } qs2$ } by (simp only:
len-qs1 len-ss1)
    hence mset qs1 = mset qs2 using * by (rule mset-bij) (simp only: len-qs1)
    hence sum-list qs2 = sum-list qs1 by (simp flip: sum-mset-sum-list)
    also from assms(1) qs1-in have ... ∈ A by (rule direct-decompD)
    finally show sum-list qs2 ∈ A .
next
  fix a
  assume a ∈ A
  with assms(1) obtain qs where a: a = sum-list qs and qs-in: qs ∈ listset ss1
  by (rule direct-decompE)
  from qs-in obtain qs2 where qs2-in: qs2 ∈ listset ss2 and len-qs2: length qs2
= length qs
    and 1:  $\bigwedge i. i < \text{length } qs \implies qs2 ! i = qs ! g i$  using g-bij g by (rule
listset-permE) blast+
    show  $\exists ! qs \in \text{listset } ss2. a = \text{sum-list } qs$ 
    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-betw g {.. $\text{length } qs2$ } {.. $\text{length } qs$ } 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-list qs2 by (simp only: a flip: sum-mset-sum-list)

      fix qs'
      assume qs' ∈ listset ss2 ∧ a = sum-list qs'
      hence qs'-in: qs' ∈ listset ss2 and a': a = sum-list qs' by simp-all
      from this(1) obtain qs1 where qs1-in: qs1 ∈ listset ss1 and len-qs1: length
qs1 = length qs'
        and 2:  $\bigwedge i. i < \text{length } qs' \implies qs1 ! i = qs' ! f i$  using f-bij f by (rule
listset-permE) blast+
        from ⟨qs' ∈ listset ss2⟩ have length qs' = length ss2 by (rule listsetD)
        with f-bij have bij-betw f {.. $\text{length } qs1$ } {.. $\text{length } qs'$ } by (simp only:
len-qs1 len-ss1)
        hence mset qs1 = mset qs' using 2 by (rule mset-bij) (simp only: len-qs1)
        hence sum-list qs1 = sum-list qs' by (simp flip: sum-mset-sum-list)
        hence sum-list qs1 = sum-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 $eq: \text{length } qs' = \text{length } qs2$ **by** (*simp only: len-qs2 len-qs len-ss1*)

fix i
 assume $i < \text{length } qs'$
 hence $i < \text{length } qs2$ **by** (*simp only: eq*)
 hence $i \in \{..<\text{length } qs2\}$ **and** $i < \text{length } qs$ **and** $i < \text{length } ss1$
 by (*simp-all add: len-qs2 len-qs*)
 from *this*(1) **have** $g\ i \in g\ \{..<\text{length } qs2\}$ **by** (*rule imageI*)
 also from *g-bij2* **have** $\dots = \{..<\text{length } qs\}$ **by** (*simp only: bij-betw-def*)
 finally **have** $g\ i < \text{length } qs'$ **by** (*simp add: eq len-qs2*)
 from $\langle i < \text{length } qs \rangle$ **have** $qs2\ !\ i = qs\ !\ g\ i$ **by** (*rule 1*)
 also **have** $\dots = qs1\ !\ g\ i$ **by** (*simp only: $\langle qs1 = qs \rangle$*)
 also from $\langle g\ i < \text{length } qs' \rangle$ **have** $\dots = qs'\ !\ f\ (g\ i)$ **by** (*rule 2*)
 also from $\langle i < \text{length } ss1 \rangle$ **have** $\dots = qs'\ !\ i$ **by** (*simp only: f-g*)
 finally **show** $qs'\ !\ i = qs2\ !\ i$ **by** (*rule sym*)

qed
 qed *fact*
 qed
 qed

lemma *direct-decomp-split-map*:
 $\text{direct-decomp } A\ (\text{map } f\ ss) \implies \text{direct-decomp } A\ (\text{map } f\ (\text{filter } P\ ss) @ \text{map } f\ (\text{filter } (-\ P)\ ss))$
proof (*rule direct-decomp-perm*)
 show $\text{mset } (\text{map } f\ ss) = \text{mset } (\text{map } f\ (\text{filter } P\ ss) @ \text{map } f\ (\text{filter } (-\ P)\ ss))$
 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 $qs \in \text{listset } ?ss$
 then **obtain** $qs1\ qs2$ **where** $qs1: qs1 \in \text{listset } ss$ **and** $qs2: qs2 \in \text{listset } rs$ **and**
 $qs: qs = qs1\ @\ qs2$
 by (*rule listset-appendE*)
 have $\text{sum-list } qs = \text{sum-list } ((\text{sum-list } qs2) \# qs1)$ **by** (*simp add: qs add.commute*)
 also from *assms*(1) **have** $\dots \in A$
proof (*rule direct-decompD*)
 from *assms*(2) $qs2$ **have** $\text{sum-list } qs2 \in s$ **by** (*rule direct-decompD*)
 thus $\text{sum-list } qs2 \# qs1 \in \text{listset } (s\ \#\ ss)$ **using** $qs1\ \text{refl}$ **by** (*rule listset-ConsI*)
 qed
 finally **show** $\text{sum-list } qs \in A$.

next
 fix a
 assume $a \in A$

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 $\exists !qs \in \text{listset } ?ss. a = \text{sum-list } qs$
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** $\bigwedge a b. x * a = x * b \implies x \neq 0 \implies a = b$
shows *direct-decomp ((*) x ' A) (map ((*) x) ss)* **(is** *direct-decomp ?A ?ss***)**
proof (*rule direct-decompI-alt*)
fix *qs*
assume *qs* ∈ *listset ?ss*

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

 fix $qs0$
 assume $qs0 \in \text{listset } ?ss \wedge a = \text{sum-list } qs0$
 hence $qs0 \in \text{listset } ?ss$ and $a0$: $a = \text{sum-list } qs0$ by *simp-all*
 from *this(1)* obtain $qs1$ where $qs1$ -in: $qs1 \in \text{listset } ss$ and $qs0$: $qs0 = \text{map } ((*) x) qs1$
 by (rule *listset-map-imageE*)
 show $qs0 = qs$
 proof (cases $x = 0$)
 case *True*
 from $qs1$ -in have $\text{length } qs1 = \text{length } ss$ by (rule *listsetD*)
 moreover from qs' -in have $\text{length } qs' = \text{length } ss$ by (rule *listsetD*)
 ultimately show *?thesis* by (simp add: *qs-def qs0 list-eq-iff-nth-eq True*)
 next
 case *False*
 have $x * \text{sum-list } qs1 = a$ by (simp only: *a0 qs0 sum-list-map-times*)
 also have $... = x * \text{sum-list } qs'$ by (simp only: *a' a*)
 finally have $\text{sum-list } qs1 = \text{sum-list } qs'$ 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 $\{\} \notin \text{set } ss2 \implies \text{direct-decomp } (\text{sum-list } \cdot \text{listset } ss1) ss1$ (is - \implies *?thesis1*)
 and $\{\} \notin \text{set } ss1 \implies \text{direct-decomp } (\text{sum-list } \cdot \text{listset } ss2) ss2$ (is - \implies *?thesis2*)
 and *direct-decomp* A [$\text{sum-list } \cdot \text{listset } ss1, \text{sum-list } \cdot \text{listset } ss2$] (is *direct-decomp* - *?ss*)
 proof -
 have *rl*: *direct-decomp* ($\text{sum-list } \cdot \text{listset } ts1$) $ts1$

```

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

  {
    assume  $\{\} \notin \text{set } ss2$ 
    with assms show ?thesis1 by (rule rl)
  }

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

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

```

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 ∈ A
with assms obtain qs' where qs': qs' ∈ listset ss and a: a = sum-list qs'
  by (rule direct-decompE)
define qs where qs = 0 # qs'
show ∃!qs. qs ∈ listset ({0} # ss) ∧ a = sum-list qs
proof (intro ex1I conjI)
  from - qs' qs-def show qs ∈ listset ({0} # ss) by (rule listset-ConsI) simp
next
  fix qs0
  assume qs0 ∈ listset ({0} # ss) ∧ a = sum-list qs0
  hence qs0 ∈ listset ({0} # ss) and a0: a = sum-list qs0 by simp-all
  from this(1) obtain q0 qs0' where q0 ∈ {0} and qs0': qs0' ∈ listset ss
    and qs0: qs0 = q0 # qs0' by (rule listset-ConsE)
  from this(1, 3) have sum-list qs0' = sum-list qs' by (simp add: a0 flip: a)
  with assms qs0' qs' have qs0' = qs' by (rule direct-decomp-unique)
  with ⟨q0 ∈ {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 ⋀s0. s0 ∈ set ss ⟹ 0 ∈ s0
  shows s ⊆ A
proof
  fix x
  assume x ∈ s
  moreover from assms(2) have map (λ-. 0) ss ∈ listset ss
    by (induct ss, auto simp del: listset.simps(2) intro: listset-ConsI)
  ultimately have x # (map (λ-. 0) ss) ∈ listset (s # ss) using refl by (rule
listset-ConsI)
  with assms(1) have sum-list (x # (map (λ-. 0) ss)) ∈ A by (rule direct-decompD)
  thus x ∈ A by simp
qed

lemma direct-decomp-Int-zero:
  assumes direct-decomp A ss and i < j and j < length ss and ⋀s. s ∈ set ss
  ⟹ 0 ∈ s
  shows ss ! i ∩ ss ! j = {0}
proof -
  from assms(2, 3) have i < length ss by (rule less-trans)

```

```

hence i-in:  $ss ! i \in \text{set } ss$  by simp
from assms(3) have j-in:  $ss ! j \in \text{set } ss$  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:  $(\text{map } (\lambda-. 0) ss)[k := y] \in \text{listset } ss$  if  $k < \text{length } ss$  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 *:  $\bigwedge s'. s' \in \text{set } ss \implies 0 \in s'$  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  $\text{map } (\lambda-. 0) ss \in \text{listset } ss$ 
      by (induct ss) (auto simp del: listset.simps(2) intro: listset-ConsI)
      moreover have  $(\text{map } (\lambda-. 0) (s \# ss))[k := y] = y \# \text{map } (\lambda-. 0) ss$  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  $(\text{map } (\lambda-. 0) ss)[k' := y] \in \text{listset } ss$ 
      proof (rule Cons.hyps)
        from Cons.prems(2) show  $k' < \text{length } ss$  by (simp add: k)
      next
        from Cons.prems(3) show  $y \in ss ! k'$  by (simp add: k)
      qed
      moreover have  $(\text{map } (\lambda-. 0) (s \# ss))[k := y] = 0 \# (\text{map } (\lambda-. 0) ss)[k'$ 
:=  $y]$ 
      by (simp add: k)
      ultimately show ?thesis by (rule listset-ConsI)
    qed
  qed
  have 2: sum-list  $((\text{map } (\lambda-. 0) ss)[k := y]) = y$  if  $k < \text{length } ss$  for k and
 $y :: 'a$ 
    using that by (induct ss arbitrary: k) (auto simp: add-ac split: nat.split)
  define qs1 where  $qs1 = (\text{map } (\lambda-. 0) ss)[i := x]$ 
  define qs2 where  $qs2 = (\text{map } (\lambda-. 0) ss)[j := x]$ 
  note assms(1)
  moreover from  $\langle i < \text{length } ss \rangle$  x-i have  $qs1 \in \text{listset } ss$  unfolding qs1-def

```

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

corollary *direct-decomp-pairwise-zero:*

assumes *direct-decomp* $A \ ss$ and $\bigwedge s. s \in \text{set } ss \implies 0 \in s$
 shows *pairwise* $(\lambda s1 \ s2. s1 \cap s2 = \{0\})$ (set ss)
 proof (rule *pairwiseI*)
 fix $s1 \ s2$
 assume $s1 \in \text{set } ss$
 then obtain i where $i < \text{length } ss$ and $s1: s1 = ss ! i$ by (metis *in-set-conv-nth*)
 assume $s2 \in \text{set } ss$
 then obtain j where $j < \text{length } ss$ and $s2: s2 = ss ! j$ by (metis *in-set-conv-nth*)
 assume $s1 \neq s2$
 hence $i < j \vee j < i$ by (auto simp: $s1 \ s2$)
 thus $s1 \cap s2 = \{0\}$
 proof
 assume $i < j$
 with $\text{assms}(1)$ show ?thesis unfolding $s1 \ s2$ using $\langle j < \text{length } ss \rangle$ $\text{assms}(2)$
 by (rule *direct-decomp-Int-zero*)
 next
 assume $j < i$
 with $\text{assms}(1)$ have $s2 \cap s1 = \{0\}$ unfolding $s1 \ s2$ using $\langle i < \text{length } ss \rangle$
 $\text{assms}(2)$
 by (rule *direct-decomp-Int-zero*)
 thus ?thesis by (simp only: *Int-commute*)
 qed
 qed
 qed

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

assumes *direct-decomp* $A \ ss$ and $1 < \text{count-list } ss \ X$ and $\bigwedge s. s \in \text{set } ss \implies 0 \in s$
 shows $X = \{0\}$
 proof –
 from $\text{assms}(2)$ obtain $i \ j$ where $i < j$ and $j < \text{length } ss$ 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 $ss ! i \cap ss ! j = \{0\}$ 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 $s1 \in \text{set } ss$ and $s2 \in \text{set } ss$ and $s1 \neq s2$

and $\bigwedge s. s \in \text{set } ss \implies 0 \in f s$

shows $f s1 \cap f s2 = \{0\}$

proof –

from assms(2) obtain i where $i < \text{length } ss$ and $s1: s1 = ss ! i$ by (metis in-set-conv-nth)

from this(1) have $i: i < \text{length } (\text{map } f ss)$ by simp

from assms(3) obtain j where $j < \text{length } ss$ and $s2: s2 = ss ! j$ by (metis in-set-conv-nth)

from this(1) have $j: j < \text{length } (\text{map } f ss)$ by simp

have $*$: $0 \in s$ if $s \in \text{set } (\text{map } f ss)$ for s

proof –

from that obtain s' where $s' \in \text{set } ss$ and $s: s = f s'$ unfolding set-map ..

from this(1) show $0 \in s$ unfolding s by (rule assms(5))

qed

show ?thesis

proof (rule linorder-cases)

assume $i < j$

with assms(1) have $(\text{map } f ss) ! i \cap (\text{map } f ss) ! j = \{0\}$

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 $(\text{map } f ss) ! j \cap (\text{map } f ss) ! i = \{0\}$

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* :: $'b \text{ set} \Rightarrow 'b \text{ set} \Rightarrow \text{bool}$

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

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

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

```

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

context vector-space
begin

lemma dim-empty [simp]:  $\dim \{\} = 0$ 
  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  $\text{span } A \cap \text{span } B = \{0\}$ 
  shows independent (A  $\cup$  B)
proof
  from span-superset have  $A \cap B \subseteq \text{span } A \cap \text{span } B$  by blast
  hence  $A \cap B = \{\}$  unfolding assms(3) using assms(1, 2) dependent-zero by
blast
  assume dependent (A  $\cup$  B)
  then obtain  $T \ u \ v$  where finite T and  $T \subseteq A \cup B$  and eq:  $(\sum_{v \in T}. u \ v \ * \ s \ v) = 0$ 
  and  $v \in T$  and  $u \ v \neq 0$  unfolding dependent-explicit by blast
  define  $TA$  where  $TA = T \cap A$ 
  define  $TB$  where  $TB = T \cap B$ 
  from  $\langle T \subseteq A \cup B \rangle$  have  $T: T = TA \cup TB$  by (auto simp: TA-def TB-def)
  from  $\langle \text{finite } T \rangle$  have finite TA and  $TA \subseteq A$  by (simp-all add: TA-def)
  from  $\langle \text{finite } T \rangle$  have finite TB and  $TB \subseteq B$  by (simp-all add: TB-def)
  from  $\langle A \cap B = \{\} \rangle \langle TA \subseteq A \rangle$  this(2) have  $TA \cap TB = \{\}$  by blast
  have  $0 = (\sum_{v \in TA \cup TB}. u \ v \ * \ s \ v)$  by (simp only: eq flip: T)
  also have  $\dots = (\sum_{v \in TA}. u \ v \ * \ s \ v) + (\sum_{v \in TB}. u \ v \ * \ s \ v)$  by (rule sum.union-disjoint)
fact+
  finally have  $(\sum_{v \in TA}. u \ v \ * \ s \ v) = (\sum_{v \in TB}. (- \ u) \ v \ * \ s \ v)$  (is ?x = ?y)
  by (simp add: sum-negf eq-neg-iff-add-eq-0)
  from  $\langle \text{finite } TB \rangle \langle TB \subseteq B \rangle$  have  $?y \in \text{span } B$  by (auto simp: span-explicit simp
del: uminus-apply)
  moreover from  $\langle \text{finite } TA \rangle \langle TA \subseteq A \rangle$  have  $?x \in \text{span } A$  by (auto simp:
span-explicit)
  ultimately have  $?y \in \text{span } A \cap \text{span } B$  by (simp add:  $\langle ?x = ?y \rangle$ )
  hence  $?x = 0$  and  $?y = 0$  by (simp-all add:  $\langle ?x = ?y \rangle$  assms(3))
  from  $\langle v \in T \rangle$  have  $v \in TA \cup TB$  by (simp only: T)
  hence  $u \ v = 0$ 
proof
  assume  $v \in TA$ 
  with assms(1)  $\langle \text{finite } TA \rangle \langle TA \subseteq A \rangle \langle ?x = 0 \rangle$  show  $u \ v = 0$  by (rule
independentD)
  next
  assume  $v \in TB$ 
  with assms(2)  $\langle \text{finite } TB \rangle \langle TB \subseteq B \rangle \langle ?y = 0 \rangle$  have  $(- \ u) \ v = 0$  by (rule
independentD)

```

```

    thus  $u \ v = 0$  by simp
  qed
  with  $\langle u \ v \neq 0 \rangle$  show False ..
qed

```

lemma *subspace-direct-decomp*:

```

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

```

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

by (metis (full-types) is-basis-def dim-eq-card span-eq span-eq-iff)

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

unfolding is-basis-def using independent-span-bound by auto

lemma *some-basis-is-basis*: $\text{is-basis } V \ (\text{some-basis } V)$

proof –

obtain B **where** $B \subseteq V$ **and** *independent* B **and** $V \subseteq \text{span } B$ **and** $\text{card } B = \dim V$
 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*: $V \subseteq \text{span } (\text{some-basis } V)$
and *card-some-basis*: $\text{card } (\text{some-basis } V) = \dim V$
using *some-basis-is-basis*[of V] **by** (simp-all add: *is-basis-def*)

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

lemma *span-some-basis*: *subspace* $V \implies \text{span } (\text{some-basis } V) = V$
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** $\bigwedge s. s \in \text{set } ss \implies \text{subspace } s$
shows *pairwise* ($\lambda s1 s2. \text{disjnt } (\text{some-basis } s1) (\text{some-basis } s2)$) (*set* ss)

proof (rule *pairwiseI*)

fix $s1 s2$

assume $s1 \in \text{set } ss$ **and** $s2 \in \text{set } ss$ **and** $s1 \neq s2$

have *some-basis* $s1 \cap \text{some-basis } s2 \subseteq s1 \cap s2$ **using** *some-basis-subset* **by** *blast*

also from *direct-decomp-pairwise-zero* **have** $\dots = \{0\}$

proof (rule *pairwiseD*)

fix s

assume $s \in \text{set } ss$

hence *subspace* s **by** (rule *assms*(2))

thus $0 \in s$ **by** (rule *subspace-0*)

qed *fact+*

finally have *some-basis* $s1 \cap \text{some-basis } s2 \subseteq \{0\}$.

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** $\bigwedge s. s \in \text{set } ss \implies \text{subspace } s$

shows $\text{span } (\bigcup (\text{some-basis ' set } ss)) = A$

proof –

from *assms*(1) **have** *eq0*[*symmetric*]: $\text{sum-list ' listset } ss = A$ **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.prem) simp
  hence eq1: span (some-basis s) = s by (rule span-some-basis)
  have  $\bigwedge s'. s' \in \text{set } ss \implies \text{subspace } s'$  by (rule Cons.prem) 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 | 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  $\langle x \in s \rangle$  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
  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  $\langle x \in s \rangle$  by blast
  qed
qed
finally show ?case .
qed
qed

```

lemma *direct-decomp-independent-some-basis:*
 assumes *direct-decomp* A ss and $\bigwedge s. s \in \text{set } ss \implies \text{subspace } s$
 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 $1: \bigwedge s'. s' \in \text{set } ss \implies \text{subspace } s'$ **by** (*rule Cons.prem*s) *simp*
have *subspace s* **by** (*rule Cons.prem*s) *simp*
hence $0 \in s$ **and** *eq1*: $\text{span } (\text{some-basis } s) = s$ **by** (*rule subspace-0*, *rule span-some-basis*)
from *Cons.prem*s(1) **have** *: *direct-decomp A* ($[s] @ ss$) **by** *simp*
moreover from $\langle 0 \in s \rangle$ **have** $\{ \} \notin \text{set } [s]$ **by** *auto*
ultimately have 2: *direct-decomp* (*sum-list ' listset ss*) *ss* **by** (*rule direct-decomp-appendD*)
hence *eq2*: $\text{span } (\bigcup (\text{some-basis ' set } ss)) = \text{sum-list ' listset } ss$ **using** 1
by (*rule direct-decomp-span-some-basis*)

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

corollary *direct-decomp-is-basis*:
assumes *direct-decomp A ss* **and** $\bigwedge s. s \in \text{set } ss \implies \text{subspace } s$
shows *is-basis A* ($\bigcup (\text{some-basis ' set } ss)$)
proof –
from *assms* **have** *subspace A* **by** (*rule subspace-direct-decomp*)
moreover from *assms* **have** $\text{span } (\bigcup (\text{some-basis ' set } ss)) = A$
by (*rule direct-decomp-span-some-basis*)
moreover from *assms* **have** *independent* ($\bigcup (\text{some-basis ' set } ss)$)
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** $A \subseteq \text{span } B$ **and** $\bigwedge s. s \in \text{set } ss \implies \text{subspace } s$
shows $\text{dim } A = (\sum s \in \text{set } ss. \text{dim } s)$
proof –
from *assms*(1, 4) **have** *is-basis A* ($\bigcup (\text{some-basis ' set } ss)$)

```

    (is is-basis A ?B) by (rule direct-decomp-is-basis)
  hence  $\dim A = \text{card } ?B$  and independent ?B and  $?B \subseteq A$  by (simp-all add:
is-basis-def)
  from this(3) assms(3) have  $?B \subseteq \text{span } B$  by (rule subset-trans)
  with assms(2) ⟨independent ?B⟩ have finite ?B using independent-span-bound
by blast
  note ⟨ $\dim A = \text{card } ?B$ ⟩
  also from finite-set have  $\text{card } ?B = (\sum s \in \text{set } ss. \text{card } (\text{some-basis } s))$ 
  proof (intro card-UN-disjoint ballI impI)
    fix s
    assume  $s \in \text{set } ss$ 
    with ⟨finite ?B⟩ show finite (some-basis s) by auto
  next
    fix s1 s2
    have pairwise ( $\lambda s t. \text{disjnt } (\text{some-basis } s) (\text{some-basis } t)$ ) (set ss)
      using assms(1, 4) by (rule direct-decomp-some-basis-pairwise-disjnt)
    moreover assume  $s1 \in \text{set } ss$  and  $s2 \in \text{set } ss$  and  $s1 \neq s2$ 
    thm pairwiseD
    ultimately have  $\text{disjnt } (\text{some-basis } s1) (\text{some-basis } s2)$  by (rule pairwiseD)
    thus  $\text{some-basis } s1 \cap \text{some-basis } s2 = \{\}$  by (simp only: disjnt-def)
  qed
  also from refl card-some-basis have  $\dots = (\sum s \in \text{set } ss. \dim s)$  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  $\bigwedge s. s \in \text{set } ss \implies \text{homogeneous-set } s$ 
  shows homogeneous-set A
proof (rule homogeneous-setI)
  fix a n
  assume  $a \in A$ 
  with assms(1) obtain qs where  $qs \in \text{listset } ss$  and  $a: a = \text{sum-list } qs$  by (rule
direct-decompE)
  have hom-component a n = hom-component (sum-list qs) n by (simp only: a)
  also have  $\dots = \text{sum-list } (\text{map } (\lambda q. \text{hom-component } q n) qs)$ 
    by (induct qs) (simp-all add: hom-component-plus)
  also from assms(1) have  $\dots \in A$ 
  proof (rule direct-decompD)
    show  $\text{map } (\lambda q. \text{hom-component } q n) qs \in \text{listset } ss$ 
    proof (rule listset-closed-map)
      fix s q
      assume  $s \in \text{set } ss$ 
      hence homogeneous-set s by (rule assms(2))
      moreover assume  $q \in s$ 
      ultimately show hom-component q n  $\in s$  by (rule homogeneous-setD)
    qed
  qed

```

qed fact
 qed
 finally show *hom-component* *a* *n* $\in A$.
 qed

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

lemma *hom-deg-setD*:
 assumes *p* \in *hom-deg-set* *z* *A*
 shows *homogeneous* *p* and *p* $\neq 0 \implies$ *poly-deg* *p* = *z*
proof –
 from *assms* obtain *a* where *a* $\in A$ and *p*: *p* = *hom-component* *a* *z* **unfolding**
hom-deg-set-def ..
 show *: *homogeneous* *p* **by** (*simp* only: *p* *homogeneous-hom-component*)

assume *p* $\neq 0$
 hence *keys* *p* $\neq \{\}$ **by** *simp*
 then obtain *t* where *t* \in *keys* *p* **by** *blast*
 with * have *deg-pm* *t* = *poly-deg* *p* **by** (*rule* *homogeneousD-poly-deg*)
 moreover from $\langle t \in \text{keys } p \rangle$ 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* $\in A$
 shows *0* \in *hom-deg-set* *z* *A*
proof –
 have *0* = *hom-component* *0* *z* **by** *simp*
 also from *assms* have ... \in *hom-deg-set* *z* *A* **unfolding** *hom-deg-set-def* **by**
 (*rule* *imageI*)
 finally show ?*thesis* .
 qed

lemma *hom-deg-set-closed-uminus*:
 assumes $\bigwedge a. a \in A \implies - a \in A$ and *p* \in *hom-deg-set* *z* *A*
 shows $- p \in$ *hom-deg-set* *z* *A*
proof –
 from *assms*(2) obtain *a* where *a* $\in A$ and *p*: *p* = *hom-component* *a* *z* **unfolding**
hom-deg-set-def ..
 from *this*(1) have $- a \in A$ **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 $\bigwedge a1\ a2. a1 \in A \implies a2 \in A \implies a1 + a2 \in A$
 and *p* \in *hom-deg-set* *z* *A* and *q* \in *hom-deg-set* *z* *A*

shows $p + q \in \text{hom-deg-set } z \ A$
proof –
 from *assms*(2) **obtain** $a1$ **where** $a1 \in A$ **and** $p: p = \text{hom-component } a1 \ z$
unfolding *hom-deg-set-def* ..
 from *assms*(3) **obtain** $a2$ **where** $a2 \in A$ **and** $q: q = \text{hom-component } a2 \ z$
unfolding *hom-deg-set-def* ..
 from $\langle a1 \in A \rangle$ *this*(1) **have** $a1 + a2 \in A$ **by** (rule *assms*(1))
moreover **have** $p + q = \text{hom-component } (a1 + a2) \ z$ **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 $\bigwedge a1 \ a2. \ a1 \in A \implies a2 \in A \implies a1 - a2 \in A$
 and $p \in \text{hom-deg-set } z \ A$ **and** $q \in \text{hom-deg-set } z \ A$
 shows $p - q \in \text{hom-deg-set } z \ A$
proof –
 from *assms*(2) **obtain** $a1$ **where** $a1 \in A$ **and** $p: p = \text{hom-component } a1 \ z$
unfolding *hom-deg-set-def* ..
 from *assms*(3) **obtain** $a2$ **where** $a2 \in A$ **and** $q: q = \text{hom-component } a2 \ z$
unfolding *hom-deg-set-def* ..
 from $\langle a1 \in A \rangle$ *this*(1) **have** $a1 - a2 \in A$ **by** (rule *assms*(1))
moreover **have** $p - q = \text{hom-component } (a1 - a2) \ z$ **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 $\bigwedge a. \ a \in A \implies c \cdot a \in A$ **and** $p \in \text{hom-deg-set } z \ A$
 shows $(c::'a::\text{semiring-0}) \cdot p \in \text{hom-deg-set } z \ A$
proof –
 from *assms*(2) **obtain** a **where** $a \in A$ **and** $p: p = \text{hom-component } a \ z$ **unfolding**
hom-deg-set-def ..
 from *this*(1) **have** $c \cdot a \in A$ **by** (rule *assms*(1))
moreover **have** $c \cdot p = \text{hom-component } (c \cdot a) \ z$
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 $0 \in A$ **and** $\bigwedge a1 \ a2. \ a1 \in A \implies a2 \in A \implies a1 + a2 \in A$
 and $\bigwedge i. \ i \in I \implies f \ i \in \text{hom-deg-set } z \ A$
 shows $\text{sum } f \ I \in \text{hom-deg-set } z \ A$
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  $\text{sum } f \text{ (insert } j \text{ } I) = f \text{ } j + \text{sum } f \text{ } I$  by simp
  also from assms(2) have  $\dots \in \text{hom-deg-set } z \text{ } A$ 
  proof (intro hom-deg-set-closed-plus insert.hyps)
    show  $f \text{ } j \in \text{hom-deg-set } z \text{ } A$  by (rule insert.prems) simp
  next
    fix i
    assume  $i \in I$ 
    hence  $i \in \text{insert } j \text{ } I$  by simp
    thus  $f \text{ } i \in \text{hom-deg-set } z \text{ } A$  by (rule insert.prems)
  qed
finally show ?case .
qed

lemma hom-deg-set-subset: homogeneous-set A  $\implies$  hom-deg-set z A  $\subseteq$  A
  by (auto dest: homogeneous-setD simp: hom-deg-set-def)

lemma Polys-closed-hom-deg-set:
  assumes  $A \subseteq P[X]$ 
  shows  $\text{hom-deg-set } z \text{ } A \subseteq P[X]$ 
proof
  fix p
  assume  $p \in \text{hom-deg-set } z \text{ } A$ 
  then obtain p' where  $p' \in A$  and  $p: p = \text{hom-component } p' \text{ } z \text{ unfolding }$ 
hom-deg-set-def ..
  from this(1) assms have  $p' \in P[X]$  ..
  have  $\text{keys } p \subseteq \text{keys } p'$  by (simp add: p keys-hom-component)
  also from  $\langle p' \in P[X] \rangle$  have  $\dots \subseteq P[X]$  by (rule PolysD)
  finally show  $p \in P[X]$  by (rule PolysI)
qed

lemma hom-deg-set-alt-homogeneous-set:
  assumes homogeneous-set A
  shows  $\text{hom-deg-set } z \text{ } A = \{p \in A. \text{homogeneous } p \wedge (p = 0 \vee \text{poly-deg } p = z)\}$ 
  (is ?A = ?B)
proof
  show ?A  $\subseteq$  ?B
  proof
    fix h
    assume  $h \in ?A$ 
    also from assms have  $\dots \subseteq A$  by (rule hom-deg-set-subset)
    finally show  $h \in ?B$  using  $\langle h \in ?A \rangle$  by (auto dest: hom-deg-setD)
  qed
next
  show ?B  $\subseteq$  ?A
  proof
    fix h

```

```

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

lemma hom-deg-set-sum-list-listset:
  assumes  $A = \text{sum-list } ' \text{ listset } ss$ 
  shows hom-deg-set  $z \ A = \text{sum-list } ' \text{ listset } (\text{map } (\text{hom-deg-set } z) \ ss)$  (is  $?A = ?B$ )
proof
  show  $?A \subseteq ?B$ 
  proof
    fix  $h$ 
    assume  $h \in ?A$ 
    then obtain  $a$  where  $a \in A$  and  $h: h = \text{hom-component } a \ z$  unfolding
hom-deg-set-def ..
    from this(1) obtain  $qs$  where  $qs \in \text{listset } ss$  and  $a: a = \text{sum-list } qs$  unfolding
assms ..
    have  $h = \text{hom-component } (\text{sum-list } qs) \ z$  by (simp only: a h)
    also have  $\dots = \text{sum-list } (\text{map } (\lambda q. \text{hom-component } q \ z) \ qs)$ 
    by (induct qs) (simp-all add: hom-component-plus)
    also have  $\dots \in ?B$ 
    proof (rule imageI)
      show  $\text{map } (\lambda q. \text{hom-component } q \ z) \ qs \in \text{listset } (\text{map } (\text{hom-deg-set } z) \ ss)$ 
      unfolding hom-deg-set-def using  $\langle qs \in \text{listset } ss \rangle$  refl by (rule list-set-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  $qs \in \text{listset } (\text{map } (\text{hom-deg-set } z) \ ss)$  and  $h: h = \text{sum-list } qs$  ..
    from this(1) obtain  $qs'$  where  $qs' \in \text{listset } ss$  and  $qs: qs = \text{map } (\lambda q. \text{hom-component } q \ z) \ qs'$ 
    unfolding hom-deg-set-def by (rule listset-map-imageE)

```

```

    have h = sum-list (map (λq. 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 ⇒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 ⇒₀ nat) ⇒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 ∈ A by (rule phull.subspace-0)
  thus 0 ∈ ?A by (rule zero-in-hom-deg-set)
next
  fix p q
  assume p ∈ ?A and q ∈ ?A
  with phull.subspace-add show p + q ∈ ?A by (rule hom-deg-set-closed-plus)
(rule assms)
next
  fix c p
  assume p ∈ ?A
  with phull.subspace-scale show c · p ∈ ?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 ⊆ ?B
  proof
    fix p
    assume p ∈ ?A
    also from this have ... = {p ∈ P[X]. homogeneous p ∧ (p = 0 ∨ poly-deg p
= z)}
    by (simp only: hom-deg-set-alt-homogeneous-set[OF homogeneous-set-Polys])
    finally have p ∈ P[X] and homogeneous p and p ≠ 0 ⇒ poly-deg p = z by
simp-all
    thus p ∈ ?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 ∈ keys ?m by simp
    hence t ∈ keys (?m + p) using 2(2) by (rule in-keys-plusI1)
    hence ?m + p ≠ 0 by auto
    hence poly-deg (monomial c t + p) = z by (rule 2)
    from 2(4) have keys (?m + p) ⊆ .[X] by (rule PolysD)
    with ⟨t ∈ keys (?m + p)⟩ have t ∈ .[X] ..
    hence ?m ∈ P[X] by (rule Polys-closed-monomial)
    have t ∈ deg-sect X z
    proof (rule deg-sectI)

```

