Gröbner Bases, Macaulay Matrices and Dubé’s Degree Bounds

Alexander Maletzky

June 17, 2019

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 Sums 29
  5.2 Inequalities 29
  5.3 Backward Difference Operator 33

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

6.1 Definition and Basic Properties 35
6.2 Closure Properties 36

# Monomial Modules

7.1 Sets of Monomials 40
7.2 Modules 40
7.3 Reduction 42
7.4 Gröbner Bases 45

# Preliminaries

8.1 Sets 52
8.2 Sums 54
8.3 count-list 54
8.4 listset 55

# Direct Decompositions and Hilbert Functions

9.1 Direct Decompositions 60
9.2 Direct Decompositions and Vector Spaces 73
9.3 Homogeneous Sets of Polynomials with Fixed Degree 79
9.4 Interpreting Polynomial Rings as Vector Spaces over the Coefficient Field 84
9.5 (Projective) Hilbert Function 86

# Cone Decompositions

10.1 More Properties of Reduced Gröbner Bases 92
10.2 Quotient Ideals 92
10.3 Direct Decompositions of Polynomial Rings 93
10.4 Basic Cone Decompositions 100
10.5 Splitting w.r.t. Ideals 126
10.6 Function split 137
10.7 Splitting Ideals 158
10.8 Exact Cone Decompositions 163
10.9 Functions shift and exact 178

# Dubé’s Degree-Bound for Homogeneous Gröbner Bases

11.1 Hilbert Function and Hilbert Polynomial 203
11.2 Dubé’s Bound 213

# Sample Computations of Gröbner Bases via Macaulay Matrices

12.2 Preparations 241
12.2.1 Connection between \((x \Rightarrow_0 a) \Rightarrow_0 b\) and \((x, 'a) pp \Rightarrow_0 'b\) 242
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>12.2.2 Locale \textit{pp-powerprod}</td>
<td>244</td>
</tr>
<tr>
<td>12.3 Computations</td>
<td>254</td>
</tr>
</tbody>
</table>
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 ⇒ nat ⇒ ('x::countable ⇒0 nat) set
  where deg-sect X d = .[X] ∩ {t. deg-pm t = d}

definition deg-le-sect :: 'x set ⇒ nat ⇒ ('x::countable ⇒0 nat) set
  where deg-le-sect X d = (⋃ d0≤d. deg-sect X d0)

lemma deg-sectI: t ∈ .[X] ⇒ deg-pm t = d ⇒ t ∈ deg-sect X d
  by (simp add: deg-sect-def)

lemma deg-sectD:
  assumes t ∈ deg-sect X d
  shows t ∈ .[X] and deg-pm t = d
  using assms by (simp-all add: deg-sect-def)

lemma deg-le-sect-alt: deg-le-sect X d = .[X] ∩ {t. deg-pm t ≤ d}
  by (auto simp: deg-le-sect-def deg-sect-def)

lemma deg-le-sectI: t ∈ .[X] ⇒ deg-pm t ≤ d ⇒ t ∈ deg-le-sect X d
by (simp add: deg-le-sect-alt)

lemma deg-le-sectD:
  assumes t ∈ deg-le-sect X d
  shows t ∈ [X] and deg-pm t ≤ d
  using assms by (simp-all add: deg-le-sect-alt)

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

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

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

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

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

lemma deg-sect-mono: X ⊆ Y =⇒ deg-sect X d ⊆ deg-sect Y d
  by (auto simp: deg-sect-def dest: PPs-mono)

lemma deg-le-sect-mono-1: X ⊆ Y =⇒ deg-le-sect X d ⊆ deg-le-sect Y d
  by (auto simp: deg-le-sect-def dest: PPs-mono)

lemma deg-le-sect-mono-2: d1 ≤ d2 =⇒ deg-le-sect X d1 ⊆ deg-le-sect X d2
  by (auto simp: deg-le-sect-alt)

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

lemma deg-sect-disjoint: d1 ≠ d2 =⇒ deg-sect X d1 ∩ deg-sect Y d2 = {}
  by (auto simp: deg-sect-def)

lemma deg-le-sect-deg-sect-disjoint: d1 < d2 =⇒ deg-le-sect X d1 ∩ deg-sect X d2 = {}
  by (auto simp: deg-sect-def)

lemma deg-sect-Suc:
  deg-sect X (Suc d) = (∪ x∈X. (+) (Poly-Mapping.single x 1) ' deg-sect X d) (is ?A = ?B)
proof (rule set-eqI)
  fix t
show \( t \in \mathcal{A} \iff t \in \mathcal{B} \)

proof

assume \( t \in \mathcal{A} \)

hence \( t \in \mathcal{X} \) and \( \degpm t = \text{Succ } d \) by (rule \( \degsectD \))

from this(2) have keys \( t \neq \{\} \) by auto

then obtain \( x \) where \( x \in \text{keys } t \) by blast

hence \( \{t\} \subseteq \mathcal{X} \) by (rule \( \mathcal{PPsD} \))

with \( x \in \text{keys } t \) have \( x \in \mathcal{X} \)

let \( \mathcal{S} = \text{Poly-Mapping.single } x \) (1:nat)

from \( \{t\} \subseteq \text{lookup } t \) have \( \mathcal{S} \) adds \( t \) by (auto simp add: in-keys-iff)

from \( \{t\} \subseteq \mathcal{X} \) have \( \mathcal{S} \) adds \( t \) by (rule \( \mathcal{PPs-closed-minus} \))

next

assume \( t \in \mathcal{B} \)

then obtain \( x \) where \( x \in \mathcal{X} \) and \( t \in (\mathcal{X} \setminus \mathcal{A}) \) by (simp only: \( \degsect X d \))

qed
proof
  assume \( t \in ?A \)
  hence \( t \in \{ \text{insert } x \ X \} \) and \( \text{deg-pm } t = d \) by (rule \text{deg-sectD})
  from this(1) obtain \( e \ tx \) where \( tx \in \{ x \} \) and \( t = \text{Poly-Mapping.single } x \ e + tx \)
  by (rule \text{PPs-insertE})
  have \( e + \text{deg-pm } tx = \text{deg-pm } t \) by (simp add: \( t \) \text{deg-pm-plus} \( \text{deg-pm-single} \))
  hence \( e + \text{deg-pm } tx = d \) by (simp only: (\( t \) \text{deg-pm } \( t = d \)))
  hence \( \text{deg-pm } tx \in \{ d \} \) and \( e : e = d - \text{deg-pm } tx \) by simp-all
  from \( tx \in \{ x \} \) refl have \( tx \in \text{deg-sect } X \) (\( \text{deg-pm } tx \)) by (rule \text{deg-sectI})
  hence \( t \in (+) \) (\( \text{Poly-Mapping.single } x \ (d - \text{deg-pm } tx) \)) \( t \) \text{deg-sect } X \ (\( \text{deg-pm } tx \))
  unfolding \( t \) \( e \) by (rule \text{imageI})
  with \( \text{deg-pm } tx \in \{ d \} \) show \( t \in ?B \).
next
  assume \( t \in ?B \)
  then obtain \( d0 \) where \( d0 \in \{ d \} \) and \( t \in (+) \) (\( \text{Poly-Mapping.single } x \ (d - d0) \)) \( t \) \text{deg-sect } X \ (\( d0 \)).
  from this(2) obtain \( s \) where \( s : s \in \text{deg-sect } X \ d0 \)
  and \( t : t = \text{Poly-Mapping.single } x \ (d - d0) + s \) (\( \text{is } t = ?s + s \)).
  show \( t \in ?A \)
  proof (rule \text{deg-sectI})
    have ?\( s \) : \( \text{insert } x \ X \) by (rule \text{PPs-closed-single}, simp)
    from \( s \) have \( ?s \in \{ x \} \) by (rule \text{deg-sectD})
    also have \( \cdots \subseteq \{ x \} \) by (rule \text{PPs-mono}, blast)
    finally have \( ?s \in \{ x \} \).
    with \( ?s \) : \( \text{insert } x \ X \) show \( t \in \{ x \} \) unfolding \( t \) by (rule \text{PPs-closed-plus})
next
  from \( s \) have \( \text{deg-pm } s = d0 \) by (rule \text{deg-sectD})
  moreover from \( d0 \in \{ d \} \) have \( d0 \leq d \) by simp
  ultimately show \( \text{deg-pm } t = d \) by (simp add: \( t \) \text{deg-pm-single} \( \text{deg-pm-plus} \))
  qed
  qed
lemma \( \text{deg-le-sect-Suc} \): \( \text{deg-le-sect } X \ (\text{Suc } d) = \text{deg-le-sect } X \ (\text{Suc } d) \cup \text{deg-sect } X \ (\text{Suc } d) \)
  by (simp add: \( \text{deg-le-sect-def atMost-Suc Un-commute} \))
lemma \( \text{deg-le-sect-Suc-2} \):
  \( \text{deg-le-sect } X \ (\text{Suc } d) = \text{insert } 0 \ (\bigcup x \in X. (+) \ (\text{Poly-Mapping.single } x \ 1)) \) \( \text{deg-le-sect } X \ (\text{Suc } d) \)
  (\( \text{is } ?A = ?B \))
proof
  have eq1: \( \text{Suc } 0.\text{Suc } d) = \text{Suc } \{ \text{Suc } d \} \) by (simp add: \text{image-Suc-atMost})
  have insert 0 \( \{ \text{Suc } d \} = \{ \text{Suc } d \} \) by fastforce
  hence ?\( A \in \{ d0 \in \text{insert } 0 \ \{ \text{Suc } d \} \) \text{deg-sect } X \ (\text{Suc } d0) \) by (simp add: \( \text{deg-le-sect-def} \))
  also have \( \cdots = \text{insert } 0 \ (\bigcup d0 \leq d. \text{deg-sect } X \ (\text{Suc } d0)) \) by (simp add: eq1)
also have \( \sum_0 \bigcup_{x \in X} (\text{Poly-Mapping.} \text{single} x 1) \) = \( \sum_0 \bigcup_{d \leq d} \bigcup_{x \in X} (\text{Poly-Mapping.} \text{single} x 1) \)

by (simp only: \text{deg-sect-Suc})
also have \( \sum_0 \bigcup_{x \in X} (\text{Poly-Mapping.} \text{single} x 1) \) = \( \sum_0 \bigcup_{d \leq d} \bigcup_{x \in X} (\text{Poly-Mapping.} \text{single} x 1) \)

by \text{fastforce}
also have \( \sum_0 \bigcup_{x \in X} (\text{Poly-Mapping.} \text{single} x 1) \) = \( \sum_0 \bigcup_{d \leq d} \bigcup_{x \in X} (\text{Poly-Mapping.} \text{single} x 1) \)

by (simp only: \text{deg-le-sect-def})
finally show \( ?\text{thesis} \).
qed

lemma \text{finite-deg-sect}:
assumes finite \( X \)
shows finite \( \bigcup_{d \leq d} \bigcup_{x \in X} (\text{Poly-Mapping.} \text{single} x 1) \)
proof (induct \( d \))
case 0
show \( ?\text{case} \) by simp
next
case (Suc \( d \))
with \( \text{assms} \) show \( ?\text{case} \) by (simp add: \text{deg-sect-Suc})
qed

corollary \text{finite-deg-le-sect}:
finite \( X \) = \( \Rightarrow \) finite \( \bigcup_{d \leq d} \bigcup_{x \in X} (\text{Poly-Mapping.} \text{single} x 1) \)

by (simp add: \text{deg-le-sect-def} \text{finite-deg-sect})

lemma \text{keys-subset-deg-le-sectI}:
assumes \( p \in P[X] \) and \( \text{poly-deg} p \leq d \)
shows keys \( p \subseteq \bigcup_{d \leq d} \bigcup_{x \in X} (\text{Poly-Mapping.} \text{single} x 1) \)
proof
fix \( t \)
assume \( t \in \text{keys} p \)
also from \( \text{assms(1)} \) have \( \subseteq .[X] \) by (rule \text{PolysD})
finally have \( t \in .[X] \).
from \( t \in \text{keys} p \) have \( \text{deg-pm} t \leq \text{poly-deg} p \) by (rule \text{poly-deg-max-keys})
from this \( \text{assms(2)} \) have \( \text{deg-pm} t \leq d \) by (rule \text{le-trans})
with \( t \in .[X] \) show \( t \in \bigcup_{d \leq d} \bigcup_{x \in X} (\text{Poly-Mapping.} \text{single} x 1) \)
qed

lemma \text{binomial-symmetric-plus}:
\( \binom{n+k}{n} \) = \( \binom{n+k}{k} \)

by (metis \text{add-diff-cancel-left} \text{binomial-symmetric} \text{le-add1})

lemma \text{card-deg-sect}:
assumes finite \( X \) and \( X \neq {} \)
shows \( \text{card} (\bigcup_{d \leq d} \bigcup_{x \in X} (\text{Poly-Mapping.} \text{single} x 1)) \) = \( \text{card} X - 1 \)
using \( \text{assms} \)
proof (induct \( X \) arbitrary: \( d \))
case empty
thus \( ?\text{case} \) by simp
next

8
case (insert x X)
from insert(1, 2) have eq1: card (insert x X) = Suc (card X) by simp
show ?case
proof (cases X = { })
  case True
  thus ?thesis by simp
next
  case False
  with insert.hyps(1) have 0 < card X by (simp add: card-gt-0-iff)
  let ?f = λd0. Poly-Mapping.single x (d - d0)
  from False have eq2: card (deg-sect X d0) = d0 + (card X - 1) choose (card X - 1) for d0
    by (rule insert.hyps)
  have finite {...} by simp
  moreover from insert.hyps(1) have ∀ d0∈{..d}. finite ((+) (?f d0) · deg-sect X d0)
    by (simp add: finite-deg-sect)
  moreover have ∀ d0∈{..d}. ∀ d1∈{..d}. d0 ≠ d1 → ((+) (?f d0) · deg-sect X d0) ∩ ((+) (?f d1) · deg-sect X d1) = {}
    proof (intro ballI impI, rule ccontr)
      fix d1 d2 :: nat
      assume d1 ≠ d2
      assume ((+) (?f d1) · deg-sect X d1) ∩ ((+) (?f d2) · deg-sect X d2) ≠ {}
      then obtain t where t ∈ ((+) (?f d1) · deg-sect X d1) ∩ ((+) (?f d2) · deg-sect X d2)
        by blast
      hence t1: t ∈ (+) (?f d1) · deg-sect X d1 and t2: t ∈ (+) (?f d2) · deg-sect X d2 by simp-all
      from t1 obtain s1 where s1 ∈ deg-sect X d1 and s1: t = ?f d1 + s1 ..
      from this(1) have s1 ∈ [X] by (rule deg-sectD)
      hence keys s1 ⊆ X by (rule PPsD)
      with insert.hyps(2) have eq3: lookup s1 x = 0 by (auto simp: in-keys-iff)
      from t2 obtain s2 where s2 ∈ deg-sect X d2 and s2: t = ?f d2 + s2 ..
      from this(1) have s2 ∈ [X] by (rule deg-sectD)
      hence keys s2 ⊆ X by (rule PPsD)
      with insert.hyps(2) have eq4: lookup s2 x = 0 by (auto simp: in-keys-iff)
      from s2 have lookup (?f d1 + s1) x = lookup (?f d2 + s2) x by (simp only: s1)
      hence d - d1 = d - d2 by (simp add: lookup-add eq3 eq4)
      moreover assume d1 ∈ {..d} and d2 ∈ {..d}
      ultimately have d1 = d2 by simp
      with ⟨d1 ≠ d2⟩ show False ..
    qed
    ultimately have card (deg-sect (insert x X) d) = (∑ d0≤d. card ((+) (monomial (d - d0) x) · deg-sect X d0))
      unfolding deg-sect-insert by (rule card-UN-disjoint)
    also from refl have ... = (∑ d0≤d. card (deg-sect X d0))
    proof (rule sum.cong)
fix \( d_0 \)

have inj-on \((+ (\text{monomial} (d - d_0) x)) (\text{deg-sect} X d_0)\) by (rule, rule add-left-imp-eq)

thus \( \text{card} ((+ (\text{monomial} (d - d_0) x) \cdot \text{deg-sect} X d_0)) = \text{card} (\text{deg-sect} X d_0) \)

by (rule card-image)

qed

also have ... = \( \sum_{d_0 \leq d} (\text{card} X - 1) + d_0 \text{ choose } d_0 \) by (simp only: eq2 add-commute)

also have ... = \( \sum_{d_0 \leq d} (\text{card} X - 1) + d_0 \text{ choose } d_0 \) by (simp only: binomial-symmetric-plus)

also have ... = Suc \((\text{card} X - 1) + d) \text{ choose } d \) by (rule sum-choose-lower)

also from \(\emptyset < \text{card} X\) have ... = \( d + (\text{card} (\text{insert} x X) - 1) \text{ choose } d \)

by (simp add: eq1 add-commute)

also have ... = \( d + (\text{card} (\text{insert} x X) - 1) \text{ choose } (\text{card} (\text{insert} x X) - 1) \)

by (fact binomial-symmetric-plus)

finally show \(?\)thesis .

qed

\textbf{corollary card-deg-sect-Suc:}

assumes finite X

shows \( \text{card} (\text{deg-sect} X (\text{Suc} d)) = (d + \text{card} X) \text{ choose } (\text{Suc} d) \)

proof (cases \( X = \{\} \))

case True

thus \(?\)thesis by (simp add: deg-sect-empty)

next

case False

with assms have \(\emptyset < \text{card} X\) by (simp add: card-gt-0-iff)

from assms False have \( \text{card} (\text{deg-sect} X (\text{Suc} d)) = (\text{Suc} d + (\text{card} X - 1)) \text{ choose } (\text{card} X - 1) \)

by (rule card-deg-sect)

also have ... = \((\text{Suc} d + (\text{card} X - 1)) \text{ choose } (\text{Suc} d) \) by (rule sym, rule binomial-symmetric-plus)

also from \(\emptyset < \text{card} X\) have ... = \((d + \text{card} X) \text{ choose } (\text{Suc} d) \) by simp

finally show \(?\)thesis .

qed

\textbf{corollary card-deg-le-sect:}

assumes finite X

shows \( \text{card} (\text{deg-le-sect} X d) = (d + \text{card} X) \text{ choose } \text{card} X \)

proof (induct d)

case \(\emptyset\)

show \(?\)case by simp

next

case \((\text{Suc} d)\)

from assms have finite \((\text{deg-le-sect} X d)\) by (rule finite-deg-le-sect)

moreover from assms have finite \((\text{deg-sect} X (\text{Suc} d))\) by (rule finite-deg-sect)

moreover from \(\text{lessI}\) have \(\text{deg-le-sect} X d \cap \text{deg-sect} X (\text{Suc} d) = \{\} \)
ultimately have \( \text{card} \ (\text{deg-le-sect} \ X \ \text{(Suc} \ d)) = \text{card} \ (\text{deg-le-sect} \ X \ d) + \text{card} \ (\text{deg-sect} \ X \ \text{(Suc} \ d)) \)

unfolding \( \text{deg-le-sect-Suc} \) by (rule Un-disjoint)
also from assms have \( \ldots = \text{(Suc} \ d + \text{card} \ X) \) choose Suc d
also have \( \ldots = \text{(Suc} \ d + \text{card} \ X) \) 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 \:: \(((\uparrow x \Rightarrow \uparrow 0 \nat) \Rightarrow \uparrow 0 \text{::field}) \text{ set} \Rightarrow \nat \Rightarrow \text{bool})
where is-GB-cofactor-bound \( F \ b \) \leftrightarrow
  (\exists G. \ punit.\text{is-Groebner-basis} \ G \land \text{ideal} \ G = \text{ideal} \ F \land \text{UNION} \ G \ \text{indets} \subseteq \text{UNION} \ F \ \text{indets} \land
  (\forall g \in G. \exists F' \ q. \text{finite} \ F' \land F' \subseteq F \land g = (\sum f \in F'. \ q \ f \ast f) \land (\forall f \in F'. \ \text{poly-deg} \ (q \ f \ast f) \leq b)))

definition is-hom-GB-bound \:: \(((\uparrow x \Rightarrow \uparrow 0 \nat) \Rightarrow \uparrow 0 \text{::field}) \text{ set} \Rightarrow \nat \Rightarrow \text{bool})
where is-hom-GB-bound \( F \ b \) \leftrightarrow ((\forall f \in F. \ \text{homogeneous} \ f) \rightarrow (\forall g \in \text{punit.\text{reduced-GB}} \ F. \ \text{poly-deg} \ g \leq b)))

lemma is-GB-cofactor-boundI:
  assumes punit.\text{is-Groebner-basis} \ G \land \text{ideal} \ G = \text{ideal} \ F \land \text{UNION} \ G \ \text{indets} \subseteq \text{UNION} \ F \ \text{indets} \land
  \land (\forall f \in F'. \ \text{poly-deg} \ (q \ f \ast f) \leq b)
shows is-GB-cofactor-bound \( F \ b \)

lemma is-GB-cofactor-boundE:
  fixes \( F \) :: \(((\uparrow x \Rightarrow \uparrow 0 \nat) \Rightarrow \uparrow 0 \text{::field}) \text{ set})
  assumes is-GB-cofactor-bound \( F \ b \)
  obtains \( G \) where punit.\text{is-Groebner-basis} \ G \land \text{ideal} \ G = \text{ideal} \ F \land \text{UNION} \ G \ \text{indets} \subseteq \text{UNION} \ F \ \text{indets} \land
  (\forall f \in F'. \ \text{poly-deg} \ (q \ f \ast f) \leq b)
\( (f \notin F' \implies q \cdot f = 0) \)

proof
- let \(?X = \text{UNION } F \text{ indets}\)
- from assms obtain \(G\) where \text{punit.is-Groebner-basis } G \text{ and ideal } G = \text{ideal } F\) and \(\text{UNION } G \text{ indets } \subseteq ?X\)
  and 1: \( \forall g. g \in G \implies \exists F' q. \text{finite } F' \land F' \subseteq F \land g = (\sum f \in F'. \ q \cdot f) \land
  (\forall f \in F'. \text{ poly-deg } (q \cdot f) \leq b) \)
    by (auto simp: is-GB-cofactor-bound-def)
- from this(1, 2, 3) show \(?\text{thesis}\)
proof
  fix \(g\)
  assume \(g \in G\)
  show \(\exists F' q. \text{finite } F' \land F' \subseteq F \land g = (\sum f \in F'. \ q \cdot f) \land
  (\forall f \in F'. \text{ poly-deg } (q \cdot f) \leq b) \land (f \notin F' \implies q \cdot f = 0) \)
  proof (cases \(g = 0\))
  case \(\text{True}\)
  define \(q\) where \(q = (\lambda f. (\exists : 'x \Rightarrow 0 \cdot \text{nat}) \Rightarrow 0 \cdot 'b :: (\exists : 'x \Rightarrow 0 \cdot \text{nat}) \Rightarrow 0 \cdot 'b)\)
  show \(?\text{thesis}\)
    proof (intro ex1 conjI allI)
    show \(q = (\sum f \in \{\}, q \cdot f)\) by (simp add: True q-def)
    qed (simp-all add: q-def)
  qed
  next
  case \(\text{False}\)
  let \(?X = \text{UNION } F \text{ indets}\)
  from \(g \in G\) have \(\exists F' q. \text{finite } F' \land F' \subseteq F \land g = (\sum f \in F'. \ q \cdot f) \land
  (\forall f \in F'. \text{ poly-deg } (q \cdot f) \leq b) \land
  (\forall f \in F'. \text{poly-deg } (q \cdot f) \leq b) \)
    by (rule 1)
  then obtain \(F' \ q0 \) where \(\text{finite } F' \land F' \subseteq F \) and \(g = (\sum f \in F'. \ q0 \cdot f)\)
  and \(q0 : \forall f. f \in F' \implies \text{poly-deg } (q0 \cdot f) \leq b\) by blast
  define \(\text{sub}\) where \(\text{sub} = (\lambda x. 'x \cdot 'x. \text{if } x \in ?X \text{ then monomial } (1::'b) \langle \text{Poly-Mapping.single } x (1::\text{nat}) \rangle
  \text{ else } 1)\)
  have 1: \(\text{sub } x = \text{monomial } 1 \) (monomial 1 \(x\)) if \(x \in \text{indets } g\) for \(x\)
    proof (simp add: sub-def, rule)
    from that \(g \in G\) have \(x \in \text{UNION } G \text{ indets}\) by blast
    also have \(\ldots \subseteq ?X\) by fact
    finally obtain \(f\) where \(f \in F \) and \(x \in \text{indets } f\) ..
    assume \(\forall f \in F. \ x \notin \text{indets } f\)
    hence \(x \notin \text{indets } f\) using \(f \in F\) ..
    thus \(\text{monomial } 1 \) (monomial \((\text{Suc } 0) \) \(x\)) = \(1\) using \(x \in \text{indets } f\) ..
  qed
  have 2: \(\text{sub } x = \text{monomial } 1 \) (monomial 1 \(x\)) if \(f \in F' \) and \(x \in \text{indets } f\) for \(f x\)
    proof (simp add: sub-def, rule)
    assume \(\forall f \in F. \ x \notin \text{indets } f\)
    moreover from that(1) \(F' \subseteq F\) have \(f \in F\) ..
    ultimately have \(x \notin \text{indets } f\) ..
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))

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 (Suc 0) (monomial (1::nat)) y)

by (simp-all add: sub-def split: if-splits)

from this(2) have x = y by (simp add: indets-monomial)

with y show x ∈ ?X by (simp only:)

qed

next

from f ∈ F' have poly-subst sub f = f by (rule 3)

hence poly-deg (q f * f) = poly-deg (q f * poly-subst sub f) by (simp only:)

also have ... = poly-deg (poly-subst sub (q0 f * f)) by (simp only: qf poly-subst-times)

also have ... ≤ poly-deg (q0 f * f)

proof (rule poly-deg-poly-subst-le)

fix x

show poly-deg (sub x) ≤ 1 by (simp add: sub-def poly-deg-monomial deg-pm-single)
qed
also from \( f \in F' \) have \( \ldots \leq b \) by (rule q0)
finally show \( \text{poly-deg} (q f * f) \leq b \).
qed
next
case False
thus \( \text{?thesis} \) 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

lemma is-GB-cofactor-boundE-Polys:
fixes \( F :: (('x \Rightarrow 0 \text{ nat}) \Rightarrow 0) \) \( b :: \text{field} \) 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 \( \bigwedge g. \ g \in G \implies \exists F'. \ \text{finite} F' \land F' \subseteq F \land g = (\sum f \in F'. q f * f) \land \\
(\forall f. \ q f \in P[X] \land \text{poly-deg} (q f * f) \leq b \land (f \notin F' \implies q f = 0)) \)
proof -
let \( \ ?X = \text{UNION} F \) \( \text{indets} \)
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) \( \text{assms(2)} \) have \( f \in P[X] \).