```

    from 2(5)  $\langle t \in \text{keys } (?m + p) \rangle$  have  $\text{deg-pm } t = \text{poly-deg } (?m + p)$ 
      by (rule homogeneousD-poly-deg)
    also have  $\dots = z$  by fact
    finally show  $\text{deg-pm } t = z$  .
  qed fact
  hence monomial 1  $t \in \text{monomial 1 } \text{' deg-sect } X z$  by (rule imageI)
  hence monomial 1  $t \in ?B$  by (rule phull.span-base)
  hence  $c \cdot \text{monomial 1 } t \in ?B$  by (rule phull.span-scale)
  hence  $?m \in ?B$  by simp
  moreover have  $p \in ?B$ 
  proof (rule 2)
  from 2(4)  $\langle ?m \in P[X] \rangle$  have  $(?m + p) - ?m \in P[X]$  by (rule Polys-closed-minus)
    thus  $p \in P[X]$  by simp
  next
    have 1:  $\text{deg-pm } s = z$  if  $s \in \text{keys } p$  for  $s$ 
    proof -
      from that 2(2) have  $s \neq t$  by blast
      hence  $s \notin \text{keys } ?m$  by simp
      with that have  $s \in \text{keys } (?m + p)$  by (rule in-keys-plusI2)
      with 2(5) have  $\text{deg-pm } s = \text{poly-deg } (?m + p)$  by (rule homogeneousD-poly-deg)
      also have  $\dots = z$  by fact
      finally show  $?thesis$  .
    qed
  show homogeneous  $p$  by (rule homogeneousI) (simp add: 1)

  assume  $p \neq 0$ 
  show  $\text{poly-deg } p = z$ 
  proof (rule antisym)
    show  $\text{poly-deg } p \leq z$  by (rule poly-deg-leI) (simp add: 1)
  next
    from  $\langle p \neq 0 \rangle$  have  $\text{keys } p \neq \{\}$  by simp
    then obtain  $s$  where  $s \in \text{keys } p$  by blast
    hence  $z = \text{deg-pm } s$  by (simp only: 1)
    also from  $\langle s \in \text{keys } p \rangle$  have  $\dots \leq \text{poly-deg } p$  by (rule poly-deg-max-keys)
    finally show  $z \leq \text{poly-deg } p$  .
  qed
qed
ultimately show  $?case$  by (rule phull.span-add)
qed
next
  show  $?B \subseteq ?A$ 
  proof
    fix  $p$ 
    assume  $p \in ?B$ 
    then obtain  $M u$  where  $M \subseteq \text{monomial 1 } \text{' deg-sect } X z$  and finite  $M$  and
       $p = (\sum_{m \in M}. u m \cdot m)$ 
    by (auto simp: phull.span-explicit)

```



```

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 di-
rect-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 ... = ( $\sum p \in \text{set } ps. \text{Hilbert-fun } p \ z$ ) 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 $t \in \text{lpp } (A - \{0\})$ **and** $\text{deg-}t[\text{symmetric}]$: $\text{deg-pm } t = z$ **by** *simp-all*
from *this(1)* **obtain** p **where** $p \in A - \{0\}$ **and** $t: t = \text{lpp } p$..
from *this(1)* **have** $p \in A$ **and** $p \neq 0$ **by** *simp-all*
from *this(1)* **have** $1: \text{hom-component } p \ z \in \text{hom-deg-set } z \ A$ **(is** $?p \in -$ **)**
unfolding *hom-deg-set-def* **by** (rule *imageI*)
from $\langle p \neq 0 \rangle$ **have** $?p \neq 0$ **and** $\text{lpp } ?p = t$ **unfolding** $t \ \text{deg-}t$ **by** (rule
hom-component-lpp)**+**
note *this(2)[symmetric]*
moreover from $1 \ \langle ?p \neq 0 \rangle$ **have** $?p \in \text{hom-deg-set } z \ A - \{0\}$ **by** *simp*
ultimately show $t \in ?B$ **by** (rule *image-eqI*)
next
fix t
assume $t \in ?B$
then obtain p **where** $p \in \text{hom-deg-set } z \ A - \{0\}$ **and** $t: t = \text{lpp } p$..
from *this(1)* **have** $p \in \text{hom-deg-set } z \ A$ **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)* $\langle p \neq 0 \rangle$ **have** $p \in A - \{0\}$ **by** *simp*
hence $1: t \in \text{lpp } (A - \{0\})$ **using** t **by** (rule *rev-image-eqI*)
from $\langle p \neq 0 \rangle$ **have** $t \in \text{keys } p$ **unfolding** t **by** (rule *punit.lt-in-keys*)
with $\langle \text{homogeneous } p \rangle$ **have** $\text{deg-pm } t = \text{poly-deg } p$ **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* $A \ z = \text{card } (\text{lpp } (A - \{0\}))$ **(is** $- = \text{card } ?A$ **)**
proof –
have $?A \subseteq \text{lpp } (A - \{0\})$ **by** *simp*
then obtain B **where** $\text{sub}: B \subseteq \text{hom-deg-set } z \ A - \{0\}$ **and** $\text{eq1}: ?A = \text{lpp } B$
and *inj-on* $\text{lpp } B$ **by** (rule *subset-imageE-inj*)
have *Hilbert-fun* $A \ z = \text{phull.dim } (\text{hom-deg-set } z \ A)$ **by** (fact *Hilbert-fun-def*)
also have $\dots = \text{card } B$
proof (rule *phull.dim-eq-card*)
show $\text{phull.span } B = \text{phull.span } (\text{hom-deg-set } z \ A)$
proof
from *sub* **have** $B \subseteq \text{hom-deg-set } z \ A$ **by** *blast*
thus $\text{phull.span } B \subseteq \text{phull.span } (\text{hom-deg-set } z \ A)$ **by** (rule *phull.span-mono*)
next
from *assms(3)* **have** *phull.subspace* $(\text{hom-deg-set } z \ A)$ **by** (rule *subspace-hom-deg-set*)
hence $\text{phull.span } (\text{hom-deg-set } z \ A) = \text{hom-deg-set } z \ A$ **by** (*simp only:*
phull.span-eq-iff)
also have $\dots \subseteq \text{phull.span } B$
proof (rule *ccontr*)
assume $\neg \text{hom-deg-set } z \ A \subseteq \text{phull.span } B$
then obtain $p0$ **where** $p0 \in \text{hom-deg-set } z \ A - \text{phull.span } B$ **(is** $- \in ?B$ **)**
by *blast*
note *assms(1)* *this*

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

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

```

    with  $\langle lpp\ b \preceq t \rangle$  have  $lpp\ b = lpp\ b0$ 
      unfolding  $t$  by simp
    from  $inj\ \langle B0 \subseteq B \rangle$  have  $inj\text{-}on\ lpp\ B0$  by (rule inj-on-subset)
      hence  $b = b0$  using  $\langle lpp\ b = lpp\ b0 \rangle\ \langle b \in B0 \rangle\ \langle b0 \in B0 \rangle$  by (rule
inj-onD)
    with  $\langle b \neq b0 \rangle$  show False ..
  qed
  thus  $u\ b * lookup\ b\ t = 0$  by (simp add: in-keys-iff)
qed
qed
also from  $\langle t \in keys\ b0 \rangle\ \langle b0 \in B0 \rangle$  have  $\dots \neq 0$  by (simp add: B0-def
in-keys-iff)
  finally show False by (simp add: eq)
qed
qed
also have  $\dots = card\ ?A$  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 \div 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]: $\text{ideal } (ideal B \div x) = (ideal B) \div (x:::comm-ring)$

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*: $\text{inj } ((*) x) \Longrightarrow ((*) x - ' A) \div x = A$
 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* $F p = (SOME\ q. (punit.red\ (punit.reduced-GB\ F))^{**}\ p\ q \wedge \neg\ punit.is-red\ (punit.reduced-GB\ F)\ q)$

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

```
context
  fixes  $X :: 'x\ set$ 
  assumes fin-X: finite  $X$ 
begin
```

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

lemma *normal-form*:

```
  shows  $(punit.red\ (punit.reduced-GB\ F))^{**}\ p\ (normal-form\ F\ p)$  (is ?thesis1)
    and  $\neg\ punit.is-red\ (punit.reduced-GB\ F)\ (normal-form\ F\ p)$  (is ?thesis2)
proof -
  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  $\neg\ punit.is-red\ (punit.reduced-GB\ F)\ q$  unfolding punit.is-red-def not-not
    by (rule relation.wf-imp-nf-ex)
  hence  $(punit.red\ (punit.reduced-GB\ F))^{**}\ p\ q \wedge \neg\ punit.is-red\ (punit.reduced-GB\ F)\ q$  ..
  hence  $?thesis1 \wedge ?thesis2$  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  $\neg\ punit.is-red\ (punit.reduced-GB\ F)\ q$ 
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  $\neg\ punit.is-red\ (punit.reduced-GB\ F)\ (normal-form\ F\ p)$  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*: $\text{normal-form } F \ p = p \longleftrightarrow (\neg \text{punit.is-red } (\text{punit.reduced-GB } F) \ p)$

proof

assume $\text{normal-form } F \ p = p$

with $\text{normal-form}(2)[\text{of } p]$ **show** $\neg \text{punit.is-red } (\text{punit.reduced-GB } F) \ p$ **by** *simp*

next

assume $\neg \text{punit.is-red } (\text{punit.reduced-GB } F) \ p$

with *rtranclp.rtrancl-refl* **show** $\text{normal-form } F \ p = p$ **by** (*rule normal-form-unique*)

qed

lemma *normal-form-normal-form*: $\text{normal-form } F \ (\text{normal-form } F \ p) = \text{normal-form } F \ p$

by (*simp add: normal-form-id-iff normal-form*)

lemma *normal-form-zero*: $\text{normal-form } F \ 0 = 0$

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

lemma *normal-form-map-scale*: $\text{normal-form } F \ (c \cdot p) = c \cdot (\text{normal-form } F \ p)$

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*: $\text{normal-form } F \ (-\ p) = -\ \text{normal-form } F \ p$

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*:

$\text{normal-form } F \ (\text{normal-form } F \ p + \text{normal-form } F \ q) = \text{normal-form } F \ p + \text{normal-form } F \ q$

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

lemma *normal-form-minus-normal-form*:

$\text{normal-form } F \ (\text{normal-form } F \ p - \text{normal-form } F \ q) = \text{normal-form } F \ p - \text{normal-form } F \ q$

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

lemma *normal-form-ideal-Polys*: $\text{normal-form } (\text{ideal } F \cap P[X]) = \text{normal-form } F$

proof –

let $?F = \text{ideal } F \cap P[X]$

from *fin-X* **have** *eq*: $\text{punit.reduced-GB } ?F = \text{punit.reduced-GB } F$

proof (*rule reduced-GB-unique-Polys*)

from *fin-X F-sub* **show** $\text{punit.is-reduced-GB } (\text{punit.reduced-GB } F)$

by (*rule reduced-GB-is-reduced-GB-Polys*)

next

from *fin-X F-sub* **have** $\text{ideal } (\text{punit.reduced-GB } F) = \text{ideal } F$ **by** (*rule reduced-GB-ideal-Polys*)

also have $\dots = \text{ideal } (\text{ideal } F \cap P[X])$

proof (*intro subset-antisym ideal.span-subset-spanI*)

```

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

```

```

lemma normal-form-diff-in-ideal:  $p - \text{normal-form } F p \in \text{ideal } F$ 
proof -
  from normal-form(1) have  $p - \text{normal-form } F p \in \text{ideal } (\text{punit.reduced-GB } F)$ 
  by (rule punit.red-rtrancp-diff-in-pmdl[simplified])
  also from fin-X F-sub have  $\dots = \text{ideal } F$  by (rule reduced-GB-ideal-Polys)
  finally show ?thesis .
qed

```

```

lemma normal-form-zero-iff:  $\text{normal-form } F p = 0 \longleftrightarrow p \in \text{ideal } F$ 
proof
  assume normal-form F p = 0
  with normal-form-diff-in-ideal[of p] show  $p \in \text{ideal } F$  by simp
next
  assume  $p \in \text{ideal } F$ 
  hence  $p - (p - \text{normal-form } F p) \in \text{ideal } F$  using normal-form-diff-in-ideal
  by (rule ideal.span-diff)
  also from fin-X F-sub have  $\dots = \text{ideal } (\text{punit.reduced-GB } F)$  by (rule re-
duced-GB-ideal-Polys[symmetric])
  finally have *:  $\text{normal-form } F p \in \text{ideal } (\text{punit.reduced-GB } F)$  by simp
  show  $\text{normal-form } F p = 0$ 
  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  $\text{normal-form } F p \neq 0$ 
    ultimately obtain g where  $g \in \text{punit.reduced-GB } F$  and  $g \neq 0$ 
    and a: lpp g adds lpp (normal-form F p) by (rule punit.GB-adds-lt[simplified])
    note this(1, 2)
    moreover from  $\langle \text{normal-form } F p \neq 0 \rangle$  have lpp (normal-form F p)  $\in \text{keys}$ 
(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:  $\text{normal-form } F p = \text{normal-form } F q \longleftrightarrow p - q \in \text{ideal } F$ 
proof -

```



```

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

```

lemma *Polys-closed-normal-form*:

```

  assumes  $p \in P[X]$ 
  shows normal-form  $F p \in P[X]$ 
proof -
  from fin-X F-sub have 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-rtrancI[OF dickson-grading-varnum, where m=0, simplified dgrad-p-set-varnum])
qed

```

lemma *image-normal-form-iff*:

$p \in \text{normal-form } F \text{ ' } P[X] \longleftrightarrow (p \in P[X] \wedge \neg \text{punit.is-red } (\text{punit.reduced-GB } F) p)$

proof

```

  assume  $p \in \text{normal-form } F \text{ ' } P[X]$ 
  then obtain  $q$  where  $q \in P[X]$  and  $p: p = \text{normal-form } F q$  ..
  from this(1) show  $p \in P[X] \wedge \neg \text{punit.is-red } (\text{punit.reduced-GB } F) p$  unfolding
  p
  by (intro conjI Polys-closed-normal-form normal-form)

```

next

```

  assume  $p \in P[X] \wedge \neg \text{punit.is-red } (\text{punit.reduced-GB } F) p$ 
  hence  $p \in P[X]$  and  $\neg \text{punit.is-red } (\text{punit.reduced-GB } F) p$  by simp-all
  from this(2) have normal-form  $F p = p$  by (simp add: normal-form-id-iff)
  from this[symmetric]  $\langle p \in P[X] \rangle$  show  $p \in \text{normal-form } F \text{ ' } P[X]$  by (rule
  image-eqI)

```

qed

end

```

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

    have  $L \subseteq P[X]$  by (simp add: L-def)
    hence  $\text{normal-form } L \ p \in P[X]$  using  $\langle p \in P[X] \rangle$  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-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  $\text{insert } f \ F \subseteq P[X]$  by simp
then obtain  $F0 \ q0$  where  $F0 \subseteq \text{insert } f \ F$  and finite  $F0$  and  $q0: \bigwedge f0. \ q0 \ f0 \in P[X]$ 
and  $a: a = (\sum f0 \in F0. \ q0 \ f0 * f0)$ 
using  $\langle a \in P[X] \rangle \ \langle a \in I \rangle$  unfolding I-def by (rule in-idealE-Polys) blast
obtain  $q \ a'$  where  $a': a' \in \text{ideal } F$  and  $a' \in P[X]$  and  $q \in P[X]$  and  $a: a = q * f + a'$ 
proof (cases  $f \in F0$ )
  case True
    with  $\langle F0 \subseteq \text{insert } f \ F \rangle$  have  $F0 - \{f\} \subseteq F$  by blast
    show ?thesis
  proof
    have  $(\sum f0 \in F0 - \{f\}. \ q0 \ f0 * f0) \in \text{ideal } (F0 - \{f\})$  by (rule ideal.sum-in-spanI)
    also from  $\langle F0 - \{f\} \subseteq F \rangle$  have  $\dots \subseteq \text{ideal } F$  by (rule ideal.span-mono)
    finally show  $(\sum f0 \in F0 - \{f\}. \ q0 \ f0 * f0) \in \text{ideal } F$  .

```

```

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

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

```

next
fix $qs0$
assume $qs0 \in \text{listset } ?ss \wedge a = \text{sum-list } qs0$
hence $qs0 \in \text{listset } ?ss$ **and** $a = \text{sum-list } qs0$ **by** *simp-all*
from *this(1)* **obtain** $x\ y$ **where** $x \in \text{ideal } F \cap P[X]$ **and** $y \in (*)\ f \text{ ' normal-form }$
 $L \text{ ' } P[X]$
and $qs0: qs0 = [x, y]$ **by** *(rule listset-doubletonE)*
from *this(2)* **obtain** $a0$ **where** $a0 \in P[X]$ **and** $y: y = f * \text{normal-form } L\ a0$
by *blast*
from $\langle x \in \text{ideal } F \cap P[X] \rangle$ **have** $x \in \text{ideal } F$ **by** *simp*
have $x: x = a - y$ **by** *(simp add: $\langle a = \text{sum-list } qs0 \rangle\ qs0$)*
have $f * (\text{normal-form } L\ q - \text{normal-form } L\ a0) = x - (a - ?a)$ **by** *(simp add: $x\ y\ a\ \text{algebra-simps}$)*
also from $\langle x \in \text{ideal } F \rangle\ \langle a - ?a \in \text{ideal } F \rangle$ **have** $\dots \in \text{ideal } F$ **by** *(rule ideal.span-diff)*
finally have $\text{normal-form } L\ q - \text{normal-form } L\ a0 \in \text{ideal } F \div f$ **by** *(rule quot-setI)*
moreover from $\langle L \subseteq P[X] \rangle\ \langle q \in P[X] \rangle\ \langle a0 \in P[X] \rangle$ **have** $\text{normal-form } L\ q - \text{normal-form } L\ a0 \in P[X]$
by *(intro Polys-closed-minus Polys-closed-normal-form)*
ultimately have $\text{normal-form } L\ q - \text{normal-form } L\ a0 \in L$ **by** *(simp add: L-def)*
also have $\dots \subseteq \text{ideal } L$ **by** *(fact ideal.span-superset)*
finally have $\text{normal-form } L\ q - \text{normal-form } L\ a0 = 0$ **using** $\langle L \subseteq P[X] \rangle$
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* $P[X]$ $[\text{ideal } F \cap P[X], \text{normal-form } F \text{ ' } P[X]]$
proof –
from *assms one-in-Polys* **have** *direct-decomp* $(\text{ideal } (\text{insert } 1\ F) \cap P[X])$ $[\text{ideal } F \cap P[X],$
 $(*)\ 1 \text{ ' normal-form } ((\text{ideal } F \div 1) \cap P[X])$
 $\text{ ' } P[X]]$
by *(rule direct-decomp-ideal-insert)*
moreover have $\text{ideal } (\text{insert } 1\ F) = \text{UNIV}$
by *(simp add: ideal-eq-UNIV-iff-contains-one ideal.span-base)*
moreover from *reft* **have** $((*)\ 1 \circ \text{normal-form } F) \text{ ' } P[X] = \text{normal-form } F \text{ ' } P[X]$
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* :: $((('x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a) \times 'x \text{ set}) \Rightarrow (('x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a :: \text{comm-semiring-0})$
set

where *cone* *hU* = (*) (*fst* *hU*) ' *P*[*snd* *hU*]

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

lemma *coneE*:

assumes $p \in \text{cone } (h, U)$

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

using *assms* **by** (*auto simp: cone-def mult.commute*)

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

by (*auto simp: Polys-empty map-scale-eq-times intro: coneI elim!: coneE*)

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

by (*auto simp: cone-def intro: zero-in-Polys*)

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

by (*auto simp: cone-def*)

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

by (*auto simp: cone-def intro!: image-eqI zero-in-Polys*)

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

using *zero-in-cone* **by** *fastforce*

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

using - *one-in-Polys* **by** (*rule coneI*) *simp*

lemma *cone-closed-plus*:

assumes $a \in \text{cone } hU$ **and** $b \in \text{cone } hU$

shows $a + b \in \text{cone } hU$

proof –

obtain *h* *U* **where** *hU*: $hU = (h, U)$ **using** *prod.exhaust* **by** *blast*

with *assms* **have** $a \in \text{cone } (h, U)$ **and** $b \in \text{cone } (h, U)$ **by** *simp-all*

from *this*(1) **obtain** *a'* **where** $a' \in P[U]$ **and** $a: a = a' * h$ **by** (*rule coneE*)

from $\langle b \in \text{cone } (h, U) \rangle$ **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 $\langle a' \in P[U] \rangle \langle b' \in P[U] \rangle$ **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 $(a :: \Rightarrow_0 - :: \text{comm-ring}) \in \text{cone } hU$

shows $- a \in \text{cone } hU$

proof –

obtain $h \ U$ **where** $hU: hU = (h, \ U)$ **using** *prod.exhaust* **by** *blast*
with *assms* **have** $a \in \text{cone } (h, \ U)$ **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 $\langle a' \in P[U] \rangle$ **have** $- \ a' \in P[U]$ **by** (rule *Polys-closed-uminus*)
ultimately show *?thesis* **unfolding** hU **by** (rule *coneI*)

qed

lemma *cone-closed-minus*:

assumes $(a::- \Rightarrow_0 \ -::\text{comm-ring}) \in \text{cone } hU$ **and** $b \in \text{cone } hU$
shows $a - b \in \text{cone } hU$

proof –

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

qed

lemma *cone-closed-times*:

assumes $a \in \text{cone } (h, \ U)$ **and** $q \in P[U]$
shows $q * a \in \text{cone } (h, \ U)$

proof –

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) $\langle a' \in P[U] \rangle$ **have** $q * a' \in P[U]$ **by** (rule *Polys-closed-times*)
ultimately show *?thesis* **by** (rule *coneI*)

qed

corollary *cone-closed-monom-mult*:

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

proof –

from *assms*(2) **have** *monomial* $c \ t \in P[U]$ **by** (rule *Polys-closed-monomial*)
with *assms*(1) **have** *monomial* $c \ t * a \in \text{cone } (h, \ U)$ **by** (rule *cone-closed-times*)
thus *?thesis* **by** (*simp only: times-monomial-left*)

qed

lemma *coneD*:

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

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* $p = \text{lpp } a + \text{lpp } h$ **unfolding** p **by** (rule *lp-times*)
also have $\dots = \text{lpp } h + \text{lpp } a$ **by** (rule *add.commute*)
finally show *?thesis* **by** (rule *addsI*)

qed

lemma *cone-mono-1*:

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

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

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

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

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

lemma *cone-subset-PolysD*:

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

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

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 $\text{monomial } (1::'a) (\text{Poly-Mapping.single } x \ 1) \in P[U]$ (**is** $?p \in -$)

by (*intro Polys-closed-monomial PPs-closed-single*)

with *refl* **have** $?p * h \in \text{cone } (h, U)$ **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** $\langle h \in P[X] \rangle \langle h \neq 0 \rangle$ **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 $h \neq 0 \implies U \subseteq X$

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

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 $a \in \text{cone } (h, U)$

then **obtain** q **where** $q \in P[U]$ and $a = q * h$ **by** (*rule coneE*)

from *this(1)* $\langle P[U] \subseteq P[X] \rangle$ **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*: $(*) a \text{ ' cone } (h, U) = \text{cone } (a * h, U)$

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 = 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 ∈ cone hU
moreover obtain h U where hU: hU = (h, U) using prod.exhaust by blast
ultimately have a ∈ cone (h, U) by simp
thus c · a ∈ 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 :: - ⇒0 'a::{comm-ring-1,ring-no-zero-divisors}
  assumes x ∉ 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 ∈ listset ?ss
    then obtain a b where a ∈ cone (h, U) and b: b ∈ cone (xx * h, insert x U)
      and qs: qs = [a, b] by (rule listset-doubletonE)
    note this(1)
    also have cone (h, U) ⊆ cone (h, insert x U) by (rule cone-mono-2) blast
    finally have a: a ∈ cone (h, insert x U) .
    have cone (xx * h, insert x U) ⊆ cone (h, insert x U)
      by (rule cone-mono-1) (simp add: xx-def Polys-def PPs-closed-single)
    with b have b ∈ cone (h, insert x U) ..
    with a have a + b ∈ cone (h, insert x U) by (rule cone-closed-plus)
    thus sum-list qs ∈ cone (h, insert x U) by (simp add: qs)
  next
    fix a
    assume a ∈ cone (h, insert x U)
    then obtain q where q ∈ 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: ex-
cept-decomp)
    have qU ∈ P[U] by (rule PolysI) (simp add: qU-def keys-except)
    have x-adds: ?x adds t if t ∈ keys qx for t unfolding adds-poly-mapping
le-fun-def
    proof
      fix y
      show lookup ?x y ≤ lookup t y
      proof (cases y = x)

```

```

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

```

definition *valid-decomp* :: 'x set ⇒ (((x ⇒₀ nat) ⇒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 ⇒₀ nat) ⇒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*:
 $(\bigwedge h\ U.\ (h,\ U) \in set\ ps \implies h \in P[X]) \implies (\bigwedge h\ U.\ (h,\ U) \in set\ ps \implies h \neq 0)$
 \implies
 $(\bigwedge h\ U.\ (h,\ U) \in set\ ps \implies U \subseteq X) \implies valid-decomp\ X\ ps$
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 $\bigwedge ps.\ ps \in set\ pss \implies valid-decomp\ X\ ps$
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** $\langle (h,\ U) \in set\ ps \rangle$ **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 ∈ P[X] and s ≠ (0::- ⇒₀ -::semiring-no-zero-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 ∈ P[X] and a ≠ 0 and U ⊆ X by (rule
valid-decompD)+
  from assms(2) this(1) show h ∈ P[X] unfolding h by (rule Polys-closed-times)
  from assms(3) ⟨a ≠ 0⟩ show h ≠ 0 unfolding h by (rule times-not-zero)
  from ⟨U ⊆ X⟩ show U ⊆ 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** $(a, U) \in \text{set } ps$ **by** *simp-all*
from *assms(1)* **this(2)** **have** *is-monomial* a **and** $\text{punit.lc } a = 1$ **by** (rule *monomial-decompD*)
from *this(1)* **have** *monomial* $(\text{punit.lc } a) (\text{lpp } a) = a$ **by** (rule *punit.monomial-eq-itself*)
moreover define t **where** $t = \text{lpp } a$
ultimately have a : $a = \text{monomial } 1\ t$ **by** (*simp only*: $\langle \text{punit.lc } a = 1 \rangle$)
from *assms(2)* **have** *monomial* $(\text{punit.lc } f) (\text{lpp } f) = f$ **by** (rule *punit.monomial-eq-itself*)
moreover define s **where** $s = \text{lpp } f$
ultimately have f : $f = \text{monomial } 1\ s$ **by** (*simp only*: *assms(3)*)
show *is-monomial* h **by** (*simp add*: $h\ a\ f\ \text{times-monomial-monomial monomial-is-monomial}$)
show $\text{punit.lc } h = 1$ **by** (*simp add*: $h\ a\ f\ \text{times-monomial-monomial}$)
qed

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

assumes *monomial-decomp* ps **and** $hU \in \text{set } ps$ **and** $a \in \text{cone } hU$
shows *monomial* $(\text{lookup } a\ t)$ $t \in \text{cone } hU$
proof (*cases* $t \in \text{keys } a$)
case *True*
obtain $h\ U$ **where** hU : $hU = (h, U)$ **using** *prod.exhaust* **by** *blast*
with *assms(2)* **have** $(h, U) \in \text{set } ps$ **by** *simp*
with *assms(1)* **have** *is-monomial* h **by** (rule *monomial-decompD*)
then obtain $c\ s$ **where** h : $h = \text{monomial } c\ s$ **by** (rule *is-monomial-monomial*)
from *assms(3)* **obtain** q **where** $q \in P[U]$ **and** $a = q * h$ **unfolding** hU **by** (rule *coneE*)
from *this(2)* **have** $a = h * q$ **by** (*simp only*: *mult.commute*)
also have $\dots = \text{punit.monom-mult } c\ s\ q$ **by** (*simp only*: $h\ \text{times-monomial-left}$)
finally have a : $a = \text{punit.monom-mult } c\ s\ q$.
with *True* **have** $t \in \text{keys } (\text{punit.monom-mult } c\ s\ q)$ **by** *simp*
hence $t \in (+)\ s\ \text{'keys } q$ **using** *punit.keys-monom-mult-subset[simplified]* ..
then obtain u **where** $u \in \text{keys } q$ **and** t : $t = s + u$..
note *this(1)*
also from $\langle q \in P[U] \rangle$ **have** $\text{keys } q \subseteq .[U]$ **by** (rule *PolysD*)
finally have $u \in .[U]$.
have *monomial* $(\text{lookup } a\ t)$ $t = \text{monomial } (\text{lookup } q\ u)\ u * h$
by (*simp add*: $a\ t\ \text{punit.lookup-monom-mult } h\ \text{times-monomial-monomial mult.commute}$)
moreover from $\langle u \in .[U] \rangle$ **have** *monomial* $(\text{lookup } q\ u)\ u \in P[U]$ **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** $a \in \text{sum-list 'listset } (\text{map } \text{cone } ps)$ **and** $t \in \text{keys } a$
obtains $c\ h\ U$ **where** $(h, U) \in \text{set } ps$ **and** $c \neq 0$ **and** *monomial* $c\ t \in \text{cone } (h,$

U)
proof –
 from *assms*(2) **obtain** qs **where** $qs\text{-in}$: $qs \in \text{listset } (\text{map } \text{cone } ps)$ **and** a : $a = \text{sum-list } qs$..
 from *assms*(3) $keys\text{-sum-list-subset}$ **have** $t \in \text{Keys } (\text{set } qs)$ **unfolding** a ..
 then **obtain** q **where** $q \in \text{set } qs$ **and** $t \in \text{keys } q$ **by** (*rule in-KeysE*)
 from *this*(1) **obtain** i **where** $i < \text{length } qs$ **and** q : $q = qs ! i$ **by** (*metis in-set-conv-nth*)
 moreover from $qs\text{-in}$ **have** $\text{length } qs = \text{length } (\text{map } \text{cone } ps)$ **by** (*rule listsetD*)
 ultimately **have** $i < \text{length } (\text{map } \text{cone } ps)$ **by** *simp*
 moreover from $qs\text{-in}$ *this* **have** $qs ! i \in (\text{map } \text{cone } ps) ! i$ **by** (*rule listsetD*)
 ultimately **have** $ps ! i \in \text{set } ps$ **and** $q \in \text{cone } (ps ! i)$ **by** (*simp-all add: q*)
 with *assms*(1) **have** $*$: $\text{monomial } (\text{lookup } q t) t \in \text{cone } (ps ! i)$
 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 $\langle ps ! i \in \text{set } ps \rangle$ **show** $(h, U) \in \text{set } ps$ **by** (*simp only: psi*)
next
 from $\langle t \in \text{keys } q \rangle$ **show** $\text{lookup } q t \neq 0$ **by** (*simp add: in-keys-iff*)
next
 from $*$ **show** $\text{monomial } (\text{lookup } q t) t \in \text{cone } (h, U)$ **by** (*simp only: psi*)
qed
qed

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

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

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

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

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

ultimately have $h: h = f * a$ and $(a, U) \in \text{set } ps$ 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 $(h, U) \in \text{set } ps$
 with *assms* have *is-monomial h* by (rule *monomial-decompD*)
 then obtain $c t$ where $h: h = \text{monomial } c t$ by (rule *is-monomial-monomial*)
 show homogeneous h unfolding h by (fact *homogeneous-monomial*)
 qed

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

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

lemma *cone-decomp-cone-subset*:
 assumes *cone-decomp T ps* and $hU \in \text{set } ps$
 shows *cone hU \subseteq T*
proof
 fix p
 assume $p \in \text{cone } hU$
 from *assms(2)* obtain i where $i < \text{length } ps$ and $hU: hU = ps ! i$ by (metis
in-set-conv-nth)
 define qs where $qs = (\text{map } 0 ps)[i := p]$
 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 $j < \text{length } (\text{map cone } ps)$
 with $\langle i < \text{length } ps \rangle \langle p \in \text{cone } hU \rangle$ show $qs ! j \in \text{map cone } ps ! j$
 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* $\langle i < \text{length } ps \rangle$)
 also from $\langle i < \text{length } ps \rangle$ have $\dots = p$ 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 $(h, U) \in \text{set } ps$
 shows $h \in P[X]$ and $h \neq (0::-\Rightarrow_0 \text{::}\{\text{comm-semiring-1, semiring-no-zero-divisors}\})$

$\Rightarrow U \subseteq X$

proof –

from *assms*(1, 3) have $\text{cone } (h, U) \subseteq T$ **by** (rule *cone-decomp-cone-subset*)

hence $\text{cone } (h, U) \subseteq P[X]$ **using** *assms*(2) **by** (rule *subset-trans*)

thus $h \in P[X]$ and $h \neq 0 \Rightarrow U \subseteq X$ **by** (rule *cone-subset-PolysD*)+

qed

lemma *cone-decomp-closed-plus*:

assumes *cone-decomp* T *ps* and $a \in T$ and $b \in T$

shows $a + b \in T$

proof –

from *assms*(1) have *dd*: *direct-decomp* T (map *cone ps*) **by** (rule *cone-decompD*)

then obtain *qsa* where *qsa*: $qsa \in \text{listset } (\text{map } \text{cone } ps)$ and $a = \text{sum-list } qsa$ **using** *assms*(2)

by (rule *direct-decompE*)

from *dd* *assms*(3) obtain *qsb* where *qsb*: $qsb \in \text{listset } (\text{map } \text{cone } ps)$ and $b = \text{sum-list } qsb$

by (rule *direct-decompE*)

from *qsa* have $\text{length } qsa = \text{length } (\text{map } \text{cone } ps)$ **by** (rule *listsetD*)

moreover from *qsb* have $\text{length } qsb = \text{length } (\text{map } \text{cone } ps)$ **by** (rule *listsetD*)

ultimately have $a + b = \text{sum-list } (\text{map2 } (+) qsa qsb)$ **by** (simp only: *sum-list-map2-plus* a b)

also from *dd* have $\text{sum-list } (\text{map2 } (+) qsa qsb) \in T$

proof (rule *direct-decompD*)

from *qsa* *qsb* show $\text{map2 } (+) qsa qsb \in \text{listset } (\text{map } \text{cone } ps)$

proof (rule *listset-closed-map2*)

fix *c* *p1* *p2*

assume $c \in \text{set } (\text{map } \text{cone } ps)$

then obtain hU where $c: c = \text{cone } hU$ **by** *auto*

assume $p1 \in c$ and $p2 \in c$

thus $p1 + p2 \in c$ **unfolding** *c* **by** (rule *cone-closed-plus*)

qed

qed

finally show *?thesis* .

qed

lemma *cone-decomp-closed-uminus*:

assumes *cone-decomp* T *ps* and $(a::\Rightarrow_0 \text{::comm-ring}) \in T$

shows $- a \in T$

proof –

from *assms*(1) have *dd*: *direct-decomp* T (map *cone ps*) **by** (rule *cone-decompD*)

then obtain *qsa* where *qsa*: $qsa \in \text{listset } (\text{map } \text{cone } ps)$ and $a = \text{sum-list } qsa$ **using** *assms*(2)

by (rule *direct-decompE*)

from *qsa* have $\text{length } qsa = \text{length } (\text{map } \text{cone } ps)$ **by** (rule *listsetD*)

have $- a = \text{sum-list } (\text{map } \text{uminus } qsa)$ **unfolding** *a* **by** (induct *qsa*, *simp-all*)

also from *dd* have $\dots \in T$

proof (rule *direct-decompD*)

from *qsa* show $\text{map } \text{uminus } qsa \in \text{listset } (\text{map } \text{cone } ps)$

```

proof (rule listset-closed-map)
  fix  $c$   $p$ 
  assume  $c \in \text{set } (\text{map cone } ps)$ 
  then obtain  $hU$  where  $c: c = \text{cone } hU$  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  $(a::\Rightarrow_0 \text{::comm-ring}) \in T$  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  $T$  ( $[S2] @ \text{map cone } ps$ ) by (rule direct-decomp-direct-decomp)
  hence direct-decomp  $T$  ( $S2 \# \text{map cone } ps$ ) by simp
  moreover from assms(3) have direct-decomp  $S2$  (map cone  $qs$ ) by (rule cone-decompD)
  ultimately have direct-decomp  $T$  (map cone  $ps @ \text{map cone } qs$ ) 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  $\text{length } pss = \text{length } ss$ 
and  $\bigwedge i. i < \text{length } ss \implies \text{cone-decomp } (ss ! i) (pss ! i)$ 
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 $0 < \text{length } (s \# ss)$ **by** *simp*
hence *cone-decomp* $((s \# ss) ! 0) ((ps \# pss) ! 0)$ **by** (*rule Cons.prem*s)
hence *cone-decomp s ps* **by** *simp*
hence $*$: *direct-decomp s* (*map cone ps*) **by** (*rule cone-decompD*)
with *Cons.prem*s(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 $i < \text{length } ss$
hence *Suc i* $< \text{length } (s \# ss)$ **by** *simp*
hence *cone-decomp* $((s \# ss) ! \text{Suc } i) ((ps \# pss) ! \text{Suc } i)$ **by** (*rule Cons.prem*s)
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* $((*) s ' T) (\text{map } (\text{apfst } ((*) (s :: \Rightarrow_0 :: \{\text{comm-ring-1, ring-no-zero-divisors}\}))) ps)$
proof (*rule cone-decompI*)
from *assms* **have** *direct-decomp T (map cone ps)* **by** (*rule cone-decompD*)
hence *direct-decomp* $((*) s ' T) (\text{map } ((') ((*) s)) (\text{map cone ps}))$
by (*rule direct-decomp-image-times*) (*rule times-canc-left*)
also have $\text{map } ((') ((*) s)) (\text{map cone ps}) = \text{map cone } (\text{map } (\text{apfst } ((*) s)) ps)$
by (*simp add: cone-image-times'*)
finally show *direct-decomp* $((*) s ' T) (\text{map cone } (\text{map } (\text{apfst } ((*) s)) ps))$.
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 $\langle \text{mset } ps = \text{mset } qs \rangle$ **show** $\langle \text{mset } (\text{map cone } ps) = \text{mset } (\text{map cone } qs) \rangle$
 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** $qs \in \text{listset } (\text{map cone } ps)$ and $p: p = \text{sum-list } qs$ **using**
 $\langle p \in T \rangle$
 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 $(h, U) \in \text{set } (a \# ps)$ **by** simp
 with Cons.prem1 **have** $h \in P[X]$ and $U \subseteq X$ **by** (rule valid-decompD)+
 hence cone $a \subseteq P[X]$ **unfolding** a **by** (rule cone-subset-PolysI)
 from Cons.prem1 **have** valid-decomp X ps **by** (simp add: valid-decomp-def)
 from Cons.prem2 **have** $qs \in \text{listset } (\text{cone } a \# \text{map cone } ps)$ **by** simp
 then obtain q qs' **where** $q \in \text{cone } a$ and $qs': qs' \in \text{listset } (\text{map cone } ps)$ and
 $qs: qs = q \# qs'$
 by (rule listset-ConsE)
 from this(1) $\langle \text{cone } a \subseteq P[X] \rangle$ **have** $q \in P[X]$..
 moreover from $\langle \text{valid-decomp } X \text{ } ps \rangle$ qs' **have** $\text{sum-list } qs' \in P[X]$ **by** (rule
 Cons.hyps)
 ultimately **have** $q + \text{sum-list } qs' \in P[X]$ **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 $cn \in \text{set } (\text{map cone } ps)$
 then obtain hU **where** $hU \in \text{set } ps$ and $cn: cn = \text{cone } hU$ **unfolding** set-map
 ..
 moreover obtain h U **where** $hU: hU = (h, U)$ **using** prod.exhaust **by** blast

ultimately have $(h, U) \in \text{set } ps$ 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* $(T::(- \Rightarrow_0 \text{::field}) \text{ set})$
proof (rule *phull.subspace-direct-decomp*)
 from *assms* show *direct-decomp* $T \ (\text{map cone } ps)$ by (rule *cone-decompD*)
 next
 fix cn
 assume $cn \in \text{set } (\text{map cone } ps)$
 then obtain hU where $hU \in \text{set } ps$ and $cn: cn = \text{cone } hU$ unfolding *set-map*
 ..
 show *phull.subspace* cn unfolding cn by (rule *subspace-cone*)
 qed

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

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

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

lemma *pos-decomp-subset*: $\text{set } (ps_+) \subseteq \text{set } ps$
 by (*simp add: pos-decomp-def*)

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

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

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

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

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

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

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

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

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

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

ultimately show $\exists h' U'. (h', U') \in \text{set } (\text{concat } pss) \wedge \text{poly-deg } h' = d \wedge \text{card } U \leq \text{card } U'$
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 $(\text{concat } [ps, qs])$ **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** $s \neq (0::\text{'a}::\text{semiring-no-zero-divisors})$
shows *standard-decomp* $(k + \text{poly-deg } s)$ $(\text{map } (\text{apfst } ((*)) s)) ps$

proof (rule *standard-decompI*)

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

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

lemma *standard-decomp-nonempty-unique:*

assumes *finite X* and *valid-decomp X ps* and *standard-decomp k ps* and $ps_+ \neq$
 \square
 shows $k = \text{Min } (\text{poly-deg } \langle \text{fst } \langle \text{set } (ps_+) \rangle \rangle)$
proof –
 let $?A = \text{poly-deg } \langle \text{fst } \langle \text{set } (ps_+) \rangle \rangle$
 define m where $m = \text{Min } ?A$
 have *finite ?A* by *simp*
 moreover from *assms(4)* have $?A \neq \{\}$ by *simp*
 ultimately have $m \in ?A$ unfolding *m-def* by (rule *Min-in*)
 then obtain $h \ U$ where $(h, U) \in \text{set } (ps_+)$ and $m: m = \text{poly-deg } h$ by *fastforce*
 have *m-min*: $m \leq \text{poly-deg } h'$ if $(h', U') \in \text{set } (ps_+)$ for $h' \ U'$
proof –
 from *that* have $\text{poly-deg } (\text{fst } (h', U')) \in ?A$ by (intro *imageI*)
 with $\langle \text{finite } ?A \rangle$ have $m \leq \text{poly-deg } (\text{fst } (h', U'))$ unfolding *m-def* by (rule
Min-le)
 thus *?thesis* by *simp*
qed
 show *?thesis*
proof (rule *linorder-cases*)
 assume $k < m$
 hence $k \leq \text{poly-deg } h$ by (simp add: *m*)
 with *assms(3)* $\langle (h, U) \in \text{set } (ps_+) \rangle$ le-refl obtain $h' \ U'$
 where $(h', U') \in \text{set } ps$ and $\text{poly-deg } h' = k$ and $\text{card } U \leq \text{card } U'$ by (rule
standard-decompE)
 from *this(2)* $\langle k < m \rangle$ have $\neg m \leq \text{poly-deg } h'$ by *simp*
 with *m-min* have $(h', U') \notin \text{set } (ps_+)$ by *blast*
 with $\langle (h', U') \in \text{set } ps \rangle$ have $U' = \{\}$ by (simp add: *pos-decomp-def*)
 with $\langle \text{card } U \leq \text{card } U' \rangle$ have $U = \{\} \vee \text{infinite } U$ by (simp add: *card-eq-0-iff*)
 thus *?thesis*
proof
 assume $U = \{\}$
 with $\langle (h, U) \in \text{set } (ps_+) \rangle$ 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 $\langle (h, U) \in \text{set } (ps_+) \rangle$ show $(h, U) \in \text{set } ps$ by (simp add: *pos-decomp-def*)
qed
 ultimately show *?thesis* ..
qed
next
 assume $m < k$
 hence $\neg k \leq m$ by *simp*
 moreover from *assms(3)* $\langle (h, U) \in \text{set } (ps_+) \rangle$ have $k \leq m$ 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 $h \neq (0::-\Rightarrow_0 'a::\{\text{comm-ring-1, ring-no-zero-divisors}\})$*

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 $h \Rightarrow \text{punit.lc } h = 1 \Rightarrow \text{monomial-decomp ps and homogeneous } h \Rightarrow \text{hom-decomp ps}$*

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.prem(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 $h \Rightarrow \text{punit.lc } h = 1 \Rightarrow \text{monomial-decomp ps}$*

and *4: homogeneous $h \Rightarrow \text{hom-decomp ps}$* **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.prem(1))*

from *$\langle x \in X \rangle$* **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** *$\langle ?x \neq 0 \rangle$* *assms(4)* **have** *?x * h $\neq 0$* **by** *(rule times-not-zero)*

ultimately **have** *valid-decomp X qs* **using** *insert.hyps(1) $\langle x \in X \rangle \langle U \subseteq X \rangle$*

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 stan-
dard-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  $\leq$  d
obtains qs where valid-decomp X qs and cone-decomp T qs and standard-decomp
d qs
and monomial-decomp ps  $\implies$  monomial-decomp qs and hom-decomp ps  $\implies$ 
hom-decomp qs
proof –
have  $\exists qs. \text{valid-decomp } X \text{ } qs \wedge \text{cone-decomp } T \text{ } qs \wedge \text{standard-decomp } (k + i) \text{ } qs$ 
 $\wedge$ 
 $(\text{monomial-decomp } ps \longrightarrow \text{monomial-decomp } qs) \wedge (\text{hom-decomp } ps \longrightarrow$ 
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  $\implies$  mono-
mial-decomp qs

```

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

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

define f **where** $f = (\lambda hU. \text{SOME } ps'. \text{valid-decomp } X \text{ } ps' \wedge \text{cone-decomp } (\text{cone } hU) \text{ } ps' \wedge$
 $\text{standard-decomp } (\text{Suc } (\text{poly-deg } ((\text{fst } hU)::('x \Rightarrow_0$
 $\text{--}) \Rightarrow_0 'a))) \text{ } ps' \wedge$
 $(\text{monomial-decomp } ps \longrightarrow \text{monomial-decomp } ps') \wedge$
 $(\text{hom-decomp } ps \longrightarrow \text{hom-decomp } ps'))$

have $\text{valid-decomp } X (f hU) \wedge \text{cone-decomp } (\text{cone } hU) (f hU) \wedge \text{standard-decomp } (\text{Suc } (k + i)) (f hU) \wedge$
 $(\text{monomial-decomp } ps \longrightarrow \text{monomial-decomp } (f hU)) \wedge (\text{hom-decomp } ps \longrightarrow \text{hom-decomp } (f hU))$
if $hU \in \text{set } rs$ **for** hU

proof –
obtain $h \text{ } U$ **where** $hU: hU = (h, U)$ **using** *prod.exhaust* **by** *blast*
with $that$ **have** $eq: \text{poly-deg } (\text{fst } hU) = k + i$ **by** (*simp add: rs-def*)
from $that$ $\langle \text{set } rs \subseteq \text{set } qs \rangle$ **have** $(h, U) \in \text{set } qs$ **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** $\text{valid-decomp } X \text{ } ps'$ **and** $\text{cone-decomp } (\text{cone } (h, U)) \text{ } ps'$
and $\text{standard-decomp } (\text{Suc } (\text{poly-deg } h)) \text{ } ps'$
and $md: \text{is-monomial } h \implies \text{punit.lc } h = 1 \implies \text{monomial-decomp } ps'$
and $hd: \text{homogeneous } h \implies \text{hom-decomp } ps'$ **by** (*rule standard-decomp-SucE*)

blast

note *this*(1–3)
moreover **have** $\text{monomial-decomp } ps'$ **if** $\text{monomial-decomp } ps$

proof –
from $that$ **have** $\text{monomial-decomp } qs$ **by** (*rule 3*)
hence $\text{is-monomial } h$ **and** $\text{punit.lc } h = 1$ **using** $\langle (h, U) \in \text{set } qs \rangle$ **by** (*rule monomial-decompD*) +
thus $?thesis$ **by** (*rule md*)

qed

moreover **have** $\text{hom-decomp } ps'$ **if** $\text{hom-decomp } ps$

proof –
from $that$ **have** $\text{hom-decomp } qs$ **by** (*rule 4*)
hence $\text{homogeneous } h$ **using** $\langle (h, U) \in \text{set } qs \rangle$ **by** (*rule hom-decompD*)
thus $?thesis$ **by** (*rule hd*)

qed

ultimately **have** $\text{valid-decomp } X \text{ } ps' \wedge \text{cone-decomp } (\text{cone } hU) \text{ } ps' \wedge$
 $\text{standard-decomp } (\text{Suc } (\text{poly-deg } (\text{fst } hU))) \text{ } ps' \wedge (\text{monomial-decomp } ps \longrightarrow$
 $\text{monomial-decomp } ps') \wedge$
 $(\text{hom-decomp } ps \longrightarrow \text{hom-decomp } ps')$ **by** (*simp add: hU*)
thus $?thesis$ **unfolding** $f\text{-def } eq$ **by** (*rule someI*)

```

qed
hence f1:  $\bigwedge ps. ps \in \text{set } (\text{map } f \text{ } rs) \implies \text{valid-decomp } X \text{ } ps$ 
and f2:  $\bigwedge hU. hU \in \text{set } rs \implies \text{cone-decomp } (\text{cone } hU) (f \text{ } hU)$ 
and f3:  $\bigwedge ps. ps \in \text{set } (\text{map } f \text{ } rs) \implies \text{standard-decomp } (\text{Suc } (k + i)) \text{ } ps$ 
and f4:  $\bigwedge ps'. \text{monomial-decomp } ps \implies ps' \in \text{set } (\text{map } f \text{ } rs) \implies \text{monomial-decomp } ps'$ 
and f5:  $\bigwedge ps'. \text{hom-decomp } ps \implies ps' \in \text{set } (\text{map } f \text{ } rs) \implies \text{hom-decomp } ps'$ 
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  $\langle \text{set } ss \subseteq \text{set } qs \rangle$  ..
    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 $(h, U) \in \text{set } (qs_+)$ and $\text{poly-deg } h \neq k + i$ by (simp-all add:
 pos-decomp-def ss-def)
 from 2 this(1) have $k + i \leq \text{poly-deg } h$ by (rule standard-decompD)
 with $\langle \text{poly-deg } h \neq k + i \rangle$ show $\text{Suc } (k + i) \leq \text{poly-deg } h$ by simp

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

10.5 Splitting w.r.t. Ideals

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

definition *splits-wrt* :: $((('x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a) \times 'x \text{ set}) \text{ list} \times ((('x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a) \times 'x \text{ set}) \text{ list}) \Rightarrow$
 $((('x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a::\text{comm-ring-1}) \text{ set} \Rightarrow ((('x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a) \text{ set} \Rightarrow \text{bool})$
where *splits-wrt* *pqs* *T* *F* \longleftrightarrow *cone-decomp* *T* (*fst* *pqs* @ *snd* *pqs*) \wedge
 $(\forall hU \in \text{set } (\text{fst } pqs). \text{cone } hU \subseteq \text{ideal } F \cap P[X]) \wedge$
 $(\forall (h, U) \in \text{set } (\text{snd } pqs). \text{cone } (h, U) \subseteq P[X] \wedge \text{cone } (h,$
 $U) \cap \text{ideal } F = \{0\})$

lemma *splits-wrtI*:

assumes *cone-decomp* *T* (*ps* @ *qs*)
and $\bigwedge h U. (h, U) \in \text{set } ps \implies \text{cone } (h, U) \subseteq P[X]$ **and** $\bigwedge h U. (h, U) \in \text{set } ps \implies h \in \text{ideal } F$
and $\bigwedge h U. (h, U) \in \text{set } qs \implies \text{cone } (h, U) \subseteq P[X]$
and $\bigwedge h U a. (h, U) \in \text{set } qs \implies a \in \text{cone } (h, U) \implies a \in \text{ideal } F \implies a = 0$
shows *splits-wrt* (*ps*, *qs*) *T* *F*
unfolding *splits-wrt-def* *fst-conv* *snd-conv*
proof (*intro conjI ballI*)
fix *hU*
assume *hU* $\in \text{set } ps$
moreover obtain *h* *U* **where** *hU*: *hU* = (*h*, *U*) **using** *prod.exhaust* **by** *blast*
ultimately have (*h*, *U*) $\in \text{set } ps$ **by** *simp*
hence $\text{cone } (h, U) \subseteq P[X]$ **and** $h \in \text{ideal } F$ **by** (*rule asms*) +
from - *this*(1) **show** $\text{cone } hU \subseteq \text{ideal } F \cap P[X]$ **unfolding** *hU*
proof (*rule Int-greatest*)
show $\text{cone } (h, U) \subseteq \text{ideal } F$
proof
fix *a*
assume *a* $\in \text{cone } (h, U)$
then obtain *a'* **where** *a'* $\in P[U]$ **and** *a*: *a* = *a'* * *h* **by** (*rule coneE*)
from $\langle h \in \text{ideal } F \rangle$ **show** *a* $\in \text{ideal } F$ **unfolding** *a* **by** (*rule ideal.span-scale*)
qed
qed
next
fix *hU*
assume *hU* $\in \text{set } qs$
moreover obtain *h* *U* **where** *hU*: *hU* = (*h*, *U*) **using** *prod.exhaust* **by** *blast*
ultimately have (*h*, *U*) $\in \text{set } qs$ **by** *simp*
hence $\text{cone } (h, U) \subseteq P[X]$ **and** $\bigwedge a. a \in \text{cone } (h, U) \implies a \in \text{ideal } F \implies a = 0$ **by** (*rule asms*) +
moreover have $0 \in \text{cone } (h, U) \cap \text{ideal } F$
by (*simp add: zero-in-cone ideal.span-zero*)
ultimately show *case* *hU* *of* (*h*, *U*) $\implies \text{cone } (h, U) \subseteq P[X] \wedge \text{cone } (h, U) \cap \text{ideal } F = \{0\}$
by (*fastforce simp: hU*)
qed (*fact asms*) +

lemma *splits-wrtI-valid-decomp*:

```

assumes valid-decomp  $X$   $ps$  and valid-decomp  $X$   $qs$  and cone-decomp  $T$  ( $ps$  @
 $qs$ )
  and  $\bigwedge h U. (h, U) \in \text{set } ps \implies h \in \text{ideal } F$ 
  and  $\bigwedge h U a. (h, U) \in \text{set } qs \implies a \in \text{cone } (h, U) \implies a \in \text{ideal } F \implies a = 0$ 
shows splits-wrt ( $ps, qs$ )  $T$   $F$ 
using assms(3) - - - assms(5)
proof (rule splits-wrtI)
  fix  $h U$ 
  assume  $(h, U) \in \text{set } ps$ 
  thus  $h \in \text{ideal } F$  by (rule assms(4))
  from assms(1)  $\langle (h, U) \in \text{set } ps \rangle$  have  $h \in P[X]$  and  $U \subseteq X$  by (rule valid-decompD)+
  thus  $\text{cone } (h, U) \subseteq P[X]$  by (rule cone-subset-PolysI)
next
  fix  $h U$ 
  assume  $(h, U) \in \text{set } qs$ 
  with assms(2) have  $h \in P[X]$  by (rule valid-decompD)
  moreover from assms(2)  $\langle (h, U) \in \text{set } qs \rangle$  have  $U \subseteq X$  by (rule valid-decompD)
  ultimately show  $\text{cone } (h, U) \subseteq P[X]$  by (rule cone-subset-PolysI)
qed

lemma splits-wrtD:
  assumes splits-wrt ( $ps, qs$ )  $T$   $F$ 
  shows cone-decomp  $T$  ( $ps$  @  $qs$ ) and  $hU \in \text{set } ps \implies \text{cone } hU \subseteq \text{ideal } F \cap P[X]$ 
  and  $hU \in \text{set } qs \implies \text{cone } hU \subseteq P[X]$  and  $hU \in \text{set } qs \implies \text{cone } hU \cap \text{ideal } F = \{0\}$ 
  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)  $\subseteq \text{ideal } F \cap P[X]$ 
proof
  fix  $x$ 
  assume  $x\text{-in}: x \in \text{sum-list ' listset } (\text{map cone } ps)$ 
  have listset (map cone ps)  $\subseteq \text{listset } (\text{map } (\lambda-. \text{ideal } F \cap P[X]) \text{ } ps)$ 
  proof (rule listset-mono)
    fix  $i$ 
    assume  $i: i < \text{length } (\text{map } (\lambda-. \text{ideal } F \cap P[X]) \text{ } ps)$ 
    hence  $ps ! i \in \text{set } ps$  by simp
    with assms(1) have  $\text{cone } (ps ! i) \subseteq \text{ideal } F \cap P[X]$  by (rule splits-wrtD)
    with  $i$  show  $\text{map cone } ps ! i \subseteq \text{map } (\lambda-. \text{ideal } F \cap P[X]) \text{ } ps ! i$  by simp
  qed simp
  hence sum-list ' listset (map cone ps)  $\subseteq \text{sum-list ' listset } (\text{map } (\lambda-. \text{ideal } F \cap P[X]) \text{ } ps)$ 
  by (rule image-mono)
  with  $x\text{-in}$  have  $x \in \text{sum-list ' listset } (\text{map } (\lambda-. \text{ideal } F \cap P[X]) \text{ } ps)$  ..
  then obtain  $xs$  where  $xs: xs \in \text{listset } (\text{map } (\lambda-. \text{ideal } F \cap P[X]) \text{ } ps)$  and  $x: x = \text{sum-list } xs$  ..
  have  $1: y \in \text{ideal } F \cap P[X]$  if  $y \in \text{set } xs$  for  $y$ 

```


proof –
 from *that* obtain i where $i < \text{length } xs$ and $y: y = xs ! i$ by (*metis in-set-conv-nth*)
 moreover from xs have $\text{length } xs = \text{length } (\text{map } (\lambda-. \text{ideal } F \cap P[X]) \text{ } ps)$
 by (*rule listsetD*)
 ultimately have $i < \text{length } (\text{map } (\lambda-. \text{ideal } F \cap P[X]) \text{ } ps)$ by *simp*
 moreover from xs this have $xs ! i \in (\text{map } (\lambda-. \text{ideal } F \cap P[X]) \text{ } ps) ! i$ by
 (*rule listsetD*)
 ultimately show $y \in \text{ideal } F \cap P[X]$ by (*simp add: y*)
qed
 show $x \in \text{ideal } F \cap P[X]$ unfolding x
proof
 show $\text{sum-list } xs \in \text{ideal } F$
proof (*rule ideal.span-closed-sum-list[simplified]*)
 fix y
 assume $y \in \text{set } xs$
 hence $y \in \text{ideal } F \cap P[X]$ by (*rule 1*)
 thus $y \in \text{ideal } F$ by *simp*
qed
 next
 show $\text{sum-list } xs \in P[X]$
proof (*rule Polys-closed-sum-list*)
 fix y
 assume $y \in \text{set } xs$
 hence $y \in \text{ideal } F \cap P[X]$ 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 $\text{sum-list } ' \text{listset } (\text{map cone } qs) \subseteq P[X]$
proof
 fix x
 assume $x\text{-in}: x \in \text{sum-list } ' \text{listset } (\text{map cone } qs)$
 have $\text{listset } (\text{map cone } qs) \subseteq \text{listset } (\text{map } (\lambda-. P[X]) \text{ } qs)$
proof (*rule listset-mono*)
 fix i
 assume $i: i < \text{length } (\text{map } (\lambda-. P[X]) \text{ } qs)$
 hence $qs ! i \in \text{set } qs$ by *simp*
 with *assms(1)* have $\text{cone } (qs ! i) \subseteq P[X]$ by (*rule splits-wrtD*)
 with i show $\text{map cone } qs ! i \subseteq \text{map } (\lambda-. P[X]) \text{ } qs ! i$ by *simp*
qed *simp*
 hence $\text{sum-list } ' \text{listset } (\text{map cone } qs) \subseteq \text{sum-list } ' \text{listset } (\text{map } (\lambda-. P[X]) \text{ } qs)$
 by (*rule image-mono*)
 with $x\text{-in}$ have $x \in \text{sum-list } ' \text{listset } (\text{map } (\lambda-. P[X]) \text{ } qs)$..
 then obtain xs where $xs: xs \in \text{listset } (\text{map } (\lambda-. P[X]) \text{ } qs)$ and $x: x = \text{sum-list } xs$..

```

show  $x \in P[X]$  unfolding  $x$ 
proof (rule Polys-closed-sum-list)
  fix  $y$ 
  assume  $y \in \text{set } xs$ 
  then obtain  $i$  where  $i < \text{length } xs$  and  $y = xs ! i$  by (metis in-set-conv-nth)
  moreover from  $xs$  have  $\text{length } xs = \text{length } (\text{map } (\lambda-. P[X]::(- \Rightarrow_0 'a) \text{ set}) \text{ } qs)$ 
    by (rule listsetD)
  ultimately have  $i < \text{length } (\text{map } (\lambda-. P[X]) \text{ } qs)$  by simp
  moreover from  $xs$  this have  $xs ! i \in (\text{map } (\lambda-. P[X]) \text{ } qs) ! i$  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
    ( $F::(- \Rightarrow_0 'a::\text{field}) \text{ set}$ )
    — The last two assumptions are missing in the paper.
  shows cone-decomp ( $T \cap \text{ideal } F$ )  $ps$ 
proof —
  from assms(1) have  $*$ : cone-decomp  $T$  ( $ps @ qs$ ) by (rule splits-wrtD)
  hence direct-decomp  $T$  ( $\text{map cone } ps @ \text{map cone } qs$ ) by (simp add: cone-decomp-def)
  hence 1: direct-decomp ( $\text{sum-list ' listset } (\text{map cone } ps)$ ) ( $\text{map cone } ps$ )
    and 2: direct-decomp  $T$  [ $\text{sum-list ' listset } (\text{map cone } ps), \text{sum-list ' listset } (\text{map cone } qs)$ ]
  by (auto dest: direct-decomp-appendD intro!: empty-not-in-map-cone)
  let  $?ss = [\text{sum-list ' listset } (\text{map cone } ps), \text{sum-list ' listset } (\text{map cone } qs)]$ 
  show ?thesis
proof (intro cone-decompI direct-decompI)
  from 1 show inj-on  $\text{sum-list } (\text{listset } (\text{map cone } ps))$  by (simp only: direct-decomp-def bij-betw-def)
next
  from assms(1) have  $\text{sum-list ' listset } (\text{map cone } ps) \subseteq \text{ideal } F \cap P[X]$ 
    by (rule splits-wrt-image-sum-list-fst-subset)
  hence sub:  $\text{sum-list ' listset } (\text{map cone } ps) \subseteq \text{ideal } F$  by simp
  show  $\text{sum-list ' listset } (\text{map cone } ps) = T \cap \text{ideal } F$ 
proof (rule set-eqI)
  fix  $x$ 
  show  $x \in \text{sum-list ' listset } (\text{map cone } ps) \longleftrightarrow x \in T \cap \text{ideal } F$ 
proof
  assume  $x\text{-in}$ :  $x \in \text{sum-list ' listset } (\text{map cone } ps)$ 
  show  $x \in T \cap \text{ideal } F$ 
proof (intro IntI)
  have  $\text{map } (\lambda-. 0) \text{ } qs \in \text{listset } (\text{map cone } qs)$  (is ?ys  $\in$  -)
    by (induct qs) (auto intro: listset-ConsI zero-in-cone simp del: list-set.simps(2))
  hence  $\text{sum-list ' listset } (\text{map cone } qs)$  by (rule imageI)
  hence  $0 \in \text{sum-list ' listset } (\text{map cone } qs)$  by simp
  with  $x\text{-in}$  have  $[x, 0] \in \text{listset } ?ss$  using refl by (rule listset-doubletonI)
  with 2 have  $\text{sum-list } [x, 0] \in T$  by (rule direct-decompD)

```

```

    thus  $x \in T$  by simp
  next
    from  $x$ -in sub show  $x \in \text{ideal } F$  ..
  qed
next
  assume  $x \in T \cap \text{ideal } F$ 
  hence  $x \in T$  and  $x \in \text{ideal } F$  by simp-all
  from 2 this(1) obtain  $xs$  where  $xs \in \text{listset } ?ss$  and  $x: x = \text{sum-list } xs$ 
    by (rule direct-decompE)
  from this(1) obtain  $p \ q$  where  $p: p \in \text{sum-list } \text{'listset } (\text{map cone } ps)$ 
    and  $q: q \in \text{sum-list } \text{'listset } (\text{map cone } qs)$  and  $xs: xs = [p, q]$ 
    by (rule listset-doubletonE)
  from  $\langle x \in \text{ideal } F \rangle$  have  $p + q \in \text{ideal } F$  by (simp add:  $x \ xs$ )
  moreover from  $p$  sub have  $p \in \text{ideal } F$  ..
  ultimately have  $p + q - p \in \text{ideal } F$  by (rule ideal.span-diff)
  hence  $q \in \text{ideal } F$  by simp
  have  $q = 0$ 
  proof (rule ccontr)
    assume  $q \neq 0$ 
    hence  $\text{keys } q \neq \{\}$  by simp
    then obtain  $t$  where  $t \in \text{keys } q$  by blast
    with  $\text{assms}(2)$   $q$  obtain  $c \ h \ U$  where  $(h, U) \in \text{set } qs$  and  $c \neq 0$ 
    and  $\text{monomial } c \ t \in \text{cone } (h, U)$  by (rule monomial-decomp-sum-list-monomial-in-cone)
    moreover from  $\text{assms}(3)$   $\langle q \in \text{ideal } F \rangle \langle t \in \text{keys } q \rangle$  have  $\text{monomial } c \ t$ 
       $\in \text{ideal } F$ 
      by (rule punit.monomial-pmdl-field[simplified])
    ultimately have  $\text{monomial } c \ t \in \text{cone } (h, U) \cap \text{ideal } F$  by simp
    also from  $\text{assms}(1)$   $\langle (h, U) \in \text{set } qs \rangle$  have  $\dots = \{0\}$  by (rule splits-wrtD)
    finally have  $c = 0$  by (simp add: monomial-0-iff)
    with  $\langle c \neq 0 \rangle$  show False ..
  qed
  with  $p$  show  $x \in \text{sum-list } \text{'listset } (\text{map cone } ps)$  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 $T \cap \text{ideal } F \cap P[X]$.

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

assumes *finite* X **and** *splits-wrt* (ps, qs) $T \ F$ **and** *monomial-decomp* qs **and** *is-monomial-set* F

and $F \subseteq P[X]$

shows *cone-decomp* $(T \cap \text{normal-form } F \text{' } P[X]) \ qs$

proof –

from $\text{assms}(2)$ **have** $*$: *cone-decomp* $T \ (ps \ @ \ qs)$ **by** (rule *splits-wrtD*)

hence *direct-decomp* $T \ (\text{map cone } ps \ @ \ \text{map cone } qs)$ **by** (simp add: *cone-decomp-def*)

hence 1: *direct-decomp* $(\text{sum-list } \text{'listset } (\text{map cone } qs)) \ (\text{map cone } qs)$

and 2: *direct-decomp* $T \ [\text{sum-list } \text{'listset } (\text{map cone } ps), \text{sum-list } \text{'listset } (\text{map$

```

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  $\subseteq$  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)  $\subseteq$  ideal F  $\cap$  P[X]
    by (rule splits-wrt-image-sum-list-fst-subset)
    hence sub: sum-list ' listset (map cone ps)  $\subseteq$  ideal F by simp
    show sum-list ' listset (map cone qs) = T  $\cap$  normal-form F ' P[X]
    proof (rule set-eqI)
      fix x
      show x  $\in$  sum-list ' listset (map cone qs)  $\longleftrightarrow$  x  $\in$  T  $\cap$  normal-form F ' P[X]
      proof
        assume x-in: x  $\in$  sum-list ' listset (map cone qs)
        show x  $\in$  T  $\cap$  normal-form F ' P[X]
        proof (intro IntI)
          have map ( $\lambda$ -. 0) ps  $\in$  listset (map cone ps) (is ?ys  $\in$  -)
          by (induct ps) (auto intro: listset-ConsI zero-in-cone simp del: list-set.simps(2))
          hence sum-list ?ys  $\in$  sum-list ' listset (map cone ps) by (rule imageI)
          hence 0  $\in$  sum-list ' listset (map cone ps) by simp
          from this x-in have [0, x]  $\in$  listset ?ss using refl by (rule listset-doubletonI)
          with 2 have sum-list [0, x]  $\in$  T by (rule direct-decompD)
          thus x  $\in$  T by simp
        next
          from assms(2) have sum-list ' listset (map cone qs)  $\subseteq$  P[X]
          by (rule splits-wrt-image-sum-list-snd-subset)
          with x-in have x  $\in$  P[X] ..
          moreover have  $\neg$  punit.is-red ?G x
          proof
            assume punit.is-red ?G x
            then obtain g t where g  $\in$  ?G and t  $\in$  keys x and g  $\neq$  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)  $\in$  set qs and c
 $\neq$  0
            and monomial c t  $\in$  cone (h, U) by (rule monomial-decomp-sum-list-monomial-in-cone)
            note this(3)
            moreover have monomial c t  $\in$  ideal ?G
            proof (rule punit.is-red-monomial-monomial-set-in-pmdl[simplified])
              from  $\langle c \neq 0 \rangle$  show is-monomial (monomial c t) by (rule monomial-is-monomial)
            end
          end
        end
      end
    end
  end

```