hence \( \text{indets} f \subseteq X \) by (rule PolysD)
with \( x \in \text{indets} f \), show \( x \in X \).
qed
from \( \text{assms(1)} \) obtain \( G \) where \( \text{punit.is-Groebner-basis} G \) and \( \text{ideal} G = \text{ideal} F \)
and 1: \( \text{UNION} G \) \( \text{indets} \subseteq \ ?X \)
and 2: \( \bigwedge g. \ g \in G \implies \exists F'. \ \text{finite} F' \land F' \subseteq F \land g = (\sum f \in F'. q f * f) \land \\
(\forall f. \ q f \in P[X] \land \text{poly-deg} (q f * f) \leq b \land (f \notin F' \implies q f = 0)) \)
by (rule is-GB-cofactor-boundE)
blast
from this(1, 2) show \( \text{?thesis} \)
proof
show \( G \subseteq P[X] \)
proof
fix \( g \)
assume \( g \in G \)
hence \( \text{indets } g \subseteq \text{UNION G indets} \) by \text{blast}
also have \( \ldots \subseteq ?X \) by \text{fact}
also have \( \ldots \subseteq X \) by \text{fact}
finally show \( g \in P[X] \) by (rule PolysI-alt)
\text{qed}
next
\text{fix } g
\text{assume } g \in G
hence \( \exists F' \ q. \text{finite } F' \wedge F' \subseteq F \wedge g = (\sum f \in F'. \ q \ f \ast f) \wedge (\forall f. \text{indets } (q \ f) \subseteq ?X \wedge \text{poly-deg } (q \ f \ast f) \leq b \wedge (f \notin F' \limp q \ f = 0)) \)
by (rule 2)
then obtain \( F' \ q \) where \( \text{finite } F' \) and \( F' \subseteq F \) and \( g = (\sum f \in F'. \ q \ f \ast f) \wedge (\forall f. \text{indets } (q \ f) \subseteq ?X \wedge \text{poly-deg } (q \ f \ast f) \leq b \wedge (f \notin F' \limp q \ f = 0)) \)
\text{by blast}
\text{show } \exists F' \ q. \text{finite } F' \wedge F' \subseteq F \wedge g = (\sum f \in F'. \ q \ f \ast f) \wedge (\forall f. \text{q } f \in P[X] \wedge \text{poly-deg } (q \ f \ast f) \leq b \wedge (f \notin F' \limp q \ f = 0)) \)
\text{proof (intro exI allI conjI impI)}
\text{fix } f
from \( \langle \text{indets } (q \ f) \subseteq ?X \rangle \) \( (?X \subseteq X) \) have \( \text{indets } (q \ f) \subseteq X \) by (rule subset-trans)
\text{thus } q \ f \in P[X] \text{ by (rule PolysI-alt)}
\text{qed fact+}
\text{qed}
\text{lemma is-GB-cofactor-boundE-finite-Polys:}
\text{fixes } F :: (''x \Rightarrow \text{nat} \Rightarrow \text{0 ''b::field} set
\text{assumes is-GB-cofactor-bound } F b \text{ and finite } F \text{ and } F \subseteq P[X]
\text{obtains } G \text{ where } \text{punit.is-Groebner-basis } G \text{ and } \text{ideal } G = \text{ideal } F \text{ and } G \subseteq P[X]
\text{and } \bigwedge g. \ g \in G \limp \exists q. \ g = (\sum f \in F. \ q \ f \ast f) \wedge (\forall f. \ q \ f \in P[X] \wedge \text{poly-deg } (q \ f \ast f) \leq b)
\text{proof}
\text{from } \text{assms}(1, 3) \text{ obtain } G \text{ where } \text{punit.is-Groebner-basis } G \text{ and } \text{ideal } G = \text{ideal } F \text{ and } G \subseteq P[X]
\text{and } \text{1: } \bigwedge g. \ g \in G \limp \exists F' \ q. \text{finite } F' \wedge F' \subseteq F \wedge g = (\sum f \in F'. \ q \ f \ast f) \wedge (\forall f. \ q \ f \in P[X] \wedge \text{poly-deg } (q \ f \ast f) \leq b \wedge (f \notin F' \limp q \ f = 0)) \)
\text{by (rule is-GB-cofactor-boundE-Polys) blast}
\text{from } \text{this}(1, 2, 3) \text{ show } \text{?thesis}
\text{proof}
\text{fix } g
\text{assume } g \in G
\text{hence } \exists F' \ q. \text{finite } F' \wedge F' \subseteq F \wedge g = (\sum f \in F'. \ q \ f \ast f) \wedge (\forall f. \ q \ f \in P[X] \wedge \text{poly-deg } (q \ f \ast f) \leq b \wedge (f \notin F' \limp q \ f = 0)) \)
\text{by (rule 1)}
then obtain $F' q$ where $F' \subseteq F$ and $q: g = (\sum f \in F', q f \ast f)$
and $\forall f \in P[X]$ and $\forall f. \text{poly-deg} (q f \ast f) \leq b$ and $2: \forall f. f \notin F' \implies q f = 0$ by blast

show $\exists g. g = (\sum f \in F. q f \ast f) \land (\forall f. q f \in P[X] \land \text{poly-deg} (q f \ast f) \leq b)$

proof
(intro exI conjI impI allI)

from assms(2) $F' \subseteq F$ have $(\sum f \in F'. q f \ast f) = (\sum f \in F. q f \ast f)$

fix f
assume f $\in F - F'$
hence f $\notin F'$ by simp
hence q f = 0 by (rule 2)
thus q f \ast f = 0 by simp
qed
thus g = $(\sum f \in F. q f \ast f)$ by (simp only: g)
qed fact+

lemma is-GB-cofactor-boundI-subset-zero:
assumes F $\subseteq \{0\}$
shows is-GB-cofactor-bound F b
using punit.is-Groebner-basis-empty

proof (rule is-GB-cofactor-boundI)

from assms show ideal {} = ideal F by (metis ideal.span-empty ideal-eq-zero-iff)

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

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

lemma is-hom-GB-boundD:
is-hom-GB-bound F b $\implies (\forall f. f \in F \implies \text{homogeneous} f) \implies g \in \text{punit.reduced-GB}$
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 None ' extend-indets ' F) b
shows is-GB-cofactor-bound F b

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

proof
fix $x$

assume $x \in Y$

then obtain $f$ where $f \in F$ and $x \in \text{indets } f$ unfolding $Y\text{-def}$

from this(1) assms(2) have $f \in P[X]$.

hence $\text{indets } f \subseteq X$ by (rule PolysD)

with $x \in \text{indets } f$; show $x \in X$.

qed

hence $\text{finite } Y$ using assms(1) by (rule finite-subset)

moreover have $F \subseteq P[Y]$ by (auto simp: $Y$-def Polys-alt)

ultimately have $\text{punit} \cdot \text{is-Groebner-basis } G$ and $\text{ideal } G = \text{ideal } F$ and $G \subseteq P[Y]$

unfolding $G\text{-def}$ by (rule restrict-indets-reduced-GB)

from this(1, 2) show ?thesis

proof (rule is-GB-cofactor-boundI)

from $\langle g \subseteq P[Y] \rangle$ show $\bigcup (\text{indets } G) \subseteq \bigcup (\text{indets } F)$ by (auto simp: $Y$-def Polys-alt)

next

fix $g$

assume $g \in G$

then obtain $g'$ where $g': g' \in \text{extended-ord-punit-reduced-GB } ?F$

and $g = \text{restrict-inds } g' \text{ unfolding } G\text{-def}$

have $f \in ?F \implies \text{homogeneous } f$ for $f$ by (auto simp: homogeneous-homogenize)

with assms(3) have $\text{poly-deg } g' \leq b$ using $g$ by (rule extended-ord.is-hom-GB-boundD)

from $g'$ have $g' \in \text{ideal } (\text{extended-ord-punit-reduced-GB } ?F)$ by (rule ideal.span-base)

also have $\ldots = \text{ideal } ?F$

proof (rule extended-ord.reduced-GB-ideal-Polys)

from $\text{finite } Y$ show $\exists \text{ Insert } (\text{Some } Y)$ by simp

next

show $?F \subseteq P[\text{insert } None \ (\text{Some } Y)]$

proof

fix $f_0$

assume $f_0 \in ?F$

then obtain $f$ where $f \in F$ and $f_0 = \text{homogenize None } (\text{extend-inds } f)$ by blast

from this(1) $\langle F \subseteq P[Y] \rangle$; have $f \in P[Y]$.

hence $\text{extend-inds } f \in P[\text{Some } Y]$ by (auto simp: indets-extend-inds Polys-alt)

thus $f_0 \in P[\text{insert } None \ (\text{Some } Y)]$ unfolding $f_0$ by (rule homogenize-in-Polys)

qed

qed

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

with $\exists f. f \in ?F \implies \text{homogeneous } f$ obtain $F_0$ $q$ where $\text{finite } F_0$ and $F_0 \subseteq ?F$

and $g': g' = (\sum f \in F_0. q \cdot f \cdot f)$ and $\text{deg-le: } \bigwedge f. \text{poly-deg } (q \cdot f \cdot f) \leq \text{poly-deg } g'$

by (rule homogeneous-idealE) blast+

from this(2) obtain $F'$ where $F' \subseteq F$ and $F_0 = \text{homogenize None } \cdot \text{extend-inds } F'$

and inj-on: inj-on (homogenize None $\circ$ extend-inds) $F'$
unfolding image-comp by (rule subset-imageE-inj)

show \( \exists F' \). finite \( F' \) \& \( F' \subseteq F \) \& \( g = (\sum f \in F'. q \ f \ast f) \land (\forall f \in F'. \text{poly-deg}(q f \ast f) \leq b) \)

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 \( g = (\sum f \in F'. (\text{restrict-indets} \circ q \circ \text{homogenize} \ None \circ \text{extend-indets}) f \ast f) \)

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 \( \text{poly-deg} ((\text{restrict-indets} \circ q \circ \text{homogenize} \ None \circ \text{extend-indets}) f \ast f) = \text{poly-deg} (\text{restrict-indets} (q (\text{homogenize} \ None (\text{extend-indets} f))) \ast \text{homogenize} \ None (\text{extend-indets} f))) \)

by (simp add: restrict-indets-times)

also have \( \ldots \leq \text{poly-deg} (q (\text{homogenize} \ None (\text{extend-indets} f))) \ast \text{homogenize} \ None (\text{extend-indets} f)) \)

by (rule poly-deg-restrict-indets-le)

also have \( \ldots \leq \text{poly-deg} g' \) by (rule deg-le)

also have \( \ldots \leq b \) by fact

finally show \( \text{poly-deg} ((\text{restrict-indets} \circ q \circ \text{homogenize} \ None \circ \text{extend-indets}) f \ast f) \leq b \).

qed fact

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)

fix $f$
assume $f \in \text{pmdl} \ (\text{set} \ ps)$
also from assms(2) have $\ldots = \text{pmdl} \ G$ .
finally have $f \in \text{pmdl} \ G$ .
assume $f \neq 0$ with assms(1) $\langle f \in \text{pmdl} \ G \rangle$
obtain $g$ where $g \in G$ and $g \neq 0$ and $\lt g \ adds t \ LT f$
  by (rule GB-adds-lt)
from assms(3) $\langle g \in G \rangle$
have $g \in \text{phull} \ (\text{set} \ ps)$ ..
this $\langle g \neq 0 \rangle$ obtain $g'$ where $g' \in (\text{set} \ (\text{Macaulay-list} \ ps))$ and $g' \neq 0$ and $\lt g = \lt g'$
  by (rule Macaulay-list-lt)
show $\exists g \in (\text{set} \ (\text{Macaulay-list} \ ps)). \ g \neq 0 \wedge \lt g \ adds t \ LT f$
proof (rule, rule)
  from $\langle \lt g \ adds t \ LT f \rangle$
  show $\lt g' \ adds t \ LT f$ by (simp only: $\langle \lt g = \lt g' \rangle$
qed fact+

4.2 Bounds
context pm-powerprod
begin

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

lemma set-deg-shifts:
  $\text{set} \ (\text{deg-shifts} \ d \ fs) = (\bigcup f \in \text{set} \ fs. \ (\lambda t. \text{punit} \ \text{monom-mult} \ 1 \ t) \ f) \ \text{pp-to-list} \ (\text{deg-le-sect} \ X \ (d - \text{poly-deg} \ f)))$
proof –
  from $\text{fin-X}$ have $\text{finite} \ (\text{deg-le-sect} \ X \ d0)$ for $d0$ by (rule finite-deg-le-sect)
  thus $\exists \text{thesis}$ by (simp add: deg-shifts-def punit.set-pps-to-list)
qed

corollary set-deg-shifts-singleton:
  $\text{set} \ (\text{deg-shifts} \ d \ [f]) = (\lambda t. \text{punit} \ \text{monom-mult} \ 1 \ t) \ f) \ \text{pp-to-list} \ (\text{deg-le-sect} \ X \ (d - \text{poly-deg} \ f))$
lemma deg-shifts-superset: set fs ⊆ set (deg-shifts d fs)
proof -
  have set fs = (⋃f∈set fs. {punit.monom-mult 1 0 f}) by simp
  also have ... ⊆ set (deg-shifts d fs) unfolding set-deg-shifts using subset-refl
proof (rule UN-mono)
  fix f
  assume f ∈ set fs
  have punit.monom-mult 1 0 f ∈ (λt. punit.monom-mult 1 t f) " deg-le-sect X (d − poly-deg f)
    using zero-in-deg-le-sect by (rule imageI)
  thus {punit.monom-mult 1 0 f} ⊆ (λt. punit.monom-mult 1 t f) " deg-le-sect X (d − poly-deg f)
    by simp
  qed
finally show ?thesis.
qed

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

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

lemma thm-2-3-6:
  assumes set fs ⊆ P[X] and is-GB-cofactor-bound (set fs) b
  shows punit.is-Groebner-basis (set (punit.Macaulay-list (deg-shifts b fs)))
proof -
  from assms(2) finite-set assms(1) obtain G where punit.is-Groebner-basis G
    and ideal-G: ideal G = ideal (set fs) and G-sub: G ⊆ P[X]
    and f: ∀g. g ∈ G ⇒ ∃q. q = (∑f∈set fs. q f * f) ∧ (∀f. q f ∈ P[X] ∧ poly-deg (q f * f) ≤ b)
    by (rule is-GB-cofactor-boundE-finite-Polys blast)
  from this(1) show ?thesis
  proof (rule punit.Macaulay-list-is-GB)
    show G ⊆ phull (set (deg-shifts b fs)) (is ⊆ ?H)
    proof
      fix g
  qed
assume \( g \in G \)

hence \( \exists q. g = (\sum f \in \text{set } fs. \, q \cdot f) \land (\forall f. \, q \cdot f \in P[X] \land \text{poly-deg } (q \cdot f) \leq b) \) by (rule I)

then obtain \( q \) where \( g = (\sum f \in \text{set } fs. \, q \cdot f) \land (\forall f. \, q \cdot f \in P[X] \land \text{poly-deg } (q \cdot f) \leq b) \) 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 \( q \cdot f \in \{H \} \)

proof (cases \( f = 0 \lor q \cdot f = 0 \))

  case True

  hence \( \{\text{thesis}\} \) by (auto simp add: phull.span-zero)

next

  case False

  hence \( q \cdot f \neq 0 \) and \( f \neq 0 \) by simp-all

  with \( \text{poly-deg } (q \cdot f) \leq b \) have \( \text{poly-deg } (q \cdot f) \leq b - \text{poly-deg } f \)

  by (simp add: poly-deg-times)

  with \( q \cdot f \in P[X] \) have \( \text{keys } (q \cdot f) \subseteq \text{deg-le-sect } X (b - \text{poly-deg } f) \)

  by (rule keys-subset-deg-le-sectI)

  with \( \text{finite-deg-le-sect[OF fin-X]} \)

  have \( q \cdot f = (\sum t \in \text{deg-le-sect } X (b - \text{poly-deg } f). \, \text{punit.monom-mult} (\text{lookup } (q \cdot f) \, \text{t} \, \text{t}) \)

  unfolding \( \text{punit.monom-mult[associated]} \)

  by (rule sum.mon-update[simplified])

  also have \( \ldots = (\sum t \in \text{deg-le-sect } X (b - \text{poly-deg } f). \, (\text{lookup } (q \cdot f) \, \text{t} \cdot \text{punit.monom-mult } 1 \, \text{t} \, \text{f})) \)

  by (simp add: punit.monom-mult-assoc punit.map-scale-eq-monom-mult)

  also have \( \ldots = (\sum t \in \text{deg-le-sect } X (b - \text{poly-deg } f). \, ((\lambda t. \, (\text{lookup } (q \cdot f) \, \text{punit.lp f0 } - \text{punit.lp f})), \, \text{f0}) \circ (\lambda t. \, \text{punit.monom-mult } 1 \, \text{t} \, \text{f}) \, \text{t}) \)

  using refl by (rule sum.cong) (simp add: punit.lt-mon-update[OF \( \neq 0 \) \( \neq 0 \)])

  also have \( \ldots = (\sum f0 \in \text{set } (\text{deg-shifts } b [f]), \, (\text{lookup } (q \cdot f) \, (\text{punit.lp f0 } - \text{punit.lp f})), \, \text{f0}) \)

  unfolding \( \text{set-deg-shifts-singleton} \)

  proof (intro sum.reindex[symmetric] inj-onI)

  fix \( s \, t \)

  assume \( \text{punit.monom-mult } 1 \, \text{s} \, \text{f} = \text{punit.monom-mult } 1 \, \text{t} \, \text{f} \)

  thus \( s = t \) using \( \{1 \neq 0; \, f \neq 0\} \) by (rule punit.monom-mult-inj-2)

  qed

  finally have \( q \cdot f \in \text{phull } (\text{deg-shifts } b [f]) \)

  by (simp add: phull.sum-in-span)

  also have \( \ldots \subseteq \{H \} \) by (rule phull.span-union, rule deg-shifts mono, simp add: \( \text{if } \in \text{set } fs \))

  finally show \( \{\text{thesis}\} \)

  qed

  qed
lemma thm-2-3-7:
assumes set fs ⊆ P[X] and is-GB-cofactor-bound (set fs) b
shows 1 ∈ ideal (set fs) ←→ 1 ∈ set (punit.Macaulay-list (deg-shifts b fs)) (is ?L ↔ ?R)
proof
assume ?L
let ?G = set (punit.Macaulay-list (deg-shifts b fs))
from assms have punit.is-Groebner-basis ?G by (rule thm-2-3-6)
moreover from ⟨?L⟩ have 1 ∈ ideal ?G by (simp add: punit.pmdl-Macaulay-list [simplified])
moreover have 1 ≠ (0::- ⇒ 0′ a) by simp
ultimately obtain g where g ∈ ?G and g ≠ 0 and punit.lt g adds punit.lt (1::- ⇒ 0′ a)
by (rule punit.GB-adds-lt [simplified])
from this(3) have lp-g: punit.lt g = 0 by (simp add: punit.lt-monomial adds-zero flip: single-one)
from punit.Macaulay-list-is-monic-set ⟨g ∈ ?G⟩ ⟨g ≠ 0⟩ have lc-g: punit.lc g = 1
by (rule punit.is-monic-setD)
have g = 1
proof (rule poly-mapping-eqI)
  fix t
  show lookup g t = lookup 1 t
  proof (cases t = 0)
    case True
    thus ?thesis using lc-g by (simp add: lookup-one punit.lc-def lp-g)
  next
    case False
    with zero-min[of t] have ¬ t ≤ punit.lt g by (simp add: lp-g)
    with punit.lt-max-keys have t ∉ keys g by blast
    with False show ?thesis by (simp add: lookup-one in-keys-iff)
  qed
qed
with ⟨g ∈ ?G⟩ show 1 ∈ ?G by simp
next
assume ?R
also have . . . ⊆ phull (set (punit.Macaulay-list (deg-shifts b fs)))
by (rule phull.span-superset)
also have . . . = phull (set (deg-shifts b fs)) by (fact punit.phull-Macaulay-list)
also have . . . ⊆ ideal (set (deg-shifts b fs)) using punit.phull-subset-module by force
finally show ?L by simp
qed
end
lemma thm-2-3-6-indets:
  assumes is-GB-cofactor-bound (set fs) b
  shows punit.is-Groebner-basis (set (punit.Macaulay-list (deg-shifts (UNION (set fs) indets) b fs)))
  using - - assms
proof (rule thm-2-3-6)
  from finite-set show finite (UNION (set fs) indets) by (simp add: finite-indets)
next
  show set fs ⊆ P[UNION (set fs) indets] by (auto simp: Polys-alt)
qed

lemma thm-2-3-7-indets:
  assumes is-GB-cofactor-bound (set fs) b
  shows 1 ∈ ideal (set fs) ↔ 1 ∈ set (punit.Macaulay-list (deg-shifts (UNION (set fs) indets) b fs))
  using - - assms
proof (rule thm-2-3-7)
  from finite-set show finite (UNION (set fs) indets) by (simp add: finite-indets)
next
  show set fs ⊆ P[UNION (set fs) indets] by (auto simp: Polys-alt)
qed

end

5 Integer Binomial Coefficients

theory Binomial-Int
  imports Complex-Main
begin

lemma upper-le-binomial:
  assumes 0 < k and k < n
  shows n ≤ n choose k
proof
  from assms have 1 ≤ n by simp
  define k' where k' = (if n div 2 ≤ k then k else n - k)
  from assms have 1: k' ≤ n - 1 and 2: n div 2 ≤ k' by (auto simp: k'-def)
  from assms(2) have k ≤ n by simp
  have n choose k = n choose k' by (simp add: k'-def binomial-symmetric[OF k ≤ n])
  have n = n choose 1 by (simp only: choose-one)
  also from 1 ≤ n have ... = n choose (n - 1) by (rule binomial-symmetric)
  also from 1 ≤ n have ... ≤ n choose k' by (rule binomial-antimono simp)
  also have ... = n choose k by (simp add: k'-def binomial-symmetric[OF k ≤ n])
  finally show ?thesis .
qed
Restore original sort constraints:

setup: Sign.add-const-constraint (@{ const-name gbinomial }, SOME @{ typ 'a:: {semidom-divide, semiring-char } ⇒ nat ⇒ 'a })

lemma gbinomial-0-left: 0 gchoose k = (if k = 0 then 1 else 0)
  by (cases k) simp-all

lemma gbinomial-eq-0-int:
  assumes n < k
  shows (int n) gchoose k = 0
proof
  have ∃ a ∈ {0..<k}. int n − int a = 0
  proof
    show int n − int n = 0 by simp
  next
    from assms show n ∈ {0..<k} by simp
  qed
  with finite-atLeastLessThan have eq: prod (λ i. int n − int i) {0..<k} = 0
  by (rule prod-zero)
  show ?thesis by (simp add: gbinomial-prod-rev eq)
qed

corollary gbinomial-eq-0: 0 ≤ a ⇒ a < int k ⇒ a gchoose k = 0
by (metis nat-eq-iff2 nat-less-iff gbinomial-eq-0-int)

lemma int-binomial: int (n choose k) = (int n) gchoose k
proof (cases k ≤ n)
  case True
  from refl have eq: (∏ i = 0..<k. int n − int i) = (∏ i = 0..<k. int n − int i)
  proof (rule prod.cong)
    fix i
    assume i ∈ {0..<k}
    with True show int (n − i) = int n − int i by simp
  qed
  show ?thesis
  by (simp add: gbinomial-binomial[symmetric] gbinomial-prod-rev zdiv-int eq)
next
  case False
  thus ?thesis by (simp add: gbinomial-eq-0-int)
qed

lemma falling-fact-pochhammer: prod (λ i. a − int i) {0..<k} = (− 1) ^ k * pochhammer (- a) k
proof
  have eq: z ^ Suc n * prod f {0..n} = prod (λ x. z * f x) {0..n} for z::int and n f
    by (induct n) (simp add: ac-simps)
  show ?thesis
  proof (cases k)
case 0
thus ?thesis by (simp add: pochhammer-minus)
next
case (Suc n)
thus ?thesis
  by (simp only: pochhammer-prod atLeastLessThanSuc-atLeastAtMost
        prod.atLeast-Suc-atMost-Suc-shift eq flip: power-mult-distrib) (simp add: of-nat-diff)
qed
qed

lemma falling-fact-pochhammer': prod (λi. a − int i) {0..<k} = pochhammer (a − int k + 1) k
  by (simp add: falling-fact-pochhammer pochhammer-minus')

lemma gbinomial-int-pochhammer: (a::int) gchoose k = (− 1) ^ k * pochhammer (− a) k div fact k
  by (simp only: gbinomial-prod-rev falling-fact-pochhammer)

lemma gbinomial-int-pochhammer': a gchoose k = pochhammer (a − int k + 1) k div fact k
  by (simp only: gbinomial-prod-rev falling-fact-pochhammer')

lemma fact-dvd-pochhammer: fact k dvd pochhammer (a::int) k
proof
  have dvd: y ≠ 0 =⇒ ((of-int (x div y))':a::field-char-0) = of-int x / of-int y
    =⇒ y dvd x
    for x y :: int
    by (smt dvd-triv-left mult.commute nonzero-eq-divide-eq of-int-eq-0-iff of-int-eq-iff
        of-int-mult)
  show ?thesis
  proof (cases 0 < a)
    case True
    moreover define n where n = nat (a − 1) + k
    ultimately have a: a = int n − int k + 1 by simp
    from fact-nonzero show ?thesis unfolding a
    proof (rule dvd)
      have of-int (pochhammer (int n − int k + 1) k div fact k) = (of-int (int n gchoose k))::rat
        by (simp only: gbinomial-int-pochhammer')
      also have ... = of-int (int (n choose k)) by (simp only: int-binomial)
      also have ... = of-nat (n choose k) by simp
      also have ... = (of-nat n) gchoose k by (fact binomial-gbinomial)
      also have ... = pochhammer (of-nat n − of-nat k + 1) k / fact k
        by (fact gbinomial-pochhammer')
      also have ... = pochhammer (of-int (int n − int k + 1)) k / fact k by simp
      also have ... = (of-int (pochhammer (int n − int k + 1) k)) / (of-int (fact k))
        by (simp only: of-int-fact pochhammer-of-int)
    qed
  qed

25
finally show of-int (pochhammer (int n - int k + 1) k div fact k) =
    of-int (pochhammer (int n - int k + 1) k) / rat-of-int (fact k) .

qed
next
case False
moreover define n where n = nat (- a)
ultimately have a: a = - int n by simp
from fact-nonzero have fact k dvd (-1) ^ k * pochhammer (- int n) k
proof (rule dvd)
  have of-int ((-1) ^ k * pochhammer (- int n) k div fact k) = (of-int (int n choose k))::rat
    by (simp only: gbinomial-int-pochhammer)
  also have ... = of-int (int (n_choose k)) by (simp only: int-binomial)
  also have ... = of-nat (n choose k) by simp
  also have ... = (of-nat n) gchoose k by (fact binomial-gbinomial)
  also have ... = (-1) ^ k * pochhammer (- of-nat n) k / fact k
    by (fact gbinomial-pochhammer)
  also have ... = (-1) ^ k * pochhammer (of-int (- int n)) k / fact k by simp
  also have ... = (-1) ^ k * (of-int (pochhammer (- int n) k)) / (of-int (fact k))
    by (simp only: of-int-int-fact pochhammer-of-int)
  also have ... = (of-int ((-1) ^ k * pochhammer (- int n) k)) / (of-int (fact k)) by simp
finally show of-int ((-1) ^ k * pochhammer (- int n) k div fact k) =
    of-int ((-1) ^ k * pochhammer (- int n) k) / rat-of-int (fact k) .