```

      next
        from assms(1, 5, 4) show is-monomial-set ?G by (rule re-
duced-GB-is-monomial-set-Polys)
      next
        from  $\langle c \neq 0 \rangle$  have  $t \in \text{keys } (\text{monomial } c \ t)$  by simp
        with  $\langle g \in ?G \rangle \langle g \neq 0 \rangle$  show punit.is-red ?G (monomial c t) using
adds
          by (rule punit.is-red-addsI[simplified])
        qed
        ultimately have monomial c  $t \in \text{cone } (h, U) \cap \text{ideal } F$  by (simp add:
ideal-G)
        also from assms(2)  $\langle (h, U) \in \text{set } qs \rangle$  have  $\dots = \{0\}$  by (rule splits-wrtD)
        finally have  $c = 0$  by (simp add: monomial-0-iff)
        with  $\langle c \neq 0 \rangle$  show False ..
        qed
        ultimately show  $x \in \text{normal-form } F \text{ ' } P[X]$ 
          using assms(1, 5) by (simp add: image-normal-form-iff)
        qed
      next
        assume  $x \in T \cap \text{normal-form } F \text{ ' } P[X]$ 
        hence  $x \in T$  and  $x \in \text{normal-form } F \text{ ' } P[X]$  by simp-all
        from this(2) assms(1, 5) have  $x \in P[X]$  and irred:  $\neg \text{punit.is-red } ?G \ x$ 
          by (simp-all add: image-normal-form-iff)
        from 2  $\langle x \in T \rangle$  obtain xs where  $xs \in \text{listset } ?ss$  and  $x = \text{sum-list } xs$ 
          by (rule direct-decompE)
        from this(1) obtain p q where  $p: p \in \text{sum-list ' listset } (\text{map cone } ps)$ 
          and  $q: q \in \text{sum-list ' listset } (\text{map cone } qs)$  and  $xs = [p, q]$ 
          by (rule listset-doubletonE)
        have  $x = p + q$  by (simp add: x xs)
        from p sub have  $p \in \text{ideal } F$  ..
        have  $p = 0$ 
        proof (rule ccontr)
          assume  $p \neq 0$ 
          hence  $\text{keys } p \neq \{\}$  by simp
          then obtain t where  $t \in \text{keys } p$  by blast
          from assms(4)  $\langle p \in \text{ideal } F \rangle \langle t \in \text{keys } p \rangle$  have  $\exists: \text{monomial } c \ t \in \text{ideal}$ 
F for c
            by (rule punit.monomial-pmdl-field[simplified])
          have  $t \notin \text{keys } q$ 
          proof
            assume  $t \in \text{keys } q$ 
            with assms(3) q obtain c h U where  $(h, U) \in \text{set } qs$  and  $c \neq 0$ 
            and monomial c  $t \in \text{cone } (h, U)$  by (rule monomial-decomp-sum-list-monomial-in-cone)
            from this(3)  $\exists$  have monomial c  $t \in \text{cone } (h, U) \cap \text{ideal } F$  by simp
            also from assms(2)  $\langle (h, U) \in \text{set } qs \rangle$  have  $\dots = \{0\}$  by (rule splits-wrtD)
            finally have  $c = 0$  by (simp add: monomial-0-iff)
            with  $\langle c \neq 0 \rangle$  show False ..
          qed
          with  $\langle t \in \text{keys } p \rangle$  have  $t \in \text{keys } x$  unfolding  $\langle x = p + q \rangle$  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 $?x + t \in \text{keys } (\text{punit.monom-mult } 1 \text{ } ?x \text{ } a)$
 also have $\dots \subseteq (+) \text{ } ?x \text{ } \text{'keys } a \text{ by (rule punit.keys-monom-mult-subset[simplified])}$
 finally obtain s where $s \in \text{keys } a$ and $?x + t = ?x + s \dots$
 from $\text{this}(2)$ have $t = s$ by *simp*
 with $\langle s \in \text{keys } a \rangle 2(2)$ show *False* by *simp*
 qed
 ultimately obtain f where $f \in \text{monomial } (1::'a) \text{ } S$ and $\text{adds: lpp } f \text{ adds}$
 $?x + t$
 by (rule *punit.keys-monomial-pmdl[simplified]*)
 from $\text{this}(1)$ obtain s where $s \in S$ and $f: f = \text{monomial } 1 \text{ } s \dots$
 from adds have $s \text{ adds } ?x + t$ by (*simp add: f punit.lt-monomial*)
 hence $s - ?x \text{ adds } t$
 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 $\langle s \in S \rangle$ have $\text{monomial } 1 \text{ } (s - ?x) \in ?S$ by (*intro imageI*)
 also have $\dots \subseteq \text{ideal } ?S$ by (*rule ideal.span-superset*)
 finally have $\text{monomial } c \text{ } s' * \text{monomial } 1 \text{ } (s - ?x) \in \text{ideal } ?S$
 by (*rule ideal.span-scale*)
 hence $\text{monomial } c \text{ } t \in \text{ideal } ?S$ by (*simp add: times-monomial-monomial t*
add.commute)
 moreover have $a \in \text{ideal } ?S$
 proof –
 from $\langle f \in \text{monomial } 1 \text{ } S \rangle$ have $f \in \text{ideal } (\text{monomial } 1 \text{ } S)$ by (*rule*
ideal.span-base)
 hence $\text{punit.monom-mult } c \text{ } (?x + t - s) f \in \text{ideal } (\text{monomial } 1 \text{ } S)$
 by (*rule punit.pmdl-closed-monom-mult[simplified]*)
 with $\langle s \text{ adds } ?x + t \rangle$ have $\text{monomial } c \text{ } (?x + t) \in \text{ideal } (\text{monomial } 1 \text{ } S)$
 by (*simp add: f punit.monom-mult-monomial adds-minus*)
 with 1 have $\text{monomial } c \text{ } (?x + t) + \text{punit.monom-mult } 1 \text{ } ?x \text{ } a - \text{monomial}$
 $c \text{ } (?x + t) \in \text{ideal } (\text{monomial } 1 \text{ } S)$
 by (*rule ideal.span-diff*)
 thus $?thesis$ by (*simp add: 2(3) del: One-nat-def*)
 qed
 ultimately show $\text{monomial } c \text{ } t + a \in \text{ideal } ?S$
 by (*rule ideal.span-add*)
 next
 have $\text{is-monomial-set } ?S$ by (*auto intro!: is-monomial-setI monomial-is-monomial*)
 moreover assume $1: \text{monomial } c \text{ } t + a \in \text{ideal } ?S$
 moreover from $- 2(2)$ have $t \in \text{keys } (\text{monomial } c \text{ } t + a)$
 proof (*rule in-keys-plusI1*)
 from $2(1)$ show $t \in \text{keys } (\text{monomial } c \text{ } t)$ by *simp*
 qed
 ultimately obtain f where $f \in ?S$ and $\text{adds: lpp } f \text{ adds } t$
 by (*rule punit.keys-monomial-pmdl[simplified]*)
 from $\text{this}(1)$ obtain s where $s \in S$ and $f: f = \text{monomial } 1 \text{ } (s - ?x)$ by
blast
 from adds have $s - ?x \text{ adds } t$ by (*simp add: f punit.lt-monomial*)
 hence $s \text{ adds } ?x + t$
 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 $\langle s \in S \rangle$ **have** $\text{monomial } 1 \ s \in \text{monomial } 1 \ ' S$ **by** (rule imageI)
also have $\dots \subseteq \text{ideal } (\text{monomial } 1 \ ' S)$ **by** (rule ideal.span-superset)
finally have $\text{monomial } c \ s' * \text{monomial } 1 \ s \in \text{ideal } (\text{monomial } 1 \ ' S)$
by (rule ideal.span-scale)
hence $\text{monomial } c \ (?x + t) \in \text{ideal } (\text{monomial } 1 \ ' S)$
by (simp only: t) (simp add: times-monomial-monomial add.commute)
moreover have $\text{punit.monom-mult } 1 \ ?x \ a \in \text{ideal } (\text{monomial } 1 \ ' S)$
proof –
from $\langle f \in ?S \rangle$ **have** $f \in \text{ideal } ?S$ **by** (rule ideal.span-base)
hence $\text{punit.monom-mult } c \ (t - (s - ?x)) \ f \in \text{ideal } ?S$
by (rule punit.pmdl-closed-monom-mult[simplified])
with $\langle s - ?x \text{ adds } t \rangle$ **have** $\text{monomial } c \ t \in \text{ideal } ?S$
by (simp add: f punit.monom-mult-monomial adds-minus)
with 1 **have** $\text{monomial } c \ t + a - \text{monomial } c \ t \in \text{ideal } ?S$
by (rule ideal.span-diff)
thus $?thesis$ **by** (simp add: 2(3) del: One-nat-def)
qed
ultimately have $\text{monomial } c \ (?x + t) + \text{punit.monom-mult } 1 \ ?x \ a \in \text{ideal } (\text{monomial } 1 \ ' S)$
by (rule ideal.span-add)
thus $\text{punit.monom-mult } 1 \ ?x \ (\text{monomial } c \ t + a) \in \text{ideal } (\text{monomial } 1 \ ' S)$
by (simp add: punit.monom-mult-monomial punit.monom-mult-dist-right)
qed
qed
finally show $a \in \text{ideal } (\text{monomial } 1 \ ' S) \div \text{monomial } 1 \ ?x \longleftrightarrow a \in \text{ideal } (\lambda s. s - ?x) \ ' S$.
qed

lemma *lem-4-2-1*:
assumes $\text{ideal } F \div \text{monomial } 1 \ t = \text{ideal } (\text{monomial } (1::'a::\text{comm-ring-1}) \ ' S)$
shows $\text{cone } (\text{monomial } 1 \ t, U) \subseteq \text{ideal } F \longleftrightarrow 0 \in S$
proof
have $\text{monomial } 1 \ t \in \text{cone } (\text{monomial } (1::'a) \ t, U)$ **by** (rule tip-in-cone)
also assume $\text{cone } (\text{monomial } 1 \ t, U) \subseteq \text{ideal } F$
finally have $*$: $\text{monomial } 1 \ t * 1 \in \text{ideal } F$ **by** simp
have *is-monomial-set* $(\text{monomial } (1::'a) \ ' S)$
by (auto intro!: *is-monomial-setI* *monomial-is-monomial*)
moreover from $*$ **have** $1 \in \text{ideal } (\text{monomial } (1::'a) \ ' S)$ **by** (simp only: *quot-set-iff flip: assms*)
moreover have $0 \in \text{keys } (1::'a \Rightarrow_0 'a)$ **by** simp
ultimately obtain g **where** $g \in \text{monomial } (1::'a) \ ' S$ **and** *adds: lpp g adds 0*
by (rule punit.keys-monomial-pmdl[simplified])
from *this*(1) **obtain** s **where** $s \in S$ **and** $g: g = \text{monomial } 1 \ s$..
from *adds* **have** $s \text{ adds } 0$ **by** (simp add: *g punit.lt-monomial flip: single-one*)
with $\langle s \in S \rangle$ **show** $0 \in S$ **by** (simp only: *adds-zero*)
next


```

assume  $0 \in S$ 
hence  $\text{monomial } 1 \ 0 \in \text{monomial } (1::'a) \text{ ' } S$  by (rule imageI)
hence  $1 \in \text{ideal } (\text{monomial } (1::'a) \text{ ' } S)$  unfolding single-one by (rule ideal.span-base)
hence  $\text{eq: ideal } F \div \text{monomial } 1 \ t = \text{UNIV}$  (is  $- \div ?t = -$ )
  by (simp only: assms ideal-eq-UNIV-iff-contains-one)
show  $\text{cone } (\text{monomial } 1 \ t, \ U) \subseteq \text{ideal } F$ 
proof
  fix  $a$ 
  assume  $a \in \text{cone } (?t, \ U)$ 
  then obtain  $q$  where  $a = q * ?t$  by (rule coneE)
  have  $q \in \text{ideal } F \div ?t$  by (simp add: eq)
  thus  $a \in \text{ideal } F$  by (simp only: a quot-set-iff mult.commute)
qed
qed

```

lemma *lem-4-2-2*:

```

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

```

proof

```

let  $?t = \text{monomial } (1::'a) \ t$ 
assume  $\text{eq: cone } (?t, \ U) \cap \text{ideal } F = \{0\}$ 
{
  fix  $s$ 
  assume  $s \in S$ 
  hence  $\text{monomial } 1 \ s \in \text{monomial } (1::'a) \text{ ' } S$  (is  $?s \in -$ ) by (rule imageI)
  hence  $?s \in \text{ideal } (\text{monomial } 1 \text{ ' } S)$  by (rule ideal.span-base)
  also have  $\dots = \text{ideal } F \div ?t$  by (simp only: assms)
  finally have  $*$ :  $?s * ?t \in \text{ideal } F$  by (simp only: quot-set-iff mult.commute)
  assume  $s \in .[U]$ 
  hence  $?s \in P[U]$  by (rule Polys-closed-monomial)
  with refl have  $?s * ?t \in \text{cone } (?t, \ U)$  by (rule coneI)
  with * have  $?s * ?t \in \text{cone } (?t, \ U) \cap \text{ideal } F$  by simp
  hence False by (simp add: eq times-monomial-monomial monomial-0-iff)
}
thus  $S \cap .[U] = \{\}$  by blast

```

next

```

let  $?t = \text{monomial } (1::'a) \ t$ 
assume  $\text{eq: } S \cap .[U] = \{\}$ 
{
  fix  $a$ 
  assume  $a \in \text{cone } (?t, \ U)$ 
  then obtain  $q$  where  $q \in P[U]$  and  $a = q * ?t$  by (rule coneE)
  assume  $a \in \text{ideal } F$ 
  have  $a = 0$ 
  proof (rule ccontr)
    assume  $a \neq 0$ 
    hence  $q \neq 0$  by (auto simp: a)
    from  $\langle a \in \text{ideal } F \rangle$  have  $*$ :  $q \in \text{ideal } F \div ?t$  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:  $q \in \text{ideal } (1::'a) ' S$  by (simp only:
assms)
  moreover from  $\langle q \neq 0 \rangle$  have lpp  $q \in \text{keys } q$  by (rule punit.lt-in-keys)
  ultimately obtain g where  $g \in \text{monomial } (1::'a) ' S$  and adds: lpp g adds
lpp q
    by (rule punit.keys-monomial-pmdl[simplified])
    from this(1) obtain s where  $s \in S$  and g:  $g = \text{monomial } 1 \ s ..$ 
    from  $\langle q \neq 0 \rangle$  have lpp  $q \in \text{keys } q$  by (rule punit.lt-in-keys)
    also from  $\langle q \in P[U] \rangle$  have  $\dots \subseteq \cdot[U]$  by (rule PolysD)
    finally have lpp  $q \in \cdot[U]$  .
    moreover from adds have s adds lpp q by (simp add: g punit.lt-monomial)
    ultimately have  $s \in \cdot[U]$  by (rule PPs-closed-adds)
    with eq  $\langle s \in S \rangle$  show False by blast
  qed
}
thus cone (?t, U)  $\cap \text{ideal } F = \{0\}$  using zero-in-cone ideal.span-zero by blast
qed

```

10.6 Function *split*

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

lemma *max-subset*:

assumes *finite* *A* **and** $B \subseteq A$ **and** *P* *B*
shows *max-subset* *A* *P* $\subseteq A$ (**is** *?thesis1*)
and *P* (*max-subset* *A* *P*) (**is** *?thesis2*)
and *card* *B* $\leq \text{card } (\text{max-subset } A \ P)$ (**is** *?thesis3*)

proof –

from *assms*(2, 3) **have** $B \subseteq A \wedge P \ B$ **by** *simp*
moreover have $\forall C. C \subseteq A \wedge P \ C \longrightarrow \text{card } C < \text{Suc } (\text{card } A)$
proof (*intro allI impI, elim conjE*)

fix *C*

assume $C \subseteq A$

with *assms*(1) **have** *card* *C* $\leq \text{card } A$ **by** (*rule card-mono*)

thus *card* *C* $< \text{Suc } (\text{card } A)$ **by** *simp*

qed

ultimately have *?thesis1* \wedge *?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) \Rightarrow 'x set \Rightarrow ('x \Rightarrow_0 nat) set \Rightarrow
 $(((((('x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a) \times ('x \text{set})) \text{list}) \times$
 $(((((('x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a::\{\text{zero}, \text{one}\}) \times ('x \text{set})) \text{list}))$

where

split *t* *U* *S* =

```

    (if 0 ∈ S then
      [(monomial 1 t, U)], [])
    else if S ∩ .[U] = {} then
      ([], [(monomial 1 t, U)])
    else
      let x = SOME x'. x' ∈ U - (max-subset U (λV. S ∩ .[V] = {}));
      (ps0, qs0) = split t (U - {x}) S;
      (ps1, qs1) = split (Poly-Mapping.single x 1 + t) U ((λf. f -
Poly-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) ⊆ X and finite (snd (snd args))
  shows split-dom TYPE('a::{zero,one}) args
proof -
  let ?m = λargs'. card (fst (snd args')) + sum deg-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.premis have U ⊆ 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' ⊆ X and finite F' and card U' + sum deg-pm F' < card U + sum
deg-pm F
  for t' U' F'
  using less.hyps that by (simp add: args)

```

define *S* where $S = \text{max-subset } U (\lambda V. F \cap .[V] = \{\})$

define *x* where $x = (\text{SOME } x'. x' \in U \wedge x' \notin S)$

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 $\langle \text{finite } U \rangle$ *empty-subsetI* **have** $S \subseteq U$ and $F \cap .[S] = \{\}$

unfolding *S-def* **by** (rule *max-subset*)+

have $x \in U \wedge x \notin S$ **unfolding** *x-def*

proof (rule *someI-ex*)

from $\langle f \in F \rangle \langle f \in .[U] \rangle \langle F \cap .[S] = \{\} \rangle$ **have** $S \neq U$ **by** *blast*

with $\langle S \subseteq U \rangle$ **show** $\exists y. y \in U \wedge y \notin S$ **by** *blast*

qed

hence $x \in U$ and $x \notin S$ **by** *simp-all*

{

assume $\neg \text{split-dom TYPE('a)} (t, U - \{x\}, F)$

```

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)
      }
    contradiction
  qed

```

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

corollary *split-domI*: $\text{finite } X \implies U \subseteq X \implies \text{finite } S \implies \text{split-dom TYPE('a::\{zero,one\})}$
 (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 {}* = $([], [(\text{monomial } (1::'a::\{zero,one\}) \ t, U)])$
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 $P :: ('x \Rightarrow_0 \text{nat}) \Rightarrow -$
 assumes *finite X* **and** $U \subseteq X$ **and** *finite S*
 assumes $\bigwedge t \ U \ S. U \subseteq X \implies \text{finite } S \implies 0 \in S \implies P \ t \ U \ S$ $((\text{monomial } (1::'a::\{zero,one\}) \ t, U), [])$
 assumes $\bigwedge t \ U \ S. U \subseteq X \implies \text{finite } S \implies 0 \notin S \implies S \cap \cdot.[U] = \{\} \implies P \ t \ U$
 $S \ ([], [(\text{monomial } 1 \ t, U)])$
 assumes $\bigwedge t \ U \ S \ V \ x \ ps0 \ ps1 \ qs0 \ qs1. U \subseteq X \implies \text{finite } S \implies 0 \notin S \implies S \cap$
 $\cdot.[U] \neq \{\} \implies V \subseteq U \implies$
 $S \cap \cdot.[V] = \{\} \implies (\bigwedge V'. V' \subseteq U \implies S \cap \cdot.[V'] = \{\} \implies \text{card } V' \leq$
 $\text{card } V) \implies$
 $x \in U \implies x \notin V \implies V = \text{max-subset } U \ (\lambda V'. S \cap \cdot.[V'] = \{\}) \implies x$
 $= (\text{SOME } x'. x' \in U - V) \implies$

```

      (ps0, qs0) = split t (U - {x}) S  $\implies$ 
      (ps1, qs1) = split (Poly-Mapping.single x 1 + t) U (( $\lambda f$ . f -
Poly-Mapping.single x 1) ' S)  $\implies$ 
      split t U S = (ps0 @ ps1, qs0 @ qs1)  $\implies$ 
      P t (U - {x}) S (ps0, qs0)  $\implies$ 
      P (Poly-Mapping.single x 1 + t) U (( $\lambda f$ . f - Poly-Mapping.single x 1)
' S) (ps1, qs1)  $\implies$ 
      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)  $\times$  -) 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'  $\leq$  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  $\langle 0 \notin F \rangle \langle F \cap .[U] \neq \{\} \rangle$  - step(5)
  proof (rule step(2))
    from  $\langle U \subseteq X \rangle$  show  $U - \{x\} \subseteq X$  by fastforce
  qed (simp add: x-def S-def)
next
  show  $P \ (Poly\text{-}Mapping.single \ x \ 1 + t) \ U \ ((\lambda f. f - Poly\text{-}Mapping.single \ x \ 1) \ 'F) \ (ps1, \ qs1)$ 
    unfolding  $PQ1$  using  $\langle 0 \notin F \rangle \langle F \cap .[U] \neq \{\} \rangle$  - refl PQ0  $\langle U \subseteq X \rangle$ 
  proof (rule step(3))
    from  $\langle finite \ F \rangle$  show  $finite \ ((\lambda f. f - Poly\text{-}Mapping.single \ x \ 1) \ 'F)$  by
(rule finite-imageI)
  qed (simp add: x-def S-def)
next
  show  $split \ t \ U \ F = (ps0 \ @ \ ps1, \ qs0 \ @ \ qs1)$  using  $\langle 0 \notin F \rangle \langle F \cap .[U] \neq \{\} \rangle$ 
    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  $U \subseteq X$  and finite S and  $t \in .[X]$ 
shows valid-decomp X (fst ((split t U S)::(-  $\times$  (((-  $\Rightarrow_0$  'a::zero-neq-one)  $\times$  -) list))))
and valid-decomp X (snd ((split t U S)::(-  $\times$  (((-  $\Rightarrow_0$  'a::zero-neq-one)  $\times$  -) list))))
  (is valid-decomp - (snd ?s))

```

proof -

```

from assms have valid-decomp X (fst ?s)  $\wedge$  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  $x \in X$  ..
      hence  $Poly\text{-}Mapping.single \ x \ 1 \in .[X]$  by (rule PPs-closed-single)
      hence  $Poly\text{-}Mapping.single \ x \ 1 + t \in .[X]$  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)  $\wedge$  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  $t \in \cdot[X]$ 
    and  $\text{ideal } F \div \text{monomial } 1 \ t = \text{ideal } (\text{monomial } 1 \ 'S)$ 
  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  $\text{cone } (\text{monomial } 1 \ t, U) \subseteq \text{ideal } F$  by (simp only: lem-4-2-1
base1(5))
    show ?case
    proof (rule splits-wrtI)
      fix h0 U0
      assume  $(h0, U0) \in \text{set } [(\text{monomial } (1::'a) \ t, U)]$ 
      hence  $h0: h0 = \text{monomial } 1 \ t$  and  $U0 = U$  by simp-all
      note this(1)
      also have  $\text{monomial } 1 \ t \in \text{cone } (\text{monomial } (1::'a) \ t, U)$  by (fact tip-in-cone)
      also have  $\dots \subseteq \text{ideal } F$  by fact
      finally show  $h0 \in \text{ideal } F$  .

    from base1(4) have  $h0 \in P[X]$  unfolding h0 by (rule Polys-closed-monomial)
    moreover from base1(1) have  $U0 \subseteq X$  by (simp only:  $\langle U0 = U \rangle$ )
    ultimately show  $\text{cone } (h0, U0) \subseteq P[X]$  by (rule cone-subset-PolysI)
  qed (simp-all add: cone-decomp-singleton  $\langle U \subseteq X \rangle$ )
next
  case (base2 t U S)

```



```

from base2(4) have cone (monomial 1 t, U)  $\cap$  ideal F = {0} by (simp only:
lem-4-2-2 base2(6))
show ?case
proof (rule splits-wrtI)
  fix h0 U0
  assume (h0, U0)  $\in$  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  $\in$  P[X] by (rule Polys-closed-monomial)
  finally have h0  $\in$  P[X] .
  moreover from base2(1) have U0  $\subseteq$  X by (simp only:  $\langle U0 = U \rangle$ )
  ultimately show cone (h0, U0)  $\subseteq$  P[X] by (rule cone-subset-PolysI)
next
  fix h0 U0 a
  assume (h0, U0)  $\in$  set [(monomial (1::'a) t, U)] and a  $\in$  cone (h0, U0)
  hence a  $\in$  cone (monomial 1 t, U) by simp
  moreover assume a  $\in$  ideal F
  ultimately have a  $\in$  cone (monomial 1 t, U)  $\cap$  ideal F by (rule IntI)
  also have ... = {0} by fact
  finally show a = 0 by simp
  qed (simp-all add: cone-decomp-singleton  $\langle U \subseteq X \rangle$ )
next
  case (step t U S V x ps0 ps1 qs0 qs1)
  let ?x = Poly-Mapping.single x 1
  from step.premis 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  $\in$  X ..
    hence ?x  $\in$  .[X] by (rule PPs-closed-single)
    thus ?x + t  $\in$  .[X] using step.premis(1) by (rule PPs-closed-plus)
  next
    have ideal F  $\div$  monomial 1 (?x + t) = ideal F  $\div$  monomial 1 t  $\div$  monomial
    1 ?x
    by (simp add: times-monomial-monomial add.commute)
    also have ... = ideal (monomial 1 ' S)  $\div$  monomial 1 ?x by (simp only:
step.premis)
    finally show ideal F  $\div$  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  $\bigwedge a. a \in \text{cone } (h0, U0) \implies a \in \text{ideal } F \implies a = 0$  by blast
  qed
  qed

```

lemma *lem-4-5*:

```

  assumes finite X and  $U \subseteq X$  and  $t \in \cdot[X]$  and  $F \subseteq P[X]$ 
    and  $\text{ideal } F \div \text{monomial } 1 \ t = \text{ideal } (\text{monomial } (1::'a) \ 'S)$ 
    and  $\text{cone } (\text{monomial } (1::'a::\text{field}) \ t', V) \subseteq \text{cone } (\text{monomial } 1 \ t, U) \cap \text{normal-form } F \ 'P[X]$ 
  shows  $V \subseteq U$  and  $S \cap \cdot[V] = \{\}$ 
  proof -
    let ?t = monomial (1::'a) t
    let ?t' = monomial (1::'a) t'
    from assms(6) have 1:  $\text{cone } (?t', V) \subseteq \text{cone } (?t, U)$  and 2:  $\text{cone } (?t', V) \subseteq \text{normal-form } F \ 'P[X]$ 
    by blast+
    from this(1) show  $V \subseteq U$  by (rule cone-subsetD) (simp add: monomial-0-iff)
  
```

show $S \cap \cdot[V] = \{\}$

proof

```

  let ?t = monomial (1::'a) t
  let ?t' = monomial (1::'a) t'
  show  $S \cap \cdot[V] \subseteq \{\}$ 

```

proof

fix s

assume $s \in S \cap \cdot[V]$

hence $s \in S$ and $s \in \cdot[V]$ by simp-all

from this(2) have $\text{monomial } (1::'a) \ s \in P[V]$ (is ?s ∈ -) by (rule Polys-closed-monomial)

with refl have $?s * ?t \in \text{cone } (?t, V)$ by (rule coneI)

from tip-in-cone 1 have $?t' \in \text{cone } (?t, U)$..

then obtain s' where $s' \in P[U]$ and $t': ?t' = s' * ?t$ by (rule coneE)

note this(1)

also from assms(2) have $P[U] \subseteq P[X]$ by (rule Polys-mono)

finally have $s' \in P[X]$.

have $s' * ?s * ?t = ?s * ?t'$ by (simp add: t')

also from refl $\langle ?s \in P[V] \rangle$ have $\dots \in \text{cone } (?t', V)$ by (rule coneI)

finally have $s' * ?s * ?t \in \text{cone } (?t', V)$.

hence 1: $s' * ?s * ?t \in \text{normal-form } F \ 'P[X]$ using 2 ..

from $\langle s \in S \rangle$ have $?s \in \text{monomial } 1 \ 'S$ by (rule imageI)

hence $?s \in \text{ideal } (\text{monomial } 1 \ 'S)$ by (rule ideal.span-base)

hence $s' * ?s \in \text{ideal } (\text{monomial } 1 \ 'S)$ by (rule ideal.span-scale)

hence $s' * ?s \in \text{ideal } F \div ?t$ by (simp only: assms(5))

hence $s' * ?s * ?t \in \text{ideal } F$ by (simp only: quot-set-iff mult.commute)

hence $s' * ?s * ?t \in \text{ideal } F \cap \text{normal-form } F \ 'P[X]$ using 1 by (rule IntI)

also from assms(1, 4) have $\dots \subseteq \{0\}$

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.prem(2))
    next
      case (base2 t U S)
      show ?case
      proof (rule base2.prem(1))
        from base2.hyps(1) assms(1) have finite U by (rule finite-subset)
        thus card V ≤ card U using base2.prem(1) by (rule card-mono)
      qed simp
    next
      case (step t U S V0 x ps0 ps1 qs0 qs1)
      from step.prem(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.prem(1))
        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
qed

```

lemma *lem-4-7*:

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

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

and *monomial-decomp* ps

obtains U **where** $(g, U) \in \text{set } ps$

proof –

let $?S = \text{monomial} (1::'a)$ ‘ S

let $?G = \text{punit.reduced-GB } ?S$

note $\text{assms}(1)$

moreover from $\text{assms}(2)$ **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** $\text{assms}(3)$ **by** (*rule is-monomial-setD*)

hence $g \neq 0$ **by** (*rule monomial-not-0*)

moreover from $\text{assms}(1)$ ‘ $?S \subseteq P[X]$ ‘ **have** *punit.is-monic-set* $?G$

by (*rule reduced-GB-is-monic-set-Polys*)

ultimately have *punit.lc* $g = 1$ **using** $\text{assms}(3)$ **by** (*simp add: punit.is-monic-set-def*)

moreover define t **where** $t = \text{lpp } g$

moreover from ‘*is-monomial* g ’ **have** *monomial* $(\text{punit.lc } g) (\text{lpp } g) = g$

by (*rule punit.monomial-eq-itself*)

ultimately have $g: g = \text{monomial } 1 \ t$ **by** *simp*

hence $t \in \text{keys } g$ **by** *simp*

from $\text{assms}(3)$ **have** $g \in \text{ideal } ?G$ **by** (*rule ideal.span-base*)

also from $\text{assms}(1)$ ‘ $?S \subseteq P[X]$ ‘ **have** *ideal-G: ... = ideal* $?S$ **by** (*rule reduced-GB-ideal-Polys*)

finally have $g \in \text{ideal } ?S$.

moreover from $\text{assms}(3)$ **have** $g \in P[X]$ **by** *rule (intro reduced-GB-Polys*

$\text{assms}(1)$ ‘ $?S \subseteq P[X]$ ‘)

ultimately have $g \in P[X] \cap \text{ideal } ?S$ **by** *simp*

with $\text{assms}(4)$ **have** $g \in \text{sum-list ‘ listset (map cone } ps)$

by (*simp only: cone-decomp-def direct-decompD*)

with $\text{assms}(5)$ **obtain** $d \ h \ U$ **where** $*(h, U) \in \text{set } ps$ **and** $d \neq 0$ **and** *monomial* $d \ t \in \text{cone } (h, U)$

using ‘ $t \in \text{keys } g$ ’ **by** (*rule monomial-decomp-sum-list-monomial-in-cone*)

from $\text{this}(3)$ *zero-in-PPs* **have** *punit.monom-mult* $(1 / d) \ 0$ (*monomial* $d \ t \in \text{cone } (h, U)$)

by (*rule cone-closed-monom-mult*)

with ‘ $d \neq 0$ ’ **have** $g \in \text{cone } (h, U)$ **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 $\text{lt-}g': \text{lpp } g = \text{lpp } q + \text{lpp } h$ **unfolding** g' **by** (*rule lp-times*)

hence $\text{adds1: lpp } h \text{ adds } t$ **by** (*simp add: t-def*)

from $\text{assms}(5) *$ **have** *is-monomial* h **and** *punit.lc* $h = 1$ **by** (*rule monomial-decompD*) +

moreover from $\text{this}(1)$ **have** *monomial* $(\text{punit.lc } h) (\text{lpp } h) = h$

by (*rule punit.monomial-eq-itself*)

moreover define s **where** $s = \text{lpp } h$

ultimately have $h: h = \text{monomial } 1 \ s$ by *simp*
 have $\text{punit.lc } q = \text{punit.lc } g$ by (*simp add: g' lc-times punit.lc h = 1*)
 hence $\text{punit.lc } q = 1$ by (*simp only: punit.lc g = 1*)
 note *tip-in-cone*
 also from *assms*(4) * have $\text{cone } (h, U) \subseteq P[X] \cap \text{ideal } ?S$ by (*rule cone-decomp-cone-subset*)
 also have $\dots \subseteq \text{ideal } ?G$ by (*simp add: ideal-G*)
 finally have $h \in \text{ideal } ?G$.
 from *assms*(1) $\langle ?S \subseteq P[X] \rangle$ have *punit.is-Groebner-basis* $?G$ by (*rule re-duced-GB-is-GB-Polys*)
 then obtain g' where $g' \in ?G$ and $g' \neq 0$ and *adds2: lpp g' adds lpp h*
 using $\langle h \in \text{ideal } ?G \rangle \langle h \neq 0 \rangle$ by (*rule punit.GB-adds-lt[simplified]*)
 from *this*(3) *adds1* have *lpp g' adds t* by (*rule adds-trans*)
 with $\langle g' \neq 0 \rangle \langle t \in \text{keys } g \rangle$ have *punit.is-red* $\{g'\}$ g
 by (*rule punit.is-red-addsI[simplified]*) *simp*
 have $g' = g$
 proof (*rule ccontr*)
 assume $g' \neq g$
 with $\langle g' \in ?G \rangle$ have $\{g'\} \subseteq ?G - \{g\}$ by *simp*
 with $\langle \text{punit.is-red } \{g'\} \ g \rangle$ have *red: punit.is-red* $(?G - \{g\}) \ g$ by (*rule punit.is-red-subset*)
 from *assms*(1) $\langle ?S \subseteq P[X] \rangle$ have *punit.is-auto-reduced* $?G$ by (*rule re-duced-GB-is-auto-reduced-Polys*)
 hence $\neg \text{punit.is-red } (?G - \{g\}) \ g$ 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* $(\text{punit.lc } q) \ 0 = q$ by (*rule punit.lt-eq-min-term-monomial[simplified]*)
 hence $g = h$ by (*simp add: punit.lc q = 1 g'*)
 with * have $(g, U) \in \text{set } ps$ by *simp*
 thus *?thesis ..*
 qed

lemma *snd-splitI*:
 assumes *finite* X and $U \subseteq X$ and *finite* S and $0 \notin S$
 obtains V where $V \subseteq U$ and $(\text{monomial } 1 \ t, V) \in \text{set } (\text{snd } (\text{split } t \ U \ S))$
 using *assms*
 proof (*induct t U S arbitrary: thesis rule: split-induct*)
 case (*base1 t U S*)
 from *base1.prem*s(2) *base1.hyps*(3) show *?case ..*
 next
 case (*base2 t U S*)
 from *subset-refl* show *?case* by (*rule base2.prem*s) *simp*
 next
 case (*step t U S V0 x ps0 ps1 qs0 qs1*)
 from *step.hyps*(3) obtain V where 1: $V \subseteq U - \{x\}$ and 2: $(\text{monomial } 1 \ t, V) \in \text{set } (\text{snd } (ps0, qs0))$
 using *step.hyps*(15) by *blast*

```

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

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

```

```

from step.hyps(2) have fin: finite (( $\lambda f. f - \text{Poly-Mapping.single } x \ 1$ ) ‘ S)
  by (rule finite-imageI)
have split (Poly-Mapping.single x 1 + t) U (( $\lambda f. f - \text{Poly-Mapping.single } x$ 
1) ‘ S) =
  ((monomial (1::'a) (Poly-Mapping.single x 1 + t), U), [])
  by (simp only: split.psims[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  $\langle (\text{monomial } 1 \ s, \ V) \in \text{set } ps1 \rangle$  have s: s = Poly-Mapping.single x 1 +
t and V = U
  by (auto dest!: monomial-inj)
show ?thesis
proof (rule step.prems)
  show  $0 \in .[X]$  by (fact zero-in-PPs)
next
  from step.hyps(8, 1) show  $x \in X$  ..
next
  show  $V \subseteq U$  by (simp add:  $\langle V = U \rangle$ )
next
  from step.hyps(3) show  $0 \notin (\lambda s. s - 0)$  ‘ S by simp
next
  show  $s = 0 + t + \text{Poly-Mapping.single } x \ 1$  by (simp add: s add.commute)
next
  show (monomial (1::'a) s, V)  $\in \text{set } (\text{fst } (\text{split } (0 + t) \ V \ ((\lambda s. s - 0)$  ‘ S)))
    using  $\langle (\text{monomial } 1 \ s, \ V) \in \text{set } ps1 \rangle$  by (simp add: step.hyps(14)  $\langle V =$ 
U  $\rangle$ )
  next
  show  $\text{set } (\text{snd } (\text{split } (0 + t) \ V \ ((\lambda s. s - 0)$  ‘ S)))  $\subseteq \text{set } (\text{snd } (ps0 \ @ \ ps1,$ 
qs0  $\ @ \ qs1))$ 
    by (simp add: step.hyps(14)  $\langle V = U \rangle$ )
  qed
next
  case False
  moreover from  $\langle (\text{monomial } 1 \ s, \ V) \in \text{set } ps1 \rangle$  have (monomial 1 s, V)  $\in$ 
set (fst (ps1, qs1))
    by (simp only: fst-conv)
  ultimately obtain t' x' where  $t' \in .[X]$  and  $x' \in X$  and  $V \subseteq U$ 
    and 1:  $0 \notin (\lambda s. s - t')$  ‘ ( $\lambda f. f - \text{Poly-Mapping.single } x \ 1$ ) ‘ S
    and  $s = t' + (\text{Poly-Mapping.single } x \ 1 + t) + \text{Poly-Mapping.single } x' \ 1$ 
    and 2: (monomial (1::'a) s, V)  $\in \text{set } (\text{fst } (\text{split } (t' + (\text{Poly-Mapping.single}$ 
x 1 + t)) V
      (( $\lambda s. s - t'$ ) ‘ ( $\lambda f. f - \text{Poly-Mapping.single } x$ 
1) ‘ S)))
    and 3:  $\text{set } (\text{snd } (\text{split } (t' + (\text{Poly-Mapping.single } x \ 1 + t)) \ V \ ((\lambda s. s - t')$ 
‘ ( $\lambda f. f - \text{monomial } 1 \ x$ ) ‘ S)))  $\subseteq$ 
      set (snd (ps1, qs1))
    using step.hyps(16) by blast
  have eq: ( $\lambda s. s - t'$ ) ‘ ( $\lambda f. f - \text{Poly-Mapping.single } x \ 1$ ) ‘ S =

```



```

      (λs. s - (t' + Poly-Mapping.single x 1)) ' S
    by (simp add: image-image add.commute diff-diff-eq)
  show ?thesis
  proof (rule step.prem)
    from step.hyps(8, 1) have x ∈ X ..
    hence Poly-Mapping.single x 1 ∈ .[X] by (rule PPs-closed-single)
    with ⟨t' ∈ .[X]⟩ show t' + Poly-Mapping.single x 1 ∈ .[X] by (rule
PPs-closed-plus)
  next
    from 1 show 0 ∉ (λs. s - (t' + Poly-Mapping.single x 1)) ' S
    by (simp only: eq not-False-eq-True)
  next
    show s = t' + Poly-Mapping.single x 1 + t + 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
                                                    ((λs. s - (t' + 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 ((λs. 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 ⊆ .[X] and 0 ∉ S
  and g ∈ punit.reduced-GB (monomial (1::'a) ' S)
  obtains t U where U ⊆ 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)::(- × (((- ⇒₀ 'a) × -) list))))
      and md2: monomial-decomp (snd ((split 0 X S)::(- × (((- ⇒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 ∉ ?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)
  qed

```

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** $\langle \text{punit.lc } g = 1 \rangle$ **by** (*metis* *punit.monomial-eq-itself*)
note *assms*(1) *subset-refl* *assms*(2) *zero-in-PPs*
moreover **have** *ideal* ?*G* \div *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] \cap ideal ?G*) (*fst (split 0 X S)*)
using *md2* $\langle \text{is-monomial-set } ?G \rangle$ **by** (*rule splits-wrt-cone-decomp-1*)
hence *cone-decomp* (*P[X] \cap ideal ?S*) (*fst (split 0 X S)*) **by** (*simp only: ideal-G*)
with *assms*(1, 3, 5) **obtain** *U* **where** (*g, U*) \in *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' \in .[X]* **and** *x \in X*
and *U \subseteq X*
and *0 \notin ($\lambda s. s - t'$) ' S* **and** *s: s = t' + 0 + Poly-Mapping.single x 1*
and (*g, U*) \in *set (fst (split (t' + 0) U (($\lambda s. s - t'$) ' S)))*
and *set (snd (split (t' + 0) U (($\lambda s. s - t'$) ' S))) \subseteq (set (snd (split 0 X S))) ::*
((\Rightarrow_0 'a) \times -) set)
unfolding *g* **by** (*rule fst-splitE*)
let ?*S* = ($\lambda s. s - t'$) ' *S*
from *assms*(2) **have** *finite* ?*S* **by** (*rule finite-imageI*)
with *assms*(1) $\langle U \subseteq X \rangle$ **obtain** *V* **where** *V \subseteq U*
and (*monomial (1::'a) (t' + 0), V*) \in *set (snd (split (t' + 0) U ?S))*
using $\langle 0 \notin ?S \rangle$ **by** (*rule snd-splitI*)
note *this*(2)
also **have** $\dots \subseteq$ *set (snd (split 0 X S))* **by** *fact*
finally **have** (*monomial (1::'a) t', V*) \in *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 $\langle V \subseteq U \rangle \langle U \subseteq X \rangle$ **have** *V \subseteq X* **by** (*rule subset-trans*)
show ?*thesis* **by** *rule fact+*
qed

corollary *cor-4-9*:

assumes *finite X* **and** *finite S* **and** *S \subseteq .[X]*
and *g \in punit.reduced-GB (monomial (1::'a)::field) ' S*
shows *poly-deg g \leq Suc (Max (poly-deg 'fst ' (set (snd (split 0 X S))) :: ((\Rightarrow_0 'a) \times -) set)))*
(is - \leq Suc (Max (poly-deg 'fst ' ?S)))
proof (*cases 0 \in S*)
case *True*
hence *1 \in monomial (1::'a) ' S* **by** (*rule rev-image-eqI*) (*simp only: single-one*)
hence *1 \in 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 \subseteq 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 ⊆ X and finite S and S ⊆ .[X] and t ∈ .[X]
  shows standard-decomp (deg-pm t) (snd (split t U S)) :: ((- ⇒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 (λs. s - Poly-Mapping.single x 1) ' S ⊆ .[X]
  proof
    fix u
    assume u ∈ (λs. s - Poly-Mapping.single x 1) ' S
    then obtain s where s ∈ S and u: u = s - Poly-Mapping.single x 1 ..
    from this(1) step.prems(1) have s ∈ .[X] ..
    thus u ∈ .[X] unfolding u by (rule PPs-closed-minus)
  qed
  moreover from - step.prems(2) have Poly-Mapping.single x 1 + t ∈ .[X]
  proof (rule PPs-closed-plus)
    from step.hyps(8, 1) have x ∈ X ..
    thus Poly-Mapping.single x 1 ∈ .[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) ∈ set ((qs0 @ qs1)+)
hence *: (h, U0) ∈ set (qs0+) ∪ set (qs1+) by (simp add: pos-decomp-append)
thus deg-pm t ≤ poly-deg h
proof
  assume (h, U0) ∈ set (qs0+)
  with qs0 show ?thesis by (rule standard-decompD)
next
  assume (h, U0) ∈ set (qs1+)
  with qs1 have Suc (deg-pm t) ≤ poly-deg h by (rule standard-decompD)
  thus ?thesis by simp
qed

fix d
assume d1: deg-pm t ≤ d and d2: d ≤ poly-deg h
from * show ∃ t' U'. (t', U') ∈ set (qs0 @ qs1) ∧ poly-deg t' = d ∧ card U0
≤ card U'
proof
  assume (h, U0) ∈ set (qs0+)
  with qs0 obtain h' U' where (h', U') ∈ set qs0 and poly-deg h' = d and
card U0 ≤ card U'
  using d1 d2 by (rule standard-decompE)
  moreover from this(1) have (h', U') ∈ set (qs0 @ qs1) by simp
  ultimately show ?thesis by blast
next
  assume (h, U0) ∈ set (qs1+)
  hence (h, U0) ∈ set qs1 by (simp add: pos-decomp-def)
  from assms(1) step.hyps(1, 2) have monomial-decomp (snd (split t U S) ::
((- ⇒0 'a) × -) list)
  by (rule monomial-decomp-split)
  hence md: monomial-decomp (qs0 @ qs1) by (simp add: step.hyps(14))
  moreover from ⟨(h, U0) ∈ set qs1⟩ have (h, U0) ∈ 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 ∨ Suc (deg-pm t) ≤ d by auto
  thus ?thesis
proof
  assume deg-pm t = d
  define F where F = (*) (monomial 1 t) ‘ monomial (1::'a) ‘ S
  have F ⊆ P[X]
  proof
    fix f
    assume f ∈ F
    then obtain u where u ∈ S and f: f = monomial 1 (t + u)
    by (auto simp: F-def times-monomial-monomial)
    from this(1) step.premis(1) have u ∈ .[X] ..

```

```

    with step.premis(2) have  $t + u \in \cdot[X]$  by (rule PPs-closed-plus)
    thus  $f \in P[X]$  unfolding  $f$  by (rule Polys-closed-monomial)
  qed
  have ideal  $F = (*)$  (monomial 1  $t$ )  $\cdot$  ideal (monomial 1  $\cdot S$ )
    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!:
  puniit.monom-mult-inj-3)
  ultimately have eq: ideal  $F \div$  monomial 1  $t =$  ideal (monomial 1  $\cdot S$ )
    by (simp only: quot-set-image-times)
  with assms(1) step.hyps(1, 2) step.premis(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)  $t$ ,  $U$ )  $\cap$  nor-
  mal-form  $F \cdot P[X]$ ) ( $qs0 @ qs1$ )
    using md -  $\langle F \subseteq P[X] \rangle$ 
    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  $s$ ,  $U0$ )  $\subseteq$  cone (monomial (1::'a)  $t$ ,  $U$ )  $\cap$  nor-
  mal-form  $F \cdot P[X]$ 
    using  $\langle (h, U0) \in \text{set } (qs0 @ qs1) \rangle$  unfolding  $h$  by (rule cone-decomp-cone-subset)
    with assms(1) step.hyps(1, 2) step.premis(2)  $\langle F \subseteq P[X] \rangle$  eq
    obtain  $U'$  where (monomial (1::'a)  $t$ ,  $U'$ )  $\in \text{set } (\text{snd } (\text{split } t \text{ } U \text{ } S))$  and
  card  $U0 \leq \text{card } U'$ 
    by (rule lem-4-6)
    from this(1) have (monomial 1  $t$ ,  $U'$ )  $\in \text{set } (qs0 @ qs1)$  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  $t$ )  $\leq d$ 
    with  $qs1$   $\langle (h, U0) \in \text{set } (qs1_+) \rangle$  obtain  $h' U'$  where  $(h', U')$   $\in \text{set } qs1$ 
  and poly-deg  $h' = d$ 
      and card  $U0 \leq \text{card } U'$  using  $d2$  by (rule standard-decompE)
      moreover from this(1) have  $(h', U') \in \text{set } (qs0 @ qs1)$  by simp
      ultimately show ?thesis by blast
    qed
  qed
  qed
  qed
  theorem standard-cone-decomp-snd-split:
    fixes  $F$ 

```

```

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

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

  assume  $\bigwedge f. f \in F \Rightarrow \text{homogeneous } f$ 
  moreover note  $\langle \text{punit.is-reduced-GB } G \rangle \ \langle \text{ideal } G = \text{ideal } F \rangle$ 
  moreover assume  $g \in G$ 
  ultimately have  $\text{homogeneous } g$  by  $(\text{rule is-reduced-GB-homogeneous})$ 
  moreover have  $\text{lpp } g \in \text{keys } g$ 
  proof  $(\text{rule punit.lt-in-keys})$ 
    from  $\langle g \in G \rangle \ \langle 0 \notin G \rangle$  show  $g \neq 0$  by blast
  qed

```

ultimately have $\text{deg-lt: deg-pm } (lpp\ g) = \text{poly-deg } g$ by (rule homogeneousD-poly-deg)
 from $\langle g \in G \rangle$ have monomial 1 $(lpp\ g) \in ?S$ unfolding S-def by (intro imageI)
 also have $\dots = \text{punit.reduced-GB } ?S$ unfolding S-def G-def using $\text{assms}(4, 5)$
 by (rule reduced-GB-monomial-lt-reduced-GB-Polys[symmetric])
 finally have monomial 1 $(lpp\ g) \in \text{punit.reduced-GB } ?S$.
 with $\text{assms}(4) \langle \text{finite } S \rangle \langle S \subseteq .[X] \rangle$ 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 $\text{ideal-decomp-aux} :: ((x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a) \text{ set} \Rightarrow ((x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a) \Rightarrow$
 $((x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a::\text{field}) \text{ set} \times (((x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a) \times 'x \text{ set}) \text{ list}$
 where $\text{ideal-decomp-aux } F\ f =$
 $(\text{let } J = \text{ideal } F; L = (J \div f) \cap P[X]; L' = lpp\ ' \text{punit.reduced-GB } L \text{ in}$
 $((*)\ f\ ' \text{normal-form } L\ ' P[X], \text{ map } (\text{apfst } ((*)\ f))\ (\text{snd } (\text{split } 0\ X\ L'))))$

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

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

proof –

define J where $J = \text{ideal } F$
 define L where $L = (J \div f) \cap P[X]$
 define S where $S = (*)\ f\ ' \text{normal-form } L\ ' P[X]$
 define L' where $L' = lpp\ ' \text{punit.reduced-GB } L$

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

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

```

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

show ?thesis2
proof (rule set-eqI)
  fix h
  show h ∈ ideal F ∩ fst (ideal-decomp-aux F f) ⟷ h ∈ {0}
  proof
    assume h ∈ ideal F ∩ fst (ideal-decomp-aux F f)
    hence h ∈ J and h ∈ S by (simp-all add: J-def S-def eq)
    from this(2) obtain q where q ∈ P[X] and h: h = f * normal-form L q by
(auto simp: S-def)
    from fin-X L-sub this(1) have normal-form L q ∈ P[X] by (rule Polys-closed-normal-form)
    moreover from ⟨h ∈ J⟩ have f * normal-form L q ∈ J by (simp add: h)
    ultimately have normal-form L q ∈ L by (simp add: L-def quot-set-iff)
    hence normal-form L q ∈ ideal L by (rule ideal.span-base)
    with normal-form-diff-in-ideal[OF fin-X L-sub] have (q - normal-form L q)
+ normal-form L q ∈ ideal L
    by (rule ideal.span-add)
    hence normal-form L q = 0 using fin-X L-sub by (simp add: normal-form-zero-iff)
    thus h ∈ {0} by (simp add: h)
  next
    assume h ∈ {0}
    moreover have 0 ∈ (*) f ‘ normal-form L ‘ P[X]
    proof (intro image-eqI)
      from fin-X L-sub show 0 = normal-form L 0 by (simp only: nor-
mal-form-zero)
    qed (simp-all add: zero-in-Polys)
    ultimately show h ∈ ideal F ∩ fst (ideal-decomp-aux F f) by (simp add:
ideal.span-zero eq S-def)
  qed
qed

have direct-decomp (ideal (insert f F) ∩ P[X]) [ideal F ∩ P[X], fst (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: standard-decomp 0 (snd (split 0 X L') :: ((- ⇒₀ 'a) × -) list)
  and cone-decomp (normal-form L ‘ P[X]) (snd (split 0 X L'))

```


unfolding L' -def **using** $\text{fin-}X \langle L \subseteq P[X] \rangle$ **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 $\text{fin-}X \langle L \subseteq P[X] \rangle$ **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 $\text{fin-}X$ subset-refl $\langle \text{finite } L' \rangle$ **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 $\text{fin-}X$ subset-refl $\langle \text{finite } L' \rangle$ 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 $\langle f \neq 0 \rangle$ **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** $\bigwedge f. f \in F \implies \text{poly-deg } f$
 $\leq \text{poly-deg } f0$
 obtains T ps **where** valid-decomp X ps **and** standard-decomp (poly-deg f0) ps
and cone-decomp T ps
 and $(\bigwedge f. f \in F \implies \text{homogeneous } f) \implies \text{hom-decomp } ps$
 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.premis)
 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.premis(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.premis)
qed
ultimately obtain T ps where valid-ps: valid-decomp X ps and std-ps: stan-
dard-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.premis)
  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.premis(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.premis) simp

```

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

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 ($\forall (h, U) \in \text{set } ps. h \in P[X] \wedge U \subseteq X$) \wedge
 ($\forall (h, U) \in \text{set } ps. \forall (h', U') \in \text{set } ps. \text{poly-deg } h = \text{poly-deg } h' \longrightarrow$
 $m < \text{card } U \longrightarrow m < \text{card } U' \longrightarrow (h, U) = (h', U')$)

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

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

lemma *exact-decompI-zero*:
 assumes $\bigwedge h U. (h, U) \in \text{set } ps \implies h \in P[X]$ and $\bigwedge h U. (h, U) \in \text{set } ps \implies$

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

lemma *exact-decompD-zero*:
assumes *exact-decomp* 0 *ps* **and** $(h, U) \in \text{set } (ps_+)$ **and** $(h', U') \in \text{set } (ps_+)$
and $\text{poly-deg } h = \text{poly-deg } h'$
shows $(h, U) = (h', U')$
proof –
from *assms*(2) **have** $(h, U) \in \text{set } ps$ **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 $\langle U \neq \{\} \rangle$ **have** $0 < \text{card } U$ **by** (*simp add: card-gt-0-iff*)
from *assms*(3) **have** $(h', U') \in \text{set } ps$ **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 $\langle U' \neq \{\} \rangle$ **have** $0 < \text{card } U'$ **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** $\bigwedge h \ U. (h, U) \in \text{set } ps \implies h \neq 0$
shows *valid-decomp* X *ps*
proof (*rule valid-decompI*)
fix $h \ U$
assume $*(h, U) \in \text{set } ps$
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** $\text{card } X \leq m$

```

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

```

```

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

```

```

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

```

```

lemma a: standard-decomp k ps  $\implies$  standard-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-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-deg 'fst 'set (ps+)) by (rule a-nonempty)
  moreover from fin-X assms have k = Min (poly-deg 'fst 'set (ps+))
  by (rule standard-decomp-nonempty-unique)
  ultimately show ?thesis by simp

```

qed

lemma b:

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

proof –

let $?A = \text{poly-deg } \text{fst } \text{set ps}$
define A where $A = \text{insert } (a \text{ ps}) ?A$
define m where $m = \text{Suc } (\text{Max } A)$
from *finite-set* have *finite* $?A$ by (intro *finite-imageI*)
hence *finite* A by (simp add: *A-def*)
have $a \text{ ps} \leq b \text{ ps } i \wedge (\forall (h', U') \in \text{set ps}. i \leq \text{card } U' \longrightarrow \text{poly-deg } h' < b \text{ ps } i)$
unfolding *b-def*
proof (rule *LeastI*)
have $a \text{ ps} \in A$ by (simp add: *A-def*)
with $\langle \text{finite } A \rangle$ have $a \text{ ps} \leq \text{Max } A$ by (rule *Max-ge*)
hence $a \text{ ps} \leq m$ by (simp add: *m-def*)
moreover {
fix $h \ U$
assume $(h, U) \in \text{set ps}$
hence $\text{poly-deg } (\text{fst } (h, U)) \in ?A$ by (intro *imageI*)
hence $\text{poly-deg } h \in A$ by (simp add: *A-def*)
with $\langle \text{finite } A \rangle$ have $\text{poly-deg } h \leq \text{Max } A$ by (rule *Max-ge*)
hence $\text{poly-deg } h < m$ by (simp add: *m-def*)
}
ultimately show $a \text{ ps} \leq m \wedge (\forall (h, U) \in \text{set ps}. i \leq \text{card } U \longrightarrow \text{poly-deg } h < m)$ by *blast*
qed
thus $a \text{ ps} \leq b \text{ ps } i$ and $(h, U) \in \text{set ps} \implies i \leq \text{card } U \implies \text{poly-deg } h < b \text{ ps } i$
by *blast+*
qed

lemma b-le:

$a \text{ ps} \leq d \implies (\bigwedge h' U'. (h', U') \in \text{set ps} \implies i \leq \text{card } U' \implies \text{poly-deg } h' < d) \implies b \text{ ps } i \leq d$
unfolding *b-def* by (intro *Least-le*) *blast*

lemma b-decreasing:

assumes $i \leq j$
shows $b \text{ ps } j \leq b \text{ ps } i$
proof (rule *b-le*)
fix $h \ U$
assume $(h, U) \in \text{set ps}$
assume $j \leq \text{card } U$
with *assms(1)* have $i \leq \text{card } U$ by (rule *le-trans*)
with $\langle (h, U) \in \text{set ps} \rangle$ show $\text{poly-deg } h < b \text{ ps } i$ by (rule *b*)
qed (fact *b*)

lemma b-Nil:

assumes $ps_+ = []$ and $Suc\ 0 \leq i$
 shows $b\ ps\ i = 0$
 unfolding $b\text{-def}$
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 $a\ ps \leq 0 \wedge (\forall (h, U) \in set\ ps. i \leq card\ U \longrightarrow poly\text{-deg}\ h < 0)$
by *blast*
qed

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

corollary *b-zero-gr*:
 assumes $(h, U) \in set\ ps$
 shows $poly\text{-deg}\ h < b\ ps\ 0$
proof –
 have $poly\text{-deg}\ h \leq Max\ (poly\text{-deg}\ 'fst\ 'set\ ps)$
proof (rule *Max-ge*)
 from *finite-set* **show** $finite\ (poly\text{-deg}\ 'fst\ 'set\ ps)$ **by** (*intro finite-imageI*)
next
 from $assms$ have $poly\text{-deg}\ (fst\ (h, U)) \in poly\text{-deg}\ 'fst\ 'set\ ps$ **by** (*intro imageI*)
 thus $poly\text{-deg}\ h \in poly\text{-deg}\ 'fst\ 'set\ ps$ **by** *simp*
qed
 also have $\dots < Suc\ \dots$ **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-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-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 $\text{Suc } 0 \leq i$
 and $i \leq \text{card } X$ and $\text{b } ps (\text{Suc } i) \leq d$ and $d < \text{b } ps i$
 obtains $h \ U$ where $(h, U) \in \text{set } (ps_+)$ and $\text{poly-deg } h = d$ and $\text{card } U = i$
 proof –
 have $ps_+ \neq \emptyset$
 proof
 assume $ps_+ = \emptyset$
 hence $\text{b } ps i = 0$ using *assms*(3) by (rule *b-Nil*)
 with *assms*(6) show *False* by *simp*
 qed
 have *eq1*: $\text{b } ps (\text{Suc } (\text{card } X)) = \text{a } ps$ using *assms*(2) *le-reft* by (rule *b-card-X*)
 from *assms*(1) have *std*: *standard-decomp* $(\text{b } ps (\text{Suc } (\text{card } X)))$ *ps* unfolding
eq1 by (rule *a*)
 from *assms*(4) have $\text{Suc } i \leq \text{Suc } (\text{card } X)$..
 hence $\text{b } ps (\text{Suc } (\text{card } X)) \leq \text{b } ps (\text{Suc } i)$ by (rule *b-decreasing*)
 hence $\text{a } ps \leq \text{b } ps (\text{Suc } i)$ by (*simp only: eq1*)
 have $\exists h \ U. (h, U) \in \text{set } ps \wedge i \leq \text{card } U \wedge \text{b } ps i \leq \text{Suc } (\text{poly-deg } h)$
 proof (rule *ccontr*)
 assume *: $\nexists h \ U. (h, U) \in \text{set } ps \wedge i \leq \text{card } U \wedge \text{b } ps i \leq \text{Suc } (\text{poly-deg } h)$
 note $\langle \text{a } ps \leq \text{b } ps (\text{Suc } i) \rangle$
 also from *assms*(5, 6) have $\text{b } ps (\text{Suc } i) < \text{b } ps i$ by (rule *le-less-trans*)
 finally have $\text{a } ps < \text{b } ps i$.
 hence $\text{a } ps \leq \text{b } ps i - 1$ by *simp*
 hence $\text{b } ps i \leq \text{b } ps i - 1$
 proof (rule *b-le*)
 fix $h \ U$
 assume $(h, U) \in \text{set } ps$ and $i \leq \text{card } U$
 show $\text{poly-deg } h < \text{b } ps i - 1$
 proof (rule *ccontr*)
 assume $\neg \text{poly-deg } h < \text{b } ps i - 1$
 hence $\text{b } ps i \leq \text{Suc } (\text{poly-deg } h)$ by *simp*
 with * $\langle (h, U) \in \text{set } ps \rangle \langle i \leq \text{card } U \rangle$ show *False* by *auto*
 qed
 qed
 thus *False* using $\langle \text{a } ps < \text{b } ps i \rangle$ by *linarith*
 qed
 then obtain $h \ U$ where $(h, U) \in \text{set } ps$ and $i \leq \text{card } U$ and $\text{b } ps i \leq \text{Suc } (\text{poly-deg } h)$ by *blast*
 from *assms*(3) *this*(2) have $U \neq \{\}$ by *auto*
 with $\langle (h, U) \in \text{set } ps \rangle$ have $(h, U) \in \text{set } (ps_+)$ by (*simp add: pos-decomp-def*)
 note *std this*
 moreover have $\text{b } ps (\text{Suc } (\text{card } X)) \leq d$ unfolding *eq1* using $\langle \text{a } ps \leq \text{b } ps (\text{Suc } i) \rangle$ *assms*(5)
 by (rule *le-trans*)
 moreover have $d \leq \text{poly-deg } h$
 proof –

from $\text{assms}(6) \langle \text{b } ps \ i \leq \text{Suc } (\text{poly-deg } h) \rangle$ **have** $d < \text{Suc } (\text{poly-deg } h)$ **by** (rule *less-le-trans*)
thus *?thesis* **by** *simp*
qed
ultimately obtain $h' \ U'$ **where** $(h', U') \in \text{set } ps$ **and** $d: \text{poly-deg } h' = d$ **and** $\text{card } U \leq \text{card } U'$
by (rule *standard-decompE*)
from $\langle i \leq \text{card } U \rangle$ *this*(3) **have** $i \leq \text{card } U'$ **by** (rule *le-trans*)
with $\text{assms}(3)$ **have** $U' \neq \{\}$ **by** *auto*
with $\langle (h', U') \in \text{set } ps \rangle$ **have** $(h', U') \in \text{set } (ps_+)$ **by** (*simp add: pos-decomp-def*)
moreover note $\langle \text{poly-deg } h' = d \rangle$
moreover have $\text{card } U' = i$
proof (rule *ccontr*)
assume $\text{card } U' \neq i$
with $\langle i \leq \text{card } U' \rangle$ **have** $\text{Suc } i \leq \text{card } U'$ **by** *simp*
with $\langle (h', U') \in \text{set } ps \rangle$ **have** $\text{poly-deg } h' < \text{b } ps \ (\text{Suc } i)$ **by** (rule *b*)
with $\text{assms}(5)$ **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** $\text{Suc } 0 \leq i$
and $i \leq \text{card } X$ **and** $\text{b } ps \ (\text{Suc } i) \leq d$ **and** $d < \text{b } ps \ i$
obtains $h \ U$ **where** $\{(h', U') \in \text{set } (ps_+). \text{poly-deg } h' = d\} = \{(h, U)\}$ **and** $\text{card } U = i$
proof –
from assms **obtain** $h \ U$ **where** $(h, U) \in \text{set } (ps_+)$ **and** $\text{poly-deg } h = d$ **and** $\text{card } U = i$
by (rule *lem-6-1-1*)
hence $\{(h, U)\} \subseteq \{(h', U') \in \text{set } (ps_+). \text{poly-deg } h' = d\}$ (*is - \subseteq ?A*) **by** *simp*
moreover have $?A \subseteq \{(h, U)\}$
proof
fix x
assume $x \in ?A$
then obtain $h' \ U'$ **where** $(h', U') \in \text{set } (ps_+)$ **and** $\text{poly-deg } h' = d$ **and** $x: x = (h', U')$
by *blast*
note $\text{assms}(2) \langle (h, U) \in \text{set } (ps_+) \rangle$ *this*(1)
moreover have $\text{poly-deg } h = \text{poly-deg } h'$ **by** (*simp only: $\langle \text{poly-deg } h = d \rangle$*)
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** $\langle \text{card } U = i \rangle$..
qed

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

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

corollary *lem-6-1-3*:

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

```

    from ⟨Suc i < j⟩ show i ≤ j - 1 by simp
qed
moreover have j - 1 ≤ card X
proof -
  have j ≤ 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 ≤ 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 ⇒0 nat) ⇒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 mono-
mial-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 $\langle x \in U \rangle \langle U \subseteq X \rangle$ show $x \in X$..
 qed
 moreover from $\langle U \subseteq X \rangle$ have $U - \{x\} \subseteq X$ by blast
 ultimately show ?thesis
 using assms(1) $\langle h \neq 0 \rangle$ 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 $(h, U) \in set\ ((shift-list\ (h2, U2)\ x\ ps)_+)$

hence *disj*: $(h, U) = ?p1 \vee ((h, U) = ?p2 \wedge U2 - \{x\} \neq \{\}) \vee (h, U) \in \text{set } (ps_+)$
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** $(h2, U2) \in \text{set } (ps_+)$ **by** (*simp add: pos-decomp-def*)
with *assms*(1) **have** *k-le*: $k \leq \text{poly-deg } h2$ **by** (*rule standard-decompD*)

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

fix d
assume $k \leq d$ **and** $d \leq \text{poly-deg } h$
from *disj* **obtain** $h' \ U'$ **where** $1: (h', U') \in \text{set } (?p1 \# ps)$ **and** $\text{poly-deg } h' = d$
and $\text{card } U \leq \text{card } U'$
proof (*elim disjE*)
assume $(h, U) = ?p1$
hence $h: h = \text{punit.monom-mult } 1 \ ?x \ h2$ **and** $U = U2$ **by** *simp-all*
from $\langle d \leq \text{poly-deg } h \rangle$ **have** $d \leq \text{poly-deg } h2 \vee \text{poly-deg } h = d$
by (*cases h2 = 0*) (*auto simp: h poly-deg-monom-mult deg-pm-single*)
thus *?thesis*
proof
assume $d \leq \text{poly-deg } h2$
with *assms*(1) $\langle (h2, U2) \in \text{set } (ps_+) \rangle \langle k \leq d \rangle$ **obtain** $h' \ U'$
where $(h', U') \in \text{set } ps$ **and** $\text{poly-deg } h' = d$ **and** $\text{card } U2 \leq \text{card } U'$
by (*rule standard-decompE*)
from *this*(1) **have** $(h', U') \in \text{set } (?p1 \# ps)$ **by** *simp*
moreover **note** $\langle \text{poly-deg } h' = d \rangle$
moreover **from** $\langle \text{card } U2 \leq \text{card } U' \rangle$ **have** $\text{card } U \leq \text{card } U'$ **by** (*simp only: $\langle U = U2 \rangle$*)
ultimately **show** *?thesis* ..
next
have $(h, U) \in \text{set } (?p1 \# ps)$ **by** (*simp add: $\langle (h, U) = ?p1 \rangle$*)
moreover **assume** $\text{poly-deg } h = d$
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  $\langle d \leq \text{poly-deg } h \rangle$  this(1) have  $d \leq \text{poly-deg } h2$  by simp
  with assms(1)  $\langle (h2, U2) \in \text{set } (ps_+) \rangle \langle k \leq d \rangle$  obtain  $h' U'$ 
    where  $(h', U') \in \text{set } ps$  and  $\text{poly-deg } h' = d$  and  $\text{card } U2 \leq \text{card } U'$ 
    by (rule standard-decompE)
  from this(1) have  $(h', U') \in \text{set } (?p1 \# ps)$  by simp
  moreover note  $\langle \text{poly-deg } h' = d \rangle$ 
  moreover from  $\langle \text{card } U2 \leq \text{card } U' \rangle$  have  $\text{card } U \leq \text{card } U'$  unfolding  $U$ 
    by (rule le-trans) (metis Diff-empty card-Diff1-le card.infinite finite-Diff-insert
order-refl)
  ultimately show ?thesis ..
next
  assume  $(h, U) \in \text{set } (ps_+)$ 
  from assms(1) this  $\langle k \leq d \rangle \langle d \leq \text{poly-deg } h \rangle$  obtain  $h' U'$ 
    where  $(h', U') \in \text{set } ps$  and  $\text{poly-deg } h' = d$  and  $\text{card } U \leq \text{card } U'$ 
    by (rule standard-decompE)
  from this(1) have  $(h', U') \in \text{set } (?p1 \# ps)$  by simp
  thus ?thesis using  $\langle \text{poly-deg } h' = d \rangle \langle \text{card } U \leq \text{card } U' \rangle$  ..
qed
show  $\exists h' U'. (h', U') \in \text{set } (\text{shift-list } (h2, U2) \times ps) \wedge \text{poly-deg } h' = d \wedge \text{card } U \leq \text{card } U'$ 
proof (cases  $(h', U') = (h2, U2)$ )
  case True
    hence  $h' = h2$  and  $U' = U2$  by simp-all
    from assms(2, 6) have  $(h1, U1) \in \text{set } (\text{shift-list } (h2, U2) \times ps)$  by (simp add: shift-list.simps)
    moreover from  $\langle \text{poly-deg } h' = d \rangle$  have  $\text{poly-deg } h1 = d$  by (simp only:  $\langle h' = h2 \rangle$  assms(4))
    moreover from  $\langle \text{card } U \leq \text{card } U' \rangle$  assms(5) have  $\text{card } U \leq \text{card } U1$  by (simp add:  $\langle U' = U2 \rangle$ )
    ultimately show ?thesis by blast
  next
    case False
      with 1 have  $(h', U') \in \text{set } (\text{shift-list } (h2, U2) \times ps)$  by (auto simp: shift-list.simps)
      thus ?thesis using  $\langle \text{poly-deg } h' = d \rangle \langle \text{card } U \leq \text{card } U' \rangle$  by blast
qed
qed

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

```

```

from assms(3) obtain ps1 ps2 where ps: ps = ps1 @ (h, U) # ps2 and *: (h,
U) ∉ set ps1
  by (meson split-list-first)
have count-list ps2 (h, U) = 0
proof (rule ccontr)
  from assms(1, 3) have h ≠ 0 by (rule valid-decompD)
  assume count-list ps2 (h, U) ≠ 0
  hence 1 < count-list ps (h, U) by (simp add: ps)
also have ... ≤ count-list (map cone ps) (cone (h, U)) by (fact count-list-map-ge)
finally have 1 < count-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 ∈ set (map cone ps)
  thus 0 ∈ 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 ≠ 0⟩ show False ..
qed
hence *: (h, U) ∉ 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: re-
move1-append ps)
  finally show ?thesis by (simp only: shift-list.simps)
qed

```

10.9 Functions *shift* and *exact*

context

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

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

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

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

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

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

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

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

declare $\text{shift1-inv.simps[simp del]}$

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

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

lemma *shift1-inv-some-snd*:
assumes *shift1-inv* (*qs*, *B*) **and** $1 < \text{card } B$ **and** $(h, U) = (\text{SOME } b. b \in B \wedge \text{card } (snd\ b) = \text{Suc } m)$
shows $(h, U) \in B$ **and** $(h, U) \in \text{set } qs$ **and** $\text{poly-deg } h = d$ **and** $\text{card } U = \text{Suc } m$
proof –
define *A* **where** $A = \{q \in B. \text{card } (snd\ q) = \text{Suc } m\}$
define *Y* **where** $Y = \{q \in \text{set } qs. \text{poly-deg } (fst\ q) = d \wedge \text{Suc } m < \text{card } (snd\ q)\}$
from *assms*(1) **have** *B*: $B = \{q \in \text{set } qs. \text{poly-deg } (fst\ q) = d \wedge m < \text{card } (snd\ q)\}$
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 $Y \subseteq \text{set } qs$ **unfolding** *Y*-def **by** *blast*
qed (fact finite-set)
moreover have $A \cap Y = \{\}$ **by** (auto simp: *A*-def *Y*-def)
ultimately have $\text{card } (A \cup Y) = \text{card } A + \text{card } Y$ **by** (rule card-Un-disjoint)
with *assms*(2) **have** $1 < \text{card } A + \text{card } Y$ **by** (simp only: *B'*)
thm *card-le-Suc0-iff-eq*[OF $\langle \text{finite } Y \rangle$]
moreover have $\text{card } Y \leq 1$ **unfolding** *One-nat-def* *card-le-Suc0-iff-eq*[OF $\langle \text{finite } Y \rangle$]
proof (intro ballI)
fix *q1 q2* :: $((x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a) \times 'x \text{ set}$
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 $(h1, U1) \in \text{set } qs$ **and** $\text{poly-deg } h1 = d$ **and** $\text{Suc } m < \text{card } U1$ **by**
(simp-all add: *q1* *Y*-def)
assume $q2 \in Y$
hence $(h2, U2) \in \text{set } qs$ **and** $\text{poly-deg } h2 = d$ **and** $\text{Suc } m < \text{card } U2$ **by**
(simp-all add: *q2* *Y*-def)
from *this*(2) **have** $\text{poly-deg } h1 = \text{poly-deg } h2$ **by** (simp only: $\langle \text{poly-deg } h1 = d \rangle$)
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 $0 < \text{card } A$ **by** *simp*
hence $A \neq \{\}$ **by** *auto*
then obtain a **where** $a \in A$ **by** *blast*
have $(h, U) \in B \wedge \text{card } (\text{snd } (h, U)) = \text{Suc } m$ **unfolding** *assms(3)*
proof (*rule someI*)
from $\langle a \in A \rangle$ **show** $a \in B \wedge \text{card } (\text{snd } a) = \text{Suc } m$ **by** (*simp add: A-def*)
qed
thus $(h, U) \in B$ **and** $\text{card } U = \text{Suc } m$ **by** *simp-all*
from *this(1)* **show** $(h, U) \in \text{set } qs$ **and** $\text{poly-deg } h = d$ **by** (*simp-all add: B*)
qed

lemma *shift1-inv-preserved*:
assumes *shift1-inv* (*qs*, *B*) **and** $1 < \text{card } B$ **and** $(h, U) = (\text{SOME } b. b \in B \wedge \text{card } (\text{snd } b) = \text{Suc } m)$
and $x = (\text{SOME } y. y \in U)$
shows *shift1-inv* (*shift-list* ($(h, U) \ x \ qs$, $B - \{(h, U)\}$)
proof –
let *?p1* = (*punit.monom-mult* 1 (*Poly-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** $(h, U) \in \text{set } qs$ **and** *deg-h*: $\text{poly-deg } h = d$
and *card-U*: $\text{card } U = \text{Suc } m$ **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*: $\text{card } (U - \{x\}) = m$
by (*metis card-Diff-singleton card.infinite diff-Suc-1 nat.simps(3)*)
from *assms(1)* **have** *B*: $B = \{q \in \text{set } qs. \text{poly-deg } (\text{fst } q) = d \wedge m < \text{card } (\text{snd } q)\}$
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** $\langle (h, U) \in \text{set } qs \rangle$ **by** (*rule valid-decompD*)
show *?thesis*
proof (*intro shift1-invI shift2-invI*)
show $?B = \{q \in \text{set } (\text{shift-list } (h, U) \ x \ qs). \text{poly-deg } (\text{fst } q) = d \wedge m < \text{card } (\text{snd } q)\}$ (*is* - = *?C*)
proof (*rule Set.set-eqI*)
fix b
show $b \in ?B \longleftrightarrow b \in ?C$
proof
assume $b \in ?C$
hence $b \in \text{insert } ?p1 \ (\text{insert } ?p2 \ (\text{set } ?qs))$ **and** *b1*: $\text{poly-deg } (\text{fst } b) = d$
and *b2*: $m < \text{card } (\text{snd } b)$ **by** (*simp-all add: shift-list.simps*)
from *this(1)* **show** $b \in ?B$
proof (*elim insertE*)
assume $b = ?p1$