qed
thus thesis unfolding a by (metis dvdI dvd-mult-unit-iff' minus-one-mult-self)
qed
qed

lemma gbinomial-int-negated-upper: (a choose k) = (-1) ^ k * ((int k - a - 1) choose k)
by (simp add: gbinomial-int-pochhammer pochhammer-minus algebra-simps fact-dvd-pochhammer
div-mult-swap)

lemma gbinomial-int-mult-fact: fact k * (a choose k) = (∏ i = 0..<k. a - int i)
by (simp only: gbinomial-int-pochhammer fact-dvd-pochhammer dvd-mult-div-cancel
falling-fact-pochhammer')
corollary gbinomial-int-mult-fact': (a choose k) * fact k = (∏ i = 0..<k. a - int i)
  using gbinomial-int-mult-fact[of k a] by (simp add: ac-simps)

lemma gbinomial-int-binomial:
a choose k = (if 0 ≤ a then int ((nat a) choose k) else (-1::int) ^ k * int ((k +
    (nat (- a)) - 1) choose k))
by (auto simp: int-binomial gbinomial-int-negated-upper[of a] int-ops(6))
corollary gbinomial-nneg: 0 ≤ a ==> a choose k = int ((nat a) choose k)
by (simp add: gbinomial-int-binomial)

corollary gbinomial-neg: \( a < 0 \Rightarrow a \text{ choose } k = (-1::int)^k \ast \text{int} ((k + (\text{nat} \ (-a))) - 1) \text{ choose } k \)
  by (simp add: gbinomial-int-binomial)

lemma of-int-gbinomial: \( \text{of-int} (a \text{ choose } k) = (\text{of-int } a :: 'a::field-char-0) \text{ choose } k \)
proof –
  have of-int-div: \( y \text{ dvd } x \Rightarrow \text{of-int} (x \div y) = \text{of-int } x / (\text{of-int } y :: 'a) \) for \( x \) \( y \)
    :: int by auto
  show ?thesis
    by (simp add: gbinomial-int-pochhammer' gbinomial-pochhammer' of-int-div
      fact-dvd-pochhammer
      pochhammer-of-int[symmetric])
qed

lemma uminus-one-gbinomial [simp]: \( (-1::int) \text{ choose } k = (-1) \cdot k \)
  by (simp add: gbinomial-int-binomial)

lemma gbinomial-int-Suc-Suc: \( (x + 1::int) \text{ choose } (\text{Suc } k) = (x \text{ choose } k) + (x \text{ choose } (\text{Suc } k)) \)
proof (rule linorder-cases)
  assume 1: \( x + 1 < 0 \)
  hence 2: \( x < 0 \) by simp
  then obtain \( n \) where 3: \( \text{nat} \ (-x) = \text{Suc } n \) using not0-implies-Suc by fastforce
  hence 4: \( \text{nat} \ (-x - 1) = n \) by simp
  show ?thesis
    proof (cases \( k \))
      case 0
      show ?thesis by (simp add: \( k = 0 \))
    next
      case (Suc \( k' \))
      from 1 2 3 4 show ?thesis by (simp add: \( k = \text{Suc } k' \) gbinomial-int-binomial int-distrib(2))
    qed
  next
    assume \( x + 1 = 0 \)
    hence \( x = -1 \) by simp
    thus ?thesis by simp
  next
    assume \( 0 < x + 1 \)
    hence \( 0 \leq x + 1 \) and \( 0 \leq x \) and \( \text{nat} (x + 1) = \text{Suc } (\text{nat } x) \) by simp-all
    thus ?thesis by (simp add: gbinomial-int-binomial)
  qed

corollary plus-Suc-gbinomial:
  \( (x + (1 + \text{int } k)) \text{ choose } (\text{Suc } k) = ((x + \text{int } k) \text{ choose } k) + ((x + \text{int } k) \text{ choose } (\text{Suc } k)) \)
(is $l = r$)

proof -
have $l = (x + \text{int } k + 1) \text{ choose } (\text{Suc } k)$ by (simp only: ac-simps)
also have $\ldots = r$ by (fact gbinomial-int-Suc-Suc)
finally show $\text{thesis}$.

qed

lemma gbinomial-int-n-n [simp]: $\text{(int } n \text{) choose } n = 1$

proof (induct n)
case 0
show $\text{case}$ by simp
next
case (Suc n)
have $\text{int } (\text{Suc } n) \text{ choose Suc } n = (\text{int } n + 1) \text{ choose } \text{Suc } n$ by (simp add: add.commute)
also have $\ldots = (\text{int } n \text{ choose } n) + (\text{int } n \text{ choose } (\text{Suc } n))$ by (fact gbinomial-int-Suc-Suc)
finally show $\text{case}$ by (simp add: Suc gbinomial-eq-0)

qed

lemma gbinomial-int-Suc-n [simp]: $(1 + \text{int } n) \text{ choose } n = 1 + \text{int } n$

proof (induct n)
case 0
show $\text{case}$ by simp
next
case (Suc n)
have $1 + \text{int } (\text{Suc } n) \text{ choose Suc } n = (1 + \text{int } n) + 1 \text{ choose } \text{Suc } n$ by simp
also have $\ldots = (1 + \text{int } n \text{ choose } n) + (1 + \text{int } n \text{ choose } (\text{Suc } n))$ by (fact gbinomial-int-Suc-Suc)
also have $\ldots = 1 + \text{int } n + (\text{int } (\text{Suc } n) \text{ choose } (\text{Suc } n))$ by (simp add: Suc)
also have $\ldots = 1 + \text{int } (\text{Suc } n)$ by (simp only: gbinomial-int-n-n)
finally show $\text{case}$.

qed

lemma zbinomial-eq-0-iff [simp]: $a \text{ choose } k = 0 \iff (0 \leq a \land a < \text{int } k)$

proof
assume $a: a \text{ choose } k = 0$
have $1: b < \text{int } k$ if $b \text{ choose } k = 0$ for $b$
proof (rule ccontr)
assume $\neg b < \text{int } k$
hence $0 \leq b$ and $k \leq \text{nat } b$ by simp-all
from this(1) have $\text{int } ((\text{nat } b) \text{ choose } k) = b \text{ choose } k$ by (simp add: gbinomial-int-binomial)
also have $\ldots = 0$ by (fact that)
finally show False using $k \leq \text{nat } b$ by simp

qed

show $0 \leq a \land a < \text{int } k$

proof
show $0 \leq a$
proof (rule ccontr)

qed

28
assume $0 \leq a$

hence $(-1)^{k \ast ((\text{int } k - a - 1) \text{ choose } k)} = a \text{ choose } k$

by (simp add: gbinomial-int-negated-upper[of $a$])

also have ... $= 0$ by (fact $a$

finally have $(\text{int } k - a - 1) \text{ choose } k = 0$ by simp

hence $(\text{int } k - a - 1 < \text{int } k$ by (rule 1)

with $(\sim 0 \leq a)$ show False by simp

qed

next

from $a$

show $a < \text{int } k$ by (rule 1)

qed

qed (auto intro: gbinomial-eq-0)

5.1 Sums

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

proof −

have $(\sum_{j \leq n.} \text{int } j + \text{int } k \text{ choose } k) = \text{int } (\sum_{j \leq n.} \text{int } j + \text{int } k \text{ choose } k)$

by (simp add: int-binomial add.commute)

also have $(\sum_{j \leq n.} \text{int } j + \text{int } k \text{ choose } k) = (k + \text{int } k + 1) \text{ choose } (k + 1)$ by (fact choose-rising-sum(1))

also have $(\sum_{j \leq n.} \text{int } j + \text{int } k \text{ choose } k) = (\text{int } n + \text{int } k + 1) \text{ choose } (\text{Suc } k)$

by (simp add: int-binomial ac-simps del: binomial-Suc-Suc)

finally show ?thesis .

qed

lemma gchoose-rising-sum:

assumes $0 \leq n$ — Necessary condition.

shows $(\sum_{j \leq \text{nat } n.} \text{int } j + \text{int } k \text{ choose } k) = (\text{int } n + \text{int } k + 1) \text{ choose } (\text{Suc } k)$

proof −

from - refl have $(\sum_{j \leq \text{nat } n.} \text{int } j + \text{int } k \text{ choose } k) = (\sum_{j \in \text{int } \{0..\text{nat } n\}.} \text{int } j + \text{int } k \text{ choose } k)$

proof (rule sum.cong)

from assms show $\{0..n\} = \text{int } \{0..\text{nat } n\}$ by (simp add: image-int-atLeastAtMost)

qed

also have ... $(\sum_{j \leq \text{nat } n.} \text{int } j + \text{int } k \text{ choose } k) = (\sum_{j \in \text{int } \{0..\text{nat } n\}.} \text{int } j + \text{int } k \text{ choose } k)$

by (simp add: sum.reindex atMost-atLeast0)

also have ... $(\text{int } (\text{nat } n) + \text{int } k + 1) \text{ choose } (\text{Suc } k)$ by (fact gchoose-rising-sum-nat)

also from assms have ... $(\text{int } (\text{nat } n) + \text{int } k + 1) \text{ choose } (\text{Suc } k)$ by (simp add: add.assoc add.commute)

finally show ?thesis .

qed

5.2 Inequalities

lemma binomial-mono:

assumes $m \leq n$

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

proof −
define \( l \) where \( l = n - m \)

with \( \text{assms} \) have \( n = m + l \) by simp
have \( m \choose k \leq (m + l) \choose k \)
proof (induct \( l \))
  case \( \theta \)
  show \( l \) case by simp
next
  case \( * : (\text{Suc } l) \)
  show \( l \) case
  proof (cases \( k \))
    case \( \theta \)
    thus \( \text{thesis} \) by simp
next
  case \( k : (\text{Suc } k0) \)
  note \( * \)
  also have \( m + l \choose k \leq m + l \choose k + (m + l \choose k0) \) by simp
  also have \( \ldots = m + \text{Suc } l \choose k \) by (simp add: \( k \))
  finally show \( \text{thesis} \).
qed
qed

thus \( \text{thesis} \) by (simp only: \( n \))
qed

lemma binomial-plus-le:
  assumes \( \theta < k \)
  shows \( m \choose k + (n \choose k) \leq (m + n) \choose k \)
proof --
  define \( k0 \) where \( k0 = k - 1 \)
  with \( \text{assms} \) have \( k = \text{Suc } k0 \) by simp
  show \( \text{thesis} \) unfolding \( k \)
  proof (induct \( n \))
    case \( \theta \)
    show \( l \) case by simp
next
  case \( \text{(Suc } n) \)
  have \( m \choose k0 + (\text{Suc } n \choose k0) = m \choose k0 + (n \choose k0) \)
  by (simp only: binomial-Suc-Suc)
  also from \( \text{Suc } \) have \( \ldots \leq (m + n) \choose k0 + ((m + n) \choose k0) \)
  proof (rule add-mono)
    have \( n \leq m + n \) by simp
    thus \( n \) choose \( k \) \( \leq m + n \) choose \( k0 \) by (rule binomial-mono)
  qed
  also have \( \ldots = m + \text{Suc } n \choose k0 \) by simp
  finally show \( \text{thesis} \).
  qed
  qed

lemma binomial-ineq-1: \( 2 \star (n + i) \choose k \leq n \choose k + ((n + 2 \star i) \)}
choose \( k \)

proof (cases \( k \))

  case \( 0 \)
  thus \( \text{thesis} \) by simp

next

  case \( k \) \( : \) \( (\text{Suc} \ k) \)
  show \( \text{thesis} \) unfolding \( k \)
    proof (induct \( i \))
      case \( 0 \)
      thus \( \text{?case} \) by simp
    next
      case \( (\text{Suc} \ i) \)
      have \( 2 \ast (n + \text{Suc} \ i \choose \text{Suc} \ k) = 2 \ast (n + i \choose k) + 2 \ast (n + i \choose \text{Suc} \ k) \)
        by simp
      also have \( \ldots \leq (n + 2 \ast i \choose k) + (n + 2 \ast i \choose \text{Suc} \ k) \)
        proof (rule add-mono)
          have \( n + i \choose k \leq n + 2 \ast i \choose k \) by (rule binomial-mono) simp
          moreover have \( n + 2 \ast i \choose k \leq \text{Suc} \ (n + 2 \ast i) \choose k \) by (rule binomial-mono) simp
          ultimately show \( 2 \ast (n + i \choose k) + (\text{Suc} \ (n + 2 \ast i) \choose k) \leq (n + 2 \ast i) \choose \text{Suc} \ k) \)
            by simp
        qed (fact Suc)
      also have \( \ldots = n \choose k + (n + 2 \ast \text{Suc} \ i \choose \text{Suc} \ k) \) by simp
      finally show \( \text{?case} \).
    qed

qed

lemma \( \text{gbinomial-int-nonneg} \):

assumes \( 0 \leq (x :: \text{int}) \)

shows \( 0 \leq x \text{ gchoose} \ k \)

proof –

  have \( 0 \leq \text{int} \ (n \text{ choose} \ k) \) by simp
  also from \( \text{assms} \) have \( \ldots = x \text{ gchoose} \ k \) by (simp add: int-binomial)
  finally show \( \text{thesis} \).

qed

lemma \( \text{gbinomial-int-mono} \):

assumes \( 0 \leq x \) and \( x \leq (y :: \text{int}) \)

shows \( x \text{ gchoose} \ k \leq y \text{ gchoose} \ k \)

proof –

  from \( \text{assms} \) have \( \text{nat} \ x \leq \text{nat} \ y \) by simp
  hence \( \text{nat} \ x \choose k \leq \text{nat} \ y \choose k \) by (rule binomial-mono)
  hence \( \text{int} \ (\text{nat} \ x \choose k) \leq \text{int} \ (\text{nat} \ y \choose k) \)
    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 \( \text{assms} \) show \( \text{thesis} \) by simp

qed
lemma \text{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 \text{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: \text{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: \text{int-plus int-binomial})
  with assms(2, 3) show \(?\)thesis by simp
qed

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

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

corollary \text{binomial-int-ineq-3}:
  assumes $0 \leq y$ and $y \leq 2 \ast (x::\text{int})$
  shows $2 \ast (x \text{ gchoose } k) \leq y \text{ gchoose } k + (2 \ast x - y \text{ gchoose } k)$
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 $2 \ast (x \text{ gchoose } k) \leq x - (x - y) \text{ gchoose } k + (x + (x - y))$
gchoose k)
    by (rule binomial-int-ineq-2)
  thus ?thesis by simp
next
case False
  hence 0 ≤ y - x by simp
moreover from assms(2) have y - x ≤ x by simp
ultimately have 2 * (x gchoose k) ≤ x - (y - x) gchoose k + (x + (y - x)
    by (rule binomial-int-ineq-2)
  thus ?thesis by simp
qed

5.3 Backward Difference Operator

definition bw-diff :: ('a ⇒ 'a) ⇒ 'a ⇒ 'a
  where bw-diff f x = f x - f (x - 1)

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

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

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

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

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

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

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

lemma bw-diff-plus-pow [simp]:
  (bw-diff ^^ k) (λx. f x + g x) = (λx. (bw-diff ^^ k) f x + (bw-diff ^^ k) g x)
  by (induct k, simp-all)

lemma bw-diff-uminus-pow [simp]: (bw-diff ^^ k) (λx. - f x) = (λx. - (bw-diff
```

lemma bw-diff-minus-pow [simp]:
  (bw-diff ^^ k) (\(\lambda x. f x - g x\)) = (\(\lambda x. (bw-diff ^^ k) f x - (bw-diff ^^ k) g x\))
by (induct k, simp-all)

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

lemma bw-diff-gbinomial:
  assumes 0 < k
  shows bw-diff (\(\lambda x::int. (x + n) gchoose k\)) = (\(\lambda x. (x + n - 1) gchoose (k - 1)\))
proof (rule ext)
  fix x::int
  from assms have eq: Suc (k - Suc 0) = k by simp
  have x + n gchoose k = (x + n - 1) + 1 gchoose (Suc (k - 1)) by (simp add: eq)
  also have \(\ldots = (x + n - 1) gchoose (k - 1) + ((x + n - 1) gchoose (Suc (k - 1))\))
  by (fact gbinomial-int-Suc-Suc)
  finally show bw-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-diff ^^ l) (\(\lambda x::int. (x + n) gchoose k\)) =
  (if l \leq k then (\(\lambda x. (x + n - \text{int } l) gchoose (k - l)\)) else (\(\lambda \cdot 0\)))
proof -
  have *: l0 \leq k \Longrightarrow (bw-diff ^^ l0) (\(\lambda x::int. (x + n) gchoose k\)) = (\(\lambda x. (x + n - \text{int } l0) gchoose (k - l0)\))
  for l0
  proof (induct l0)
    case 0
    show ?case by simp
  next
    case (Suc l0)
    from Suc.prems have 0 < k - l0 and l0 \leq k by simp-all
    from this(2) have eq: (bw-diff ^^ l0) (\(\lambda x. x + n gchoose k\)) = (\(\lambda x. x + n - \text{int } l0 gchoose (k - l0)\))
      by (rule Suc.hyps)
    have (bw-diff ^^ Suc l0) (\(\lambda x. x + n gchoose k\)) = bw-diff (\(\lambda x. x + (n - \text{int } l0) gchoose (k - l0)\))
      by (simp add: eq algebra-simps)
    also from 0 < k - l0 have \(\ldots = (\lambda x. (x + (n - \text{int } l0) - 1) gchoose (k - l0 - 1))\)
      by (rule bw-diff-gbinomial)
end
```
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: (∀a. rat-of-int (g a) = poly q (rat-of-int a))
    by (rule poly-funE, blast)
  have p = q
  proof (rule ccontr)
let \(?A = \{ a. \text{poly} p \ (\text{rat-of-int} \ a) = \text{poly} q \ (\text{rat-of-int} \ a)\}\)

assume \(p \neq q\)

hence \(p - q \neq 0\) by simp

hence \(\text{fin} \ \{ x. \text{poly} \ (p - q) \ x = 0\}\) by (rule poly-roots-finite)

have \(\text{rat-of-int} \ ?A \subseteq \{ x. \text{poly} \ (p - q) \ x = 0\}\) by (simp add: image-Collect-subsetI)

moreover have \(\text{inj-on} \ \text{rat-of-int} \ ?A \) by (simp add: inj-on-def)

ultimately have \(\text{finite} \ \text{rat-of-int} \ ?A\) using fin by (rule finite-subset)

finally show \(\text{False}\) using assms (3) by simp

qed

hence \(\text{rat-of-int} \ (f \ a) = \text{rat-of-int} \ (g \ a)\) by (simp add: p q)

thus \(f \ a = g \ a\) by simp

qed

corollary \(\text{poly-fun-eqI-ge}\):

assumes \(\text{poly-fun} \ f\) and \(\text{poly-fun} \ g\) and \(\forall a. b \leq a \implies f \ a = g \ a\)

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 \(\text{poly-fun-eqI-gr}\):

assumes \(\text{poly-fun} \ f\) and \(\text{poly-fun} \ g\) and \(\forall a. b < a \implies f \ a = g \ a\)

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 \(\text{poly-fun-const}\) [simp]: \(\text{poly-fun} \ (\lambda. \ c)\)

by (rule poly-funI[where \(p=:\text{rat-of-int} \ c\)] simp

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

proof -

show \(\text{poly-fun} \ (\lambda x. \ x)\) by (rule poly-funI[where \(p=:\text{rat-of-int} \ I\)] simp

thus \(\text{poly-fun} \ id\) by (simp only: id-def)

qed

lemma \(\text{poly-fun-uminus}\):

assumes \(\text{poly-fun} \ f\)

shows \(\text{poly-fun} \ (\lambda x. -f \ x)\) and \(\text{poly-fun} \ (-f)\)

proof -

from assms obtain \(p\) where \(\forall a. \text{rat-of-int} \ (f \ a) = \text{poly} \ p \ (\text{rat-of-int} \ a)\)
by (rule poly-funE, blast)
show poly-fun (λx. - 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 (λx. − f x) ←→ poly-fun f poly-fun (− f) ←→ poly-fun f
proof –
  show poly-fun (λx. − f x) ←→ poly-fun f
  proof
    assume poly-fun (λx. − f x)
    hence poly-fun (λx. -(− f x)) by (rule poly-fun-uminus)
    thus poly-fun f by simp
  qed (rule poly-fun-uminus)
  thus poly-fun (− f) ←→ 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 (λx. f x + g x)
proof –
  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: ∃a. rat-of-int (g a) = poly q (rat-of-int a)
    by (rule poly-funE, blast)
  show ?thesis by (rule poly-funI[where p=p + q]) (simp add: p q)
qed

lemma poly-fun-minus [simp]:
  assumes poly-fun f and poly-fun g
  shows poly-fun (λx. f x - g x)
proof –
  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: ∃a. rat-of-int (g a) = poly q (rat-of-int a)
    by (rule poly-funE, blast)
  show ?thesis by (rule poly-funI[where p=p − q]) (simp add: p q)
qed

lemma poly-fun-times [simp]:
  assumes poly-fun f and poly-fun g
  shows poly-fun (λx. f x * g x)
proof –
  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: ∃a. rat-of-int (g a) = poly q (rat-of-int a)
    by (rule poly-funE, blast)
  show ?thesis by (rule poly-funI[where p=p * q]) (simp add: p q)
qed
lemma poly-fun-divide:
assumes poly-fun f and \( \forall a. c \text{ dvd } f a \)
shows poly-fun \( (\lambda x. f x \div c) \)
proof –
from assms(1) obtain p where p: \( \forall a. \text{ rat-of-int } (f a) = \text{ poly } p \text{ (rat-of-int } a) \)
by (rule poly-funE, blast)
let \(?p = p \ast \left[ 1 / \text{ rat-of-int } c \right] \)
show ?thesis
proof (rule poly-funI)
fix a
have \( c \text{ dvd } f a \) by fact
hence \( \text{ rat-of-int } (f a \div c) = \text{ rat-of-int } (f a) / \text{ rat-of-int } c \) by auto
also have \( \ldots = \text{ poly } \ ?p \text{ (rat-of-int } a) \) by (simp add: p)
finally show \( \text{ rat-of-int } (f a \div c) = \text{ poly } \ ?p \text{ (rat-of-int } a) \).
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: \( \forall a. \text{ rat-of-int } (f a) = \text{ poly } p \text{ (rat-of-int } a) \)
by (rule poly-funE, blast)
show ?thesis by (rule poly-funI[where \( p = p \ ^{\ k} \)] (simp add: p)
qed

lemma poly-fun-comp:
assumes poly-fun f and poly-fun g
shows poly-fun \( (\lambda x. f \circ g) \) and poly-fun \( f \circ g \)
proof –
from assms(1) obtain p where p: \( \forall a. \text{ rat-of-int } (f a) = \text{ poly } p \text{ (rat-of-int } a) \)
by (rule poly-funE, blast)
from assms(2) obtain q where q: \( \forall a. \text{ rat-of-int } (g a) = \text{ poly } q \text{ (rat-of-int } a) \)
by (rule poly-funE, blast)
show poly-fun \( (\lambda x. f \ (g x)) \) by (rule poly-funI[where \( p = p \circ q \)] (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 \( \text{infinite } I \)
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.prems)
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.prems)
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.prems)
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.prems)
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
Properties of modules generated by sets of monomials, and (reduced) Gröbner bases thereof.

7.1 Sets of Monomials

definition is-monomial-set :: ('a ⇒ 'b::zero) set ⇒ bool
  where is-monomial-set A = (∀ p∈A. is-monomial p)

lemma is-monomial-setI: (∀ p∈A. is-monomial p) ⇒ is-monomial-set A
  by (simp add: is-monomial-set-def)

lemma is-monomial-setD: is-monomial-set A ⇒ p∈A ⇒ is-monomial p
  by (simp add: is-monomial-set-def)

lemma is-monomial-set-subset: is-monomial-set B ⇒ A⊆B ⇒ is-monomial-set A
  by (auto simp: is-monomial-set-def)

lemma is-monomial-set-Un: is-monomial-set (A ∪ B) ←→ (is-monomial-set A ∨ is-monomial-set B)
  by (auto simp: is-monomial-set-def)

7.2 Modules

context term-powerprod

lemma monomial-pmdl:
  assumes is-monomial-set B and p∈pmdl B
  shows monomial (lookup p v) v∈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 = monomial d u by (rule is-monomial-monomial)
  have monomial (lookup (monom-mult c t b) v) v∈pmdl B
    proof (simp add: b monom-mult-monomial lookup-single when-def pmdl.span-zero, intro impl)
      assume t ⊕ u = v
      hence monomial (c * d) v = monom-mult c t b by (simp add: b monom-mult-monomial)
also from step.hyps(3) have ... ∈ pmdl B by (rule monom-mult-in-pmdl)

finally show monomial \((c \cdot d) \cdot v \in pmdl B\).

qed

with step.hyps(2) show \(?case\) unfolding eq by (rule pmdl.span-add)

qed

lemma monomial-pmdl-field:

assumes \(is-monomial-set B \quad and \quad p \in pmdl B \quad and \quad v \in keys (p::- \Rightarrow_{_0} 'b::field)\)

shows monomial \(c \cdot v \in pmdl B\)

proof

from assms(1, 2) have monomial \((\text{lookup } p \cdot v) \in pmdl B\) by (rule monomial-pmdl)

hence monom-mult \((c / \text{lookup } p \cdot v) \cdot 0 \quad (\text{monomial } (\text{lookup } p \cdot v) \in pmdl B\)

by (rule pmdl-closed-monom-mult)

with assms(3) show \(?thesis\) by (simp add: monom-mult-monomial splus-zero in-keys-iff)

qed

end

context ordered-term

begin

lemma keys-monomial-pmdl:

assumes \(is-monomial-set F \quad and \quad p \in pmdl F \quad and \quad t \in keys p\)

obtains \(f \quad where \quad f \in F \quad and \quad f \neq 0 \quad and \quad \text{lt } f \ \text{adds}, \ t\)

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: \((\text{module-plus } p \cdot f0 \cdot c \cdot s)\)

from assms(1) step(3) have is-monomial f0 unfolding is-monomial-set-def ..

hence keys f0 = \{\text{lt } f0\} \quad and \quad f0 \neq 0\) by (rule keys-monomial, rule monomial-not-0)

from Poly-Mapping.keys-add-step(6) have \(t \in keys p \cup keys (\text{monom-mult } c \cdot s \cdot f0)\) ..

thus \(?case\)

proof

assume \(t \in keys p\)

from step(2)[OF - this] obtain \(f \quad where \quad f \in F \quad and \quad f \neq 0 \quad and \quad \text{lt } f \ \text{adds}, \ t\)

by blast

thus \(?thesis\) by (rule step(5))

next

assume \(t \in keys (\text{monom-mult } c \cdot s \cdot f0)\)

with keys-monom-mult-subset have \(t \in (\oplus) \quad s \cdot keys f0\) ..