```

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

```

with $\langle x \in U \rangle$ have $x \in X$..
 hence *Poly-Mapping.single* $x \ 1 \in \cdot[X]$ by (rule *PPs-closed-single*)
 moreover from *exct* $\langle (h, U) \in \text{set } qs \rangle$ 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* $\langle (h, U) \in \text{set } qs \rangle$ have $\dots \in P[X]$ by (rule *exact-decompD*)
 finally show *?thesis* .
 next
 assume $(h', U') \in \text{set } ?qs$
 hence $(h', U') \in \text{set } qs$ 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* $\langle (h, U) \in \text{set } qs \rangle$ have $\dots \subseteq X$ by (rule *exact-decompD*)
 finally show *?thesis* .
 next
 assume $(h', U') = ?p2$
 hence $U' = U - \{x\}$ by *simp*
 also have $\dots \subseteq U$ by *blast*
 also from *exct* $\langle (h, U) \in \text{set } qs \rangle$ have $\dots \subseteq X$ by (rule *exact-decompD*)
 finally show *?thesis* .
 next
 assume $(h', U') \in \text{set } ?qs$
 hence $(h', U') \in \text{set } qs$ by *simp*
 with *exct* show *?thesis* by (rule *exact-decompD*)
 qed
 next
 fix $h1 \ h2 \ U1 \ U2$
 assume $(h1, U1) \in \text{set } (\text{shift-list } (h, U) \ x \ qs)$ and $\text{Suc } m < \text{card } U1$
 hence $(h1, U1) \in \text{set } qs$ using *card-U card-Ux* by (auto simp: *shift-list.simps*)
 assume $(h2, U2) \in \text{set } (\text{shift-list } (h, U) \ x \ qs)$ and $\text{Suc } m < \text{card } U2$
 hence $(h2, U2) \in \text{set } qs$ using *card-U card-Ux* by (auto simp: *shift-list.simps*)
 assume *poly-deg* $h1 = \text{poly-deg } h2$
 from *exct* show $(h1, U1) = (h2, U2)$ by (rule *exact-decompD*) *fact+*
 qed
 next
 fix $d0$
 assume $d0 < d$
 have *finite* $\{q \in \text{set } qs. \text{poly-deg } (\text{fst } q) = d0 \wedge m < \text{card } (\text{snd } q)\}$ (is *finite* ?A)
 by *auto*
 moreover have $\{q \in \text{set } (\text{shift-list } (h, U) \ x \ qs). \text{poly-deg } (\text{fst } q) = d0 \wedge m <$

```

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 ⇒0 nat) ⇒0 'a) × 'x set) list × (((('x ⇒0
nat) ⇒0 'a) × 'x set) set) ⇒
  (((('x ⇒0 nat) ⇒0 'a) × 'x set) list ×
  (((('x ⇒0 nat) ⇒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** $B0::((- \Rightarrow_0\ 'a) \times -)$ **set**
using - *that(2)*
proof (*rule less*)
from *that(1)* **show** $((qs0, B0), args) \in measure\ (card \circ snd)$ **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 $hU: (h, U) = (SOME\ b. b \in B \wedge card\ (snd\ b) = Suc\ m)$
define x **where** $x = (SOME\ y. y \in U)$
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** $inv\ \langle 1 < card\ B \rangle\ hU$ **have** $(h, U) \in B$ **by** (*rule*
shift1-inv-some-snd)
ultimately **show** $card\ (B - \{(h, U)\}) < card\ B$ **by** (*rule card-Diff1-less*)
next
from $inv\ \langle 1 < card\ B \rangle\ hU\ x-def$ **show** $shift1-inv\ (shift-list\ (h, U)\ x\ qs, (B$
 $- \{(h, U)\}))$
by (*rule shift1-inv-preserved*)
qed
thus $shift1-dom\ (shift-list\ (SOME\ b. b \in B \wedge card\ (snd\ b) = Suc\ m)\ (SOME$
 $y. y \in U)\ qs,$
 $B - \{SOME\ b. b \in B \wedge card\ (snd\ b) = Suc\ m\})$ **by** (*simp add:*
 $hU\ x-def$)
qed
qed
qed

lemma *shift1-induct* [*consumes 1, case-names base step*]:

assumes *shift1-inv args*
assumes $\bigwedge qs\ B. shift1-inv\ (qs, B) \implies card\ B \leq 1 \implies P\ (qs, B)\ (qs, B)$
assumes $\bigwedge qs\ B\ h\ U\ x. shift1-inv\ (qs, B) \implies 1 < card\ B \implies$
 $(h, U) = (SOME\ b. b \in B \wedge card\ (snd\ b) = Suc\ m) \implies x = (SOME\ y.$
 $y \in U) \implies$
 $finite\ U \implies x \in U \implies card\ (U - \{x\}) = m \implies$
 $P\ (shift-list\ (h, U)\ x\ qs, B - \{(h, U)\})\ (shift1\ (shift-list\ (h, U)\ x\ qs, B$
 $- \{(h, U)\})) \implies$
 $P\ (qs, B)\ (shift1\ (shift-list\ (h, U)\ x\ qs, B - \{(h, U)\}))$
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.psims[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  $B = \{q \in set\ (fst\ (qs, B)).\ poly-deg\ (fst\ q) = d0 \wedge m <$ 
 $card\ (snd\ q)\}$ 
    unfolding  $fst-conv\ \langle d0 = d \rangle$  by  $(rule\ shift1-invD)$ 
    with  $base(2)$  show  $?thesis$  by  $simp$ 
  qed
qed

```

lemma $shift1-2$:

```

   $shift1-inv\ args \implies$ 
   $card\ \{q \in set\ (fst\ (shift1\ args)).\ m < card\ (snd\ q)\} \leq card\ \{q \in set\ (fst\ args).$ 
 $m < card\ (snd\ q)\}$ 
  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 =  $\{q \in set\ qs.\ m < card\ (snd\ q)\}$ 
    from  $step(1-3)$  have  $card-U$ :  $card\ U = Suc\ m$  and  $(h,\ U) \in set\ qs$  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  $h \neq 0$  using  $\langle (h,\ U) \in set\ qs \rangle$  by  $(rule\ valid-decompD)$ 
    have  $fin1$ :  $finite\ ?A$  by  $auto$ 
    hence  $fin2$ :  $finite\ (insert\ ?p1\ ?A)$  by  $simp$ 
    from  $\langle (h,\ U) \in set\ qs \rangle$  have  $hU-in$ :  $(h,\ U) \in insert\ ?p1\ ?A$  by  $(simp\ add:$ 
 $card-U)$ 
    have  $?p1 \neq (h,\ U)$ 
    proof
      assume  $?p1 = (h,\ U)$ 
      hence  $lpp\ (punit.monom-mult\ 1\ ?x\ h) = lpp\ h$  by  $simp$ 
      with  $\langle h \neq 0 \rangle$  show  $False$  by  $(simp\ add: punit.lt-monom-mult\ monomial-0-iff)$ 
    qed
    let ?qs =  $shift-list\ (h,\ U)\ x\ qs$ 
    have  $\{q \in set\ (fst\ (?qs, B - \{(h,\ U)\})).\ m < card\ (snd\ q)\} = (insert\ ?p1\ ?A)$ 
 $-\ \{(h,\ U)\}$ 
    using  $step(7)\ card-U\ \langle ?p1 \neq (h,\ U) \rangle$  by  $(fastforce\ simp: shift-list.simps)$ 
    also from  $fin2\ hU-in$  have  $card\ \dots = card\ (insert\ ?p1\ ?A) - 1$  by  $(simp\ add:$ 
 $card-Diff-singleton-if)$ 
    also from  $fin1$  have  $\dots \leq Suc\ (card\ ?A) - 1$  by  $(simp\ add: card-insert-if)$ 
    also have  $\dots = card\ \{q \in set\ (fst\ (qs, B)).\ m < card\ (snd\ q)\}$  by  $simp$ 
    finally have  $card\ \{q \in set\ (fst\ (?qs, B - \{(h,\ U)\})).\ m < card\ (snd\ q)\} \leq$ 
 $card\ \{q \in set\ (fst\ (qs, B)).\ m < card\ (snd\ q)\}$  .
    with  $step(8)$  show ?case by  $(rule\ le-trans)$ 
  qed

```

lemma $shift1-3$: $shift1-inv\ args \implies cone-decomp\ T\ (fst\ args) \implies cone-decomp\ T$

```

(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.prem1 have cone-decomp T qs by (simp only: fst-conv)
  moreover from step.hyps(1-3) have (h, U) ∈ 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 ⇒
    Max (poly-deg ‘fst ‘set (fst args)) ≤ 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 ∈ 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 set qs ≠ {} 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))) ≤ 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 ≠ {} show poly-deg ‘fst ‘set qs ≠ {} by simp
next
  fix a
  assume a ∈ poly-deg ‘fst ‘set qs
  then obtain q where q ∈ set qs and a: a = poly-deg (fst q) by blast
  show a ≤ Max (poly-deg ‘fst ‘set ?qs)
  proof (cases q = (h, U))
  case True
  hence a ≤ poly-deg (fst ?p1) by (cases h = 0) (simp-all add: a poly-deg-monom-mult)
  also from fin have ... ≤ Max (poly-deg ‘fst ‘set ?qs)
  proof (rule Max-ge)
  have ?p1 ∈ 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.prem1 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) ∈ set qs by (rule shift1-inv-some-snd)
  with step.premis 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 ∈ set qs. poly-deg (fst q) = d ∧ m
    < card (snd q)})))
proof -
  define args where args = (qs, {q ∈ set qs. poly-deg (fst q) = d ∧ 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 ≤ d by simp
  with inv1 show card {q ∈ set (fst (shift1 args)). poly-deg (fst q) = d0 ∧ m <
    card (snd q)} ≤ 1
    by (rule shift1-1)
qed
qed

function shift2 :: nat ⇒ nat ⇒ (((('x ⇒0 nat) ⇒0 'a) × 'x set) list ⇒
  (((('x ⇒0 nat) ⇒0 'a::{'comm-ring-1', 'ring-no-zero-divisors'}) × 'x
  set) list) where
  shift2 c d qs =
    (if c ≤ d then qs
     else shift2 c (Suc d) (fst (shift1 (qs, {q ∈ set qs. poly-deg (fst q) = d ∧ m <
       card (snd q)}))))
  by auto
termination proof
  show wf (measure (λ(c, d, -). c - d)) by (fact wf-measure)
qed simp

lemma shift2-1: shift2-inv d qs ⟹ 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  $c \leq d$ 
  show shift2-inv c qs
  proof (rule shift2-invI)
    from IH(2) show valid-decomp X qs and standard-decomp k qs and ex-
act-decomp (Suc m) qs
    by (rule shift2-invD)+
  next
  fix d0
  assume  $d0 < c$ 
  hence  $d0 < d$  using  $\langle c \leq d \rangle$  by (rule less-le-trans)
  with IH(2) show  $\text{card } \{q \in \text{set } qs. \text{poly-deg } (\text{fst } q) = d0 \wedge m < \text{card } (\text{snd } q)\} \leq 1$ 
  by (rule shift2-invD)
  qed
next
  assume  $\neg c \leq d$ 
  thus shift2-inv c (shift2 c (Suc d) (fst (shift1 (qs, {q ∈ set qs. poly-deg (fst q) = d ∧ m < card (snd q)}))))
  proof (rule IH)
    from IH(2) show shift2-inv (Suc d) (fst (shift1 (qs, {q ∈ set qs. poly-deg (fst q) = d ∧ m < card (snd q)}))))
    by (rule shift2-inv-preserved)
  qed
qed
qed

```

lemma *shift2-2*:

```

shift2-inv d qs  $\implies$ 
 $\text{card } \{q \in \text{set } (\text{shift2 } c \text{ d } qs). m < \text{card } (\text{snd } q)\} \leq \text{card } \{q \in \text{set } qs. m < \text{card } (\text{snd } q)\}$ 
proof (induct c d qs rule: shift2.induct)
  case IH: (1 c d qs)
  let ?A =  $\{q \in \text{set } (\text{shift2 } c \text{ (Suc d) (fst (shift1 (qs, \{q \in \text{set } qs. \text{poly-deg } (\text{fst } q) = d \wedge m < \text{card } (\text{snd } q)\}))))). m < \text{card } (\text{snd } q)\}$ 
  show ?case
  proof (subst shift2.simps, simp del: shift2.simps, intro impI)
    assume  $\neg c \leq d$ 
    hence  $\text{card } ?A \leq \text{card } \{q \in \text{set } (\text{fst } (\text{shift1 } (qs, \{q \in \text{set } qs. \text{poly-deg } (\text{fst } q) = d \wedge m < \text{card } (\text{snd } q)\}))). m < \text{card } (\text{snd } q)\}$ 
    proof (rule IH)
      show 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)
    qed
    also have  $\dots \leq \text{card } \{q \in \text{set } (\text{fst } (qs, \{q \in \text{set } qs. \text{poly-deg } (\text{fst } q) = d \wedge m < \text{card } (\text{snd } q)\})). m < \text{card } (\text{snd } q)\}$ 

```

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

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

lemma *shift2-4*:
 $\text{shift2-inv } d \text{ } qs \implies \text{Max } (\text{poly-deg } ' \text{fst } ' \text{set } qs) \leq \text{Max } (\text{poly-deg } ' \text{fst } ' \text{set } (\text{shift2 } c \text{ } d \text{ } qs))$
proof (*induct c d qs rule: shift2.induct*)
 case *IH*: ($1 \text{ } c \text{ } d \text{ } qs$)
 let *?args* = $(qs, \{q \in \text{set } qs. \text{poly-deg } (\text{fst } q) = d \wedge m < \text{card } (\text{snd } q)\})$
 show ?case
proof (*subst shift2.simps, simp del: shift2.simps, intro impI*)
 assume $\neg c \leq d$
 have $\text{Max } (\text{poly-deg } ' \text{fst } ' \text{set } (\text{fst } ?args)) \leq \text{Max } (\text{poly-deg } ' \text{fst } ' \text{set } (\text{fst } (\text{shift1 } ?args)))$
 using *refl IH(2)* by (*intro shift1-4 shift1-invI*)
 also from $\langle \neg c \leq d \rangle$ have $\dots \leq \text{Max } (\text{poly-deg } ' \text{fst } ' \text{set } (\text{shift2 } c (\text{Suc } d) (\text{fst } (\text{shift1 } ?args))))$
proof (*rule IH*)
 from *IH(2)* show $\text{shift2-inv } (\text{Suc } d) (\text{fst } (\text{shift1 } ?args))$
 by (*rule shift2-inv-preserved*)
 qed
 finally show $\text{Max } (\text{poly-deg } ' \text{fst } ' \text{set } qs) \leq \text{Max } (\text{poly-deg } ' \text{fst } ' \text{set } (\text{shift2 } c \text{ } d \text{ } qs))$

(Suc d) (fst (shift1 ?args)))
 by (simp only: fst-conv)
 qed
 qed

lemma shift2-5:

shift2-inv d qs \implies 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-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-inv (Suc d) (fst (shift1 ?args))
 by (rule shift2-inv-preserved)
 qed
 also from refl IH(2) have ... \longleftrightarrow fst ?args = [] **by** (intro shift1-5 shift1-invI)
 finally show shift2 c (Suc d) (fst (shift1 ?args)) = [] \longleftrightarrow qs = [] **by** (simp
 only: fst-conv)
 qed
 qed

lemma shift2-6:

shift2-inv d qs \implies monomial-decomp qs \implies monomial-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-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-inv (Suc d) (fst (shift1 ?args)) **by** (rule shift2-inv-preserved)
 next
 from refl IH(2) have shift1-inv d ?args **by** (rule shift1-invI)
 moreover from IH(3) have monomial-decomp (fst ?args) **by** simp
 ultimately show monomial-decomp (fst (shift1 ?args)) **by** (rule shift1-6)
 qed
 qed

lemma shift2-7:

shift2-inv d qs \implies hom-decomp qs \implies hom-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-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-inv (Suc d) (fst (shift1 ?args)) **by** (rule shift2-inv-preserved)
 next
 from refl IH(2) have shift1-inv d ?args **by** (rule shift1-invI)

moreover from $IH(3)$ have $\text{hom-decomp } (fst \text{ ?args})$ by simp
 ultimately show $\text{hom-decomp } (fst (\text{shift1 ?args}))$ by (rule shift1-7)
 qed
 qed

definition $\text{shift} :: (((x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a) \times 'x \text{ set}) \text{ list} \Rightarrow$
 $((x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a :: \{\text{comm-ring-1, ring-no-zero-divisors}\}) \times$
 $'x \text{ set}) \text{ list}$
 where $\text{shift } qs = \text{shift2 } (k + \text{card } \{q \in \text{set } qs. m < \text{card } (\text{snd } q)\}) \ k \ qs$

lemma shift2-inv-init :
 assumes $\text{valid-decomp } X \ qs$ and $\text{standard-decomp } k \ qs$ and $\text{exact-decomp } (\text{Suc } m) \ qs$
 shows $\text{shift2-inv } k \ qs$
 using assms
proof (rule shift2-invI)
 fix $d0$
 assume $d0 < k$
 have $\{q \in \text{set } qs. \text{poly-deg } (fst \ q) = d0 \wedge m < \text{card } (\text{snd } q)\} = \{\}$
proof –
 {
 fix q
 assume $q \in \text{set } qs$
 obtain $h \ U$ where $q = (h, \ U)$ using prod.exhaust by blast
 assume $\text{poly-deg } (fst \ q) = d0$ and $m < \text{card } (\text{snd } q)$
 hence $\text{poly-deg } h < k$ and $m < \text{card } U$ using $\langle d0 < k \rangle$ by (simp-all add: q)
 from $\text{this}(2)$ have $U \neq \{\}$ by auto
 with $\langle q \in \text{set } qs \rangle$ have $(h, \ U) \in \text{set } (qs_+)$ by (simp add: $q \text{ pos-decomp-def}$)
 with $\text{assms}(2)$ have $k \leq \text{poly-deg } h$ by (rule standard-decompD)
 with $\langle \text{poly-deg } h < k \rangle$ have False by simp
 }
 thus $?thesis$ by blast
 qed
 thus $\text{card } \{q \in \text{set } qs. \text{poly-deg } (fst \ q) = d0 \wedge m < \text{card } (\text{snd } q)\} \leq 1$ by (simp
 only: card.empty)
 qed

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

```

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 $\{q \in A. \text{poly-deg } (\text{fst } q) = d0\} \neq \{\}$ by blast
 moreover from - ⟨finite A⟩ have finite $\{q \in A. \text{poly-deg } (\text{fst } q) = d0\}$
 by (rule finite-subset) blast
 ultimately show $1 \leq \text{card } \{q \in A. \text{poly-deg } (\text{fst } q) = d0\}$
 by (simp add: card-gt-0-iff Suc-le-eq)
 qed
 also have $(\sum d0=k.. \text{poly-deg } h2. 1) = \text{Suc } (\text{poly-deg } h2) - k$ by auto
 finally show ?thesis .
 qed
 also from ⟨finite A⟩ - have $\dots \leq \text{card } A$ by (rule card-mono) (auto simp:
 B-def)
 also have $\dots \leq c$ by fact
 finally have $\text{poly-deg } h2 < k + c$ by simp
 with inv2 have $\text{card } ?C \leq 1$ by (rule shift2-invD)
 have finite ?C by auto
 moreover note $\langle \text{card } ?C \leq 1 \rangle$
 moreover from 1 3 4 have $(h1, U1) \in ?C$ by simp
 moreover from 2 5 have $(h2, U2) \in ?C$ 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 $\text{Max} (\text{poly-deg } 'fst \text{ ' set } qs) \leq \text{Max} (\text{poly-deg } 'fst \text{ ' set } (\text{shift } qs))$

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 $\text{shift } qs = [] \longleftrightarrow qs = []$

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

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.prem*s 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.prem1(1, 2, 3) by (rule shift)+
  next
    from Suc.prem1 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.prem1(1, 2, 3) by (rule shift)+
  next
    from Suc.prem1 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.prem1(1, 2, 3)* **by** (*rule shift*)
next
 from *Suc.prem1* **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.prem1* **have** *Max (poly-deg 'fst 'set qs) ≤ Max (poly-deg 'fst 'set ?qs)*
by (*rule Max-shift-ge*)
 also **have** $\dots \leq \text{Max (poly-deg 'fst 'set (exact-aux k m ?qs))}$
proof (*rule Suc*)
 from *Suc.prem1* **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 = []  $\longleftrightarrow$  ?qs = []
  proof (rule Suc)
    from Suc.prem1 show valid-decomp X ?qs and standard-decomp k ?qs and
exact-decomp m ?qs
    by (rule shift)+
  qed
  also from Suc.prem2 have ...  $\longleftrightarrow$  qs = [] by (rule shift-Nil-iff)
  finally show ?case by simp
qed

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

lemma exact:
  assumes valid-decomp X qs and standard-decomp k qs
  shows valid-decomp X (exact k qs) (is ?thesis1)
    and standard-decomp k (exact k qs) (is ?thesis2)
    and exact-decomp 0 (exact k qs) (is ?thesis3)
  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 $\bigwedge g. (\bigwedge f. f \in F \implies \text{homogeneous } f) \implies g \in \text{punit.reduced-GB } F \implies \text{poly-deg } g \leq \text{b } qs$ 0
proof –
 let ? $G = \text{punit.reduced-GB } F$
 let ? $S = \text{lpp}$ ‘? G ’

```

let ?N = normal-form F ‘ P[X]
define qs::((-  $\Rightarrow_0$  'a)  $\times$  -) 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  $\leq$  b ?qs 0 if  $\bigwedge f. f \in F \implies \text{homogeneous } f$  and  $g \in$ 
?G for g
proof (cases qs = [])
case True
  from one-in-Polys have normal-form F 1  $\in$  ?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  $\leq$  Suc (Max (poly-deg ‘ fst ‘ set qs))
  unfolding qs-def by (rule standard-cone-decomp-snd-split)
  also from valid std False have ...  $\leq$  b ?qs 0 by (rule b-zero-exact)
  finally show ?thesis .
qed
ultimately show ?thesis ..
qed
end
end
end
end
end

```

11 Dubé's Degree-Bound for Homogeneous Gröbner Bases

```

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

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

function Dube-aux ::  $\text{nat} \Rightarrow \text{nat}$  where
  Dube-aux  $j = (\text{if } j + 2 < n \text{ then}$ 
     $2 + ((\text{Dube-}i\text{aux } (j + 1)) \text{ choose } 2) + (\sum_{i=j+3..n-1}. (\text{Dube-}i\text{aux } i) \text{ choose } (\text{Suc } (i - j)))$ 
     $\text{else if } j + 2 = n \text{ then } d^2 + 2 * d \text{ else } 2 * d)$ 
  by pat-completeness auto
termination proof
  show wf (measure ((-)  $n$ )) by (fact wf-measure)
qed auto

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

lemma Dube-aux-ge- $d$ :  $d \leq \text{Dube-}i\text{aux } j$ 
proof (induct  $j$  rule: Dube-aux.induct)
  case step: (1  $j$ )
  have  $j + 2 < n \vee j + 2 = n \vee n < j + 2$  by auto
  show ?case
  proof (rule linorder-cases)
    assume *:  $j + 2 < n$ 
    hence 1:  $d \leq \text{Dube-}i\text{aux } (j + 1)$ 
    by (rule step.hyps)+
    show ?thesis
    proof (cases  $d \leq 2$ )
      case True
      also from * have  $2 \leq \text{Dube-}i\text{aux } j$  by simp
      finally show ?thesis .
    next
      case False
      hence 2:  $2 < d$  by simp
      hence 2:  $2 < \text{Dube-}i\text{aux } (j + 1)$  using 1 by (rule less-le-trans)
      with - have  $\text{Dube-}i\text{aux } (j + 1) \leq \text{Dube-}i\text{aux } (j + 1) \text{ choose } 2$  by (rule upper-le-binomial) simp
      also from * have  $\dots \leq \text{Dube-}i\text{aux } j$  by simp
      finally have  $\text{Dube-}i\text{aux } (j + 1) \leq \text{Dube-}i\text{aux } j$  .
      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 *rat-of-nat Dube* $\leq 2 * ((\text{rat-of-nat } d)^2 / 2 + (\text{rat-of-nat } d)) \wedge (2 \wedge (n - 2))$
 oops

end

11.1 Hilbert Function and Hilbert Polynomial

context *pm-powerprod*
begin

context
 fixes $X :: 'x \text{ set}$
 assumes *fin-X*: *finite X*
begin

lemma *Hilbert-fun-cone-aux*:
 assumes $h \in P[X]$ and $h \neq 0$ and $U \subseteq X$ and *homogeneous* ($h::- \Rightarrow_0 'a::\text{field}$)
 shows *Hilbert-fun* (*cone* (h , U)) $z = \text{card } \{t \in .[U]. \text{deg-pm } t + \text{poly-deg } h = z\}$
proof –
 from *assms*(2) have *lpp* $h \in \text{keys } h$ 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*} =
($\lambda t. t + \text{lpp } h$) ‘{*t* ∈ $\cdot[U]$. *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* ∈ $\cdot[U]$ **by** (*rule PPs-closed-lpp*)
ultimately have *lpp* *q* ∈ {*t* ∈ $\cdot[U]$. *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* ∈ $\cdot[U]$. *deg-pm* *t* + *poly-deg* *h* = *z*}
and *t1*: *t* = *s* + *lpp* *h* ..
from *this*(1) **have** *s* ∈ $\cdot[U]$ **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* ∈ $\cdot[U]$ ⟩ **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 $\text{card } \dots = \text{card } \{t \in .[U]. \text{deg-pm } t + \text{poly-deg } h = z\}$ 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 ($h::- \Rightarrow_0 'a::\text{field}$)
 shows *Hilbert-fun* (*cone* (h , $\{\}$)) $z = (\text{if } \text{poly-deg } h = z \text{ then } 1 \text{ else } 0)$
proof –
 have *Hilbert-fun* (*cone* (h , $\{\}$)) $z = \text{card } \{t \in .[\{\}::'x \text{ set}]. \text{deg-pm } t + \text{poly-deg } h = z\}$
 using *assms*(1, 2) *empty-subsetI* *assms*(3) by (*rule Hilbert-fun-cone-aux*)
 also have $\dots = (\text{if } \text{poly-deg } h = z \text{ then } 1 \text{ else } 0)$ 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 ($h::- \Rightarrow_0 'a::\text{field}$)
 and $U \neq \{\}$
 shows *Hilbert-fun* (*cone* (h , U)) $z = (\text{if } \text{poly-deg } h \leq z \text{ then } ((z - \text{poly-deg } h) + (\text{card } U - 1)) \text{ choose } (\text{card } U - 1) \text{ else } 0)$
proof (*cases poly-deg h ≤ 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)) $z = \text{card } \{t \in .[U]. \text{deg-pm } t + \text{poly-deg } h = z\}$
 by (*rule Hilbert-fun-cone-aux*)
 also from *True* have $\{t \in .[U]. \text{deg-pm } t + \text{poly-deg } h = z\} = \text{deg-sect } U \ (z - \text{poly-deg } h)$
 by (*auto simp: deg-sect-def*)
 also from $\langle \text{finite } U \rangle$ *assms*(5) have $\text{card } \dots = (z - \text{poly-deg } h) + (\text{card } U - 1) \text{ choose } (\text{card } U - 1)$
 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)) $z = \text{card } \{t \in .[U]. \text{deg-pm } t + \text{poly-deg } h = z\}$
 by (*rule Hilbert-fun-cone-aux*)
 also from *False* have $\{t \in .[U]. \text{deg-pm } t + \text{poly-deg } h = z\} = \{\}$ by *auto*
 hence $\text{card } \{t \in .[U]. \text{deg-pm } t + \text{poly-deg } h = z\} = \text{card } (\{\}::('x \Rightarrow_0 \text{nat}) \text{ set})$
 by (*rule arg-cong*)
 also have $\dots = 0$ by *simp*
 finally show *?thesis* by (*simp add: False*)
 qed

corollary *Hilbert-fun-Polys*:

assumes $X \neq \{\}$
shows *Hilbert-fun* ($P[X]::(- \Rightarrow_0 'a::field)$ *set*) $z = (z + (card\ X - 1))$ *choose* $(card\ X - 1)$
proof –
let $?one = 1::('x \Rightarrow_0 nat) \Rightarrow_0 'a$
have *Hilbert-fun* ($P[X]::(- \Rightarrow_0 'a)$ *set*) $z = \text{Hilbert-fun } (cone\ (?one, X))\ z$ **by** *simp*
also have $\dots = (if\ poly-deg\ ?one \leq z\ then\ ((z - poly-deg\ ?one) + (card\ X - 1))\ choose\ (card\ X - 1)\ else\ 0)$
using *one-in-Polys - subset-refl - assms* **by** (*rule Hilbert-fun-cone-nonempty*) *simp-all*
also have $\dots = (z + (card\ X - 1))\ choose\ (card\ X - 1)$ **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* $T\ z = (\sum\ hU \in set\ ps.\ \text{Hilbert-fun } (cone\ hU)\ z)$
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* $T\ z = (\sum\ s \in set\ (map\ cone\ ps).\ \text{Hilbert-fun } s\ z)$
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 $\dots = (\sum\ hU \in set\ ps.\ ((\lambda s.\ \text{Hilbert-fun } s\ z) \circ cone)\ hU)$ **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 \Rightarrow nat) \Rightarrow int \Rightarrow int

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

lemma *poly-fun-Hilbert-poly*: poly-fun (*Hilbert-poly* b)

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

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

assumes $X \neq \{\}$ **and** *valid-decomp* X ps **and** *hom-decomp* ps **and** *cone-decomp* T ps

and *standard-decomp* k ps **and** *exact-decomp* X 0 ps **and** b ps (Suc 0) \leq d
shows int (*Hilbert-fun* T d) = card {h:: \Rightarrow 'a::field. (h, $\{\}$) \in set ps \wedge 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 \leq n **and** Suc 0 \leq n **by** *simp-all*
from *pos-decomp-subset* **have** eq0: (set ps - set (ps₊)) \cup set (ps₊) = set ps **by**
blast
have set ps - set (ps₊) \subseteq set ps **by** *blast*
hence *fin2*: finite (set ps - set (ps₊)) **using** *finite-set* **by** (rule *finite-subset*)

have ($\sum_{hU \in \text{set ps} - \text{set (ps}_+)} \text{Hilbert-fun (cone hU) d}$) =
 ($\sum_{(h, U) \in \text{set ps} - \text{set (ps}_+)} \text{if poly-deg h = d then 1 else 0}$)
using *refl*

proof (rule *sum.cong*)

fix x
assume x \in set ps - set (ps₊)
moreover obtain h U **where** x = (h, U) **using** *prod.exhaust* **by** *blast*
ultimately have U = $\{\}$ **and** (h, U) \in set ps **by** (*simp-all* add: *pos-decomp-def*)
from *assms*(2) *this*(2) **have** h \in P[X] **and** h \neq 0 **by** (rule *valid-decompD*) +
moreover from *assms*(3) \langle h, U $\rangle \in$ set ps **have** *homogeneous h* **by** (rule *hom-decompD*)

ultimately show *Hilbert-fun* (cone x) d = (case x of (h, U) \Rightarrow if poly-deg h = d then 1 else 0)

by (simp add: x \langle U = $\{\}$ \rangle *Hilbert-fun-cone-empty split del: if-split*)

qed

also from *fin2* **have** ... = ($\sum_{(h, U) \in \{(h', U') \in \text{set ps} - \text{set (ps}_+). \text{poly-deg } h' = d\}. 1}$)

by (rule *sum.mono-neutral-cong-right*) (auto split: *if-splits*)
also have ... = card {(h, U) \in set ps - set (ps₊). poly-deg h = d} **by** *auto*
also have ... = card {h. (h, $\{\}$) \in set ps \wedge poly-deg h = d} **by** (fact *card-Diff-pos-decomp*)
finally have eq1: ($\sum_{hU \in \text{set ps} - \text{set (ps}_+)} \text{Hilbert-fun (cone hU) d}$) =
 card {h. (h, $\{\}$) \in set ps \wedge poly-deg h = d} .


```

let ?f =  $\lambda a b. (int\ d) - a + b\ gchoose\ b$ 
have  $int\ (\sum_{hU \in set\ (ps_+)} Hilbert\text{-}fun\ (cone\ hU)\ d) = (\sum_{hU \in set\ (ps_+)} int\ (Hilbert\text{-}fun\ (cone\ hU)\ d))$ 
by (simp add: int-sum prod.case-distrib)
also have  $\dots = (\sum_{(h, U) \in (\bigcup_{i \in \{1..n\}} \{(h, U) \in set\ (ps_+). card\ U = i\})} ?f\ (poly\text{-}deg\ h)\ (card\ U - 1))$ 
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  $x = (h, U)$  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  $\langle U \neq \{\} \rangle$  have  $0 < card\ U$  by (simp add: card-gt-0-iff)
  moreover from fin-X  $\langle U \subseteq X \rangle$  have  $card\ U \leq n$  unfolding n-def by (rule card-mono)
  ultimately have  $card\ U \in \{1..n\}$  by simp
  moreover from  $\langle (h, U) \in set\ (ps_+) \rangle$  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  $x = (h, U)$  by blast
from fin-X assms(2, 5) this(1) have  $poly\text{-}deg\ h < b\ ps\ (Suc\ 0)$  by (rule b-one-gr)
also have  $\dots \leq d$  by fact
finally have  $poly\text{-}deg\ h < d$  .
hence int1:  $int\ (d - poly\text{-}deg\ h) = int\ d - int\ (poly\text{-}deg\ h)$  by simp
from  $\langle card\ U = j \rangle \langle j \in \{1..n\} \rangle$  have  $0 < card\ U$  by simp
hence int2:  $int\ (card\ U - Suc\ 0) = int\ (card\ U) - 1$  by simp
from  $\langle (h, U) \in set\ (ps_+) \rangle$  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)  $\langle (h, U) \in set\ ps \rangle$  have homogeneous h by (rule hom-decompD)
moreover from  $\langle 0 < card\ U \rangle$  have  $U \neq \{\}$  by auto
ultimately have  $Hilbert\text{-}fun\ (cone\ (h, U))\ d =$ 
  (if  $poly\text{-}deg\ h \leq d$  then  $(d - poly\text{-}deg\ h + (card\ U - 1))$  choose  $(card\ U - 1)$  else 0)
by (rule Hilbert-fun-cone-nonempty)

```

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

```

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

```

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

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

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

corollary *Hilbert-fun-eq-Hilbert-poly*:
assumes $X \neq \{\}$ **and** *valid-decomp* $X \text{ ps}$ **and** *hom-decomp* ps **and** *cone-decomp* $T \text{ ps}$
and *standard-decomp* $k \text{ ps}$ **and** *exact-decomp* $X \text{ } 0 \text{ ps}$ **and** $\text{b ps } 0 \leq d$
shows $\text{int (Hilbert-fun (T::(- \Rightarrow_0 'a::field) set) } d) = \text{Hilbert-poly (b ps) } d$
proof –
from *fin-X* **have** $\text{b ps (Suc 0)} \leq \text{b ps } 0$ **using** *le0* **by** (*rule b-decreasing*)
also have $\dots \leq d$ **by** *fact*
finally have $\text{b ps (Suc 0)} \leq d .$
with *assms*(1–6) **have** $\text{int (Hilbert-fun } T \text{ } d) =$
 $\text{int (card } \{h. (h, \{\}) \in \text{set ps} \wedge \text{poly-deg } h = d\}) + \text{Hilbert-poly (b ps) (int } d)$

```

    by (rule Hilbert-fun-eq-Hilbert-poly-plus-card)
  also have ... = Hilbert-poly (b ps) (int d)
proof -
  have eq: {h. (h, { }) ∈ set ps ∧ poly-deg h = d} = { }
proof -
  {
    fix h
    assume (h, { }) ∈ set ps and poly-deg h = d
    from fin-X this(1) le0 have poly-deg h < b ps 0 by (rule b)
    with assms(7) have False by (simp add: ⟨poly-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 ⇒0 nat) ⇒0 'a::field
  fixes F
  assumes n-gr-1: 1 < card X and fin-F: finite F and F-sub: F ⊆ P[X] and
f-in: f ∈ F
    and hom-F: ⋀f'. f' ∈ F ⇒ homogeneous f' and f-max: ⋀f'. f' ∈ F ⇒
poly-deg f' ≤ poly-deg f
    and d-gr-0: 0 < poly-deg f and ideal-f-neq: ideal {f} ≠ ideal F
begin

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

lemma f-in-Polys: f ∈ P[X]
  using f-in F-sub ..