hence \(t = s \oplus \text{lt } f0\) by (simp add: \(\text{keys } f0 = \{\text{lt } f0\}\))

hence \(\text{lt } f0 \ \text{adds}, \ t\) by (simp add: term-simps)

with \(\{f0 \in F\} \cdot f0 \neq 0\) show \(?thesis\) by (rule step(5))

qed

qed
lemma image-lt-monomial-lt: \( \text{l}t \cdot \text{monomial} (1::'b::zero-neq-one) \cdot \text{l}t \cdot F = \text{l}t \cdot F \)
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)
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 \( rs: \text{red-single} \) p q r s unfolding red-singleton
  hence cp0: lookup p \((s \oplus \text{lt} r)\) \neq 0 and q-def0: \( q = p - \text{monom-mult} \) (lookup p \((s \oplus \text{lt} r)\) / \( \text{lc} r\) ) s r
  unfolding red-single-def by simp-all
from assms(1) obtain c t where \( c \neq 0 \) and r-def: \( r = \text{monomial} \) c t by (rule is-monomial-monomial)
  have ltr: \( \text{lt} r = t \) unfolding r-def by (rule lt-monomial, fact)
  have lcr: \( \text{lc} r = c \) 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 r unfolding u-def ltr .
  also have \( \ldots = p - \text{monomial} \) ((lookup p u) * c) u unfolding u-def r-def monom-mult-monomial .
finally have q-def: \( q = p - \text{monomial} \) (lookup p u) u using \( c \neq 0 \) 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 \((p - \text{monomial} \) (lookup p u) u\)
  hence lookup \((p - \text{monomial} \) (lookup p u) u\) x \neq 0 by (simp add: in-keys-iff)
  hence a: lookup p x - lookup \((\text{monomial} \) (lookup p u) u\) x \neq 0 unfolding lookup-minus .
  hence x \neq u unfolding lookup-single by auto
  with a have lookup p x \neq 0 unfolding lookup-single by auto
  show x \in keys p - {u}
  proof

42
from \( \text{lookup } p \ x \neq 0 \) show \( x \in \text{keys } p \) by (simp add: \( \text{in-keys-iff} \))

next
from \( x \neq u \) 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 \( x \in \text{keys } p \)

have \( \text{lookup } p \ x \neq 0 \) by (simp add: \( \text{in-keys-iff} \))

with \( x \neq u \) have \( \text{lookup } (p - \text{monomial } (\text{lookup } p \ u) \ u) \ x \neq 0 \) by (simp add: \( \text{lookup-minus lookup-single} \))

thus \( x \in \text{keys } \) \((p - \text{monomial } (\text{lookup } p \ u) \ u)\) by (simp add: \( \text{in-keys-iff} \))

qed

qed

have \( \text{Suc } (\text{card } (\text{keys } q)) = \text{card } (\text{keys } p) \) unfolding \( \text{keys } q = \text{keys } p - \{u\} \)

by (rule \( \text{card-Suc-Diff1} \), rule \( \text{finite-keys} \), fact)

thus \( ?\text{thesis} \) 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 assms(3) obtain \( b \) where \( b \in B \) and \( b \neq 0 \) and \( *: \text{red } \{b\} \ p \ q \) by (rule \( \text{red-setE2} \))

from assms(2) this(1) have \( \text{is-monomial } b \) by (rule \( \text{is-monomial-setD} \))

hence \( \text{card } (\text{keys } p) = \text{Suc } (\text{card } (\text{keys } q)) \) using \( * \) by (rule \( \text{red-monomial-keys} \))

with assms(1) show \( ?\text{thesis} \) by (simp add: \( \text{is-monomial-def} \))

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 assms(3) obtain \( q \) where \( \text{red } B \ p \ q \) by (rule \( \text{is-redE} \))

moreover from assms(1, 2) this have \( q = 0 \) by (rule \( \text{red-monomial-monomial-setD} \))

ultimately show \( ?\text{thesis} \) by simp

qed

corollary \( \text{is-red-monomial-monomial-set-in-pmdl} \): 

is-monomial \( p \implies \text{is-monomial-set } B \implies \text{is-red } B \ p \implies p \in \text{pmdl } B \)

by (intro \( \text{red-rtranclp-0-in-pmdl} \) \( r\text{-into-rtranclp} \) is-red-monomial-monomial-setD)

corollary \( \text{red-rtrancl-monomial-monomial-set-cases} \): 

assumes \( \text{is-monomial } p \) and \( \text{is-monomial-set } B \) and \( (\text{red } B)^* \ p \ q \)

obtains \( q = p \mid q = 0 \)
using assms(3)

proof (induct q arbitrary: thesis rule: rtranclp-induct)

  case base
  from refl show ?case by (rule base)

next
  case (step y z)
  show ?case
  proof (rule step.hyps)
    assume y = p
    with step.hyps(2) have red B p z by simp
    with assms(1, 2) have z = 0 by (rule red-monomial-monomial-setD)
    thus ?thesis by (rule step.prems)
  next
    assume y = 0
    from step.hyps(2) have is-red B 0 unfolding (y = 0) by (rule is-redI)
    with irred-0 show ?thesis ..
  qed

qed

lemma is-red-monomial-lt:
  assumes 0 ∉ B
  shows is-red (monomial (1::'b::field) · lt B) = is-red B

proof
  fix p
  let ?B = monomial (1::'b) · lt B
  show is-red ?B p ⇐⇒ is-red B p
  proof
    assume is-red ?B p
    then obtain f v where f ∈ ?B and v ∈ keys p and adds: lt f adds t v
    by (rule is-red-addsE)
    from this(1) have lt f ∈ lt ?B by (rule imageI)
    also have ... = lt · B by (fact image-lt-monomial-lt)
    finally obtain b where b ∈ B and eq: lt f = lt b ..
    note this(1)
    moreover from this assms have b ≠ 0 by blast
    moreover note (v ∈ keys p)
    moreover from adds have lt b adds t v by (simp only: eq)
    ultimately show is-red B p by (rule is-red-addsI)
  next
    assume is-red B p
    then obtain b v where b ∈ B and v ∈ keys p and adds: lt b adds t v
    by (rule is-red-addsE)
    from this(1) have lt b ∈ lt · B by (rule imageI)
    also from image-lt-monomial-lt have ... = lt · ?B by (rule sym)
    finally obtain f where f ∈ ?B and eq: lt b = lt f ..
    note this(1)
    moreover from this have f ≠ 0 by (auto simp: monomial-0-iff)
    moreover note (v ∈ keys p)
    moreover from adds have lt f adds t v by (simp only: eq)
ultimately show is-red \( B p \) by (rule is-red-addsI)
qed
qed
end

7.4 Gröbner Bases

category 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 \( (\text{red } G)^\ast \ast f 0 \)
  proof (induct \( f \) rule: poly-mapping-plus-induct)
    case 1
    show \( \ldots \).
  next 
    case (2 f c t)
    let \( \text{?f} = \text{monomial } c t + f \)
    from 2(1) have \( t \in \text{keys } (\text{monomial } c t) \) by simp
    from this 2(2) have \( t \in \text{keys } \text{?f} \) by (rule in-keys-plusI1)
    with assms \( (\text{?f} \in \text{pmdl } G) \) obtain \( g \) where \( g \in G \) and \( g \neq 0 \) and \( \text{lt } g \) adds \( t \)
    by (rule keys-monomial-pmdl)
    from this(1) have \( \text{red } G \) \( ?f \) \( f \)
    proof (rule red-setI)
      from \( \text{lt } g \) adds \( t \); \( \text{have } \text{component-of-term } (\text{lt } g) = \text{component-of-term } t \) and \( \text{lp } g \) adds \( \text{pp-of-term } t \)
      by (simp-all add: adds-term-def)
      from this have 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 lookup \( \text{?f } t = c \) by (simp add: lookup-add in-keys-iff)
    ultimately show \( \text{red-single } (\text{monomial } c + t) \) \( f \) \( g \) \( (\text{pp-of-term } t - \text{lp } g) \)
    proof (simp add: red-single-def \( g \neq 0 \); \( t \in \text{keys } \text{?f} \) 2(1))
      from \( g \neq 0 \) 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 \( g \in G \) have \( \text{is-monomial } g \) unfolding is-monomial-set-def
      ..
    ultimately show \( \text{monomial } c t = \text{monom-mult } (c / \text{lc } g) (\text{pp-of-term } t - \text{lp } g) \) \( g \)

45
by (simp only: monomial-eq-itself)
qed
qed
have \( f \in \text{pmdl } G \) by (rule pmdl-closed-red, fact subset-refl, fact+)
hence \( (\text{red } G)^* \ f \ 0 \) by (rule 2(3))
with \( \text{red } G \ ?f \) show \( ?\)case by simp
qed
qed

context

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

context

fixes \( F \ m \)
assumes \( \text{fin-comps}: \text{finite } (\text{component-of-term } ' \text{Keys } F) \)
and \( \text{F-sub}: F \subseteq \text{dgrad-p-set } d \ m \)
and \( \text{F-monom}: \text{is-monomial-set } (F::(- \Rightarrow \text{?}0::'b::\text{field} ) \text{set}) \)
begin

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

lemma \( \text{reduced-GB-subset-monic-dgrad-p-set}: \text{reduced-GB } F \subseteq \text{monic ' F} \)
proof
− from subset-refl obtain \( F' \) where \( F' \subseteq F - \{0\} \) and \( \text{lt } (F - \{0\}) = \text{lt } F' \)
and \( \text{inj-on } \text{lt } F' \) by (rule subset-imageE-inj)
define \( G \) where \( G = \{ f \in F', \forall f' \in F', \text{lt } f' \text{ adds t } \text{lt } f \rightarrow f' = f \} \)
have \( G \subseteq F' \) by (simp add: G-def)
hence \( G \subseteq F - \{0\} \) using \( (F' \subseteq F - \{0\}) \) by (rule subset-trans)
also have \( \ldots \subseteq F \) by blast
finally have \( G \subseteq F \).

have \( 1: \text{thesis if } f \in F \) and \( f \neq 0 \) and \( \bigwedge g. g \in G \Rightarrow \text{lt } g \text{ adds t } \text{lt } f \Rightarrow \text{thesis } f \)
for \( \text{thesis } f \)
proof
− let \( \{K = \text{component-of-term } ' \text{Keys } F \)
let \( \{A = t. \text{pp-of-term } t \in \text{dgrad-set } d \ m \land \text{component-of-term } t \in ?K \} \)
let \( \{Q = \{f' \in F'. \text{lt } f' \text{ adds t } \text{lt } f \} \}
from dgrad fin-comps have \( \text{almost-full-on } (\text{adds}_1) \ ?A \) by (rule Dickson-term)
moreover have \( \text{transp-on } (\text{adds}_1) \ ?A \) by (auto intro: transp-onI dest: adds-term-trans)
ultimately have \( \text{wfp-on } (\text{strict } (\text{adds}_1)) \ ?A \) by (rule af-trans-imp-wf)
moreover have \( \text{lt } f \in \text{lt } ?Q \)
proof
− from that(1, 2) have \( f \in F - \{0\} \) by simp
hence \( \text{lt } f \in \text{lt } (F - \{0\}) \) by (rule imageI)
also have \( \ldots = \text{lt } F' \) by fact
finally have \( \text{lt } f \in \text{lt } F' \).
with \( \text{adds-term-refl } \text{show } ?\text{thesis } \text{by fastforce} \)
qed
moreover have \( lt \cdot ?Q \subseteq ?A \)

proof

fix \( s \)
assume \( s \in lt \cdot ?Q \)
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 \( F'' \subseteq F - \{0\} \).

hence \( q \in F \) and \( q \neq 0 \) by simp all
from this(1) F-sub have \( q \in dgrad-p-set d m \).

from \( q \neq 0 \) have \( lt q \in keys q \) by (rule lt-in-keys)

hence pp-of-term \( (lt q) \in pp-of-term \) \( \langle \) keys \( q \) \( \rangle \) by (rule imageI)

also from \( q \in dgrad-p-set d m \) have \( \ldots \subseteq dgrad-set d m \) by (simp add: dgrad-p-set-def)

finally have \( I: pp-of-term s \in dgrad-set d m \) by (simp only: \( s \))
from \( (lt q \in keys q \) \( \langle \) \( q \in F \) \( \rangle \) 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 \( t \in lt \cdot ?Q \) and \( t-min: \bigwedge s. \) strict \( (adds_1) s t \implies s \notin lt \cdot ?Q \)
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 \( adds: lt g adds_1 lt f \) by simp all

show \( \langle \) thesis \( \rangle \)
proof (rule that)

\{
  fix \( f' \)
  assume \( f' \in F' \)
  assume \( lt f' \in adds_1 \) \( lt g \)
  hence \( lt f' \in adds_1 \) \( lt f \) using \( adds \) by (rule adds-term-trans)
  with \( f' \in F'' \) have \( f' \in ?Q \) by simp
  hence \( lt f' \in lt \cdot ?Q \) by (rule imageI)
  with \( t-min \) have \( \neg \) strict \( (adds_1) \) \( (lt f') \) \( (lt g) \) unfolding \( t \) by blast
  with \( (lt f' \in adds_1 \) \( lt g \) \( \) have \( lt g adds_1 \) \( lt f' \) by blast
  with \( (lt f' \in adds_1 \) \( lt g \) \( \) have \( lt f' = lt g \) by (rule adds-term-antisym)
  with \( (inj-on \) \( lt F' \) \( ) \) have \( f' = g \) using \( f' \in F'' \) \( \langle g \in F' \rangle \) by (rule inj-onD)

  with \( (q \in F' \) \( ) \) show \( g \in G \) by (simp add: G-def)

qed fact

have \( 2: is-red G q \) if \( q \in pmml F \) 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 \( *: lt f adds_1 t \)
by (rule keys-monomial-pmdl)
from this(1) \( , 2 \) obtain \( g \) where \( g \in G \) and \( lt g adds_1 \) \( lt f \) by (rule 1)
from this(2) have \( *: lt g adds_1 t \) using \( * \) by (rule adds-term-trans)
from \( g \in G \colon (G \subseteq F - \{0\}) \) have \( g \in F - \{0\} \).

hence \( g \neq 0 \) by simp

with \( (g \in G) \) show \( \text{thesis using } (t \in \text{keys } g) \) ** by (rule is-red-addsI)

qed

from \( (G \subseteq F - \{0\}) \) have \( G \subseteq F \) by blast

hence \( \text{pmdl } G \subseteq \text{pmdl } F \) by (rule pmdl.span-mono)

note dgrad fin-comps F-sub

moreover have \( \text{is-reduced-GB } (\text{monic } ' G) \) unfolding is-reduced-GB-def GB-image-monic

proof (intro conjI image-monic-is-auto-reduced image-monic-is-monic-set)

from dgrad show \( \text{is-Groebner-basis } G \)

proof (rule isGB-I-is-red)

from \( (G \subseteq F) \) F-sub show \( G \subseteq \text{dgrad-p-set } d \\ m \) by (rule subset-trans)

next

fix \( f \)

assume \( f \in \text{pmdl } G \)

hence \( f \in \text{pmdl } F \) using \( (\text{pmdl } G \subseteq \text{pmdl } F) \) ..

moreover assume \( f \neq 0 \)

ultimately show \( \text{is-red } G f \) by (rule 2)

qed

next

show \( \text{is-auto-reduced } G \) unfolding is-auto-reduced-def

proof (intro ballI notI)

fix \( g \)

assume \( g \in G \)

hence \( g \in F \) using \( (G \subseteq F) \) ..

with \( F\)-monom have \( \text{is-monomial } g \) by (rule is-monomial-setD)

hence \( \text{keys-g: keys } g = \{ \text{lt } g \} \) by (rule keys-monomial)

assume \( \text{is-red } (G - \{g\}) \)

then obtain \( g' t \) where \( g' \in G - \{g\} \) and \( t \in \text{keys } g \) and \( \text{adds: lt } g' \) \( t \) by (rule is-red-addsE)

from this(1) have \( g' \in F' \) and \( g' \neq g \) by (simp-all add: G-def)

from \( (t \in \text{keys } g) \) have \( t = \text{lt } g \) by (simp add: keys-g)

with \( (g \in G) \) \( (g' \in F') \) \( \text{adds have } g' = g \) by (simp add: G-def)

with \( (g' \neq g) \) show False ..

qed

next

show \( 0 \notin \text{monic } ' G \)

proof

assume \( 0 \in \text{monic } ' G \)

then obtain \( g \) where \( 0 = \text{monic } g \) and \( g \in G \) ..

moreover from this(2) \( (G \subseteq F - \{0\}) \) have \( g \neq 0 \) by blast

ultimately show False by (simp add: monic-0-iff)

qed

qed

moreover have \( \text{pmdl } (\text{monic } ' G) = \text{pmdl } F \) unfolding pmdl-image-monic

proof

show \( \text{pmdl } F \subseteq \text{pmdl } G \)

proof (rule pmdl.span-subset-spanI, rule)

fix \( f \)
assume \( f \in F \)

hence \( f \in \text{pmdl} F \) by (rule \text{pmdl.span-base})

\textit{note} dgrad

moreover from \( G \subseteq F \) \textit{F-sub} have \( G \subseteq \text{dgrad-p-set} d m \) by (rule \text{subset-trans})

moreover \textit{note} (\text{pmdl} G \subseteq \text{pmdl} F) \( 2 : f \in \text{pmdl F} \)

moreover from \( f \in F \) \textit{F-sub} have \( f \in \text{dgrad-p-set} d m .. \)

ultimately have \( (\text{red} G)^* f \emptyset \) by (rule \text{is-red-implies-0-red-dgrad-p-set})

thus \( f \in \text{pmdl} G \) by (rule \text{red-rtranclp-0-in-pmdl})

\textit{qed}

\textit{fact} ultimately have \( \text{reduced-GB} F = \text{monic ' G} \) by (rule \text{reduced-GB-unique-dgrad-p-set})

also from \( G \subseteq F \) have \( .. \subseteq \text{monic ' F} \) by (rule \text{image-mono})

finally show \( \text{thesis} \)

\textit{qed}

\textbf{corollary} \text{reduced-GB-is-monomial-set-dgrad-p-set: is-monomial-set (reduced-GB F)}

\textit{proof} (rule \text{is-monomial-setI})

fix \( g \)

assume \( g \in \text{reduced-GB} F \)

also have \( .. \subseteq \text{monic ' F} \) by (fact \text{reduced-GB-subset-monic-dgrad-p-set})

finally obtain \( f \) where \( f \in F \) and \( g = \text{monic ' f ..} \)

from \( F\text{-monom this(1)} \) have \( \text{is-monomial f} \) by (rule \text{is-monomial-setD})

hence card (\( \text{keys f} = 1 \)) by (simp only: \text{is-monomial-def})

hence \( f \neq 0 \) by auto

hence \( \text{lc f} \neq 0 \) by (rule \text{lc-not-0})

hence \( 1 / \text{lc f} \neq 0 \) by simp

hence \( \text{keys g} = (\oplus 0) \cdot \text{keys f} \) by (simp add: \text{keys-monom-mult monic-def g})

also from \( \text{refl} \) have \( .. = (\lambda x. x) \cdot \text{keys f} \) by (rule \text{image-cong}) (simp only: \text{splus-zero})

finally show \( \text{is-monomial g} \) using \( \langle \text{card (keys f) = 1} : \text{by (simp only: \text{is-monomial-def image-ident)} \rangle \)

\textit{qed}

\textbf{end}

\textbf{lemma} \text{is-red-reduced-GB-monomial-dgrad-set}:

\textit{assumes} finite \( \langle \text{component-of-term ' S} \rangle \) and \( \text{pp-of-term ' S} \subseteq \text{dgrad-set} d m \)

\textit{shows} \( \text{is-red (reduced-GB (monomial 1 ' S)) = is-red (monomial (1::'b::field) ' S)} \)

\textit{proof} \( \text{fix p} \)

let \( ?F = \text{monomial (1::'b) ' S} \)

from \text{assms(1)} have \( 1: \text{finite (component-of-term ' Keys ?F)} \) by (simp add: \text{Keys-def})

moreover from \text{assms(2)} have \( 2: ?F \subseteq \text{dgrad-p-set} d m \) by (auto simp: \text{dgrad-p-set-def})

moreover have \( \text{is-monomial-set ?F} \) by (auto intro: \text{is-monomial-setI monomial-is-monomial})
ultimately have $\text{reduced-GB } ?F \subseteq \text{monic } ?F$ by (rule reduced-GB-subset-monic-dgrad-p-set)
also have $\ldots = ?F$ by (auto simp: monic-def intro! image-eqI)
finally have $3: \text{reduced-GB } ?F \subseteq ?F$ .
show $\text{is-red (reduced-GB } ?F) \iff \text{is-red } ?F$ p
proof
assume $\text{is-red (reduced-GB } ?F) \ p$
thus $\text{is-red } ?F \ p$ using $3$ by (rule is-red-subset)
next
assume $\text{is-red } ?F \ p$
then obtain $f \ v$ where $f \in ?F$ and $v \in \text{keys } p$ and $f \neq 0$ and adds1: $lt \ f$
addsl $v$
  by (rule is-red-addsE)
from this(1) have $f \in \text{pmdl } ?F$ by (rule pmdl_span-base)
from dgrad 1 2 have $\text{is-Groebner-basis (reduced-GB } ?F)$ by (rule reduced-GB-is-GB-dgrad-p-set)
moreover from $f \in \text{pmdl } ?F$ dgrad 1 2 have $f \in \text{pmdl (reduced-GB } ?F)$
  by (simp only: reduced-GB-pmdl-dgrad-p-set)
ultimately obtain $g$ where $g \in \text{reduced-GB } ?F$ and $g \neq 0$ and $lt \ g$ adds1 $lt \ f$
  using $f \neq 0$ by (rule GB-adds-lt)
from this(2) adds1 have $lt \ g \ v$ by (rule adds-term-trans)
with $\langle g \in \text{reduced-GB } ?F \rangle \ (g \neq 0) \ (v \in \text{keys } p)$ show $\text{is-red (reduced-GB } ?F)$
p
  by (rule is-red-addsI)
qed

corollary $\text{is-red-reduced-GB-monomial-lt-GB-dgrad-p-set}$:
assumes $\text{finite (component-of-term } ?S \text{ Keys } G)$ and $G \subseteq \text{dgrad-p-set } d \ m$ and $0 \notin G$
shows $\text{is-red (reduced-GB (monomial (1::b::field) } ?S \text{ lt } G))} = \text{is-red } G$
proof --
let $?S = \text{lt } ?G$
let $?G = \text{monomial (1::b) } ?S$
from assms(3) have $?S \subseteq \text{Keys } G$ by (auto intro: lt-in-keys in-KeysI)
hence $\text{component-of-term } ?S \subseteq \text{component-of-term } \text{Keys } G$
  and $\ast: \text{pp-of-term } ?S \subseteq \text{pp-of-term } \text{Keys } G$ by (rule image-mono)+
from this(1) have $\text{finite (component-of-term } ?S)$ using assms(1) by (rule finite-subset)
moreover from $\ast$ have $\text{pp-of-term } ?S \subseteq \text{dgrad-set } d \ m$
proof (rule subset-trans)
  from assms(2) show $\text{pp-of-term } \text{Keys } G \subseteq \text{dgrad-set } d \ m$ by (auto simp: dgrad-p-set-def Keys-def)
qed
ultimately have $\text{is-red (reduced-GB } ?G) = \text{is-red } ?G$ by (rule is-red-reduced-GB-monomial-dgrad-set)
also from assms(3) have $\ldots = \text{is-red } G$ by (rule is-red-monomial-lt)
finally show $?thesis$. 
qed

lemma $\text{reduced-GB-monomial-lt-reduced-GB-dgrad-p-set}$:
assumes finite (component-of-term ' Keys F) and F ⊆ dgrad-p-set d m
shows reduced-GB (monomial 1 ' lt ' reduced-GB F) = monomial (1::'b::field) ' lt ' reduced-GB F
proof (rule reduced-GB-unique)
let ?G = reduced-GB F
let ?F = monomial (1::'b) ' lt ' ?G

from dgrad assms have 0 ∉ ?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 ∉ ?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 ∈ ?F
then obtain g where g ∈ ?G and f: f = monomial 1 (lt g) by blast
assume is-red (?F − {f}) f
then obtain f' v where f' ∈ ?F − {f} and v ∈ keys f and f' ≠ 0 and adds1: lt f' adds t v by (rule is-red-addsE)
from this(1) have f' ∈ ?F and f' ≠ f by simp-all
from this(1) obtain g' where g' ∈ ?G and f': f' = monomial 1 (lt g') by blast
from (v ∈ keys f) have v: v = lt g by (simp add: f)
from ar (g ∈ ?G) have ¬ is-red (?G − {g}) g by (rule is-auto-reducedD)
moreover have is-red (?G − {g}) g by (rule is-red-addsI)
from (g' ∈ ?G) (f' ≠ f) show g' ∈ ?G − {g} by (auto simp: f f')
next
from (g' ∈ ?G) (0 ∉ ?G) have g' ≠ 0 by blast
next
from (g ∈ ?G) (0 ∉ ?G) have g ≠ 0 by blast
thus lt g ∈ keys g by (rule lt-in-keys)
next
from adds1 show adds2: lt g' adds t lt g by (simp add: v f' lt-monomial)
qed
ultimately show False ..
qed
qed (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
have card \( (B \cup \{c\}) = \text{card} \ (\text{insert} \ c \ B) \) by simp
also have \( \ldots = \text{Suc} \ (\text{card} \ (B - \{c\})) \)
  by (rule card-insert, rule finite-subset, fact \( B \subseteq A \), fact)
finally show card \( (B \cup \{c\}) = \text{Suc} \ n \) unfolding \( B - \{c\} = B \); card.

next
show \( B \cup \{c\} \subseteq A \) unfolding Un-subset-iff
proof (intro conjI, fact)
  from \( \langle c \in ?C \rangle \) show \( \{c\} \subseteq A \) by auto
qed
qed

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

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

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

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

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

8.3 count-list

lemma count-list-eq-0-iff: $\text{count-list } \text{xs } x = 0 \iff x \notin \text{set } \text{xs}$
  by (induct \text{xs}) simp-all

lemma count-list-append: $\text{count-list } (\text{xs @ ys}) \, x = \text{count-list } \text{xs } x + \text{count-list } \text{ys } x$
  by (induct \text{xs}) simp-all

lemma count-list-map-ge:
  $\text{count-list } \text{xs } x \leq \text{count-list } (\text{map } f \, \text{xs}) \, (f \, x)$
  by (induct \text{xs}) simp-all

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

\textbf{lemma listset-Cons}: \( \text{listset} \ (x \ # \ xs) = (\bigcup y \in x. \ (\#) \ y \ # \ \text{listset} \ xs) \)
by (auto simp: set-Cons-def)

\textbf{lemma listset-ConsI}:
\( y \in x \implies \text{ys}' \in \text{listset} \ xs \implies \text{ys} = y \ # \ \text{ys}' \implies \text{ys} \in \text{listset} \ (x \ # \ xs) \)
by (simp add: set-Cons-def)

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

\textbf{lemma listsetI}:
\( \text{length} \ \text{ys} = \text{length} \ \text{xs} \implies (\forall i. \ i < \text{length} \ \text{xs} \implies \text{ys} ! i \in \text{xs} ! i) \implies \text{ys} \in \text{listset} \ \text{xs} \)
by (induct ys xs rule: list-induct2)
(simp-all, smt Suc-mono nth-Cons-0 nth-tl set-Cons-def zero-less-Suc)

\textbf{lemma listsetD}:
assumes \( \text{ys} \in \text{listset} \ \text{xs} \)
shows \( \text{length} \ \text{ys} = \text{length} \ \text{xs} \ \text{and} \ \forall i. \ i < \text{length} \ \text{xs} \implies \text{ys} ! i \in \text{xs} ! i \)
proof –
from assms have \( \text{length} \ \text{ys} = \text{length} \ \text{xs} \ \land \ (\forall i\ < \text{length} \ \text{xs}. \ \text{ys} ! i \in \text{xs} ! i) \) by (rule Cons.hyps)

thus \( \text{?case} \) by simp
next
\textbf{case} \( \text{Cons} \ x \ \text{xs} \)
from Cons.prems obtain \( y \ \text{ys}' \ \text{where} \ y \in x \ \text{and} \ \text{ys}' \in \text{listset} \ xs \ \text{and} \ \text{ys} = y \ # \ \text{ys}' \)
by (rule listset-ConsE)
from this(2) have \( \text{length} \ \text{ys}' = \text{length} \ \text{xs} \ \land \ (\forall i\ < \text{length} \ \text{xs}. \ \text{ys}' ! i \in \text{xs} ! i) \) by (rule Cons.hyps)
hence \( \text{1: length} \ \text{ys}' = \text{length} \ \text{xs} \) and \( \text{2: } \forall i. \ i < \text{length} \ \text{xs} \implies \text{ys}' ! i \in \text{xs} ! i \)
by simp-all
show \( \text{?case} \) proof (intro conjI allI impI)
fix \( i \)
assume \( i < \text{length} \ (x \ # \ xs) \)
show \( \text{ys} ! i \in (x \ # \ xs) ! i \)
proof (cases \( i \))
case \( 0 \)
with \( (y \in x) \) show \( \text{?thesis} \) by (simp add: ys)
next
case \( (\text{Suc} \ j) \)
with \( i < \text{length} \ (x \ # \ xs) \) have \( j < \text{length} \ \text{xs} \) by simp
hence \( \text{ys}' ! j \in \text{xs} ! j \) by (rule 2)
thus \( \exists i. \) i = Suc j

qed

qed (simp add: ys 1)

thus length ys = length xs and \( \forall i. \) i < length xs \( \rightarrow \) ys ! i \in xs ! i by simp-all

qed

lemma listset-singletonI: a \in A \Rightarrow ys = [a] \Rightarrow ys \in listset [A]

by simp

lemma listset-singletonE:
assumes ys \in listset [A]

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

using assms by auto

lemma listset-doubletonI:
\( \forall a \in A \), b \in B \Rightarrow ys = [a, b] \Rightarrow ys \in listset [A, B]

by (simp add: set-Cons-def)

lemma listset-doubletonE:
assumes ys \in listset [A, B]

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

using assms by (auto simp: set-Cons-def)

lemma listset-appendI:
ys1 \in listset xs1 \Rightarrow ys2 \in listset xs2 \Rightarrow ys = ys1 @ ys2 \Rightarrow ys \in listset (xs1 @ xs2)

by (induct xs1 arbitrary: thesis ys)

(induct, auto simp, simp del: listset.simps elim: listset-ConsE intro: listset-ConsI)

lemma listset-appendE:
assumes ys \in listset (xs1 @ xs2)

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

using assms

proof (induct xs1 arbitrary: thesis ys)

case Nil

have [] \in listset [] by simp

moreover from Nil(2) have ys \in listset xs2 by simp

ultimately show case by (rule Nil) simp

next

case (Cons x xs1)

from Cons.prems(2) have ys \in listset (x # (xs1 @ xs2)) by simp

then obtain y ys' where y \in x and ys' \in listset (xs1 @ xs2) and ys: ys = y # ys'

by (rule listset-ConsE)

from - this(2) obtain ys1 ys2 where ys1: ys1 \in listset xs1 and ys2 \in listset xs2

and ys': ys' = ys1 @ ys2 by (rule Cons.hyps)
proof (rule Cons.prems)
  from \( y \in x \): \( \text{ys1 refl show } y \# \text{ys1} \in \text{listset } (x \# \text{xs1}) \) by (rule listset-ConsI)
next
  show \( \text{ys} = (y \# \text{ys1}) \@ \text{ys2} \) by (simp add: \( \text{ys} \)'s)
qed fact
qed

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

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

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

proof (rule listsetI)

show \( \text{length } ys' = \text{length } xs' \) by (simp add: \( ys' \)-def \( \text{len-ys } \) \( \text{len-xs} \))

fix \( i \)
assume \( i < \text{length } xs' \)

hence \( i < \text{length } xs \) by (simp only: \( \text{len-xs} \))

hence \( i < \text{length } ys \) by simp

hence \( ys'! i = ys! (f i) \) by (rule 1)

also from assms (1) have \( \ldots \in xs! (f i) \)
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 \( \ldots = \{..<\text{length } xs'\} \) by (simp add: bij-betw-def)

finally show \( f i < \text{length } xs \) by simp

qed

also have ... = \( xs'! i \) 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 \( \forall x . y . \ x \in \text{set } xs \implies y \in x \implies f y \in x \)
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.prems (1) obtain \( y \) \( ys' \) where \( y \in x \) and \( ys' \in \text{listset } xs \) and \( ys' = y \# ys' \)
by (rule listset-ConsE)
show ?case
proof (rule listset-ConsI)
from \( \langle y \in x \rangle \) show \( f y \in x \) by (rule Cons.prems) 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.prems)

qed fact

qed (simp add: \( ys \))

58
lemma listset-closed-map2:
assumes ys1 ∈ listset xs and ys2 ∈ listset xs
and \( \forall x. y1. y2. x ∈ set xs \Rightarrow y1 ∈ x \Rightarrow y2 ∈ x \Rightarrow f y1 y2 ∈ x \)
shows \( \text{map2 } f \text{ ys1 ys2 } ∈ \text{ listset xs} \)
using assms
proof (induct xs arbitrary: ys1 ys2)
case Nil
from Nil(1) show ?case by simp
next
  case (Cons x xs)
  from Cons.prems(1) obtain y1 ys1' where y1 ∈ x and ys1' ∈ listset xs and 
  y1: ys1 = y1 ≠ ys1'
  by (rule listset-ConsE)
  from Cons.prems(2) obtain y2 ys2' where y2 ∈ x and ys2' ∈ listset xs and 
  ys2: ys2 = y2 ≠ ys2'
  by (rule listset-ConsE)
  show ?case by (rule listset-Cons)
  from - ⟨y1 ∈ x⟩ ⟨y2 ∈ x⟩ show f y1 y2 ∈ x by (rule Cons.prems) simp
next
  show map2 f ys1' ys2' ∈ listset xs
  proof (rule Cons.hyps)
    fix x' y1' y2'
    assume x' ∈ set xs
    hence x' ∈ set (x ≠ xs) by simp
    moreover assume y1' ∈ x' and y2' ∈ x'
    ultimately show f y1' y2' ∈ x' by (rule Cons.prems)
  qed
  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 \( \forall i. i < \text{length } ys \Rightarrow xs ! i \subseteq ys ! i \)
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 $z \in x$ and $zs \in \operatorname{listset} \, xs$ and $zs' : zs' = z \# zs$

by (rule \text{listset-ConsE})

have $0 < \text{length} \, (y \# ys)$ by simp

hence $(x \# xs) ! 0 \subseteq (y \# ys) ! 0$ by (rule \text{Cons.prems})

hence $x \subseteq y$ by simp

with $(z \in x)$ have $z \in y$ ..

moreover from $zs$ have $zs \in \operatorname{listset} \, ys$

proof
  
proof (rule \text{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 \text{Cons.prems})

thus $xs ! i \subseteq ys ! i$ by simp

qed

qed

ultimately show $zs' \in \operatorname{listset} \, (y \# ys)$ using $zs'$ by (rule \text{listset-ConsI})

qed

qed

end

9 Direct Decompositions and Hilbert Functions

theory \textit{Hilbert-Function}

imports \textit{Dube-Prelims Degree-Section HOL-Library.Permutation}

begin

9.1 Direct Decompositions

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

definition \textit{direct-decomp} :: \textit{'a set ⇒ \text{comm-monoid-add} \, set list ⇒ bool} where \textit{direct-decomp} \, A \, ss \iff \operatorname{bij-betw} \, \text{sum-list} \, (\operatorname{listset} \, ss) \, A

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

lemma \textit{direct-decompI-alt}:
  $(\forall \, qs. \, qs \in \operatorname{listset} \, ss \implies \text{sum-list} \, qs \in A) \implies (\forall a. \, a \in A \implies \exists! \, qs \in \operatorname{listset} \, ss. \, a = \text{sum-list} \, qs) \implies \text{direct-decomp} \, A \, ss$
  by (auto simp: \text{direct-decomp-def} \, \text{intro!} \, \text{bij-betwI}’) \, \text{blast}
lemma direct-decompD:
  assumes direct-decomp A ss
  shows \( qs \in \text{listset} \ ss \implies \text{sum-list} \ qs \in A \) and inj-on \text{sum-list} (\text{listset} \ ss)
  and \( \text{sum-list} \ ' \ \text{listset} \ ss = A \)
  using \text{assms} by (auto simp: direct-decomp-def bij-betw-def)

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

lemma direct-decomp-unique:
  \( \text{direct-decomp} \ A \ ss = \implies \exists q : q \in \text{listset} \ ss \)
  \( q \in \text{listset} \ ss \implies \text{sum-list} \ q = \text{sum-list} \ q' \)
  by (auto dest: direct-decompD simp: inj-on-def)

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

lemma mset-bij:
  assumes bij-betw f \{..<length xs\} \{..<length ys\} \wedge \forall i. i < length xs \Rightarrow xs ! i = ys ! f i
  shows mset xs = mset ys
proof -
  from \text{assms}(1) have 1: inj-on f \{0..<length xs\} and 2: \( f ' \ \{0..<length xs\} = \{0..<length ys\} \)
  by (simp-all add: bij-betw_def lessThan-atLeast0)
  let \( ?f = (!) \ ys o f \)
  have \( xs = \text{map} \ ?f \ \{0..<length xs\} \)
  unfolding list-eq-iff-nth-eq
proof (intro conjI allI impI)
fix i
assume i < length xs
hence xs ! i = ys ! f i by (rule assms(2))
also from (i < length xs) have ... = map (!) ys o f) [0..<length xs] ! i by simp
finally show xs ! i = map (!) ys o f) [0..<length xs] ! i .
qed simp

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

lemma direct-decomp-perm:
assumes direct-decomp A ss1 and perm ss1 ss2
shows direct-decomp A ss2

proof –
from assms(2) have len-ss1: length ss1 = length ss2 by (rule perm-length)
from assms(2) have \exists f. bij-betw f {..<length ss1} {..<length ss2} \land (\forall i<length ss1. ss1 ! i = ss2 ! f i)
by (rule permutation-Ex-bij)
then obtain f where f-bij: bij-betw f {..<length ss2} {..<length ss1}
and f: \A i. i < length ss2 \implies ss1 ! i = ss2 ! f i unfolding len-ss1 by blast

define g where g = inv-into {..<length ss2} f
from f-bij have g-bij: bij-betw g {..<length ss1} {..<length ss2}
unfolding g-def len-ss1 by (rule bij-betw-inv-into)
have f-g: f (g i) = i if i < length ss1 for i
proof –
from that f-bij have i \in f \cdot {..<length ss2} by (simp add: bij-betw-def len-ss1)
thus \?thesis by (simp only: f-inv-into-f g-def)
qed

have g-f: g (f i) = i if i < length ss2 for i
proof –
from f-bij have inj-on f {..<length ss2} by (simp only: bij-betw-def)
moreover from that have i \in {..<length ss2} by simp
ultimately show \?thesis by (simp add: g-def)
qed

have g: ss2 ! i = ss1 ! g i if i < length ss1 for i
proof –
from that have i \in {..<length ss2} by (simp add: len-ss1)
hence g i \in g \cdot {..<length ss2} by (rule image1)
also from g-bij have ... = {..<length ss2} by (simp only: len-ss1 bij-betw-def)
finally have g i < length ss2 by simp
hence ss1 ! g i = ss2 ! f (g i) by (rule f)
with that show \?thesis by (simp only: f-g)
qed

show ?thesis

proof (rule direct-decompI-alt)
  fix qs2
  assume qs2 ∈ listset ss2
  then obtain qs1 where qs1-in: qs1 ∈ listset ss1 and len-qs1: length qs1 = length qs2
    and *: ∃i. i < length qs2 ⇒ qs1 ! i = qs2 ! 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 {..<length qs1} {..<length qs2} by (simp only: len-qs1 len-ss1)
  hence mset qs1 = mset qs2 using * by (rule mset-bij) (simp only: len-qs1)
  hence sum-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: ∃i. i < length qs ⇒ qs2 ! i = qs ! i using g-bij g by (rule listset-permE) blast+
  show ∃!qs∈listset ss2. a = 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 {..<length qs2} {..<length qs} by (simp only: len-qs2 len-ss1)
      hence mset qs2 = mset qs using 1 by (rule mset-bij) (simp only: len-qs2)
      thus a2: a = sum-list qs2 by (simp only: a flip: sum-mset-sum-list)
    next
      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: ∃i. i < length qs' ⇒ qs1 ! i = qs' ! 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 {..<length qs1} {..<length qs'} by (simp only: len-qs1 len-ss1)
      hence mset qs1 = mset qs' using 2 by (rule mset-bij) (simp only: len-qs1)
      hence sum-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: length qs' = length qs2 by (simp only: len-qs2 len-qs len-ss1)

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

lemma direct-decomp-split-map:
direct-decomp A (map f ss) ⇒ direct-decomp A (map f (filter P ss) @ map f (filter (¬ P) ss))
proof (rule direct-decomp-perm)
show perm (map f ss) (map f (filter P ss) @ map f (filter (¬ P) ss))
proof (induct ss)
case Nil
  show ?case by simp
next
case (Cons s ss)
  show ?case
proof (cases P s)
  case True
  with Cons show ?thesis by simp
next
case False
  have map f (s # ss) = f s # map f ss by simp
  also from Cons have perm (f s # map f ss) (f s # map f (filter P ss) @ map f (filter (¬ P) ss))
    by (rule perm.intro)
  also have perm ... (map f (filter P ss) @ map f (s # filter (¬ P) ss))
    by (simp add: perm_append_Cons)
  also (trans) from False have ... = map f (filter P (s # ss)) @ map f (filter (¬ P) (s # ss))
    by simp
  finally show ?thesis .
qed
qed

64
lemmas \textit{direct-decomp-split} = \textit{direct-decomp-split-map}[\textit{where \textit{f}=id, simplified}]

\textbf{lemma} \textit{direct-decomp-direct-decomp}:  
\textit{assumes} \textit{direct-decomp \textit{A} (\textit{s} \# \textit{ss}) and direct-decomp \textit{s rs}}  
\textit{shows} \textit{direct-decomp \textit{A} (\textit{ss @ rs}) (is direct-decomp \textit{A} \textit{?ss})}  
\textit{proof} (rule \textit{direct-decompI-alt})  
\textit{fix} \textit{qs}  
\textit{assume} \textit{qs \in listset \textit{?ss}}  
\textit{then obtain} \textit{qs1 qs2 where} \textit{qs1: qs1 \in listset \textit{ss} and qs2: qs2 \in listset \textit{rs} and q}: \textit{qs} = \textit{qs1 @ qs2}  
\textit{by} (rule \textit{listset-appendE})  
\textit{have} \textit{sum-list qs} = \textit{sum-list ((sum-list qs2) \# qs1)} \textit{by} (simp add: \textit{qs add commute})  
\textit{also from} \textit{assms(1)} \textit{have} \ldots \in \textit{A}  
\textit{proof} (rule \textit{direct-decompD})  
\textit{from} \textit{assms(2)} \textit{qs2 have} \textit{sum-list qs2 \in s} \textit{by} (rule \textit{direct-decompD})  
\textit{thus} \textit{sum-list qs2 \# qs1 \in listset (s \# \textit{ss})} \textit{using} \textit{qs1 refl by} (rule \textit{listset-Consl})  
\textit{qed}  
\textit{finally show} \textit{sum-list qs \in A} .

\textbf{next}  
\textit{fix} \textit{a}  
\textit{assume} \textit{a \in A}  
\textit{with} \textit{assms(1)} \textit{obtain} \textit{qs1 where} \textit{qs1-in: qs1 \in listset (s \# ss) and a: a = sum-list qs1}  
\textit{by} (rule \textit{direct-decompE})  
\textit{from} \textit{qs1-in obtain} \textit{qs11 qs12 where} \textit{qs11 \in s and qs12-in: qs12 \in listset ss and qs1: qs1 = qs11 \# qs12} \textit{by} (rule \textit{listset-Consl})  
\textit{from} \textit{assms(2)} \textit{this(1) obtain} \textit{qs2 where} \textit{qs2-in: qs2 \in listset \textit{rs} and qs11: qs11 = sum-list qs2}  
\textit{by} (rule \textit{direct-decompE})  
\textit{let} \textit{qs = qs12 @ qs2}  
\textit{show} \exists!\textit{qs\in listset ?ss}. \textit{a = sum-list qs}  
\textit{proof} (intro \textit{ex1I conjI allI impI})  
\textit{from} \textit{qs12-in listset \textit{?ss} by} (rule \textit{listset-appendI})  
\textit{show} \textit{a = sum-list qs by} (simp add: \textit{a qs1 qs11 add commute})  

\textit{fix} \textit{qs0}  
\textit{assume} \textit{qs0 \in listset ?ss \wedge a = sum-list qs0}  
\textit{hence} \textit{qs0-in: qs0 \in listset ?ss and a2: a = sum-list qs0 by simp-all}  
\textit{from} \textit{this(1) obtain} \textit{qs01 qs02 where} \textit{qs01-in: qs01 \in listset ss and qs02-in: qs02 \in listset rs and qs0: qs0 = qs01 @ qs02} \textit{by} (rule \textit{listset-appendE})  
\textit{note} \textit{assms(1)}  
\textit{moreover from} \textit{- qs01-in refl have} \textit{sum-list qs02 \# qs01 \in listset (s \# ss) (is \textit{qs' \in -})}  
\textit{proof} (rule \textit{listset-Consl})  
\textit{from} \textit{assms(2) qs02-in show} \textit{sum-list qs02 \in s by} (rule \textit{direct-decompD})

\textbf{qed}
qed

moreover note \( qs' = \) sum-list \( qs1 \) by (simp add: \( qs0 \) a add.commute)

ultimately have \( qs' = qs11 \neq qs12 \) 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

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 \( (A::'a::semiring-0 set) ss \) \( \wedge a. x * a = x * b \implies a = b \)

shows direct-decomp \( (** x ' A) (map (** x) ss) \) (is direct-decomp \( ?A ?ss \))

proof (rule direct-decompI-alt)

fix \( qs \)

assume \( qs \in \) listset \( \) ?ss

then obtain \( qs0 \) where \( qs0-in: qs0 \in \) listset \( ss \) and \( qs: qs = map (** x) qs0 \)

by (rule listset-map-imageE)

have \( \) sum-list \( qs = x * \) sum-list \( qs0 \) by (simp only: \( qs \) sum-list-map-times)

moreover from assms(1) \( qs0-in \) have \( \) sum-list \( qs0 \) \( \in \) \( A \) by (rule direct-decompD)

ultimately show \( sum-list qs \in (** x ' A) \) by (rule image-eqI)

next

fix \( a \)

assume \( a \in ?A \)

then obtain \( a' \) where \( a' \in A \) and \( a: a = x * a' \).

from assms(1) \( this(1) \) obtain \( qs' \) where \( qs'-in: qs' \in \) listset \( ss \) and \( a': a' = \) sum-list \( qs' \)

by (rule direct-decompE)

define \( qs \) where \( qs = map (** x) qs' \)

show \( \exists qs \in \) listset \( ?ss. a = \) sum-list \( qs \)

proof (intro exI conjI allI implI)

from \( qs'-in \) \( qs-def \) show \( qs \in \) listset \( ?ss \) by (rule listset-map-imageI)

fix \( qs0 \)

assume \( qs0 \in \) listset \( ?ss \) \( \wedge a = \) sum-list \( qs0 \)

hence \( qs0 \in \) listset \( ?ss \) and \( a0: a = \) sum-list \( qs0 \) by simp-all

from this(1) obtain \( qs1 \) where \( qs1-in: qs1 \in \) listset \( ss \) and \( qs0: qs0 = map (** x) qs1 \)

by (rule listset-map-imageE)

show \( qs0 = qs \)

proof (cases \( x = 0 \))

case True

66
from qs1-in have length qs1 = length ss by (rule listsetD)
moreover from qs′-in have length qs′ = length ss by (rule listsetD)
ultimately show thesis by (simp add: qs-def qs0 list-eq-iff-nth-eq True)

next
case False
have x * sum-list qs1 = a by (simp only: a0 qs0 sum-list-map-times)
also have ... = x * sum-list qs′ by (simp only: a′ a)
finally have sum-list qs1 = 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

lemma direct-decomp-appendD:
  assumes direct-decomp A (ss1 @ ss2)
  shows \{\} \notin set ss2 \Longrightarrow direct-decomp (sum-list ' listset ss1) ss1 (is - \Longrightarrow thesis1)
    and \{\} \notin set ss1 \Longrightarrow direct-decomp (sum-list ' listset ss2) ss2 (is - \Longrightarrow thesis2)
    and direct-decomp A [sum-list ' listset ss1, sum-list ' listset ss2] (is direct-decomp - \& ss)
proof –
  have rl: direct-decomp (sum-list ' listset ts1) ts1
    if direct-decomp A (ts1 @ ts2) and \{\} \notin set ts2 for ts1 ts2
proof (intro direct-decompI inj-onI refl)
  fix qs1 qs2
  assume qs1: qs1 \in listset ts1 and qs2: qs2 \in listset ts1
  assume eq: sum-list qs1 = sum-list qs2
  from that(2) have listset ts2 \neq \{\} by (simp add: listset-empty-iff)
  then obtain qs3 where qs3: qs3 \in listset ts2 by blast
  note that(1)
  moreover from qs1 qs3 refl have qs1 @ qs3 \in listset (ts1 @ ts2) by (rule listset-appendI)
  moreover from qs2 qs3 refl have qs2 @ qs3 \in listset (ts1 @ ts2) by (rule listset-appendI)
  moreover have sum-list (qs1 @ qs3) = sum-list (qs2 @ qs3) by (simp add: eq)
  ultimately have qs1 @ qs3 = qs2 @ qs3 by (rule direct-decomp-unique)
  thus qs1 = qs2 by simp
  qed

  { assume \{\} \notin set ss2
    with assms show thesis1 by (rule rl)
  }
  { from assms perm-append-swap have direct-decomp A (ss2 @ ss1) by (rule direct-decomp-perm)
moreover assume \{\} \notin set ss1
ultimately show thesis2 by (rule rl)
}

direct-decomp A ?ss
proof (rule direct-decompI-alt)
fix qs
assume qs \in listset ?ss
then obtain q1 q2 where q1: q1 \in \text{sum-list} \cdot \text{listset} ss1 \land q2: q2 \in \text{sum-list} \cdot \text{listset} ss2
\land qs: qs = [q1, q2] by (rule listset-doubletonE)
from q1 obtain qs1 where qs1: qs1 \in listset ss1 \land q1: q1 = \text{sum-list} qs1
.. from q2 obtain qs2 where qs2: qs2 \in listset ss2 \land q2: q2 = \text{sum-list} qs2
.. from qs1 qs2 refl have \text{sum-list} qs \in A by (rule direct-decompD)
next
fix a
assume a \in A
with asms obtain qs0 where qs0-in: qs0 \in listset (ss1 @ ss2) \land a: a = \text{sum-list} qs0
by (rule direct-decompE)
from this(1) obtain qs1 qs2 where qs1: qs1 \in listset ss1 \land qs2: qs2 \in listset ss2
\land qs0: qs0 = qs1 @ qs2 by (rule listset-appendE)
from qs1 have len-qs1: length qs1 = length ss1 by (rule listsetD)
define qs where qs = [\text{sum-list} qs1, \text{sum-list} qs2]
show \exists!qs\in listset ?ss. a = \text{sum-list} qs
proof (intro ex1I conjI)
from qs1 have \text{sum-list} qs1 \in \text{sum-list} \cdot \text{listset} ss1 by (rule imageI)
moreover from qs2 have \text{sum-list} qs2 \in \text{sum-list} \cdot \text{listset} ss2 by (rule imageI)
ultimately show qs \in listset ?ss using qs-def by (rule listset-doubletonI)
fix qs'
assume qs' \in listset ?ss \land a = \text{sum-list} qs'
hence qs' \in listset ?ss \land a': a = \text{sum-list} qs' by simp-all
from this(1) obtain q1 q2 where q1: q1 \in listset \cdot \text{listset} ss1
\land q2: q2 \in \text{sum-list} \cdot \text{listset} ss2 \land qs': qs' = [q1, q2] by (rule listset-doubletonE)
from q1 obtain qs1' where qs1': qs1' \in listset ss1 \land q1: q1 = \text{sum-list} qs1'
.. from q2 obtain qs2' where qs2': qs2' \in listset ss2 \land q2: q2 = \text{sum-list} qs2'
.. from qs1' have len-qs1': length qs1' = length ss1 by (rule listsetD)
note asms
moreover from qs1' qs2' refl have qs1' @ qs2' \in listset (ss1 @ ss2) by
moreover note $qs0$-in

moreover have \texttt{sum-list ($qs1' @ qs2'$) = sum-list $qs0$ by (simp add: \texttt{a' $qs'$ flip: a $q1$ $q2$)}

ultimately have $qs1' @ qs2' = qs0$ by (rule direct-decomp-unique)

also have \ldots = $qs1 @ qs2$ by fact

finally show $qs' = qs$ by (simp add: \texttt{qs-def $qs'$ $q1$ $q2$ len-$qs1$ len-$qs1'$)}

qed (simp add: \texttt{qs-def a $qs0$})

qed

\textbf{lemma direct-decomp-Cons-zeroI:}

\textbf{assumes} direct-decomp $A$ $ss$

\textbf{shows} direct-decomp $A$ ($\{0\} \# ss$)

\textbf{proof} (rule direct-decompI-alt)

fix $qs$

assume $qs \in \text{listset} (\{0\} \# ss)$

then obtain $q$ $qs'$ where $q \in \{0\}$ and $qs' \in \text{listset} ss$ and $qs = q \# qs'$

by (rule listset-ConsE)

from this(1, 3) have \texttt{sum-list $qs = sum-list$ $qs'$ by simp}

also from \texttt{assms} ($qs' \in \text{listset} ss$) have \ldots \in $A$ by (rule direct-decompD)

finally show \texttt{sum-list $qs \in A$}.

next

fix $a$

assume $a \in A$

with \texttt{assms} obtain $qs'$ where $qs' : qs' \in \text{listset} ss$ and $a : a = \text{sum-list} qs'$

by (rule direct-decompE)

define $qs$ where $qs = 0 \# qs'$

show $\exists!qs. qs \in \text{listset} (\{0\} \# ss) \land a = \text{sum-list} qs$

\textbf{proof} (intro ex1I conjI)

from - $qs'$ $qs-def$ show $qs \in \text{listset} (\{0\} \# ss)$ by (rule listset-ConsI) simp

next

fix $qs0$

assume $qs0 \in \text{listset} (\{0\} \# ss) \land a = \text{sum-list} qs0$

hence $qs0 \in \text{listset} (\{0\} \# ss)$ and $a0 : a = \text{sum-list} qs0$ by simp-all

from this(1) obtain $q0$ $qs0'$ where $q0 \in \{0\}$ and $qs0' : qs0' \in \text{listset} ss$

and $qs0 : qs0 = q0 \# qs0'$ by (rule listset-ConsE)

from this(1, 3) have \texttt{sum-list $qs0' = sum-list$ $qs' by simp add: a0 flip: a$}

with \texttt{assms} $qs0' : qs' have $qs0' = qs' \land a = \texttt{sum-list}$ $qs' by$ (rule direct-decomp-unique)

with $q0 \in \{0\}$ show $qs0 = qs$ by (simp add: $qs-def$ $qs0$)

qed (simp add: $qs-def$ $a$)