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

lemma f-not-0: f ≠ 0
  using d-gr-0 by auto

lemma X-not-empty: X ≠ { }
  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 ...  $\subseteq$  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  $\langle p \in \text{ideal } \{f\} \rangle$ 
    by (rule ideal.span-finiteE)
  hence p: p = q f * f by simp
  with  $\langle p \in P[X] \rangle$  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  $\langle q \in P[X] \rangle$  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-ps = (SOME x. valid-decomp X (snd x)  $\wedge$  standard-decomp d (snd x)  $\wedge$ 
    exact-decomp X 0 (snd x)  $\wedge$  cone-decomp (fst x) (snd x)  $\wedge$ 
    hom-decomp (snd x)  $\wedge$ 
    direct-decomp (ideal F  $\cap$  P[X]) [ideal {f}  $\cap$  P[X], fst x])

private definition P where P = fst P-ps

private definition ps where ps = snd P-ps

lemma
  shows valid-ps: valid-decomp X ps (is ?thesis1)
    and std-ps: standard-decomp d ps (is ?thesis2)
    and ext-ps: exact-decomp X 0 ps (is ?thesis3)
    and cn-ps: cone-decomp P ps (is ?thesis4)
    and hom-ps: hom-decomp ps (is ?thesis5)

```

and *decomp-F*: *direct-decomp* (*ideal* $F \cap P[X]$) [*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 $P' \text{ ps'}$ **where** 1: *valid-decomp* $X \text{ ps'}$ **and** 2: *standard-decomp*
 $d \text{ ps'}$
and 3: *cone-decomp* $P' \text{ ps'}$ **and** 40: $(\bigwedge f'. f' \in F - \{f\} \implies \text{homogeneous } f')$
 $\implies \text{hom-decomp } \text{ps'}$
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 $?ps = \text{exact } X \text{ (poly-deg } f) \text{ ps'}$
from *fin-X* 1 2 **have** *valid-decomp* $X \text{ ?ps}$ **and** *standard-decomp* $d \text{ ?ps}$ **and** *ex-*
act-decomp $X \text{ 0 ?ps}$
by (*rule exact*)
moreover from *fin-X* 1 2 3 **have** *cone-decomp* $P' \text{ ?ps}$ **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* $X \text{ (snd (P', ?ps))} \wedge \text{standard-decomp } d \text{ (snd (P',$
 $\text{?ps))} \wedge$
 $\text{exact-decomp } X \text{ 0 (snd (P', ?ps))} \wedge \text{cone-decomp (fst (P', ?ps))}$
 $\text{(snd (P', ?ps))} \wedge$
 $\text{hom-decomp (snd (P', ?ps))} \wedge$
 $\text{direct-decomp (ideal } F \cap P[X]) \text{ [ideal } \{f\} \cap P[X], \text{fst (P', ?ps)]}$
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*: $\text{ps}_+ \neq []$

proof
assume $\text{ps}_+ = []$
have $\text{Keys } P \subseteq (\bigcup hU \in \text{set } \text{ps. keys (fst } hU)) \text{ (is - } \subseteq \text{ ?A)}$
proof
fix t
assume $t \in \text{Keys } P$
then obtain p **where** $p \in P$ **and** $t \in \text{keys } p$ **by** (*rule in-KeysE*)
from *cn-ps* **have** *direct-decomp* $P \text{ (map cone ps)}$ **by** (*rule cone-decompD*)
then obtain qs **where** $qs: qs \in \text{listset (map cone ps)}$ **and** $p: p = \text{sum-list } qs$
using $\langle p \in P \rangle$

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

have $\exists q \in \text{ideal } F. q \in P[X] \wedge q \neq 0 \wedge \neg \text{lpp } f \text{ adds lpp } q$
 proof (rule ccontr)
 assume $\neg (\exists q \in \text{ideal } F. q \in P[X] \wedge q \neq 0 \wedge \neg \text{lpp } f \text{ adds lpp } q)$
 hence adds: $\text{lpp } f \text{ adds lpp } q$ if $q \in \text{ideal } F$ and $q \in P[X]$ and $q \neq 0$ for q
 using that by blast
 from fin-X - F-sub have ideal $\{f\} = \text{ideal } F$
 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 $\{f\} \subseteq \text{ideal } F$ by (rule ideal.span-mono)
 next
 fix q
 assume $q \in \text{ideal } F$ and $q \in P[X]$ and $q \neq 0$
 hence $\text{lpp } f \text{ adds lpp } q$ by (rule adds)
 with f-not-0 show $\exists g \in \{f\}. g \neq 0 \wedge \text{lpp } g \text{ adds lpp } q$ by blast
 qed
 with ideal-f-neq show False ..
 qed
 then obtain $q0$ where $q0 \in \text{ideal } F$ and $q0 \in P[X]$ and $q0 \neq 0$
 and $\neg \text{lpp } f \text{ adds lpp } q0$ by blast
 define q where $q = \text{hom-component } q0 (\text{deg-pm } (\text{lpp } q0))$

from $\text{hom-}F \langle q0 \in \text{ideal } F \rangle$ **have** $q \in \text{ideal } F$ **unfolding** $q\text{-def}$ **by** (rule homogeneous-ideal)
from $\text{homogeneous-set-Polys} \langle q0 \in P[X] \rangle$ **have** $q \in P[X]$ **unfolding** $q\text{-def}$ **by** (rule homogeneous-setD)
from $\langle q0 \neq 0 \rangle$ **have** $q \neq 0$ **and** $\text{lpp } q = \text{lpp } q0$ **unfolding** $q\text{-def}$ **by** (rule hom-component-lpp)+
from $\text{nadds-}q0$ **this**(2) **have** $\text{nadds-}q: \neg \text{lpp } f \text{ adds } \text{lpp } q$ **by** *simp*
have $\text{hom-}q: \text{homogeneous } q$ **by** (simp only: $q\text{-def}$ homogeneous-hom-component)
from $\text{nadds-}q$ **obtain** x **where** $x: \neg \text{lookup } (\text{lpp } f) \ x \leq \text{lookup } (\text{lpp } q) \ x$
by (auto simp add: $\text{adds-poly-mapping le-fun-def}$)
obtain y **where** $y \in X$ **and** $y \neq x$
proof –
from $n\text{-gr-}1$ **have** $2 \leq n$ **by** *simp*
then obtain Y **where** $Y \subseteq X$ **and** $\text{card } Y = 2$ **by** (rule card-geq-ex-subset)
from $\text{this}(2)$ **obtain** $u \ v$ **where** $u \neq v$ **and** $Y = \{u, v\}$ **by** (rule card-2-E)
from this **obtain** y **where** $y \in Y$ **and** $y \neq x$ **by** *blast*
from $\text{this}(1) \langle Y \subseteq X \rangle$ **have** $y \in X$..
thus $?thesis$ **using** $\langle y \neq x \rangle$..
qed
define q' **where** $q' = (\lambda k. \text{punit.monom-mult } 1 \ (\text{Poly-Mapping.single } y \ k) \ q)$
have $\text{inj1}: \text{inj } q'$ **by** (auto intro!: $\text{injI simp: } q'\text{-def} \langle q \neq 0 \rangle \text{dest: punit.monom-mult-inj-2 monomial-inj}$)
have $q'\text{-in}: q' \ k \in \text{ideal } F \cap P[X]$ **for** k **unfolding** $q'\text{-def}$ **using** $\langle q \in \text{ideal } F \rangle \langle q \in P[X] \rangle \langle y \in X \rangle$
by (intro IntI $\text{punit.pmdl-closed-monom-mult[simplified]}$ $\text{Polys-closed-monom-mult PPs-closed-single}$)
have $\text{lpp-}q': \text{lpp } (q' \ k) = \text{Poly-Mapping.single } y \ k + \text{lpp } q$ **for** k
using $\langle q \neq 0 \rangle$ **by** (simp add: $q'\text{-def}$ $\text{punit.lt-monom-mult}$)
have $\text{inj2}: \text{inj-on } (\text{deg-pm} \circ \text{lpp}) \ (\text{range } q')$
by (auto intro!: $\text{inj-onI simp: lpp-}q' \text{ deg-pm-plus deg-pm-single dest: monomial-inj}$)
have $(\text{deg-pm} \circ \text{lpp}) \text{ ` range } q' \subseteq \text{deg-pm ` Keys } P$
proof
fix d
assume $d \in (\text{deg-pm} \circ \text{lpp}) \text{ ` range } q'$
then obtain k **where** $d: d = \text{deg-pm } (\text{lpp } (q' \ k))$ (is - = $\text{deg-pm } ?t$) **by** *auto*
from $\text{hom-}q$ **have** $\text{hom-}q': \text{homogeneous } (q' \ k)$ **by** (simp add: $q'\text{-def}$ homogeneous-monom-mult)
from $\langle q \neq 0 \rangle$ **have** $q' \ k \neq 0$ **by** (simp add: $q'\text{-def}$ $\text{punit.monom-mult-eq-zero-iff}$)
hence $?t \in \text{keys } (q' \ k)$ **by** (rule punit.lt-in-keys)
with $\text{hom-}q'$ **have** $\text{deg-}q': d = \text{poly-deg } (q' \ k)$ **unfolding** d **by** (rule homogeneousD-poly-deg)
from $\text{decomp-}F \ q'\text{-in}$ **obtain** qs **where** $qs \in \text{listset } [\text{ideal } \{f\} \cap P[X], P]$ **and** $q' \ k = \text{sum-list } qs$
by (rule direct-decompE)
moreover from $\text{this}(1)$ **obtain** $f0 \ p0$ **where** $f0: f0 \in \text{ideal } \{f\} \cap P[X]$ **and** $p0: p0 \in P$
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*) $cc \equiv (\lambda i. aa\ i + bb\ i)$

lemma

shows *valid-qs*: *valid-decomp* $X\ qs$ (**is** *?thesis1*)
and *std-qs*: *standard-decomp* $0\ qs$ (**is** *?thesis2*)
and *mon-qs*: *monomial-decomp* qs (**is** *?thesis3*)
and *hom-qs*: *hom-decomp* qs (**is** *?thesis6*)
and *cn-qs*: *cone-decomp* $N\ qs$ (**is** *?thesis4*)
and *ext-qs*: *exact-decomp* $X\ 0\ qs$ (**is** *?thesis5*)
and *deg-RGB*: $g \in \text{punit.reduced-GB } F \implies \text{poly-deg } g \leq bb\ 0$
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: $\bigwedge g. (\bigwedge f. f \in F \implies \text{homogeneous } f) \implies g \in \text{punit.reduced-GB } F \implies \text{poly-deg } g \leq b\ qs'\ 0$
by (*rule normal-form-exact-decompE*) *blast*
from *hom-F* **have** $\bigwedge g. g \in \text{punit.reduced-GB } F \implies \text{poly-deg } g \leq b\ qs'\ 0$ **by** (*rule 60*)
with 1 2 3 4 5 **have** *valid-decomp* $X\ qs' \wedge \text{standard-decomp } 0\ qs' \wedge$
monomial-decomp $qs' \wedge \text{cone-decomp } N\ qs' \wedge \text{exact-decomp } X\ 0$
 $qs' \wedge$
 $(\forall g \in \text{punit.reduced-GB } F. \text{poly-deg } g \leq b\ qs'\ 0)$ **by** (*simp add:*
N-def)
hence *?thesis1* \wedge *?thesis2* \wedge *?thesis3* \wedge *?thesis4* \wedge *?thesis5* \wedge $(\forall g \in \text{punit.reduced-GB } F. \text{poly-deg } g \leq bb\ 0)$
unfolding *qs-def* *bb-def* **by** (*rule someI*)
thus *?thesis1* **and** *?thesis2* **and** *?thesis3* **and** *?thesis4* **and** *?thesis5*
and $g \in \text{punit.reduced-GB } F \implies \text{poly-deg } g \leq bb\ 0$ **by** *simp-all*
from $\langle ?thesis3 \rangle$ **show** *?thesis6* **by** (*rule monomial-decomp-imp-hom-decomp*)
qed

lemma *N-sub*: $N \subseteq P[X]$

using *valid-qs cn-qs* **by** (*rule valid-cone-decomp-subset-Polys*)

lemma *decomp-Polys*: *direct-decomp* $P[X]\ [ideal\ \{f\} \cap P[X], P, N]$

proof –

from *fin-X F-sub* **have** *direct-decomp* $P[X]\ [ideal\ F \cap P[X], N]$ **unfolding** *N-def*
by (*rule direct-decomp-ideal-normal-form*)
hence *direct-decomp* $P[X]\ ([N] @ [ideal\ \{f\} \cap P[X], P])$ **using** *decomp-F*
by (*rule direct-decomp-direct-decomp*)
hence *direct-decomp* $P[X]\ ([ideal\ \{f\} \cap P[X], P] @ [N])$
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** $\dots = d$ **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** $\dots = 0$ **unfolding** *a-def[OF fin-X]* **by** (rule *Least-eq-0*)
finally show *?thesis* .
qed

lemma *Hilbert-fun-X*:
assumes $d \leq z$
shows $Hilbert-fun \ (P[X]::(- \Rightarrow_0 \ 'a) \ set) \ z =$
 $((z - d) + (n - 1) \ choose \ n - 1) + Hilbert-fun \ P \ z + Hilbert-fun \ N \ z$
proof –
define *ss* **where** $ss = [ideal \ \{f\} \cap P[X], P, N]$
have *homogeneous-set A* \wedge *phull.subspace A* **if** $A \in set \ ss$ **for** *A*
proof –
from *that* **have** $A = ideal \ \{f\} \cap P[X] \vee A = P \vee A = N$ **by** (*simp add: ss-def*)
thus *?thesis*
proof (*elim disjE*)
assume $A: A = ideal \ \{f\} \cap P[X]$
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 $1: \bigwedge A. A \in set \ ss \Longrightarrow homogeneous-set \ A$ **and** $2: \bigwedge A. A \in set \ ss \Longrightarrow$
 $phull.subspace \ A$
by *simp-all*
have $Hilbert-fun \ (P[X]::(- \Rightarrow_0 \ 'a) \ set) \ z = (\sum p \in set \ ss. Hilbert-fun \ p \ z)$
using *fin-X subset-refl decomp-Polys* **unfolding** *ss-def*
proof (rule *Hilbert-fun-direct-decomp*)
fix *A*
assume $A \in set \ [ideal \ \{f\} \cap P[X], P, N]$
hence $A \in set \ ss$ **by** (*simp only: ss-def*)

```

    thus homogeneous-set A and phull.subspace A by (rule 1, rule 2)
  qed
  also have ... = ( $\sum p \in \text{set } ss. \text{count-list } ss \ p * \text{Hilbert-fun } p \ z$ )
    using refl
  proof (rule sum.cong)
    fix p
    assume  $p \in \text{set } ss$ 
    hence  $\text{count-list } ss \ p \neq 0$  by (simp only: count-list-0-iff not-not)
    hence  $\text{count-list } ss \ p = 1 \vee 1 < \text{count-list } ss \ p$  by auto
    thus  $\text{Hilbert-fun } p \ z = \text{count-list } ss \ p * \text{Hilbert-fun } p \ z$ 
  proof
    assume  $1 < \text{count-list } ss \ p$ 
    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 ( $\lambda p. \text{Hilbert-fun } p \ z$ ) ss)
    by (rule sym) (rule sum-list-map-eq-sum-count)
  also have ... =  $\text{Hilbert-fun } (\text{cone } (f, X)) \ z + \text{Hilbert-fun } P \ z + \text{Hilbert-fun } N \ z$ 
    by (simp add: ss-def ideal-Int-Polys-eq-cone)
  also have  $\text{Hilbert-fun } (\text{cone } (f, X)) \ z = (z - d + (n - 1)) \text{ choose } (n - 1)$ 
    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:

```

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

```

using X -not-empty valid-ps hom-ps cn-ps std-ps ext-ps $\langle aa \ 0 \leq nat \ z \rangle$
 valid-qs hom-qs cn-qs std-qs ext-qs $\langle bb \ 0 \leq nat \ z \rangle \langle 0 \leq z \rangle$
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 $\langle 0 \leq z \rangle$
by (simp add: Hilbert-fun-Polys int-binomial) smt
qed (simp-all add: poly-fun-Hilbert-poly)

corollary dube-eq-1:

$(\lambda z::int. (z + int \ n - 1) \ gchoose \ (n - 1)) =$
 $(\lambda z::int. ((z - d + n - 1) \ gchoose \ (n - 1)) + ((z - d + n) \ gchoose \ n) + ((z$
 $+ n) \ gchoose \ n) - 2 -$
 $(\sum_{i=1..n.} ((z - aa \ i + i - 1) \ gchoose \ i) + ((z - bb \ i + i - 1) \ gchoose$
 $i)))$
by (simp only: dube-eq-0) (auto simp: Hilbert-poly-def Let-def sum.distrib)

lemma dube-eq-2:

assumes $j < n$
shows $(\lambda z::int. (z + int \ n - int \ j - 1) \ gchoose \ (n - j - 1)) =$
 $(\lambda z::int. ((z - d + n - int \ j - 1) \ gchoose \ (n - j - 1)) + ((z - d + n$
 $- j) \ gchoose \ (n - j)) +$
 $((z + n - j) \ gchoose \ (n - j)) - 2 -$
 $(\sum_{i=Suc \ j..n.} ((z - aa \ i + i - j - 1) \ gchoose \ (i - j)) + ((z -$
 $bb \ i + i - j - 1) \ gchoose \ (i - j))))$
(is $?f = ?g$ **)**
proof -
let $?h = \lambda z \ i. ((z + (int \ i - aa \ i - 1)) \ gchoose \ i) + ((z + (int \ i - bb \ i - 1))$
 $gchoose \ i)$
let $?hj = \lambda z \ i. ((z + (int \ i - aa \ i - 1) - j) \ gchoose \ (i - j)) + ((z + (int \ i -$
 $bb \ i - 1) - j) \ gchoose \ (i - j))$
from assms **have** $1: j \leq n - Suc \ 0$ **and** $2: j \leq n$ **by** simp-all

have eq1: $(bw_diff \ \sim j) (\lambda z. \sum_{i=1..j.} ?h \ z \ i) = (\lambda -. \text{if } j = 0 \text{ then } 0 \text{ else } 2)$
proof (cases j)
case 0
thus ?thesis **by** simp
next
case (Suc $j0$)
hence $j \neq 0$ **by** simp
have $(\lambda z::int. \sum_{i=1..j.} ?h \ z \ i) = (\lambda z::int. (\sum_{i=1..j0.} ?h \ z \ i) + ?h \ z \ j)$
by (simp add: $\langle j = Suc \ j0 \rangle$)
moreover have $(bw_diff \ \sim j) \dots = (\lambda z::int. (\sum_{i=1..j0.} (bw_diff \ \sim j) (\lambda z.$
 $?h \ z \ i) \ z) + 2)$
by (simp add: bw-diff-gbinomial-pow)
moreover have $(\sum_{i=1..j0.} (bw_diff \ \sim j) (\lambda z. ?h \ z \ i) \ z) = (\sum_{i=1..j0.}$
 $0)$ **for** $z::int$
using refl
proof (rule sum.cong)
fix i
assume $i \in \{1..j0\}$

hence $\neg j \leq i$ **by** (*simp add: $\langle j = \text{Suc } j0 \rangle$*)
 thus $(bw\text{-diff } \sim j) (\lambda z. ?h z i) z = 0$ **by** (*simp add: bw-diff-gbinomial-pow*)
qed
 ultimately show *?thesis* **by** (*simp add: $\langle j \neq 0 \rangle$*)
qed

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

from 1 have $?f = (bw\text{-diff } \sim j) (\lambda z::\text{int}. (z + (\text{int } n - 1)) \text{ gchoose } (n - 1))$
 by (*simp add: bw-diff-gbinomial-pow*) (*simp only: algebra-simps*)
 also have $\dots = (bw\text{-diff } \sim j) (\lambda z::\text{int}. (z + \text{int } n - 1) \text{ gchoose } (n - 1))$
 by (*simp only: algebra-simps*)
 also have $\dots = (bw\text{-diff } \sim j)$
 $(\lambda z::\text{int}. ((z - d + n - 1) \text{ gchoose } (n - 1)) + ((z - d + n) \text{ gchoose } n)$
 $+ ((z + n) \text{ gchoose } n) - 2 -$
 $(\sum i=1..n. ((z - aa i + i - 1) \text{ gchoose } i) + ((z - bb i + i - 1) \text{ gchoose } i)))$
 by (*simp only: dube-eq-1*)
 also have $\dots = (bw\text{-diff } \sim j)$
 $(\lambda z::\text{int}. ((z + (\text{int } n - d - 1)) \text{ gchoose } (n - 1)) + ((z + (\text{int } n - d))$
 $\text{gchoose } n) +$
 $((z + n) \text{ gchoose } n) - 2 - (\sum i=1..n. ?h z i))$
 by (*simp only: algebra-simps*)
 also have $\dots = (\lambda z::\text{int}. ((z + (\text{int } n - d - 1) - j) \text{ gchoose } (n - 1 - j)) +$
 $((z + (\text{int } n - d) - j) \text{ gchoose } (n - j)) + ((z + n - j) \text{ gchoose } (n -$
 $j)) - (\text{if } j = 0 \text{ then } 2 \text{ else } 0) -$
 $(bw\text{-diff } \sim j) (\lambda z. \sum i=1..n. ?h z i) z)$
 using 1 2 **by** (*simp add: bw-diff-const-pow bw-diff-gbinomial-pow del: bw-diff-sum-pow*)
 also from $\langle j \leq n \rangle$ have $(\lambda z. \sum i=1..n. ?h z i) = (\lambda z. (\sum i=1..j. ?h z i) +$
 $(\sum i=\text{Suc } j..n. ?h z i))$
 by (*simp add: sum-split-nat-ivl*)
 also have $(bw\text{-diff } \sim j) \dots = (\lambda z. (bw\text{-diff } \sim j) (\lambda z. \sum i=1..j. ?h z i) z +$
 $(bw\text{-diff } \sim j) (\lambda z. \sum i=\text{Suc } j..n. ?h z i) z)$
 by (*simp only: bw-diff-plus-pow*)
 also have $\dots = (\lambda z. (\text{if } j = 0 \text{ then } 0 \text{ else } 2) + (\sum i=\text{Suc } j..n. ?hj z i))$

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

lemma dube-eq-3:

assumes $j < n$
 shows $(1::int) = (-1)^{\wedge(n - \text{Suc } j)} * ((\text{int } d - 1) \text{ gchoose } (n - \text{Suc } j)) +$
 $(-1)^{\wedge(n - j)} * ((\text{int } d - 1) \text{ gchoose } (n - j)) - 1 -$
 $(\sum_{i=\text{Suc } j..n.} (-1)^{\wedge(i - j)} * ((\text{int } (aa \ i) \text{ gchoose } (i - j)) +$
 $(\text{int } (bb \ i) \text{ gchoose } (i - j))))$
 proof -
 from assms have 1: $\text{int } (n - \text{Suc } j) = \text{int } n - j - 1$ and 2: $\text{int } (n - j) = \text{int } n - j$ by simp-all
 from assms have $\text{int } n - \text{int } j - 1 = \text{int } (n - j - 1)$ by simp
 hence eq1: $\text{int } n - \text{int } j - 1 \text{ gchoose } (n - \text{Suc } j) = 1$
 by (simp del: of-nat-diff)
 from assms have $\text{int } n - \text{int } j = \text{int } (n - j)$ by simp
 hence eq2: $\text{int } n - \text{int } j \text{ gchoose } (n - j) = 1$
 using gbinomial-int-n-n by presburger
 have eq3: $\text{int } n - d - j - 1 \text{ gchoose } (n - \text{Suc } j) = (-1)^{\wedge(n - \text{Suc } j)} * (\text{int } d - 1 \text{ gchoose } (n - \text{Suc } j))$
 by (simp add: gbinomial-int-negated-upper[of int n - d - j - 1] 1)
 have eq4: $\text{int } n - d - j \text{ gchoose } (n - j) = (-1)^{\wedge(n - j)} * (\text{int } d - 1 \text{ gchoose } (n - j))$
 by (simp add: gbinomial-int-negated-upper[of int n - d - j] 2)
 have eq5: $(\sum_{i=\text{Suc } j..n.} (\text{int } i - aa \ i - j - 1 \text{ gchoose } i - j) + (\text{int } i - bb \ i - j - 1 \text{ gchoose } (i - j))) =$
 $(\sum_{i=\text{Suc } j..n.} (-1)^{\wedge(i - j)} * ((\text{int } (aa \ i) \text{ gchoose } (i - j)) + (\text{int } (bb \ i) \text{ gchoose } (i - j))))$
 using refl
 proof (rule sum.cong)
 fix i
 assume $i \in \{\text{Suc } j..n\}$
 hence $j \leq i$ by simp
 hence 3: $\text{int } (i - j) = \text{int } i - j$ by simp
 show $(\text{int } i - aa \ i - j - 1 \text{ gchoose } i - j) + (\text{int } i - bb \ i - j - 1 \text{ gchoose } (i - j)) =$
 $(-1)^{\wedge(i - j)} * ((\text{int } (aa \ i) \text{ gchoose } (i - j)) + (\text{int } (bb \ i) \text{ gchoose } (i - j)))$
 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
 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 $(h, \{\}) \in \text{set } ps \cup \text{set } qs$
 shows $\text{poly-deg } h < \max (aa \ 1) (bb \ 1)$

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

let $?S = \lambda A. \{h. (h, \{\}) \in A \wedge \text{poly-deg } h = z\}$
have $\text{fin: finite } (?S\ A)$ **if** $\text{finite } A$ **for** $A::(((x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a) \times 'x\ \text{set})\ \text{set}$
proof –
have $(\lambda t. (t, \{\})) ' ?S\ A \subseteq A$ **by** blast
hence $\text{finite } ((\lambda t. (t, \{\}::'x\ \text{set})) ' ?S\ A)$ **using** that **by** (rule finite-subset)
moreover **have** $\text{inj-on } (\lambda t. (t, \{\}::'x\ \text{set})) (?S\ A)$ **by** (rule inj-onI) simp
ultimately show ?thesis **by** (rule finite-imageD)
qed
from finite-set **have** 1: $\text{finite } (?S\ (\text{set } ps))$ **by** (rule fin)
from finite-set **have** 2: $\text{finite } (?S\ (\text{set } qs))$ **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** $\langle aa\ 1 \leq z \rangle$ **by** (rule le-trans)
hence $\text{eq: int } (z - d) = \text{int } z - \text{int } d$ **by** simp
from $\langle d \leq z \rangle$ **have** Hilbert-fun $(P[X]::(- \Rightarrow_0 'a)\ \text{set})\ z =$
 $((z - d) + (n - 1)\ \text{choose } n - 1) + \text{Hilbert-fun } P\ z +$
Hilbert-fun $N\ z$
by (rule Hilbert-fun-X)
also **have** $\text{int } \dots = ((\text{int } z - d + (n - 1)\ \text{gchoose } n - 1) + \text{Hilbert-poly } aa\ z$
 $+ \text{Hilbert-poly } bb\ z) +$
 $(\text{int } (\text{card } (?S\ (\text{set } ps))) + \text{int } (\text{card } (?S\ (\text{set } qs))))$
using X-not-empty valid-ps hom-ps cn-ps std-ps ext-ps $\langle aa\ 1 \leq z \rangle$
valid-qs hom-qs cn-qs std-qs ext-qs $\langle bb\ 1 \leq z \rangle$
by (simp add: Hilbert-fun-eq-Hilbert-poly-plus-card aa-def bb-def int-binomial
eq)
finally **have** $((\text{int } z - d + n - 1\ \text{gchoose } n - 1) + \text{Hilbert-poly } aa\ z + \text{Hilbert-poly } bb\ z) +$
 $(\text{int } (\text{card } (?S\ (\text{set } ps))) + \text{int } (\text{card } (?S\ (\text{set } qs)))) = \text{int } z + n -$
 $1\ \text{gchoose } (n - 1)$
using fin-X X-not-empty **by** (simp add: Hilbert-fun-Polys int-binomial alge-
bra-simps)
also **have** $\dots = (\text{int } z - d + n - 1\ \text{gchoose } n - 1) + \text{Hilbert-poly } aa\ z +$
Hilbert-poly $bb\ z$
by (fact dube-eq-0[THEN fun-cong])
finally **have** $\text{int } (\text{card } (?S\ (\text{set } ps))) + \text{int } (\text{card } (?S\ (\text{set } qs))) = 0$ **by** simp
hence $\text{card } (?S\ (\text{set } ps)) = 0$ **and** $\text{card } (?S\ (\text{set } qs)) = 0$ **by** simp-all
with 1 2 **have** $?S\ (\text{set } ps \cup \text{set } qs) = \{\}$ **by** auto
moreover **from** assms **have** $h \in ?S\ (\text{set } ps \cup \text{set } qs)$ **by** (simp add: z-def)
ultimately **have** $h \in \{\}$ **by** (rule subst)
thus False **by** simp
qed

lemma

shows $aa\text{-}n$: $aa\ n = d$ **and** $bb\text{-}n$: $bb\ n = 0$ **and** $bb\text{-}0$: $bb\ 0 \leq \max\ (aa\ 1)\ (bb\ 1)$
proof –
let $?j = n - \text{Suc}\ 0$
from $n\text{-}gr\text{-}0$ **have** $?j < n$ **and** $eq1$: $\text{Suc}\ ?j = n$ **and** $eq2$: $n - ?j = 1$ **by** *simp-all*
from $this(1)$ **have** $(1::int) = (-\ 1) \wedge (n - \text{Suc}\ ?j) * ((int\ d - 1)\ gchoose\ (n - \text{Suc}\ ?j)) +$
 $(-\ 1) \wedge (n - ?j) * ((int\ d - 1)\ gchoose\ (n - ?j)) - 1 -$
 $(\sum i=\text{Suc}\ ?j..n. (-\ 1) \wedge (i - ?j) * ((int\ (aa\ i)\ gchoose\ (i - ?j)) +$
 $(int\ (bb\ i)\ gchoose\ (i - ?j))))$
by (*rule dube-eq-3*)
hence eq : $aa\ n + bb\ n = d$ **by** (*simp add: eq1 eq2*)
hence $aa\ n \leq d$ **by** *simp*
moreover **have** $d \leq aa\ n$ **unfolding** $aa\text{-}Suc\text{-}n$ [*symmetric*] $aa\text{-}def$ **using** $fin\text{-}X$
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 $fin\text{-}X$ **have** $\dots \leq \max\ (aa\ 1)\ (bb\ 1)$ (*is - $\leq ?m$*)
proof (*rule b-le*)
from $fin\text{-}X\ ext\text{-}qs$ **have** $a\ qs = bb\ (\text{Suc}\ n)$ **by** (*simp add: b-card-X bb-def*)
also have $\dots \leq bb\ 1$ **unfolding** $bb\text{-}def$ **using** $fin\text{-}X$ **by** (*rule b-decreasing*)
simp
also have $\dots \leq ?m$ **by** (*rule max.cobounded2*)
finally show $a\ qs \leq ?m$.
next
fix $h\ U$
assume $(h,\ U) \in set\ qs$
show $poly\text{-}deg\ h < ?m$
proof (*cases card U = 0*)
case *True*
from $fin\text{-}X\ valid\text{-}qs\ \langle (h,\ U) \in set\ qs \rangle$ **have** $finite\ U$ **by** (*rule valid-decompD-finite*)
with *True* **have** $U = \{\}$ **by** *simp*
with $\langle (h,\ U) \in set\ qs \rangle$ **have** $(h,\ \{\}) \in set\ ps \cup set\ qs$ **by** *simp*
thus $?thesis$ **by** (*rule dube-aux-1*)
next
case *False*
hence $1 \leq card\ U$ **by** *simp*
with $fin\text{-}X\ \langle (h,\ U) \in set\ qs \rangle$ **have** $poly\text{-}deg\ h < bb\ 1$ **unfolding** $bb\text{-}def$ **by**
(*rule b*)
also have $\dots \leq ?m$ **by** (*rule max.cobounded2*)
finally show $?thesis$.
qed
qed
finally show $bb\ 0 \leq ?m$.
qed

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

shows $(1::int) = 2 * (-1) \wedge (n - Suc\ j) * ((int\ d - 1)\ gchoose\ (n - Suc\ j)) - 1 -$
 $(\sum_{i=Suc\ j..n-1}. (-1) \wedge (i - j) * ((int\ (aa\ i)\ gchoose\ (i - j)) +$
 $(int\ (bb\ i)\ gchoose\ (i - j))))$
proof -
from *assms* **have** $Suc\ j \leq n$ **and** $0 < n$ **and** $1: Suc\ (n - Suc\ j) = n - j$ **by** *simp-all*
have $2: (-1) \wedge (n - Suc\ j) = -((-1::int)) \wedge (n - j)$ **by** (*simp flip: 1*)
from *assms* **have** $(1::int) = (-1) \wedge (n - Suc\ j) * ((int\ d - 1)\ gchoose\ (n - Suc\ j)) +$
 $(-1) \wedge (n - j) * ((int\ d - 1)\ gchoose\ (n - j)) - 1 -$
 $(\sum_{i=Suc\ j..n}. (-1) \wedge (i - j) * ((int\ (aa\ i)\ gchoose\ (i - j)) +$
 $(int\ (bb\ i)\ gchoose\ (i - j))))$
by (*rule dube-eq-3*)
also have $\dots = (-1) \wedge (n - Suc\ j) * ((int\ d - 1)\ gchoose\ (n - Suc\ j)) +$
 $(-1) \wedge (n - j) * ((int\ d - 1)\ gchoose\ (n - j)) - 1 -$
 $(-1) \wedge (n - j) * ((int\ (aa\ n)\ gchoose\ (n - j)) + (int\ (bb\ n)\ gchoose\$
 $(n - j))) -$
 $(\sum_{i=Suc\ j..n-1}. (-1) \wedge (i - j) * ((int\ (aa\ i)\ gchoose\ (i - j)) +$
 $(int\ (bb\ i)\ gchoose\ (i - j))))$
using $\langle 0 < n \rangle \langle Suc\ j \leq n \rangle$ **by** (*simp only: sum-tail-nat*)
also have $\dots = (-1) \wedge (n - Suc\ j) * ((int\ d - 1)\ gchoose\ (n - Suc\ j)) +$
 $(-1) \wedge (n - j) * (((int\ d - 1)\ gchoose\ (n - j)) - (int\ d\ gchoose\$
 $(n - j))) - 1 -$
 $(\sum_{i=Suc\ j..n-1}. (-1) \wedge (i - j) * ((int\ (aa\ i)\ gchoose\ (i - j)) +$
 $(int\ (bb\ i)\ gchoose\ (i - j))))$
using *assms* **by** (*simp add: aa-n bb-n gbinomial-0-left right-diff-distrib*)
also have $(-1) \wedge (n - j) * (((int\ d - 1)\ gchoose\ (n - j)) - (int\ d\ gchoose\ (n - j))) =$
 $(-1) \wedge (n - Suc\ j) * (((int\ d - 1 + 1)\ gchoose\ (Suc\ (n - Suc\ j))) -$
 $((int\ d - 1)\ gchoose\ (Suc\ (n - Suc\ j))))$
by (*simp add: 1 2 flip: mult-minus-right*)
also have $\dots = (-1) \wedge (n - Suc\ j) * ((int\ d - 1)\ gchoose\ (n - Suc\ j))$
by (*simp only: gbinomial-int-Suc-Suc, simp*)
finally show *?thesis* **by** *simp*
qed

lemma *cc-Suc*:

assumes $j < n - 1$
shows $int\ (cc\ (Suc\ j)) = 2 + 2 * (-1) \wedge (n - j) * ((int\ d - 1)\ gchoose\ (n -$
 $Suc\ j)) +$
 $(\sum_{i=j+2..n-1}. (-1) \wedge (i - j) * ((int\ (aa\ i)\ gchoose\ (i - j)) +$
 $(int\ (bb\ i)\ gchoose\ (i - j))))$
proof -
from *assms* **have** $j < n$ **and** $Suc\ j \leq n - 1$ **by** *simp-all*
hence $n - j = Suc\ (n - Suc\ j)$ **by** *simp*
hence $eq: (-1) \wedge (n - Suc\ j) = -((-1::int)) \wedge (n - j)$ **by** *simp*
from $\langle j < n \rangle$ **have** $(1::int) = 2 * (-1) \wedge (n - Suc\ j) * ((int\ d - 1)\ gchoose\ (n -$
 $Suc\ j)) - 1 -$

```

      (∑ i=Suc j..n-1. (- 1)^(i - j) * ((int (aa i) gchoose (i - j)) + (int
      (bb i) gchoose (i - j))))
    by (rule dube-eq-4)
    also have ... = cc (Suc j) - 2 * (- 1)^(n - j) * ((int d - 1) gchoose (n -
      Suc j)) - 1 -
      (∑ 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)) +
    (∑ 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 $\text{card } X = 2$ is settled, we can concentrate on $2 < \text{card } X$ now.

```

context
  assumes n-gr-2: 2 < n
begin

```

```

lemma cc-n-minus-2: cc (n - 2) ≤ d2 + 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)) +
    (∑ 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 $\dots \leq (2 - 2 * (\text{int } d - 1 \text{ gchoose } 2)) + (2 * \text{int } d \text{ gchoose } 2)$
proof (rule add-left-mono)
have $(\text{int } (aa \ (n - 1)) \text{ gchoose } 2) + (\text{int } (bb \ (n - 1)) \text{ gchoose } 2) \leq \text{int } (aa \ (n - 1)) + \text{int } (bb \ (n - 1)) \text{ gchoose } 2$
by (rule gbinomial-int-plus-le) simp-all
also have $\dots = \text{int } (2 * d) \text{ gchoose } 2$ **by** (simp flip: cc-n-minus-1)
also have $\dots = 2 * \text{int } d \text{ gchoose } 2$ **by** (simp add: int-ops(7))
finally show $(\text{int } (aa \ (n - 1)) \text{ gchoose } 2) + (\text{int } (bb \ (n - 1)) \text{ gchoose } 2) \leq 2 * \text{int } d \text{ gchoose } 2$.
qed
also have $\dots = 2 - \text{fact } 2 * (\text{int } d - 1 \text{ gchoose } 2) + (2 * \text{int } d \text{ gchoose } 2)$ **by** (simp only: fact-2)
also have $\dots = 2 - (\text{int } d - 1) * (\text{int } d - 2) + (2 * \text{int } d \text{ gchoose } 2)$
by (simp only: gbinomial-int-mult-fact) (simp add: numeral-2-eq-2 prod.atLeast0-lessThan-Suc)
also have $\dots = 2 - (\text{int } d - 1) * (\text{int } d - 2) + \text{int } d * (2 * \text{int } d - 1)$
by (simp add: gbinomial-prod-rev numeral-2-eq-2 prod.atLeast0-lessThan-Suc)
also have $\dots = \text{int } (d^2 + 2 * d)$ **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 $\text{int } (cc \ (\text{Suc } j)) \leq 2 + (\text{int } (cc \ (j + 2)) \text{ gchoose } 2) + (\sum i=j+4..n-1. \text{int } (cc \ i) \text{ gchoose } (i - j))$
— Could be proved without coercing to *int*, because everything is non-negative.
proof —
let ?f = $\lambda i \ j. (\text{int } (aa \ i) \text{ gchoose } (i - j)) + (\text{int } (bb \ i) \text{ gchoose } (i - j))$
let ?S = $\lambda x \ y. (\sum i=j+x..n-y. (-1)^\wedge(i-j) * ?f \ i \ j)$
let ?S3 = $\lambda x \ y. (\sum i=j+x..n-y. (\text{int } (cc \ i) \text{ gchoose } (i - j)))$
have ie1: $(\text{int } (aa \ i) \text{ gchoose } k) + (\text{int } (bb \ i) \text{ gchoose } k) \leq \text{int } (cc \ i) \text{ gchoose } k$
if $0 < k$ **for** $i \ k$
proof —
from that **have** $(\text{int } (aa \ i) \text{ gchoose } k) + (\text{int } (bb \ i) \text{ gchoose } k) \leq \text{int } (aa \ i) + \text{int } (bb \ i) \text{ gchoose } k$
by (rule gbinomial-int-plus-le) simp-all
also have $\dots = \text{int } (cc \ i) \text{ gchoose } k$ **by** simp
finally show ?thesis .
qed
from d-gr-0 **have** $0 \leq \text{int } d - 1$ **by** simp
from assms **have** $0 < n - \text{Suc } j$ **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: $n - \text{Suc } j = 3$ **and** $j < n - 1$ **and** 2: $\text{Suc } (n - 3) = \text{Suc } (\text{Suc } j)$

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

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


```

    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 $\dots \leq 2 + (int\ (cc\ (j0 + 2))\ gchoose\ 2) + (\sum_{i=j0+4..n-1}. int\ (cc\ i)\ gchoose\ (i - j0))$
 using $\langle j0 < n - 3 \rangle$ **by** (*rule cc-Suc-le*)
 also have $\dots = 2 + (int\ (cc\ (j + 1))\ gchoose\ 2) + (\sum_{i=j0+4..n-1}. int\ (cc\ i)\ gchoose\ (i - j0))$
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*: $0 < j \implies j + 1 \leq n \implies cc\ j \leq Dube\ aux\ n\ d\ j$

proof (*induct j rule: Dube-aux.induct[where n=n]*)

case *step*: (1 j)

from *step.prem*s(2) have $j + 2 < n \vee j + 2 = n \vee j + 1 = n$ **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 $cc\ (j + 1) \leq Dube\ aux\ n\ d\ (j + 1)$ **by** (*rule step.hyps*)

hence 1: $cc\ (j + 1)\ choose\ 2 \leq Dube\ aux\ n\ d\ (j + 1)\ choose\ 2$

by (*rule Binomial-Int.binomial-mono*)

have 2: $(\sum_{i=j+3..n-1}. cc\ i\ choose\ (Suc\ (i - j))) \leq$

$(\sum_{i=j+3..n-1}. Dube\ aux\ n\ d\ i\ choose\ (i - j))$

proof (*rule sum-mono*)

fix $i::nat$

note *

```

    moreover assume  $i \in \{j + 3..n - 1\}$ 
    moreover from this  $\langle 2 < n \rangle$  have  $0 < i$  and  $i + 1 \leq n$  by auto
    ultimately have  $cc\ i \leq Dube\text{-}aux\ n\ d\ i$  by (rule step.hyps)
    thus  $cc\ i\ choose\ Suc\ (i - j) \leq Dube\text{-}aux\ n\ d\ i\ choose\ Suc\ (i - j)$ 
    by (rule Binomial-Int.binomial-mono)
  qed
  from * have  $j < n - 2$  by simp
  with step.prem1 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  $\dots \leq Dube\text{-}aux\ n\ d\ j$  by simp
  finally show ?thesis .
next
  assume  $j + 2 = n$ 
  hence  $j = n - 2$  and  $Dube\text{-}aux\ n\ d\ j = d^2 + 2 * d$  by simp-all
  thus ?thesis by (simp only: cc-n-minus-2)
next
  assume  $j + 1 = n$ 
  hence  $j = n - 1$  and  $Dube\text{-}aux\ n\ d\ j = 2 * d$  by simp-all
  thus ?thesis by (simp only: cc-n-minus-1)
qed
qed
end

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

```

theorem *Dube*:
assumes *finite* F **and** $F \subseteq P[X]$ **and** $\bigwedge f. f \in F \implies \text{homogeneous } f$ **and** $g \in \text{punit.reduced-GB } F$
shows $\text{poly-deg } g \leq \text{Dube } (\text{card } X) (\text{maxdeg } F)$
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 $\text{poly-deg } F \neq \{\}$ **by** *simp*
 from *assms*(1) **have** *fin1*: *finite* ($\text{poly-deg } F$) **by** (*rule finite-imageI*)
 from *assms*(1) **have** *finite* ($F - \{0\}$) **by** *simp*
 hence *fin*: *finite* ($\text{poly-deg } (F - \{0\})$) **by** (*rule finite-imageI*)
 moreover from $\langle F - \{0\} \neq \{\} \rangle$ **have** *: $\text{poly-deg } (F - \{0\}) \neq \{\}$ **by** *simp*
 ultimately have $\text{maxdeg } (F - \{0\}) \in \text{poly-deg } (F - \{0\})$ **unfolding** *maxdeg-def*
by (*rule Max-in*)
 then obtain f **where** $f \in F - \{0\}$ **and** $\text{md1}: \text{maxdeg } (F - \{0\}) = \text{poly-deg } f$
 ..
 note *this*(2)
 moreover have $\text{maxdeg } (F - \{0\}) \leq \text{maxdeg } F$
 unfolding *maxdeg-def* **using** *image-mono * fin1* **by** (*rule Max-mono*) *blast*
 ultimately have $\text{poly-deg } f \leq \text{maxdeg } F$ **by** *simp*
 from $\langle f \in F - \{0\} \rangle$ **have** $f \in F$ **and** $f \neq 0$ **by** *simp-all*
 from *this*(1) *assms*(2) **have** $f \in P[X]$..
 have $f\text{-max}$: $\text{poly-deg } f' \leq \text{poly-deg } f$ **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 $\text{poly-deg } f' \in \text{poly-deg } (F - \{0\})$ **by** (*rule imageI*)
 with fin show $\text{poly-deg } f' \leq \text{poly-deg } f$ **unfolding** *md1[symmetric]* *maxdeg-def*
by (*rule Max-ge*)
 qed
 have $\text{maxdeg } F \leq \text{poly-deg } f$ **unfolding** *maxdeg-def* **using** *fin1* $\langle \text{poly-deg } F \neq \{\} \rangle$
proof (*rule Max.boundedI*)
 fix d
 assume $d \in \text{poly-deg } F$
 then obtain f' **where** $f' \in F$ **and** $d = \text{poly-deg } f'$..
 note *this*(2)
 also from $\langle f' \in F \rangle$ **have** $\text{poly-deg } f' \leq \text{poly-deg } f$ **by** (*rule f-max*)
 finally show $d \leq \text{poly-deg } f$.
qed
with $\langle \text{poly-deg } f \leq \text{maxdeg } F \rangle$ **have** *md*: $\text{poly-deg } f = \text{maxdeg } F$ **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 ...  $\leq$  poly-deg f
  by (simp add: punit.monic-def)
  also have ... = maxdeg F by (fact md)
  also have ...  $\leq$  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  $\langle f \neq 0 \rangle$  have  $c \neq 0$  by (simp add: monomial-0-iff)
    from  $\langle f \in F \rangle$  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  $\langle c \neq 0 \rangle$  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  $\leq$  1  $\vee$   $1 <$  card X by auto
    thus ?thesis
    proof
      note fin-X
      moreover assume card X  $\leq$  1
      moreover note assms(2)
      moreover from  $\langle f \in F \rangle$  have  $f \in$  ideal F by (rule ideal.span-base)
      ultimately have poly-deg g  $\leq$  poly-deg f
      using  $\langle f \neq 0 \rangle$  assms(4) by (rule deg-reduced-GB-univariate-le)
    also have ...  $\leq$  Dube (card X) (maxdeg F) unfolding md by (fact Dube-ge-d)
    finally show ?thesis .
  next
    assume  $1 <$  card X

```

```

    hence  $\text{poly-deg } g \leq \text{Dube-aux } (\text{card } X) (\text{poly-deg } f) 1$ 
    using  $\text{assms}(1, 2) \langle f \in F \rangle \text{assms}(3) f\text{-max } \langle 0 < \text{poly-deg } f \rangle \langle \text{ideal } \{f\} \neq$ 
 $\text{ideal } F \rangle \text{assms}(4)$ 
    by (rule Dube-aux)
    also from  $\langle 1 < \text{card } X \rangle \langle 0 < \text{poly-deg } f \rangle$  have  $\dots = \text{Dube } (\text{card } X) (\text{maxdeg}$ 
 $F)$ 
    by (simp add: Dube-def md)
    finally show ?thesis .
  qed
qed
qed
qed

```

corollary *Dube-is-hom-GB-bound:*

```

  finite  $F \implies F \subseteq P[X] \implies \text{is-hom-GB-bound } F (\text{Dube } (\text{card } X) (\text{maxdeg } F))$ 
  by (intro is-hom-GB-boundI Dube)

```

end

corollary *Dube-indets:*

```

  assumes finite  $F$  and  $\bigwedge f. f \in F \implies \text{homogeneous } f$  and  $g \in \text{punit.reduced-GB}$ 
 $F$ 
  shows  $\text{poly-deg } g \leq \text{Dube } (\text{card } (\bigcup (\text{indets } 'F))) (\text{maxdeg } F)$ 
  using -  $\text{assms}(1)$  -  $\text{assms}(2, 3)$ 
proof (rule Dube)
  from  $\text{assms}$  show finite  $(\bigcup (\text{indets } 'F))$  by (simp add: finite-indets)
next
  show  $F \subseteq P[\bigcup (\text{indets } 'F)]$  by (auto simp: Polys-alt)
qed

```

corollary *Dube-is-hom-GB-bound-indets:*

```

  finite  $F \implies \text{is-hom-GB-bound } F (\text{Dube } (\text{card } (\bigcup (\text{indets } 'F))) (\text{maxdeg } F))$ 
  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  $F \subseteq P[X]$ 
  shows is-GB-cofactor-bound  $F (\text{Dube } (\text{Suc } (\text{card } X)) (\text{maxdeg } F))$ 
  using  $\text{assms}(1, 3)$ 
proof (rule hom-GB-bound-is-GB-cofactor-bound)
  let ? $F$  = homogenize None 'extend-indets'  $F$ 
  let ? $X$  = insert None (Some 'X')
  from  $\text{assms}(1)$  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** $f': f' = \text{homogenize None } (\text{extend-indets } f)$
by *blast*
 from *this(1) assms(3)* **have** $f \in P[X]$ **..**
 hence *extend-indets* $f \in P[\text{Some } 'X]$ **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 $\text{maxdeg } ?F = \text{maxdeg } F$
proof –
 have $\text{maxdeg } ?F = \text{maxdeg } (\text{extend-indets } 'F)$
 by (*auto simp: indets-extend-indets intro: maxdeg-homogenize*)
 also have $\dots = \text{maxdeg } F$ **by** (*simp add: maxdeg-def image-image*)
 finally show $\text{maxdeg } ?F = \text{maxdeg } F$.
qed
moreover from *assms(1)* **have** $\text{card } ?X = \text{card } X + 1$ **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* $G = \text{ideal } F$ **and** $G \subseteq P[X]$
 and $\bigwedge g. g \in G \implies \exists q. g = (\sum_{f \in F} q f * f) \wedge$
 $(\forall f. q f \in P[X] \wedge \text{poly-deg } (q f * f) \leq \text{Dube } (\text{Suc } (\text{card } X))$
 $(\text{maxdeg } F) \wedge$
 $(f \notin F \longrightarrow q f = 0))$

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* $G = \text{ideal } F$
and $G \subseteq P[X]$
 and $1: \bigwedge g. g \in G \implies \exists F' q. \text{finite } F' \wedge F' \subseteq F \wedge g = (\sum_{f \in F'} q f * f) \wedge$
 $(\forall f. q f \in P[X] \wedge \text{poly-deg } (q f * f) \leq ?b \wedge (f \notin F' \longrightarrow q$
 $f = 0))$

by (*rule is-GB-cofactor-boundE-Polys*) *blast*

from *this(1-3)* **show** *?thesis*

proof

fix g

assume $g \in G$

hence $\exists F' q. \text{finite } F' \wedge F' \subseteq F \wedge g = (\sum_{f \in F'} q f * f) \wedge$

```

      (∀f. q f ∈ P[X] ∧ poly-deg (q f * f) ≤ ?b ∧ (f ∉ F' → q
f = 0))
    by (rule 1)
    then obtain F' q where F' ⊆ F and g: g = (∑f∈F'. q f * f) and ∧f. q f
∈ P[X]
      and ∧f. poly-deg (q f * f) ≤ ?b and 2: ∧f. f ∉ F' ⇒ q f = 0 by blast
    show ∃ q. g = (∑f∈F. q f * f) ∧ (∀f. q f ∈ P[X] ∧ poly-deg (q f * f) ≤ ?b
∧ (f ∉ F → q f = 0))
    proof (intro exI allI conjI impI)
      from assms(2) ⟨F' ⊆ F⟩ have (∑f∈F'. q f * f) = (∑f∈F. q f * f)
    proof (intro sum.mono-neutral-left ballI)
      fix f
      assume f ∈ F - F'
      hence f ∉ F' by simp
      hence q f = 0 by (rule 2)
      thus q f * f = 0 by simp
    qed
    thus g = (∑f∈F. q f * f) by (simp only: g)
  next
    fix f
    assume f ∉ F
    with ⟨F' ⊆ F⟩ have f ∉ 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 (⋃(indets ' F)))) (maxdeg F))
  using - assms -
proof (rule Dube-is-GB-cofactor-bound)
  from assms show finite (⋃(indets ' F)) by (simp add: finite-indets)
next
  show F ⊆ P[⋃(indets ' 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 $1 \in \text{ideal}(\text{set } fs) \longleftrightarrow 1 \in \text{set}(\text{punit.Macaulay-list}(\text{deg-shifts } X(\text{Dube}(\text{Suc}(\text{card } X))(\text{maxdeg}(\text{set } fs))) \text{ } fs))$

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

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

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

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

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* :: $('a \Rightarrow 'b) \Rightarrow 'a \text{ list} \Rightarrow 'b \text{ list} \Rightarrow 'a \text{ list}$ **where**
remdups-wrt-rev f [] vs = [] |

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

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

proof (induct xs arbitrary: vs)

case Nil

show ?case by simp

next

case (Cons x xs)

from Cons(2) have 1: $v \notin f \text{ ' set (remdups-wrt-rev f xs vs) }$ by (rule Cons(1))

from Cons(2) have $v \in \text{set (f x # vs) }$ by simp

hence 2: $v \notin f \text{ ' set (remdups-wrt-rev f xs (f x # vs)) }$ by (rule Cons(1))

from Cons(2) show ?case by (auto simp: Let-def 1 2)

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 ($\lambda x. \text{fst } x \notin \text{set } vs$) 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 Cons, intro impI)

assume $k \neq \text{fst } x$

have map-of (filter ($\lambda y. \text{fst } y \neq \text{fst } x \wedge \text{fst } y \notin \text{set } vs$) xs) =

map-of (filter ($\lambda y. \text{fst } y \neq \text{fst } x$) (filter ($\lambda y. \text{fst } y \notin \text{set } vs$) xs))

by (simp only: filter-filter conj-commute)

also have ... = map-of (filter ($\lambda y. \text{fst } y \notin \text{set } vs$) xs) |' {y. y \neq fst x} by (rule map-of-filter)

finally show map-of (filter ($\lambda y. \text{fst } y \neq \text{fst } x \wedge \text{fst } y \notin \text{set } vs$) xs) k =

map-of (filter ($\lambda y. \text{fst } y \notin \text{set } vs$) 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 *compute-list-to-poly* [code]:
 $list\text{-}to\text{-}poly\ ts\ cs = distr_0\ DRLEX\ (remdups\text{-}wrt\text{-}rev\ fst\ (zip\ ts\ cs)\ [])$
by (rule *poly-mapping-eqI*,
simp add: *pprod.lookup-list-to-poly list-to-fun-def distr₀-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*:
 $Macaulay\text{-}list\ ps =$
 $(let\ ts = Keys\text{-}to\text{-}list\ ps\ in$
 $filter\ (\lambda p.\ p \neq 0)\ (mat\text{-}to\text{-}polys\ ts\ (row\text{-}echelon\ (polys\text{-}to\text{-}mat\ ts\ ps)))$
 $)$
by (simp add: *Macaulay-list-def Macaulay-mat-def Let-def*)

declare *conversep-iff* [code]

12.2.1 Connection between $(x \Rightarrow_0 a) \Rightarrow_0 b$ and $(x, a) pp \Rightarrow_0 b$

definition *keys-pp-to-list* :: $(x::linorder, a::zero)\ pp \Rightarrow x\ list$
where *keys-pp-to-list* $t = sorted\text{-}list\text{-}of\text{-}set\ (keys\text{-}pp\ t)$

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]:
 $mapping\text{-}of \circ PP = (\lambda x.\ x)$
 $PP \circ mapping\text{-}of = (\lambda x.\ x)$
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*: $\text{keys } (\text{Poly-Mapping.map-key } PP \ p) = \text{mapping-of } \text{'keys } p$
by (*simp add: keys-map-key inj-PP*)
(smt Collect-cong PP-inverse UNIV-I image-def pp.mapping-of-inverse vim-age-def)

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

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

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

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

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

lemma *map-key-PP-monom-mult-punit*:
 $\text{Poly-Mapping.map-key } PP \ (\text{monom-mult-punit } c \ t \ p) = \text{monom-mult-punit } c \ (\text{mapping-of } t) \ (\text{Poly-Mapping.map-key } PP \ p)$
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*:
 $\text{Poly-Mapping.map-key } PP \ (p * q) = \text{Poly-Mapping.map-key } PP \ p * \text{Poly-Mapping.map-key } PP \ (q :: (-, - :: \text{add-linorder}))$
 $pp \Rightarrow_0 -)$
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*: $\text{Poly-Mapping.map-key } PP \ (\text{sum } f \ A) = (\sum a \in A. \text{Poly-Mapping.map-key } PP \ (f \ a))$

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 $\lambda s t. \text{ord } (PP\ s) (PP\ t) \lambda s t. \text{ord-strict } (PP\ s) (PP\ t)$*

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 \Rightarrow nat \Rightarrow ('x, nat) pp \Rightarrow_0 nat* **where**

deg-le-sect-pp-aux xs 0 = 1 |

deg-le-sect-pp-aux xs (Suc n) =

(let p = deg-le-sect-pp-aux xs n in p + foldr ($\lambda x. (+)$) (monom-mult-punit 1 (single-pp x 1) p)) xs 0)

definition *deg-le-sect-pp* :: *'x list \Rightarrow nat \Rightarrow ('x, nat) pp list*

where *deg-le-sect-pp xs d = punit.keys-to-list (deg-le-sect-pp-aux xs d)*

definition *deg-shifts-pp* :: 'x list \Rightarrow nat \Rightarrow
 ((('x, nat) pp \Rightarrow_0 'b) list \Rightarrow ((('x, nat) pp \Rightarrow_0 'b::semiring-1)
 list
 where *deg-shifts-pp* xs d fs = concat (map (λf . (map (λt . 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) \Rightarrow '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 \Rightarrow '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 $PP\ t \notin PP\ ' T$ by (simp add: *PP-inject im-*
age-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 ≠ 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 ∈ keys (deg-le-sect-pp-aux xs n)
 shows deg-pp t ≤ n and keys-pp t ⊆ set xs
proof –
 from assms have deg-pp t ≤ n ∧ keys-pp t ⊆ 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.premis
 also have keys (deg-le-sect-pp-aux xs (Suc n)) ⊆ keys q ∪
 keys (foldr (λx. (+) (monom-mult-punit 1 (single-pp x 1) q)) xs 0)

```

    (is -  $\subseteq$  -  $\cup$  keys (foldr ?r xs 0)) by (simp add: Let-def Poly-Mapping.keys-add
flip: q-def)
  finally show ?case
proof
  assume  $t \in \text{keys } q$ 
  hence  $\text{deg-pp } t \leq n \wedge \text{keys-pp } t \subseteq \text{set } xs$  unfolding q-def by (rule Suc.hyps)
  thus ?thesis by simp
next
  assume  $t \in \text{keys } (\text{foldr } ?r \text{ xs } 0)$ 
  moreover have  $\text{set } xs \subseteq X$  by (simp add: X-def)
  ultimately have  $\text{deg-pp } t \leq \text{Suc } n \wedge \text{keys-pp } t \subseteq X$ 
proof (induct xs arbitrary: t)
  case Nil
  thus ?case by simp
next
  case (Cons x xs)
  from Cons.premis(2) have  $x \in X$  and  $\text{set } xs \subseteq X$  by simp-all
  note Cons.premis(1)
  also have  $\text{keys } (\text{foldr } ?r (x \# xs) 0) \subseteq \text{keys } (?r \ x \ 0) \cup \text{keys } (\text{foldr } ?r \text{ xs } 0)$ 
    by (simp add: Poly-Mapping.keys-add)
  finally show ?case
proof
  assume  $t \in \text{keys } (?r \ x \ 0)$ 
  also have  $\dots = (+) (\text{single-pp } x \ 1) \text{ 'keys } q$ 
    by (simp add: monom-mult-punit-def punit.keys-monom-mult)
  finally obtain s where  $s \in \text{keys } q$  and  $t: t = \text{single-pp } x \ 1 + s ..$ 
  from this(1) have  $\text{deg-pp } s \leq n \wedge \text{keys-pp } s \subseteq X$  by (rule 1)
  with  $\langle x \in X \rangle$  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  $t \in \text{keys } (\text{foldr } ?r \text{ xs } 0)$ 
  thus  $\text{deg-pp } t \leq \text{Suc } n \wedge \text{keys-pp } t \subseteq X$  using  $\langle \text{set } xs \subseteq X \rangle$  by (rule
Cons.hyps)
qed
qed
thus ?thesis by (simp only: X-def)
qed
qed
thus  $\text{deg-pp } t \leq n$  and  $\text{keys-pp } t \subseteq \text{set } xs$  by simp-all
qed

lemma deg-le-sect-pp-aux-2:
  assumes  $\text{deg-pp } t \leq n$  and  $\text{keys-pp } t \subseteq \text{set } xs$ 
  shows  $t \in \text{keys } (\text{deg-le-sect-pp-aux } xs \ n)$ 
  using assms
proof (induct n arbitrary: t)
  case 0
  thus ?case by simp

```



```

next
  case (Suc n)
  have foldr: foldr ( $\lambda x. (+) (f x)$ )  $ys\ 0 + y = \text{foldr } (\lambda x. (+) (f x))\ ys\ y$ 
    for  $f\ ys$  and  $y::'z::\text{monoid-add}$  by (induct  $ys$ ) (simp-all add: ac-simps)
  define  $q$  where  $q = \text{deg-le-sect-pp-aux } xs\ n$ 
  from Suc.premis(1) have  $\text{deg-pp } t \leq n \vee \text{deg-pp } t = \text{Suc } n$  by auto
  thus ?case
proof
  assume  $\text{deg-pp } t \leq n$ 
  hence  $t \in \text{keys } q$  unfolding  $q\text{-def}$  using Suc.premis(2) by (rule Suc.hyps)
  hence  $0 < \text{lookup } q\ t$  by (simp add: in-keys-iff)
  also have  $\dots \leq \text{lookup } (\text{deg-le-sect-pp-aux } xs\ (\text{Suc } n))\ t$ 
    by (simp add: Let-def lookup-add flip:  $q\text{-def}$ )
  finally show ?thesis by (simp add: in-keys-iff)
next
  assume eq:  $\text{deg-pp } t = \text{Suc } n$ 
  hence  $\text{keys-pp } t \neq \{\}$  by (auto simp: keys-pp.rep-eq deg-pp.rep-eq)
  then obtain  $x$  where  $x \in \text{keys-pp } t$  by blast
  with Suc.premis(2) have  $x \in \text{set } xs$  ..
  then obtain  $xs1\ xs2$  where  $xs: xs = xs1 @ x \# xs2$  by (meson split-list)
  define  $s$  where  $s = t - \text{single-pp } x\ 1$ 
  from  $\langle x \in \text{keys-pp } t \rangle$  have  $\text{single-pp } x\ 1 \text{ adds } t$ 
    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  $s + \text{single-pp } x\ 1 = (t + \text{single-pp } x\ 1) - \text{single-pp } x\ 1$ 
    unfolding  $s\text{-def}$  by (rule minus-plus)
  hence  $t: t = \text{single-pp } x\ 1 + s$  by (simp add: add.commute)
  with eq have  $\text{deg-pp } s \leq n$  by (simp add: deg-pp-plus deg-pp-single)
  moreover have  $\text{keys-pp } s \subseteq \text{set } xs$ 
  proof (rule subset-trans)
    from Suc.premis(2)  $\langle x \in \text{set } xs \rangle$  show  $\text{keys-pp } t \cup \text{keys-pp } (\text{single-pp } x\ (\text{Suc } 0)) \subseteq \text{set } xs$ 
      by (simp add: keys-pp.rep-eq single-pp.rep-eq)
    qed (simp add:  $s\text{-def}$  keys-pp.rep-eq minus-pp.rep-eq keys-diff)
    ultimately have  $s \in \text{keys } q$  unfolding  $q\text{-def}$  by (rule Suc.hyps)
    hence  $t \in \text{keys } (\text{monom-mult-punit } 1\ (\text{single-pp } x\ 1)\ q)$ 
      by (simp add: monom-mult-punit-def punit.keys-monom-mult  $t$ )
    hence  $0 < \text{lookup } (\text{monom-mult-punit } 1\ (\text{single-pp } x\ 1)\ q)\ t$  by (simp add:
in-keys-iff)
    also have  $\dots \leq \text{lookup } (q + (\text{foldr } (\lambda x. (+) (\text{monom-mult-punit } 1\ (\text{single-pp } x\ 1)\ q))\ xs1\ 0 +
      (\text{monom-mult-punit } 1\ (\text{single-pp } x\ 1)\ q +
        \text{foldr } (\lambda x. (+) (\text{monom-mult-punit } 1\ (\text{single-pp } x\ 1)\ q))\ xs2\ 0)))\ t$ 
      by (simp add: lookup-add)
    also have  $\dots = \text{lookup } (\text{deg-le-sect-pp-aux } xs\ (\text{Suc } n))\ t$ 
      by (simp add: Let-def foldr flip:  $q\text{-def}$ , 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-pp 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 $\dots = \text{ideal } (\text{set } fs)$ 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: map-
 ping-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 ∉ set ts
proof
 assume PP t ∈ set ts
 hence mapping-of (PP t) ∈ 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 ∈ 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 $t \in \text{keys } p$ and $s : s = \text{mapping-of } t \text{ unfolding keys-map-key-PP}$
..
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:

$\text{finite } G \implies \text{pp-pm.punit.is-Groebner-basis } (\text{Poly-Mapping.map-key PP } 'G) \longleftrightarrow$
 $\text{punit.is-Groebner-basis } G$
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 $\text{pp-pm.is-GB-cofactor-bound } (\text{Poly-Mapping.map-key PP } ' \text{set } fs) \text{ } b$
shows $\text{punit.is-Groebner-basis } (\text{set } (\text{punit.Macaulay-list } (\text{deg-shifts-pp } (\text{Indets-pp } fs) \text{ } b \text{ } fs)))$
proof –
let $?fs = \text{map } (\text{Poly-Mapping.map-key PP}) \text{ } fs$
from assms **have** $\text{pp-pm.is-GB-cofactor-bound } (\text{set } ?fs) \text{ } b$ **by** simp
hence $\text{pp-pm.punit.is-Groebner-basis}$
 $(\text{set } (\text{pp-pm.punit.Macaulay-list } (\text{pp-pm.deg-shifts } (\bigcup (\text{indets } ' \text{set } ?fs)) \text{ } b \text{ } ?fs)))$
by $(\text{rule pp-pm.thm-2-3-6-indets})$
also have $(\bigcup (\text{indets } ' \text{set } ?fs)) = \text{set } (\text{Indets-pp } fs)$ **by** $(\text{simp add: Indets-pp-def set-indets-pp})$
finally show $?thesis$
by $(\text{simp add: deg-shifts-deg-shifts-pp map-key-PP-Macaulay-list flip: set-map is-GB-map-key-PP})$
qed

lemma Dube-is-GB-cofactor-bound-pp:

$\text{pp-pm.is-GB-cofactor-bound } (\text{Poly-Mapping.map-key PP } ' \text{set } fs)$
 $(\text{Dube } (\text{Suc } (\text{length } (\text{Indets-pp } fs))) (\text{max-list } (\text{map poly-deg-pp } fs)))$
proof $(\text{cases } fs = [])$
case True
show $?thesis$ **by** $(\text{rule pp-pm.is-GB-cofactor-boundI-subset-zero}) (\text{simp add: True})$
next
case False
let $?F = \text{Poly-Mapping.map-key PP } ' \text{set } fs$
have $\text{pp-pm.is-GB-cofactor-bound } ?F (\text{Dube } (\text{Suc } (\text{card } (\bigcup (\text{indets } ' ?F)))) (\text{maxdeg } ?F))$
by $(\text{intro pp-pm.Dube-is-GB-cofactor-bound-indets finite-imageI finite-set})$
moreover have $\text{card } (\bigcup (\text{indets } ' ?F)) = \text{length } (\text{Indets-pp } fs)$
by $(\text{simp add: Indets-pp-def length-remdups-card-conv set-indets-pp})$
moreover from False **have** $\text{maxdeg } ?F = \text{max-list } (\text{map poly-deg-pp } fs)$
by $(\text{simp add: max-list-Max maxdeg-def image-image poly-deg-map-key-PP})$
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 = ($\lambda -. ()$)
and punit.monom-mult = monom-mult-punit
and punit.mult-scalar = mult-scalar-punit
and punit'.punit.min-term = min-term-punit
and punit'.punit.lt = 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 *trivariate₀-rat* .

lemma

```

  comp-red-basis-punit DRLEX (GB-Macaulay-Dube-punit DRLEX [X * Y2 + 3
  * X2 * Y, Y3 - X3]) =
  [X5, X3 * Y - C0 (1 / 9) * X4, Y3 - X3, X * Y2 + 3 * X2
  * Y]
  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. Maletsky. Formalization of Dubé’s Degree Bounds for Gröbner Bases in Isabelle/HOL. In C. Kaliszyk, E. Brady, A. Kohlhasse, 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.