\textbf{lemma direct-decomp-Cons-zeroD:}

\textbf{assumes} direct-decomp $A$ ($\{0\} \# ss$)

\textbf{shows} direct-decomp $A$ $ss$

\textbf{proof} –

have direct-decomp $\{0\} []$ by (simp add: direct-decomp-def bij-beta-def)

with \texttt{assms} have direct-decomp $A$ ($ss @ []$) by (rule direct-decomp-direct-decomp)
thus \textit{thesis} by simp

\textbf{qed}

\textbf{lemma} \textit{direct-decomp-Cons-subsetI}:
\begin{itemize}
\item \textbf{assumes} \textit{direct-decomp} \( A \) \((s \# ss)\)
\item \textbf{and} \( \land s0. s0 \in \text{set ss} \implies 0 \in s0 \)
\item \textbf{shows} \( s \subseteq A \)
\end{itemize}
\textbf{proof} \begin{itemize}
\item \textbf{fix} \( x \)
\item \textbf{assume} \( x \in s \)
\item \textbf{moreover from} \textit{assms}(2) \textbf{have} \textit{map} \((\lambda\cdot 0)\) \( ss \in \text{listset} ss \)
\item \textbf{ultimately have} \( x \# (\textit{map} \((\lambda\cdot 0)\) \( ss \)) \in \text{listset} (s \# ss) \) \textbf{using} \textit{refl} \textbf{by} \textit{rule} \textit{listset-ConsI}
\item \textbf{with} \textit{assms}(1) \textbf{have} \textit{sum-list} \((x \# (\textit{map} \((\lambda\cdot 0)\) \( ss \))) \in A \) \textbf{by} \textit{rule} \textit{direct-decompD}
\item \textbf{thus} \( x \in A \) \textbf{by} simp
\end{itemize}
\textbf{qed}

\textbf{lemma} \textit{direct-decomp-Int-zero}:
\begin{itemize}
\item \textbf{assumes} \textit{direct-decomp} \( A \) \( ss \) and \( i < j \) \textbf{and} \( j < \text{length} ss \) \textbf{and} \( \land s. s \in \text{set ss} \implies 0 \in s \)
\item \textbf{shows} \( ss ! i \cap ss ! j = \{0\} \)
\end{itemize}
\textbf{proof} \begin{itemize}
\item \textbf{from} \textit{assms}(2, 3) \textbf{have} \( i < \text{length} ss \) \textbf{by} \textit{rule} \textit{less-trans}
\item \textbf{hence} \( i\text{-in}: ss ! i \in \text{set ss} \) \textbf{by} simp
\item \textbf{from} \textit{assms}(3) \textbf{have} \( j\text{-in}: ss ! j \in \text{set ss} \) \textbf{by} simp
\item \textbf{show} \textit{thesis}
\item \textbf{proof}
\item \textbf{show} \( ss ! i \cap ss ! j \subseteq \{0\} \)
\item \textbf{proof}
\item \textbf{fix} \( x \)
\item \textbf{assume} \( x \in ss ! i \cap ss ! j \)
\item \textbf{hence} \( x\text{-i}: x \in ss ! i \) \textbf{and} \( x\text{-j}: x \in ss ! j \) \textbf{by} simp-all
\item \textbf{have} \( 1: (\textit{map} \((\lambda\cdot 0)\) ss)[k := y] \in \text{listset} ss \) \textbf{if} \( k < \text{length} ss \) \textbf{and} \( y \in ss ! k \)
\item \textbf{for} \( k y \)
\item \textbf{using} \textit{assms}(4) \textbf{that}
\item \textbf{proof} \textbf{(induct} \( ss \) \textbf{arbitrary:} \( k \))
\item \textbf{case} \textbf{Nil}
\item \textbf{from} \textbf{Nil}(2) \textbf{show} \textit{?case} \textbf{by} simp
\item \textbf{next}
\item \textbf{case} \( (\text{Cons} s ss) \)
\item \textbf{have} \( *: \land s'. s' \in \text{set ss} \implies 0 \in s' \) \textbf{by} \textit{rule} \textit{Cons.prems} \textbf{simp}
\item \textbf{show} \textit{?case}
\item \textbf{proof} \textbf{(cases} \( k \))
\item \textbf{case} \( k: 0 \)
\item \textbf{with} \textit{Cons.prems}(3) \textbf{have} \( y \in s \) \textbf{by} simp
\item \textbf{moreover from} \( * \) \textbf{have} \( \textit{map} \((\lambda\cdot 0)\) ss \in \text{listset} ss \)
\item \textbf{by} \textbf{(induct} \( ss \)) \textbf{(auto simp del:} \textit{listset.simps}(2) \textit{intro:} \textit{listset-ConsI})
\item \textbf{moreover have} \( (\textit{map} \((\lambda\cdot 0)\) (s \# ss))[k := y] = y \# (\textit{map} \((\lambda\cdot 0)\) ss) \) \textbf{by}
\item \textbf{simp add:} \( k \)
\end{itemize}

70
ultimately show \( \psi \)thesis by (rule listset-ConsI)

next

\begin{align*}
  \text{case } k \colon (\text{Suc } k') \\
  \text{have } 0 \in s \text{ by (rule Cons.prems) simp}
\end{align*}

moreover from * have \((\map \lambda \cdot. 0) s)[k' := y] \in \text{listset } ss

\begin{proof}
  \text{(rule Cons.hyps)}
  \begin{align*}
    \text{from Cons.prems(2) show } k' < \text{length } ss \text{ by (simp add: } k) \\
    \text{next}
  \end{align*}

  \text{from Cons.prems(3) show } y \in s \cdot k' \text{ by (simp add: } k)
\end{proof}

qed

moreover have \((\map \lambda \cdot. 0) (s \# ss))[k := y] = 0 \# (\map \lambda \cdot. 0) ss)[k' := y] \\
  \text{by (simp add: } k)

ultimately show \( \psi \)thesis by (rule listset-ConsI)

qed

qed

have 2: \text{sum-list } ((\map \lambda \cdot. 0) ss)[k := y] = y \text{ if } k < \text{length } ss \text{ for } k \text{ and } y::'a

using that by (induct ss arbitrary: } k) \text{ (auto simp: add-ac split: nat.split)}

\begin{align*}
  \text{define } qs1 \text{ where } qs1 = (\map \lambda \cdot. 0) ss)[i := x] \\
  \text{define } qs2 \text{ where } qs2 = (\map \lambda \cdot. 0) ss)[j := x]
\end{align*}

\text{note assms(1)}

moreover from \((i < \text{length } ss) \cdot x-i \text{ have } qs1 \in \text{listset } ss \text{ unfolding } qs1-def

\begin{proof}
  \text{by (rule 1)}
  \begin{align*}
    \text{moreover from } \text{assms(3)} \cdot \text{x-j have } qs2 \in \text{listset } ss \text{ unfolding } qs2-def \text{ by (rule 1)}
  \end{align*}

  \text{thin } \text{sum-list-update}

  \begin{align*}
    \text{moreover from } \text{(i < length ss) } \text{assms(3) have } \text{sum-list } qs1 = \text{sum-list } qs2 \\
    \text{by (simp add: qs1-def qs2-def 2)}
  \end{align*}

  \text{ultimately have } qs1 = qs2 \text{ by (rule direct-decomp-unique)}

  \text{hence } qs1 \cdot i = qs2 \cdot i \text{ by simp}

  \text{with } (i < \text{length ss}) \text{ assms(2, 3) show } x \in \{0\} \text{ by (simp add: qs1-def qs2-def)}

  \text{qed}

\text{next}

\text{from } i\text{-in have } 0 \in s \cdot i \text{ by (rule assms(4))}

\text{moreover from } j\text{-in have } 0 \in s \cdot j \text{ by (rule assms(4))}

\text{ultimately show } \{0\} \subseteq s \cdot i \cap s \cdot j \text{ by simp}

\text{qed}

\text{corollary direct-decomp-pairwise-zero: }

\text{assumes } \text{direct-decomp } A ss \text{ and } \bigwedge s. s \in \text{set } ss \implies 0 \in s

\text{shows } \text{pairwise } (\lambda s1 s2. s1 \cap s2 = \{0\}) \text{ (set } ss)

\text{proof } \text{(rule pairwiseI)}

\begin{align*}
  \text{fix } s1 s2 \\
  \text{assume } s1 \in \text{set } ss
\end{align*}

\text{then obtain } i \text{ where } i < \text{length } ss \text{ and } s1: s1 = s1 \cdot i \text{ by (metis in-set-conv-nth)}

\text{assume } s2 \in \text{set } ss

\text{then obtain } j \text{ where } j < \text{length } ss \text{ and } s2: s2 = s2 \cdot j \text{ by (metis in-set-conv-nth)}

71
assume $s_1 \neq s_2$

hence $i < j \lor j < i$ by (auto simp: $s_1 \ s_2$)
thus $s_1 \cap s_2 = \{0\}$
proof
  assume $i < j$
  with assms(1) show ?thesis unfolding $s_1 \ s_2$ using $j < \text{length \ ss}$ assms(2)
  by (rule direct-decomp-Int-zero)
next
  assume $j < i$
  with assms(1) have $s_2 \cap s_1 = \{0\}$ unfolding $s_1 \ s_2$ using $i < \text{length \ ss}$ assms(2)
  by (rule direct-decomp-Int-zero)
  thus ?thesis by (simp only: Int-commute)
qed

corollary direct-decomp-repeated-eq-zero:
  assumes direct-decomp $A \ \text{ss and} \ \ 1 < \text{count-list \ ss \ X \ and} \ \ \bigwedge \ s, \ s \in \text{set \ ss} \Longrightarrow 0 \in \ s$
  shows $X = \{0\}$
  proof
  - from assms(2) obtain $i \ j$ where $i < j$ and $j < \text{length \ ss}$ and $1: \ \text{ss} ! i = X$
  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 $A \ \text{(map \ f \ ss)} \ \text{and} \ \ s_1 \in \text{set \ ss} \ \text{and} \ \ s_2 \in \text{set \ ss} \ \text{and} \ s_1 \neq s_2$
  and $\bigwedge \ s, \ s \in \text{set \ ss} \Longrightarrow 0 \in f \ s$
  shows $f \ s_1 \cap f \ s_2 = \{0\}$
proof
  - from assms(2) obtain $i$ where $i < \text{length \ ss}$ and $s_1 : s_1 = s ! i$ by (metis \text{in-set-conv-nth})
    from this(1) have $i : i < \text{length \ (map \ f \ ss)}$ by simp
    from assms(3) obtain $j$ where $j < \text{length \ ss}$ and $s_2 : s_2 = s ! j$ by (metis \text{in-set-conv-nth})
    from this(1) have $j : j < \text{length \ (map \ f \ ss)}$ by simp
    have $s : 0 \in s$ if $s \in \text{set \ (map \ f \ ss)}$ for $s$
    proof
      - from that obtain $s'$ where $s' \in \text{set \ ss} \ \text{and} \ s : s = f \ s'$ unfolding set-map ..
        from this(1) show $0 \in s$ unfolding $s$ by (rule assms(5))
    qed
    show ?thesis
    proof (rule linorder-cases)
      assume $i < j$
    qed
with assms(1) have (map f ss) ! i ∩ (map f ss) ! j = {0}
  using j * by (rule direct-decomp-Int-zero)
with i j show ?thesis by (simp add: s1 s2)
next
  assume j < i
with assms(1) have (map f ss) ! j ∩ (map f ss) ! i = {0}
  using i * by (rule direct-decomp-Int-zero)
with i j show ?thesis by (simp add: s1 s2 Int-commute)
next
  assume i = j
with assms(4) show ?thesis by (simp add: s1 s2)
qed
qed

9.2 Direct Decompositions and Vector Spaces

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

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

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

context vector-space begin

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

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

lemma independent-UnI:
  assumes independent A and independent B and span A ∩ span B = {0}
  shows independent (A ∪ B)
proof
  from span-superset have A ∩ B ⊆ span A ∩ span B by blast
  hence A ∩ B = {} unfolding assms(3) using assms(1, 2) dependent-zero by blast
  assume dependent (A ∪ B)
  then obtain T u v where finite T and T ⊆ A ∪ B and eq: (∑ v∈T. u v * s v) = 0
    and v ∈ T and u v ≠ 0 unfolding dependent-explicit by blast
define TA where TA = T ∩ A
define TB where TB = T ∩ B
from (T ⊆ A ∪ B) have T: T = TA ∪ TB by (auto simp: TA-def TB-def)
from (finite T) have finite TA and TA ⊆ A by (simp-all add: TA-def)
from (finite T) have finite TB and TB ⊆ B by (simp-all add: TB-def)
from (A ∩ B = {}) ⟨TA ⊆ A⟩ this(2) have TA ∩ TB = {} by blast
have θ = (∑ v ∈ TA ∪ TB. u v * s v) by (simp only: eq flip: T)
also have \ldots = (∑ v ∈ TA. u v * s v) + (∑ v ∈ TB. u v * s v) by (rule sum.union-disjoint)

finally have (∑ v ∈ TA. u v * s v) = (∑ v ∈ TB. (− u) v * s v) (is {?x = ?y)
    by (simp add: sum-negf eq-neg-iff-add-eq-0)
from (finite TB) ⟨TB ⊆ B⟩ have ?y ∈ span B by (auto simp: span-explicit simp del: uminus-apply)
     moreover from (finite TA) ⟨TA ⊆ A⟩ have {?x ∈ span A by (auto simp: span-explicit)
ultimately have ?y ∈ span A ∩ span B by (simp add: {?x = ?y)
    hence ?x = 0 and ?y = 0 by (simp-all add: {?x = ?y) assms(3))
from (v ∈ T) have v ∈ TA ∪ TB by (simp only: T)
    hence u v = 0

proof
    assume v ∈ TA
    with assms(1) (finite TA) ⟨TA ⊆ A⟩ ⟨?x = 0⟩ show u v = 0 by (rule independentD)

next
    assume v ∈ TB
    with assms(2) (finite TB) ⟨TB ⊆ B⟩ ⟨?y = 0⟩ have (− u) v = 0 by (rule independentD)
    thus u v = 0 by simp

qed
with ⟨u v ≠ 0⟩ show False ..

qed

lemma subspace-direct-decomp:
assumes direct-decomp A ss and \( \bigwedge s \in \text{set } ss \rightarrow \text{subspace } s \)
shows subspace A
proof (rule subspaceI)
    let ?qs = map (λs.. 0) ss
from assms(2) have ?qs ∈ listset ss
    by (induct ss) (auto simp del: listset.simps(2) dest: subspace-0 intro: listset-ConsI)
with assms(1) have sum-list ?qs ∈ A by (rule direct-decompD)
    thus 0 ∈ A by simp

next
    fix p q
    assume p ∈ A
with assms(1) obtain ps where ps: ps ∈ listset ss and p: p = sum-list ps by (rule direct-decompE)
    assume q ∈ A
with assms(1) obtain qs where qs: qs ∈ listset ss and q: q = sum-list qs by (rule direct-decompE)
from ps qs have l: length ps = length qs by (simp only: listsetD)
from ps qs have map2 (+) ps qs ∈ listset ss (is ?qs ∈ )
    by (rule listset-closed-map2) (auto dest: assms(2) subspace-add)
with assms(1) have sum-list ?qs ∈ A by (rule direct-decompD)

74
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 $ps \in \text{listset } ss$ and $p: p = \text{sum-list } ps$ by (rule direct-decompE)
  from this(1) have $\text{map } ((\ast s) \ c) \ ps \in \text{listset } ss$ (is $qs \in -$)
    by (rule listset-closed-map) (auto dest: assms(2) subspace-scale)
  with assms(1) have $\text{sum-list } qs = c * s \text{sum-list } ps$ by (induct ps) (simp-all add: scale-right-distrib)
  finally show $c * s \in A$ by (simp only: $p$)
qed

lemma $\text{is-basis-alt}$: $\text{subspace } V \Rightarrow \text{is-basis } V \ B \iff (\text{independent } B \land \text{span } B = V)$
  by (metis (full-types) is-basis-def dim-eq-card span-eq span-eq-iff)

lemma $\text{is-basis-finite}$: $\text{is-basis } V \ A \Rightarrow \text{is-basis } V \ B \Rightarrow \text{finite } A \iff \text{finite } B$
  unfolding $\text{is-basis-def}$ using independent-span-bound by auto

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

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

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

lemma $\text{span-some-basis}$: $\text{subspace } V \Rightarrow \text{span } (\text{some-basis } V) = V$
  by (simp add: span-subspace some-basis-subset span-some-basis-supset)

lemma $\text{direct-decomp-some-basis-pairwise-disjnt}$:
  assumes $\text{direct-decomp } A \ ss$ and $\land_s. \ s \in \text{set } ss \Rightarrow \text{subspace } s$
  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 $\text{some-basis } s1 \cap \text{some-basis } s2 \subseteq s1 \cap s2$ using some-basis-subset by blast
also from direct-decomp-pairwise-zero have \ldots = \{0\}

proof (rule pairwiseD)
  fix s
  assume s ∈ set ss
  hence subspace s by (rule assms(2))
  thus ∅ ∈ s by (rule subspace-∅)
qed

fact

finally have some-basis s1 ∩ some-basis s2 ⊆ \{0\}.
with some-basis-not-zero show disjoint (some-basis s1) (some-basis s2)
  unfolding disjoint_def by blast
qed

lemma direct-decomp-span-some-basis:
  assumes direct-decomp A ss and \( \forall \mathbf{s} \in \text{set ss} = \Rightarrow \text{subspace} \mathbf{s} \)
  shows \( \text{span} \left( \bigcup (\text{some-basis} ' \text{set ss}) \right) = A \)
proof
  from assms(1) have eq0[symmetric]: sum-list ' listset ss = A by (rule direct-decompD)
  show ?thesis unfolding eq0 using assms(2)
  proof (induct ss)
    case Nil
    show ?case by simp
  next
    case (Cons s ss)
    have subspace s by (rule Cons.prems simp)
    hence eq1: \text{span} (\text{some-basis} s) = s by (rule span-some-basis)
    have \( \forall \mathbf{s'} \in \text{set ss} = \Rightarrow \text{subspace} \mathbf{s'} \)
      by (rule span-some-basis)
    hence eq2: \text{span} (\bigcup (\text{some-basis} ' \text{set ss})) = \text{sum-list} \text{listset ss}
      by (rule Cons.hyps)
    have \text{span} (\bigcup (\text{some-basis} ' \text{set ss})) = \{x + y | x, y \in \text{sum-list}
      \text{listset ss}\}
      by (simp add: span-Un eq1 eq2)
    also have \ldots = \text{sum-list} \text{listset} (s # ss) (is ?A = ?B)
    proof
      show ?A ⊆ ?B
      proof
        fix a
        assume a ∈ ?A
        then obtain x y where x ∈ s and y ∈ \text{sum-list} \text{listset ss and} a: a = x + y
        by blast
        from this(2) obtain qs where qs ∈ listset ss and y: y = \text{sum-list} qs ..
        from ⟨x ∈ s⟩ this(1) refl have x # qs ∈ listset (s # ss) by (rule listset-ConsI)
        hence sum-list (x # qs) ∈ ?B by (rule imageI)
        also have sum-list (x # qs) = a by (simp add: a y)
        finally show a ∈ ?B.
      qed
    next
    show ?B ⊆ ?A
    proof
      fix a
assume $a \in ?B$
then obtain $qs'$ where $qs' \in \text{listset}(s \# ss)$ and $a = \text{sum-list} qs'$
from this(1) obtain $x$ where $x \in s$ and $qs \in \text{listset} ss$ and $qs': qs' = x \# qs$
by (rule listset-ConsE)
from this(2) have $\text{sum-list} qs \in \text{sum-list} \": \text{listset} ss"$ by (rule imageI)
moreover have $a = x + \text{sum-list} qs$ by (simp add: $a \: qs'$)
ultimately show $a \in ?A$ using $(x \in s)$ by blast
qed

finally show $?case$.

qed

lemma direct-decomp-independent-some-basis:
assumes direct-decomp $A \: ss$ and $\forall \: s. \: s \in \text{set} \: ss \: \Rightarrow \: \text{subspace} \: s$
shows independent $(\bigcup (\text{some-basis} \: \" \text{set} \: ss\"))$
using assms
proof (induct ss arbitrary: $A$)
case Nil
from independent-empty show $?case$ by simp
next
case (Cons $s \: ss$)
have 1: $\\forall s', \: s' \in \text{set} \: ss \: \Rightarrow \: \text{subspace} \: s'$ by (rule Cons.premis) simp
have subspace $s$ by (rule Cons.premis) simp
hence $0 \in s$ and eq1: span $(\text{some-basis} \: s)$ = $s$ by (rule subspace-0, rule span-some-basis)
from Cons.premis(1) have *: direct-decomp $A \: ( [s] \: @ \: ss)$ by simp
moreover from $\{} \not\in \text{set} \: [s]$ by auto
ultimately have 2: direct-decomp $(\text{sum-list} \: \" \text{listset} ss\")$ ss by (rule direct-decomp-appendD)
hence eq2: span $(\bigcup (\text{some-basis} \: \" \text{set} \: ss\"))$ = $\text{sum-list} \: \" \text{listset} ss"$ using 1
by (rule direct-decomp-span-some-basis)

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

corollary direct-decomp-is-basis:
assumes direct-decomp A ss and \( s \in \text{set ss} \implies \text{subspace s} \)
shows is-basis \( A (\bigcup \{\text{some-basis } s \mid \text{set ss}\}) \)
proof
  from assms have subspace A by (rule subspace-direct-decomp)
  moreover from assms have span \( (\bigcup \{\text{some-basis } s \mid \text{set ss}\}) = A \)
    by (rule direct-decomp-span-some-basis)
  moreover from assms have independent \( (\bigcup \{\text{some-basis } s \mid \text{set ss}\}) \)
    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 \( s \in \text{set ss} \implies \text{subspace s} \)
shows \( \dim A = (\sum s \in \text{set ss}. \dim s) \)
proof
  from assms(1, 4) have is-basis \( A (\bigcup \{\text{some-basis } s \mid \text{set ss}\}) \)
    by (rule direct-decomp-is-basis)
  hence \( \dim A = \card \ ?B \) 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 \( \dim A = \card \ ?B \)
  also from finite-set have \( \card \ ?B = (\sum s \in \text{set ss}. \card \{\text{some-basis } s\}) \)
  proof (intro card-UN-disjoint ballI impI)
  fix \( s \)
  assume \( s \in \text{set ss} \)
  with \( \text{finite ?B} \) show \( \text{finite } \{\text{some-basis } s\} \) 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 \( s1 \neq s2 \)
  thus pairwiseD
  ultimately have disjoint \( \{\text{some-basis } s1\} \) \( \{\text{some-basis } s2\} \) by (rule pairwiseD)
  thus \( \text{some-basis } s1 \cap \text{some-basis } s2 = \{\} \) by (simp only: disjoint-def)
  qed
  also from refl card-some-basis have \( \ldots = (\sum s \in \text{set ss}. \dim s) \) 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 \( \forall s \in \text{set ss} \Rightarrow \text{homogeneous-set s} \)
  shows homogeneous-set A
proof (rule homogeneous-setI)
  fix a n
  assume a \( \in \) A
  with assms (1) obtain qs where qs \( \in \) listset ss and a = \( \text{sum-list} \) qs
  by (rule direct-decompE)
  have hom-component a n = hom-component (\( \text{sum-list} \) qs) n
  by (simp only: a)
  also have \( \ldots = \text{sum-list} \) (\( \text{map} \) (\( \lambda \) q. hom-component q n) qs)
  by (induct qs) (simp-all add: hom-component-plus)
  also from assms (1) have \( \ldots \in A \)
proof (rule direct-decompD)
  show \( \text{map} \) (\( \lambda \) q. hom-component q n) qs \( \in \) listset ss
  proof (rule listset-closed-map)
    fix s q
    assume s \( \in \) 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
  fact
  finally show hom-component a n \( \in \) A .
  qed

definition hom-deg-set :: nat \Rightarrow (('x \Rightarrow \text{nat}) \Rightarrow a) \Rightarrow \text{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 \( \Rightarrow \) \( \text{poly-deg} \) p = z
proof
  from assms obtain a where a \( \in \) A and 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 (t \( \in \) keys p)
  have deg-pm t = z unfolding p
  by (rule keys-hom-componentD)
  ultimately show poly-deg p = z
  by simp
  qed

lemma zero-in-hom-deg-set:
  assumes 0 \( \in \) A
  shows 0 \( \in \) hom-deg-set z A

79
proof
have 0 = hom-component 0 z by simp
also from assms have ... ∈ hom-deg-set z A unfolding hom-deg-set-def by
(rule imageI)
finally show ?thesis.
qed

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

lemma hom-deg-set-closed-plus:
assumes ⋀a1 a2. a1 ∈ A ⇒ a2 ∈ A ⇒ a1 + a2 ∈ A
and p ∈ hom-deg-set z A and q ∈ hom-deg-set z A
shows p + q ∈ hom-deg-set z A
proof –
from assms(2) obtain a1 where a1 ∈ A and p: p = hom-component a1 z
unfolding hom-deg-set-def ..
from assms(3) obtain a2 where a2 ∈ A and q: q = hom-component a2 z
unfolding hom-deg-set-def ..
from ⟨a1 ∈ A; this(1)⟩ have a1 + a2 ∈ A by (rule assms(1))
moreover have p + q = hom-component (a1 + a2) z by (simp only: p q
hom-component-plus)
ultimately show ?thesis unfolding hom-deg-set-def by (rule rev-image-eqI)
qed

lemma hom-deg-set-closed-minus:
assumes ⋀a1 a2. a1 ∈ A ⇒ a2 ∈ A ⇒ a1 − a2 ∈ A
and p ∈ hom-deg-set z A and q ∈ hom-deg-set z A
shows p − q ∈ hom-deg-set z A
proof –
from assms(2) obtain a1 where a1 ∈ A and p: p = hom-component a1 z
unfolding hom-deg-set-def ..
from assms(3) obtain a2 where a2 ∈ A and q: q = hom-component a2 z
unfolding hom-deg-set-def ..
from ⟨a1 ∈ A; this(1)⟩ have a1 − a2 ∈ A by (rule assms(1))
moreover have p − q = hom-component (a1 − a2) z by (simp only: p q
hom-component-minus)
ultimately show ?thesis unfolding hom-deg-set-def by (rule rev-image-eqI)
qed

lemma hom-deg-set-closed-scalar:
assumes $\bigwedge a. a \in A \Rightarrow c \cdot a \in A$ 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 $\bigwedge a1 a2. a1 \in A \Rightarrow a2 \in A \Rightarrow a1 + a2 \in A$
and $\bigwedge i. i \in I \Rightarrow f i \in \text{hom-deg-set} z A$
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 $\text{sum} f (\text{insert} j I) = f j + \text{sum} f I$ by simp
also from assms(2) have $\ldots \in \text{hom-deg-set} z A$
proof (intro hom-deg-set-closed-plus insert.hyps)
show $f j \in \text{hom-deg-set} z A$ by (rule insert.prems) simp
next
fix $i$
assume $i \in I$

hence $i \in \text{insert} j I$ by simp
thus $f i \in \text{hom-deg-set} z A$ by (rule insert.prems)
qed
finally show $?case$.
qed

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

lemma Polys-closed-hom-deg-set:
assumes $A \subseteq P[X]$
shows $\text{hom-deg-set} z A \subseteq P[X]$
proof
fix $p$
assume $p \in \text{hom-deg-set} z A$

then obtain $p' \in A$ and $p: p = \text{hom-component} p' z$ unfolding hom-deg-set-def.
from this(1) assms have $p' \in P[X]$.
have keys $p \subseteq keys p'$ by (simp add: p keys-hom-component)
also from $\langle p' \in P[X] \rangle$ have $\ldots \subseteq [X]$ 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 hom-deg-set $z A = \{ p \in A. \text{homogeneous } p \land (p = 0 \lor \text{poly-deg } p = z) \}$
(is $?A = ?B$)
proof
  show $?A \subseteq ?B$
proof
    fix $h$
    assume $h \in ?A$
    also from assms have $\ldots \subseteq A$ 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 $h = 0 \lor \text{poly-deg } h = z$ 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 $\text{thesis unfolding } (h = 0)$ by (rule zero-in-hom-deg-set)
next
      assume $\text{poly-deg } h = z$
      with $\langle \text{homogeneous } h \rangle$ have $h = \text{hom-component } h z$ by (simp add: hom-component-of-homogeneous)
      with $\langle \langle h \in A \rangle \rangle$ show $\text{thesis unfolding } \text{hom-deg-set-def}$ 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 $z A = \text{sum-list } \text{listset} (\text{map (hom-deg-set } z) \text{ ss})$ (is $?A = ?B$)
proof
  show $?A \subseteq ?B$
proof
    fix $h$
    assume $h \in ?A$
    then obtain $a$ where $a \in A$ and $h = \text{hom-component } a z$ 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 ..

82
have $h = \text{hom-component} (\text{sum-list} \ qs) \ z$ by (simp only: a h)
also have \ldots = \text{sum-list} (\map (\lambda q. \text{hom-component} \ q \ z) \ qs)
  by (induct qs) (simp-all add: hom-component-plus)
also have \ldots \in \mathcal{?B}
proof (rule imageI)
  show \map (\lambda q. \text{hom-component} \ q \ z) \ qs \in \text{listset} (\map (\text{hom-deg-set} \ z) \ \text{ss})
    unfolding hom-deg-set-def using \langle qs \in \text{listset} \ \text{ss} \rangle \text{refl} by (rule listset-map-imageI)
qed
finally show $h \in \mathcal{?B}$ .
qed

next
show \mathcal{?B} \subseteq \mathcal{?A}
proof
  fix $h$
  assume $h \in \mathcal{?B}$
  then obtain $qs$ where $qs \in \text{listset} (\map (\text{hom-deg-set} \ z) \ \text{ss})$ and $h': h = \text{sum-list} \ qs$ .
  from this(1) obtain $qs'$ where $qs' \in \text{listset} \ \text{ss}$ and $qs: qs = \map (\lambda q. \text{hom-component} \ q \ z) \ qs'$
    unfolding hom-deg-set-def by (rule listset-map-imageE)
  have $h = \text{sum-list} (\map (\lambda q. \text{hom-component} \ q \ z) \ qs')$ by (simp only: h qs)
  also have \ldots = \text{hom-component} (\text{sum-list} \ qs') \ z$ by (induct qs') (simp-all add: hom-component-plus)
  finally have $h = \text{hom-component} (\text{sum-list} \ qs') \ z$ .
  moreover have $\text{sum-list} \ qs' \in \mathcal{?A}$ unfolding assms using \langle qs' \in \text{listset} \ \text{ss} \rangle by (rule imageI)
    ultimately show $h \in \mathcal{?A}$ unfolding hom-deg-set-def by (rule image-eqI)
  qed
qed

lemma direct-decomp-hom-deg-set:
assumes direct-decomp $A \ \text{ss}$ and $\forall s. \ s \in \text{set} \ \text{ss} \Longrightarrow \text{homogeneous-set} \ s$
shows direct-decomp (\text{hom-deg-set} \ z \ \text{A}) (\map (\text{hom-deg-set} \ z) \ \text{ss})
proof (rule direct-decompI)
  from assms(1) have $\text{sum-list} \ \text{'} \ \text{listset} \ \text{ss} = \mathcal{?A}$ by (rule direct-decompD)
  from this[symmetric] show $\text{sum-list} \ \text{'} \ \text{listset} (\map (\text{hom-deg-set} \ z) \ \text{ss}) = \text{hom-deg-set} \ z \ \mathcal{?A}$
    by (simp only: hom-deg-set-sum-list-listset)
next
from assms(1) have $\text{inj-on} \ \text{sum-list} \ (\text{listset} \ \text{ss})$ by (rule direct-decompD)
moreover have $\text{listset} (\map (\text{hom-deg-set} \ z) \ \text{ss}) \subseteq \text{listset} \ \text{ss}$
proof (rule listset Mono)
  fix $i$
  assume $i < \text{length} \ \text{ss}$
  hence $\map (\text{hom-deg-set} \ z) \ \text{ss} \ \text{'} \ i = \text{hom-deg-set} \ z \ (\text{ss} \ \text{'} \ i)$ by simp
  also from $i < \text{length} \ \text{ss}$ have \ldots $\subseteq \text{ss}$ \ i by (intro hom-deg-set-subset assms(2) nth-mem)
    finally show $\map (\text{hom-deg-set} \ z) \ \text{ss} \ \text{'} \ i \subseteq \text{ss} \ \text{'} \ i$ .
  qed simp

83
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 ⇒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) ⇒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 c p
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

84
by (simp only: hom-deg-set-alt-homogeneous-set[OF homogeneous-set-Polys])
finally have \( p \in P[X] \) and homogeneous \( p \) and \( p \neq 0 \) \( \implies \) poly-deg \( p \) = \( z \) by simp-all
thus \( p \in \mathcal{B} \)
proof (induct \( p \) rule: poly-mapping-plus-induct)
case 1
from \( \text{phull.span-zero} \) show ?case .
next
case (2 \( c \) \( t \))
let \( \mathfrak{m} = \text{monomial} c t \)
from \( 2(1) \) have \( t \in \text{keys} \mathfrak{m} \) by simp
hence \( t \in \text{keys} (\mathfrak{m} + p) \) using \( 2(2) \) by (rule in-keys-plusI1)
hence \( \mathfrak{m} + p \neq 0 \) by auto
hence poly-deg (monomial \( c \) \( t \) + \( p \)) = \( z \) by (rule 2)
from \( 2(4) \) have keys (\( \mathfrak{m} + p \)) \( \subseteq \{ X \} \) by (rule PolysD)
with \( (t \in \text{keys} (\mathfrak{m} + p)) \) have \( t \in \{ X \} \) ..
hence \( \mathfrak{m} + p \in P[X] \) by (rule Polys-closed-monomial)
have \( t \in \text{deg-sect} X z \)
proof (rule deg-sectI)
from \( 2(5) \) \( (t \in \text{keys} (\mathfrak{m} + p)) \) have deg-pm \( t \) = poly-deg (\( \mathfrak{m} + p \))
by (rule homogeneousD-poly-deg)
also have \( \ldots = z \) by fact
finally show deg-pm \( t \) = \( z \) .
qed fact
hence monomial 1 \( t \in \) monomial 1 \( ^{\ast} \) deg-sect \( X \) \( z \) by (rule imageI)
hence monomial 1 \( t \in \mathcal{B} \) by (rule phull.span-base)
hence \( c \cdot \text{monomial} 1 \) \( t \in \mathcal{B} \) by (rule phull.span-scale)
hence \( \mathfrak{m} \in \mathcal{B} \) by simp
moreover have \( p \in \mathcal{B} \)
proof (rule 2)
from \( 2(4) \) \( (\mathfrak{m} \in P[X]) \) have (\( \mathfrak{m} + p \)) - \( \mathfrak{m} \in P[X] \) by (rule Polys-closed-minus)
thus \( p \in P[X] \) by simp
next
have 1: deg-pm \( s = z \) if \( s \in \text{keys} p \) for \( s \)
proof
from that \( 2(2) \) have \( s \neq t \) by blast
hence \( s \notin \text{keys} \mathfrak{m} \) by simp
with that have \( s \in \text{keys} (\mathfrak{m} + p) \) by (rule in-keys-plusI2)
with \( 2(5) \) have deg-pm \( s \) = poly-deg (\( \mathfrak{m} + p \)) by (rule homogeneousD-poly-deg)
also have \( \ldots = z \) by fact
finally show \( \text{thesis} \) .
qed
show homogeneous \( p \) by (rule homogeneousI) (simp add: 1)
assume \( p \neq 0 \)
show poly-deg \( p \) = \( z \)
proof (rule antisym)
show poly-deg \( p \) \( \leq \) \( z \) by (rule poly-deg-leI) (simp add: 1)
next
from \( p \neq 0 \) have 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 \( s \in \text{keys } p \) have \( \ldots \leq \text{poly-deg } p \) by (rule poly-deg-max-keys)
finally show \( z \leq \text{poly-deg } p \).
qed
qed
ultimately show \(?\text{case }\) by (rule phull.span-add)
qed
qed

next
show \(?B \subseteq ?A\)
proof
fix \( p \)
assume \( p \in ?B \)
then obtain \( M \ u \) where \( M \subseteq \text{monomial 1 \cdot deg-sect } X \ z \) and finite \( M \) and
\( p : p = (\sum m \in M. \ u \ m \cdot m) \)
by (auto simp: phull.span-explicit)
from this(1) obtain \( T \) where \( T \subseteq \text{deg-sect } X \ z \) and \( M : M = \text{monomial 1 \cdot } T \)
and \( \text{inj: inj-on (monomial (1::'a)) } T \) by (rule subset-imageE-inj)
define \( c \) where \( c = (\lambda t. \ u \ (\text{monomial 1 } t)) \)
from \( \text{inj} \) have \( p = (\sum t \in T. \ \text{monomial (c } t) \) by (simp add: \( p \ M \) sum.reindex)
c-def)
also have \( \ldots \in ?A\)
proof (intro hom-deg-set-closed-sum zero-in-Polys Polys-closed-plus)
fix \( t \)
assume \( t \in T \)
then \( \text{in } \text{deg-sect } X \ z \) using \( T \subseteq \text{deg-sect } X \ z \)\)
hence \( t \in [X] \) and \( \text{eq: deg-pm } t = z \) by (rule deg-sectD+)+
from this(1) have \( \text{monomial (c } t) \) \( t \in P[X] \) \( \text{is } ?m \in -\) by (rule Polys-closed-monomial)
thus \( ?m \in ?A \)
by (simp add: hom-deg-set-alt-homogeneous-set[OF homogeneous-set-Polys]
poly-deg-monomial
monomial-0-iff eq)
qed
finally show \( p \in ?A \).
qed
qed

9.5 (Projective) Hilbert Function

interpretation phull: vector-space map-scale
apply standard
subgoal by (fact map-scale-distrib-left)
subgoal by (fact map-scale-distrib-right)
subgoal by (fact map-scale-assoc)
subgoal by (fact map-scale-one-left)
definition Hilbert-fun :: (('x ⇒_0 nat) ⇒_0 'a::field) set ⇒ nat ⇒ nat
where Hilbert-fun A z = phull.dim (hom-deg-set z A)

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

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

lemma Hilbert-fun-direct-decomp:
  assumes finite X and A ⊆ P[X] and direct-decomp (A::(('x::countable ⇒_0 nat)
 ⇒_0 'a::field) set) ps
  and \( \land s. \, s \in \text{set ps} \Rightarrow \text{homogeneous-set s} \) and \( \land \, s \in \text{set ps} \Rightarrow \text{phull.subspace s} \)
  shows Hilbert-fun A z = (∑ p∈set ps. Hilbert-fun p z)
proof –
  from assms(3, 4) have dd: direct-decomp (hom-deg-set z A) (map (hom-deg-set z) ps)
  by (rule direct-decomp-hom-deg-set)
  also have Hilbert-fun A z = phull.dim (hom-deg-set z A) by (fact Hilbert-fun-def)
  next
  from assms(2) have hom-deg-set z A ⊆ hom-deg-set z P[X]
   unfolding hom-deg-set-def by (rule image-mono)
  also have hom-deg-set z A ⊆ phull.span (monomial 1 ' deg-sect X z)
   by (simp only: hom-deg-set-Polys-eq-span)
  next
  fix s
  assume s ∈ set (map (hom-deg-set z) ps)
  then obtain s' where s' ∈ set ps and 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 \( \ldots = \text{sum (phull.dim} \circ \text{hom-deg-set z} \text{ (set ps))} \) unfolding set-map
using finite-set
proof (rule sum.reindex-nontrivial)
  fix s1 s2
  note dd
  moreover assume s1 ∈ set ps and s2 ∈ set ps and s1 ≠ s2
  moreover have \( 0 \in \text{hom-deg-set z} \) s if s ∈ 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)
proof

lemma image-lt-hom-deg-set:
  assumes homogeneous-set A
  shows lpp (hom-deg-set z A - \{0\}) = \{t ∈ lpp (A - \{0\}), deg-pm t = z\}
  (is `?A)
proof (intro set-eqI iffI)
  fix t
  assume t ∈ ?A
  hence t ∈ lpp (A - \{0\}) and deg-t[symmetric]: deg-pm t = z by simp-all
  from this(1) obtain p where p ∈ A - \{0\} and t: t = lpp p ..
  from this(1) have p ∈ A and p ≠ 0 by simp-all
  from this(1) have h: hom-component p z ∈ hom-deg-set z A (is `?p ∈ -)
    unfolding hom-deg-set-alt-def by (rule imageI)
  from `p ≠ 0: have lpp ?p = t unfolding t deg-t by (rule hom-component-lpp)+
    note this(2)[symmetric]
  moreover from I (\?p ≠ 0: have ?p ∈ hom-deg-set z A - \{0\} by simp
  ultimately show t ∈ ?B by (rule image-eqI)
next
  fix t
  assume t ∈ ?B
  then obtain p where p ∈ hom-deg-set z A - \{0\} and t: t = lpp p ..
  from this(1) have p ∈ hom-deg-set z A and p ≠ 0 by simp-all
  with assms have p ∈ A and homogeneous p and poly-deg p = z
    by (simp-all add: hom-deg-set-alt-homogeneous-set)
  from this(1) (\p ≠ 0: have p ∈ A - \{0\} by simp
  hence I: t ∈ lpp (A - \{0\}) using t by (rule rev-image-eqI)
  from `p ≠ 0: have t ∈ keys p unfolding t by (rule punit.ii)
  with homogeneous p: have deg-pm t = poly-deg p by (rule homogeneousD-poly-deg)
  with I show t ∈ ?A by (simp add: poly-deg p = z)
qed

lemma Hilbert-fun-alt:
  assumes finite X and A ⊆ P[X] and phull.subspace A
  shows Hilbert-fun A z = card (lpp (hom-deg-set z A - \{0\})) (is - = card ?A)
proof -
  have A ⊆ lpp (hom-deg-set z A - \{0\}) by simp
  then obtain B where sub: B ⊆ hom-deg-set z A - \{0\} and eq1: ?A = lpp B

88
and inj: inj-on lpp B by (rule subset-image-E-inj)
have Hilbert-fun A z = phull.dim (hom-deg-set z A) by (fact Hilbert-fun-def)
also have \ldots = card B
proof (rule phull.dim-eq-card)
show phull.span B = phull.span (hom-deg-set z A)
proof
from sub have B ⊆ hom-deg-set z A by blast
thus phull.span B ⊆ phull.span (hom-deg-set z A) by (rule phull.span-mono)
next
from assms(3) have phull.subspace (hom-deg-set z A) by (rule subspace-hom-deg-set)
  hence phull.span (hom-deg-set z A) = hom-deg-set z A by (simp only: phull.span-eq-iff)
also have \ldots ⊆ phull.span B
proof (rule ccontr)
  assume \neg hom-deg-set z A ⊆ phull.span B
  then obtain p0 where p0 ∈ hom-deg-set z A − phull.span B (is - ∈ B)
by blast
note assms(1) this
moreover have ?B ⊆ P[X]
proof (rule subset-trans)
from assms(2) show hom-deg-set z A ⊆ P[X] by (rule Polys-closed-hom-deg-set)
qed blast
ultimately obtain p where p ∈ ?B and p-min: \wedge q. punit.ord-strict-p q
p \rightarrow q \notin ?B
by (rule punit.ord-p-minimum-dgrad-p-set[OF dickson-grading-variety, where m=0,
  simplified dgrad-p-set-variety]) blast
from this(1) have p ∈ hom-deg-set z A and p \notin phull.span B by simp-all
from phull.span-zero this(2) have p \neq 0 by blast
with \{p ∈ hom-deg-set z A\} have p ∈ hom-deg-set z A − \{0\} by simp
hence lpp p ∈ lpp \cdot (hom-deg-set z A − \{0\}) by (rule imageI)
also have \ldots = lpp \cdot B by (simp only: eq1)
finally obtain b where b ∈ B and eq2: lpp p = lpp b ..
from this(1) sub have b ∈ hom-deg-set z A − \{0\} ..
hence b ∈ hom-deg-set z A and b \neq 0 by simp-all
from this(2) have lc:b: punit.lc b \neq 0 by (rule punit.lc-not-0)
from (p \neq 0) have lcp: punit.lc p \neq 0 by (rule punit.lc-not-0)
from (b ∈ B) have b ∈ phull.span B by (rule phull.span-base)
  hence (punit.lc p / punit.lc b) \cdot b ∈ phull.span B (is \ w b ∈ -) by (rule phull.span-scale)
with (p \notin phull.span B: have p − \w b \neq 0 by auto
moreover from lcb lcp \{b \neq 0\}: have lpp \w b = lpp p
  by (simp add: punit.map-scale-eq-monom-mult punit.lt-monom-mult eq2)
moreover from lcb have punit.lc \w b = punit.lc p by (simp add: punit.map-scale-eq-monom-mult)
ultimately have lpp (p − \w b) < lpp p by (rule punit.lt-minus-lessI)
hence punit.ord-strict-p (p − \w b) p by (rule punit.lt-ord-p)
hence p − \w b \notin ?B by (rule p-min)
hence p − \w b \notin hom-deg-set z A \or p − \w b ∈ phull.span B by simp
thus False

89
proof
assume ∗: \( p - \frac{b}{\mathfrak{A}} \in \text{hom-deg-set } z A \)
from phull.subspace-scale have \( \frac{b}{\mathfrak{A}} \in \text{hom-deg-set } z A \)
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-scale \( \langle \frac{p}{\mathfrak{A}} \in \text{hom-deg-set } z A \rangle \) have \( \frac{p}{\mathfrak{A}} - \frac{b}{\mathfrak{A}} \in \text{hom-deg-set } z A \) by (rule assms (3))
proof (rule hom-deg-set-closed-minus)
show \( \frac{b}{\mathfrak{A}} \in \text{hom-deg-set } z A \) by fact
qed
with phull.subspace-diff \( \langle \frac{p}{\mathfrak{A}} \in \text{hom-deg-set } z A \rangle \) have \( \frac{p}{\mathfrak{A}} - \frac{b}{\mathfrak{A}} \in \text{hom-deg-set } z A \) by (rule assms)
proof (rule hom-deg-set-closed-minus)
show \( \frac{b}{\mathfrak{A}} \in \text{hom-deg-set } z A \) by fact
qed
with \( \langle \frac{p}{\mathfrak{A}} \in \text{hom-deg-set } z A \rangle \) have \( \frac{p}{\mathfrak{A}} - \frac{b}{\mathfrak{A}} \in \text{hom-deg-set } z A \) by (rule assms)
next
assume \( p - \frac{b}{\mathfrak{A}} \in \text{phull span } B \)
hence \( p - \frac{b}{\mathfrak{A}} + \frac{b}{\mathfrak{A}} \in \text{phull span } B \) using \( \langle \frac{b}{\mathfrak{A}} \in \text{phull span } B \rangle \) by (rule phull.span-add)
hence \( p \in \text{phull span } B \) by simp
with \( \langle \frac{p}{\mathfrak{A}} \notin \text{phull span } B \rangle \) show \( \text{thesis} \) ...
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 \( \text{finite } B' \) \( \text{and } B' \subseteq B \) \( \text{and } (\sum b \in B'. u b \cdot b) = 0 \)
and \( b' \in B' \) \( \text{and } u b' \neq 0 \) 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-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 sum.mono-neutral-left) (simp add: B0-def)
also have \( \ldots = 0 \) by fact
finally have \( eq: (\sum b \in B0. u b \cdot b) = 0 \).
define \( t \) where \( t = \text{ordered-powerprod-lin.Max } (\text{lpp } B0) \)
from \( \langle b' \in B' \rangle \langle u b' \neq 0 \rangle \) have \( b' \in B0 \) by (simp add: B0-def)
hence \( \text{lpp } b' \in \text{lpp } B0 \) by (rule imageI)
hence \( \text{lpp } B0 \neq \{ \} \) by blast
from \( B0 \subseteq B' \) \( \langle \text{finite } B' \rangle \) 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 ordered-powerprod-lin.Max-in)
then obtain \( b0 \) where \( b0 \in B0 \) \( \text{and } t = \text{lpp } b0 \).
note this(1)
moreover from \( B0 \subseteq B' \) \( \langle B' \subseteq B \rangle \) have \( B0 \subseteq B \) by (rule subset-trans)
also have \( \ldots \subseteq \text{hom-deg-set } z A - \{ 0 \} \) by fact
finally have \( b0 \in \text{hom-deg-set } z A - \{ 0 \} \).
hence \( b0 \neq 0 \) by simp
hence $t \in \text{keys } b_0$ unfolding $t$ by \(\text{rule } \text{punit}.lt\text{-keys}\)
have \(\text{lookup} \left( \sum_{b \in B_0} u \cdot b \right) t = \left( \sum_{b \in B_0} u \cdot \text{lookup } b \cdot t \right)\) by \(\text{simp add: lookup-sum}\)
also from \(\langle \text{finite } B_0 \rangle\) have \(\ldots = \left( \sum_{b \in \{b_0\}} u \cdot \text{lookup } b \cdot t \right)\)
proof \(\text{rule } \text{sum.mono-neutral-right}\)
from \(\langle b_0 \in B_0 \rangle\) show \(\{b_0\} \subseteq B_0\) by simp
next
show \(\forall b \in B_0 - \{b_0\}. \ u \cdot \text{lookup } b \cdot t = 0\)
proof
fix $b$
assume $b \in B_0 - \{b_0\}$
hence $b \in B_0$ and $b \neq b_0$ by simp-all
from \(\text{this}(i)\) have $\lpp b \in \lpp ' B_0$ by \(\text{rule } \text{imageI}\)
with \(\langle \text{finite } (\lpp ' B_0) \rangle\) have $\lpp b \leq t$ unfolding $t\text{-def}$
by \(\text{rule } \text{ordered-powerprod-lin}.\text{Max-ge}\)
have $t \notin \text{keys } b$
proof
assume $t \in \text{keys } b$
hence $t \leq \lpp b$ by \(\text{rule } \text{punit}.\text{lt-max-keys}\)
with \(\langle \lpp b \leq t \rangle\) have $\lpp b = \lpp b_0$
unfolding $t$ by \(\text{rule } \text{ordered-powerprod-lin.antisym}\)
from \(\text{inj } B_0 \subseteq B\) have \(\text{inj-on } \lpp B_0\) by \(\text{rule } \text{inj-on-subset}\)
\hspace{1cm} hence $b = b_0$ using \(\langle \lpp b = \lpp b_0 \rangle\) \(\langle b \in B_0 \rangle\) \(\langle b_0 \in B_0 \rangle\) by \(\text{rule } \text{inj-onD}\)
\hspace{1cm} with \(\langle b \neq b_0 \rangle\) show False ..
qed
thus $u \cdot \text{lookup } b \cdot t = 0$ by \(\text{simp add: in-keys-iff}\)
qed
qed
also from \(\langle t \in \text{keys } b_0 \rangle\) \(\langle b_0 \in B_0 \rangle\) have \(\ldots \neq 0\) by \(\text{simp add: } B_0\text{-def in-keys-iff}\)
finally show False by \(\text{simp add: eq}\)
qed
qed
also have \(\ldots = \text{card } ?A\) unfolding eq1 using inj by \(\text{rule } \text{card-image}[\text{symmetric}]\)
finally show $?\text{thesis }$.
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 ⇒ 'a ⇒ 'a::semigroup-mult set (infixl ÷ 55)
  where quot-set A x = (∗) x − ' A

lemma quot-set-iff: a ∈ A ÷ x ←→ x ∗ a ∈ A
  by (simp add: quot-set-def)

lemma quot-setI: x ∗ a ∈ A ⇒ a ∈ A ÷ x
  by (simp only: quot-set-iff)

lemma quot-setD: a ∈ A ÷ x ⇒ x ∗ a ∈ A
  by (simp only: quot-set-iff)

lemma quot-set-quot-set [simp]: A ÷ x ÷ y = A ÷ x ∗ y
  by (rule set-eqI) (simp add: quot-set-iff mult.assoc)

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

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

proof
  show ideal (ideal B ÷ x) ⊆ ideal B ÷ x

proof
  fix b
  assume b ∈ ideal (ideal B ÷ x)
  thus b ∈ ideal B ÷ x

  proof (induct b rule: ideal.span-induct')
    case base
    show ?case
      by (simp add: quot-set-iff ideal.span-zero)

end
next
case (step b q p)
hence $x \ast b \in \text{ideal } B$ and $x \ast p \in \text{ideal } B$ by (simp-all add: quot-set-iff)
hence $x \ast b + q \ast (x \ast p) \in \text{ideal } B$
  by (intro ideal.span-add ideal.span-scale[where c=q])
thus ?case by (simp only: quot-set-iff algebra-simps)
qed
qed

qed (fact ideal.span-superset)

lemma quot-set-image-times: inj (($x \ast) x) \Rightarrow (($x \ast) x \cdot 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 \text{nat}) \Rightarrow (\text{a} :: \text{field}) \Rightarrow (\text{a} :: \text{field}))
  where normal-form F p = (SOME q. (punit.red (punit.reduced-GB F))** p q \land
  \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 \text{nat}) \Rightarrow (\text{a} :: \text{field}) \Rightarrow (\text{a} :: \text{field})
  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 \land \neg punit.is-red (punit.reduced-GB F) q ..
hence ?thesis1 \land ?thesis2 unfolding normal-form-def by (rule someI)
thus ?thesis1 and ?thesis2 by simp-all
qed
lemma normal-form-unique:
  assumes \((\text{punit} . \text{red} (\text{punit} . \text{reduced-GB} \ F))^{**} \ p \ q \ \text{and} \ \neg \text{punit} . \text{is-red} (\text{punit} . \text{reduced-GB} \ F)\) \ 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 \((\text{punit} . \text{red} (\text{punit} . \text{reduced-GB} \ F))^{**} \ p \) (normal-form \(\ F \ p \)) by (rule normal-form)
next
  have \(\neg \text{punit} . \text{is-red} (\text{punit} . \text{reduced-GB} \ F)\) (normal-form \(\ F \ p \)) by (rule normal-form)
  thus relation.is-final (punit.red (punit.reduced-GB F)) (normal-form \(\ F \ p \))
  by (simp add: punit.is-red-def)
next
  from assms(2) show relation.is-final (punit.red (punit.reduced-GB F)) \(\ q\)
  by (simp add: punit.is-red-def)
qed fact

lemma normal-form-id-iff: normal-form \(\ F \ p \ = \ p \Longleftrightarrow \neg \text{punit} . \text{is-red} (\text{punit} . \text{reduced-GB} \ F)\)
proof
  assume normal-form \(\ F \ p \ = \ p\)
  with normal-form(2)[of p] show \(\neg \text{punit} . \text{is-red} (\text{punit} . \text{reduced-GB} \ F)\) \(\ p\) by simp
next
  assume \(\neg \text{punit} . \text{is-red} (\text{punit} . \text{reduced-GB} \ F)\) \(\ p\)
  with rtranclp.rtrancl-refl show normal-form \(\ F \ p \ = \ p\) by (rule normal-form-unique)
qed

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

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

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

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

lemma normal-form-plus-normal-form:
  normal-form \(\ F \) (normal-form \(\ F \) \(\ p\) + normal-form \(\ F \) \(\ q\)) = normal-form \(\ F \) \(\ p\) + normal-form \(\ F \) \(\ q\)
by (intro normal-form-unique rtranclp.rtrancl-refl punit.is-irred-plus normal-form)
lemma normal-form-minus-normal-form:
  normal-form F (normal-form F p − normal-form F q) = normal-form F p − normal-form F q
  by (intro normal-form-unique rtranclp.rtrancl-refl punit.is-irred-minus normal-form)

lemma normal-form-ideal-Polys: normal-form (ideal F ∩ P[X]) = normal-form F
proof −
  let ?F = ideal F ∩ P[X]
  from fin-X have eq: punit.reduced-GB ?F = punit.reduced-GB F
  proof (rule reduced-GB-unique-Polys)
    from fin-X F-sub show punit.is-reduced-GB (punit.reduced-GB F)
    by (rule reduced-GB-is-reduced-GB-Polys)
  next
  from fin-X F-sub have ideal (punit.reduced-GB F) = ideal F by (rule reduced-GB-ideal-Polys)
  also have ... = ideal (ideal F ∩ P[X])
  proof (intro subset-antisym ideal.span-subset-spanI)
    from ideal.span-superset[of F] F-sub have F ⊆ ideal F ∩ P[X] by simp
    thus F ⊆ ideal (ideal F ∩ P[X]) using ideal.span-superset by (rule subset-trans)
    qed
  finally show ideal (punit.reduced-GB F) = ideal (ideal F ∩ P[X]) .
  qed blast
  show ?thesis by (rule ext) (simp only: normal-form-def eq)
  qed

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

lemma normal-form-zero-iff: normal-form F p = 0 ↔ p ∈ ideal F
proof
  assume normal-form F p = 0
  with normal-form-diff-in-ideal[of p] show p ∈ ideal F by simp
next
  assume p ∈ ideal F
  hence p − (p − normal-form F p) ∈ ideal F using normal-form-diff-in-ideal
  by (rule ideal.span-diff)
  also from fin-X F-sub have ... = ideal (punit.reduced-GB F) by (rule reduced-GB-ideal-Polys[symmetric])
  finally have #: normal-form F p ∈ ideal (punit.reduced-GB F) by simp
  show normal-form F p = 0
  proof (rule contr)
    from fin-X F-sub have punit.is-Groebner-basis (punit.reduced-GB F) by (rule reduced-GB-is-GB-Polys)
    moreover note *
    moreover assume normal-form F p ≠ 0
ultimately obtain $g$ where $g \in \text{punit}\_\text{reduced-GB} F$ and $g \neq 0$
and $e$: $\text{lpp}(g) \text{ adds } \text{lpp}(\text{normal-form} F p)$ by (rule punit._GB-adds-lt[simplified])

note this(1, 2)

moreover from $\langle \text{normal-form} F p \neq 0 \rangle$ have $\text{lpp}(\text{normal-form} F p) \in \text{keys}$
(normal-form $F p$)

by (rule punit._lt-in-keys)

ultimately have $\text{punit}_{} \text{is-red}(\text{punit}\_\text{reduced-GB} F) (\text{normal-form} F p)$

using a by (rule punit._is-red-addsI[simplified])

with normal-form$(2)$ show False ...

qed

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

proof

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

by simp

also from $\text{normal-form-diff-in-ideal} \text{ normal-form-diff-in-ideal} \text{ have } \ldots \in \text{ideal} F$

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

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 $\text{normal-form} F (\text{normal-form} F p - \text{normal-form} F q) = 0$ by (simp add: normal-form-zero-iff)

thus $\text{normal-form} F p = \text{normal-form} F q$ by (simp add: normal-form-minus-normal-form)

qed

lemma Polys-closed-normal-form:

assumes $p \in P[X]$

shows $\text{normal-form} F p \in P[X]$

proof

from fin-X F-sub have $\text{punit}\_\text{reduced-GB} F \subseteq P[X]$ by (rule reduced-GB-Polys)

with fin-X show $?thesis$ using assms normal-form$(1)$

by (rule punit._dgrad-p-set-closed-red-rtrancl[OF dickson-grading-varnum, where $m=0$, simplified dgrad-p-set-varnum])

qed

lemma image-normal-form-iff:

$p \in \text{normal-form} F : P[X] \iff (p \in P[X] \land \neg \text{punit}_{} \text{is-red}(\text{punit}\_\text{reduced-GB} F) p)$

96
proof
  assume \( p \in \text{normal-form } F \cdot P[X] \)
  then obtain \( q \) where \( q \in P[X] \) and \( p = \text{normal-form } F \cdot q \).
  from this(1) show \( p \in P[X] \and \neg \text{punit.is-red } (\text{punit.reduced-GB } F) \cdot p \).
  unfolding \( p \)
  by (intro conjI Polys-closed-normal-form normal-form)

next
  assume \( p \in P[X] \and \neg \text{punit.is-red } (\text{punit.reduced-GB } F) \cdot p \)
  hence \( p \in P[X] \and \neg \text{punit.is-red } (\text{punit.reduced-GB } F) \cdot p \)
  by simp-all
  from this(2) have \( \text{normal-form } F \cdot p = p \)
  by (simp add: normal-form-id-iff)
  from this[symmetric] \( \cdot p \in P[X] \):
  show \( \cdot p \in \text{normal-form } F \cdot P[X] \)
  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 \( L \equiv (\text{ideal } F \div f) \cdot \cap P[X] \)
  assumes \( F \subseteq P[X] \) and \( f \in P[X] \)
  shows \( \text{direct-decomp } (I \cap P[X]) \cdot (\text{ideal } F \cap P[X], \cdot f \cdot \text{normal-form } L \cdot P[X]) \)
  (is \( \text{direct-decomp } - \cdot \text{ss} \))
  proof (rule direct-decompI-alt)
  fix \( qs \)
  assume \( qs \in \text{listset } ?ss \)
  then obtain \( x \) y where \( x \cdot x \in \text{ideal } F \cap P[X] \) and \( y \cdot y \in \text{(*) } f \cdot \text{normal-form } L \cdot P[X] \)
  and \( qs \cdot qs = [x, y] \)
  by (rule listset-doubletonE)
  have \( \text{sum-list } qs = x + y \)
  by (simp add: qs)
  also have \( \ldots \in I \cap P[X] \)
  unfolding I-def
  proof (intro IntI ideal.span-add Polys-closed-plus)
    have \( \text{ideal } F \subseteq \text{ideal } (\text{insert } f F) \)
    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 \( y = f \cdot \text{normal-form } L \cdot p \)
  by blast
  have \( f \in \text{ideal } (\text{insert } f F) \)
  by (rule ideal.span-base) simp
  hence \( \text{normal-form } L \cdot p \cdot f \in \text{ideal } (\text{insert } f F) \)
  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 \( \text{normal-form } L \cdot p \in P[X] \)
  using \( p \in P[X] \)
  by (rule Polys-closed-normal-form)
  with \( \text{assms(4)} \)
  show \( y \in P[X] \)
  unfolding \( y \)
  by (rule Polys-closed-times)
  qed
  finally show \( \text{sum-list } qs \in I \cap P[X] \).
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 insert \( f \subseteq P[X] \) by simp
then obtain \( F_0 \) \( q_0 \) where \( F_0 \subseteq insert f \) \( F \) and finite \( F_0 \) and \( q_0 : \bigwedge f_0. \) \( q_0 \) \( f_0 \) \( \in \) \( P[X] \)
and \( a : a = (\sum f_0 \in F_0. \) \( q_0 \) \( f_0 \) \( * \) \( f_0 \))
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 \( a' : a' \in ideal F \) and \( a' \in P[X] \) and \( q \in P[X] \) and \( a : a = q * f + a' \)
proof (cases \( f \in F_0 \))
case True
with \( \langle F_0 \subseteq insert f \rangle \) have \( F_0 - \{ f \} \subseteq F \) by blast
show \(?thesis \)
proof
have \( (\sum f_0 \in F_0 - \{ f \}. \) \( q_0 \) \( f_0 \) \( * \) \( f_0 \)) \( \in \) ideal \( (F_0 - \{ f \}) \) by (rule ideal.sum-in-spanI)
also from \( (F_0 - \{ f \} \subseteq F) \) have \( \ldots \subseteq ideal F \) by (rule ideal.span-mono)
finally show \( (\sum f_0 \in F_0 - \{ f \}. \) \( q_0 \) \( f_0 \) \( * \) \( f_0 \)) \in ideal F \).
next
show \( (\sum f_0 \in F_0 - \{ f \}. \) \( q_0 \) \( f_0 \) \( * \) \( f_0 \)) \in P[X]
proof (intro Polys-closed-sum Polys-closed-times \( q_0 \))
fix \( f_0 \)
assume \( f_0 \in F_0 - \{ f \} \)
also have \( \ldots \subseteq F_0 \) by blast
also have \( \ldots \subseteq insert f \) \( \subseteq \) \( F \) by fact
also have \( \ldots \subseteq P[X] \) by fact
finally show \( f_0 \in P[X] \).
qed
next
from \( \langle finite F_0, True \rangle \) show \( a = q_0 f * f + (\sum f_0 \in F_0 - \{ f \}. \) \( q_0 \) \( f_0 \) \( * \) \( f_0 \))
by (simp only: a sum.remove)
qed fact
next
case False
with \( \langle F_0 \subseteq insert f \rangle \) have \( F_0 \subseteq F \) by blast
show ?thesis
proof
have \( a \in ideal F_0 \) unfolding a by (rule ideal.sum-in-spanI)
also from \( \langle F_0 \subseteq F \rangle \) have \( \ldots \subseteq ideal F \) by (rule ideal.span-mono)
finally show \( a \in ideal F \).
next
show \( a = 0 * f + a \) by simp
qed (fact \( \langle a \in P[X], fact zero-in-Polys \rangle \))
next
let \( \alpha = a * \) (normal-form \( L q \))
have \( L \subseteq P[X] \) by (simp add: L-def)
hence normal-form \( L q \in P[X] \) using \( \langle q \in P[X] \rangle \) by (rule Polys-closed-normal-form)
with assms(\( \_ \)) have \( \alpha \in P[X] \) by (rule Polys-closed-times)
from \( \langle L \subseteq P[X] \rangle \) have \( q = normal-form L q \in ideal L \) by (rule normal-form-diff-in-ideal)
also have \( \ldots \subseteq ideal \) (ideal \( F \subseteq f \)) unfolding L-def by (rule ideal.span-mono) blast
finally have \( f * (q - normal-form L q) \in ideal F \) by (simp add: quot-set-iff)
with \(a' \in \text{ideal } F\) have \(a' + f \cdot (q - \text{normal-form } L \ q) \in \text{ideal } F\) 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 \(\text{assms}(4)\) \((a \in P[X])\) \((\text{normal-form } L \ q \in P[X])\) show \(a - ?a \in P[X]\)
  by (intro Polys-closed-minus Polys-closed-times)
qed fact
moreover from \(q \in P[X]\) have \(?a \in (\ast) f \cdot \text{normal-form } L \cdot P[X]\) by (intro imageI)
ultimately show \(qs \in \text{listset } ?ss\) using qs-def by (rule listset-doubletonI)
next
fix \(qs0\)
assume \(qs0 \in \text{listset } ?ss \land a = \text{sum-list } qs0\)
from \(this(1)\) obtain \(x y\) where \(x \in \text{ideal } F \cap P[X]\) and \(y \in (\ast) f \cdot \text{normal-form } L \cdot P[X]\) and \(qs0: \{x, y\}\) by (rule listset-doubletonE)
from \(this(2)\) obtain \(a0\) where \(a0 \in P[X]\) and \(y = f \cdot \text{normal-form } L \ a0\) by blast
  from \(\{x \in \text{ideal } F \cap P[X]\}\) have \(x \in \text{ideal } F\) by simp
  have \(x = a - y\) by (simp add: \(\langle a = \text{sum-list } qs0 \rangle\) \(qs0\))
  have \(f \cdot (\text{normal-form } L \ q - \text{normal-form } L \ a0) = x - (a - ?a)\) by (simp add: \(x\ y\ a\ algebra-simps\))
  also from \(\{x \in \text{ideal } F\}\) \((a - ?a \in \text{ideal } F)\) have \(\ldots \in \text{ideal } F\) by (rule ideal.span-diff)
finally have normal-form \(L \ q - \text{normal-form } L \ a0 \in \text{ideal } F \div f\) by (rule quot-set1)
moreover from \(\{L \subseteq P[X]\}\) \(\langle q \in P[X]\rangle\) \((a0 \in P[X])\) have normal-form \(L \ q - \text{normal-form } L \ a0 \in P[X]\)
  by (intro Polys-closed-minus Polys-closed-normal-form)
ultimately have normal-form \(L \ q - \text{normal-form } L \ a0 \in L\) by (simp add: L-def)
also have \(\ldots \subseteq \text{ideal } L\) by (fact ideal.span-superset)
finally have normal-form \(L \ q - \text{normal-form } L \ a0 = 0\) using \(\{L \subseteq P[X]\}\)
  by (simp only: normal-form-minus-normal-form flip: normal-form-zero-iff)
thus \(qs0 = qs\) by (simp add: qs0 qs-def \(x\ y)\)
qed (simp-all add: qs-def)

qed

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

end

10.4 Basic Cone Decompositions

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

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

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

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

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

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

lemma zero-in-cone: \[ 0 \in \text{cone } hU \]
\[ \text{by (auto simp: cone-def intro!: image-eqI zero-in-Polys)} \]

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

lemma tip-in-cone: \[ h \in \text{cone } (h::'a::\text{comm-semiring-1}, U) \]
\[ \text{using - one-in-Polys by (rule coneI) simp} \]

lemma cone-closed-plus: \[ \text{assumes } a \in \text{cone } hU \text{ and } b \in \text{cone } hU \]
\[ \text{shows } a + b \in \text{cone } hU \]

100
proof –

obtain $h$ $U$ where $hU = (h, U)$ 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 $a: a = a' \ast h$ by (rule coneE)
from $(b \in \text{cone } (h, U))$ obtain $b'$ where $b' \in P[U]$ and $b: b = b' \ast h$ by (rule coneE)

have $a + b = (a' + b') \ast h$ by (simp only: $a$ algebra-simps)
moreover from $(a' \in P[U]; b' \in P[U])$ have $a' + b' \in P[U]$ by (rule Polys-closed-plus)
ultimately show $?\text{thesis unfolding } hU$ by (rule cone1)

qed

lemma cone-closed-uminus:

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

proof –

obtain $h$ $U$ where $hU = (h, U)$ 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 $a: a = a' \ast h$ by (rule coneE)
have $-a = (-a') \ast h$ by (simp add: $a$)
moreover from $(a' \in P[U])$ have $-a' \in P[U]$ by (rule Polys-closed-uminus)
ultimately show $?\text{thesis unfolding } hU$ by (rule cone1)

qed

lemma cone-closed-times:

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

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 $?\text{thesis by simp}$

qed

lemma cone-closed-monom-mult:

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

proof –

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

corollary cone-closed-monom-mult:

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

proof –

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

101
qed

lemma coneD:
  assumes \( p \in \text{cone} (h, U) \) and \( p \neq 0 \)
  shows \( \text{lpp} h \text{ adds lpp} (p :: \Rightarrow 0 \Rightarrow \{\text{comm-semiring-0, semiring-no-zero-divisors}\}) \)
proof
  from assms(1) obtain \( a \) where \( p = a \ast h \) by (rule coneE)
  with assms(2) have \( a \neq 0 \) and \( h \neq 0 \) by auto
  hence \( \text{lpp} p = \text{lpp} a + \text{lpp} h \) unfolding \( p \) by (rule lp-times)
  also have \( \ldots = \text{lpp} h + \text{lpp} a \) by (rule add.commute)
  finally show \( ?\text{thesis} \) by (rule addsI)
qed

lemma cone-mono-1:
  assumes \( h' \in P[U] \)
  shows \( \text{cone} (h' \ast h, U) \subseteq \text{cone} (h, U) \)
proof
  fix \( p \)
  assume \( p \in \text{cone} (h' \ast h, U) \)
  then obtain \( a' \) where \( a' \in P[U] \) and \( p = a' \ast (h' \ast h) \) by (rule coneE)
  from this(2) have \( p = a' \ast h' \ast h \) by (simp only: mult.comm)
  moreover from \( a' \in P[U] \) \( \text{assms} \) have \( a' \ast h' \in P[U] \) 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 \( p = a \ast h \) by (rule coneE)
  note this(2)
  moreover from \( a \in P[U1] \) \( \text{assms} \) have \( a' \ast h' \in P[U] \) by (rule Polys-closed-times)
  ultimately show \( p \in \text{cone} (h, U2) \) by (rule coneI)
qed

lemma cone-subsetD:
  assumes \( \text{cone} (h1, U1) \subseteq \text{cone} (h2 :: \Rightarrow 0 \Rightarrow \{\text{comm-ring-1, ring-no-zero-divisors}\}, U2) \)
  shows \( h2 \text{ dvd} h1 \) and \( h1 \neq 0 \Rightarrow U1 \subseteq U2 \)
proof
  from tip-in-cone \text{assms} \ have \( h1 \in \text{cone} (h2, U2) \)
  then obtain \( a' \) where \( a' \in P[U2] \) and \( h1 = a' \ast h2 \) by (rule coneE)
  from this(2) have \( h1 = h2 \ast a' \) by (simp only: mult.comm)
  thus \( h2 \text{ dvd} h1 \)
  assume \( h1 \neq 0 \)
with \( h_1 \) have \( a_1' \neq 0 \) and \( h_2 \neq 0 \) by \textit{auto}

\begin{proof}
  \begin{enumerate}
    \item fix \( x \)
    \item assume \( x \in U_1 \)
    \item hence monomial \((1::'a) \text{ (Poly-Mapping.single } x \ 1) \in P[U_1]\) (is \( ?p \in -\))
      by (intro Polys-closed-monomial PPs-closed-single)
    \item with \textit{refl} have \( ?p \ast h_1 \in \text{cone} \ (h_1, \ U_1) \) by (rule coneI)
    \item hence \( ?p \ast h_1 \in \text{cone} \ (h_2, \ U_2) \) using \textit{assms} ..
    \item hence \( ?p \ast a_1' \neq 0 \) by \textit{simp only: h1 ac-simps}
    \item hence \( ?p \ast a_1' = a \) using \( \langle h_2 \neq 0 \rangle \) by (rule times-canc-right)
    \item with \( \langle a \in P[U_2] \rangle \) have \( ?p \in P[U_2] \) by (simp add: mult.commute)
    \item hence \( ?p \in P[U_2] \) using \( \langle a_1' \in P[U_2] \rangle \) \( (a_1' \neq 0) \) by (rule times-in-PolysD)
    \item thus \( x \in U_2 \) by \textit{simp add: Polys-def PPs-def}
  \end{enumerate}
\end{proof}

\begin{lemma}
  \textit{cone-subset-PolysD}:
  \begin{enumerate}
    \item assumes \( \text{cone} \ (h::':a' \text{ (comm-semiring-1,semiring-no-zero-divisors}, \ U) \subseteq \ P[X]\)
    \item shows \( h \in P[X] \) and \( h \neq 0 \implies U \subseteq X \)
  \end{enumerate}
\end{lemma}

\begin{proof}
  \begin{enumerate}
    \item assume \( h \neq 0 \)
    \item show \( U \subseteq X \)
      \begin{enumerate}
        \item fix \( x \)
        \item assume \( x \in U \)
        \item hence monomial \((1::'a) \text{ (Poly-Mapping.single } x \ 1) \in P[U]\) (is \( ?p \in -\))
          by (intro Polys-closed-monomial PPs-closed-single)
        \item with \textit{refl} have \( ?p \ast h \in \text{cone} \ (h, \ U) \) by (rule coneI)
        \item hence \( ?p \ast h \in P[X] \) using \textit{assms} ..
        \item hence \( h \ast ?p \in P[X] \) by (simp only: mult.commute)
        \item hence \( ?p \in P[X] \) using \( \langle h \in P[X] \rangle \) \( (h \neq 0) \) by (rule times-in-PolysD)
        \item thus \( x \in X \) by \textit{simp add: Polys-def PPs-def}
      \end{enumerate}
  \end{enumerate}
\end{proof}

\begin{lemma}
  \textit{cone-subset-PolysI}:
  \begin{enumerate}
    \item assumes \( h \in P[X] \) and \( h \neq 0 \implies U \subseteq X \)
    \item shows \( \text{cone} \ (h, \ U) \subseteq P[X] \)
  \end{enumerate}
\end{lemma}

\begin{proof}
  \begin{enumerate}
    \item \( \text{cases } h = 0 \)
      \begin{enumerate}
        \item hence \( U \subseteq X \) by \textit{(simp add: zero-in-Polys)}
      \end{enumerate}
  \end{enumerate}
\end{proof}
hence $P[U] \subseteq P[X]$ by \textit{(rule Polys-mono)}

\textbf{show} \ ?thesis

\textbf{proof}

\hspace{1em}fix $a$

\hspace{1em}assume $a \in \text{cone } (h, U)$

\hspace{1em}then obtain $q$ where $q \in P[U]$ \textbf{and} $a = q * h$ by \textit{(rule coneE)}

\hspace{1em}from this(1) : $P[U] \subseteq P[X]$. have $q \in P[X]$ ..

\hspace{1em}from this assms(1) show $a \in P[X]$ unfolding $a$ by \textit{(rule Polys-closed-times)}

\hspace{1em}qed

\hspace{1em}qed

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

\hspace{1em}by \textit{(auto simp: ac-simps image-image intro: image-eqI coneI elim: coneE)}

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

\hspace{1em}proof --

\hspace{1em}obtain $h \ U$ where $hU = (h, U)$ \textbf{using} prod.exhaust \textbf{by} blast

\hspace{1em}thus \ ?thesis \ \textbf{by} \textit{(simp add: cone-image-times)}

\hspace{1em}qed

\textbf{lemma} homogeneous-set-coneI:

\hspace{1em}assumes \textit{homogeneous }$h$

\hspace{1em}shows \textit{homogeneous-set }$\text{cone } (h, U)$

\hspace{1em}proof \textit{(rule homogeneous-setI)}

\hspace{1em}fix $a \ n$

\hspace{1em}assume $a \in \text{cone } (h, U)$

\hspace{1em}then obtain $q$ where $q \in P[U]$ \textbf{and} $a = q * h$ by \textit{(rule coneE)}

\hspace{1em}from this show $\text{hom-component } a \ n \in \text{cone } (h, U)$ unfolding $a$

\hspace{1em}proof \textit{(induct }q\text{ rule: poly-mapping-plus-induct)}

\hspace{1em}case $1$

\hspace{1em}show \ ?case \ \textbf{by} \textit{(simp add: zero-in-cone)}

\hspace{1em}next

\hspace{1em}case (2 $p \ c \ t$)

\hspace{1em}have $p \in P[U]$

\hspace{1em}proof \textit{(intro PolysI subsetI)}

\hspace{1em}fix $s$

\hspace{1em}assume $s \in \text{keys } p$

\hspace{1em}moreover from $2(2)$ this have $s \notin \text{keys } (\text{monomial } c \ t)$ \textbf{by} \textit{auto}

\hspace{1em}ultimately have $s \in \text{keys } (\text{monomial } c \ t + p)$ \textbf{by} \textit{(rule in-keys-plusI2)}

\hspace{1em}also from $2(4)$ have $\ldots \subseteq [U]$ \textbf{by} \textit{(rule PolysD)}

\hspace{1em}finally have $s \in [U]$ .

\hspace{1em}qed

\hspace{1em}hence $\ast$: \textit{hom-component } $(p * h) \ n \in \text{cone } (h, U)$ \textbf{by} \textit{(rule 2(3))}

\hspace{1em}from $2(1)$ have $t \in \text{keys } (\text{monomial } c \ t)$ \textbf{by} \textit{simp}

\hspace{1em}hence $t \in \text{keys } (\text{monomial } c \ t + p)$ \textbf{using} $2(2)$ \textbf{by} \textit{(rule in-keys-plusI1)}

\hspace{1em}also from $2(4)$ have $\ldots \subseteq [U]$ \textbf{by} \textit{(rule PolysD)}

\hspace{1em}finally have $\text{monomial } c \ t \in P[U]$ \textbf{by} \textit{(rule Polys-closed-monomial)}

\hspace{1em}with \textbf{refl} have $\text{monomial } c \ t * h \in \text{cone } (h, U)$ \textbf{is } $?h \in -$ \textbf{by} \textit{(rule coneI)}

\hspace{1em}from \textbf{assms} have \textit{homogeneous }$?h$ \textbf{by} \textit{(simp add: homogeneous-times)}
hence \( \text{hom-component } ?h \cdot n = (\ ?h \text{ when } n = \text{poly-deg } ?h) \) by (rule hom-component-of-homogeneous)

with \( ?h \in \text{cone } (h, U) \) have \( \ast \ast: \text{hom-component } ?h \cdot n \in \text{cone } (h, U) \)

by (simp add: when-def zero-in-cone)

have \( \text{hom-component } ((\text{monomial } c \cdot t + p) \ast h) \cdot n = \text{hom-component } ?h \cdot n + \text{hom-component } (p \ast h) \cdot n \)

by (simp only: distrib-right hom-component-plus)

also from \( \ast \ast \ast: \text{have } \ldots \in \text{cone } (h, U) \) by (rule cone-closed-plus)

finally show \( \text{?case } \).

qed

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

using zero-in-cone cone-closed-plus

proof (rule phull.subspaceI)

fix \( c, a \)

assume \( a \in \text{cone } hU \)

moreover obtain \( h, U \) where \( hU = (h, U) \) using prod.exhaust by blast

ultimately have \( a \in \text{cone } (h, U) \) by simp

thus \( c \cdot a \in \text{cone } hU \) unfolding \( hU = \text{Poly-Mapping}. \text{single } x \cdot (\text{Suc } 0) \)

using zero-in-PPs

by (rule cone-closed-monom-mult)

qed

lemma direct-decomp-cone-insert:

fixes \( h \cdot :: \Rightarrow 0 \cdot a :: \{\text{comm-ring-1, ring-no-zero-divisors}\} \)

assumes \( x \notin U \)

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

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

proof –

let \( ?x = \text{Poly-Mapping}. \text{single } x \cdot (\text{Suc } 0) \)

define \( xx \) where \( xx = \text{monomial } (1 :: a) \cdot ?x \)

show \( \text{direct-decomp } (\text{cone } (h, \text{insert } x U)) \cdot [\text{cone } (h, U), \text{cone } (xx \ast h, \text{insert } x U)] \)

(by direct-decomp - \( \ast \ast \ast ))

proof (rule direct-decompl-alt)

fix \( qs \)

assume \( qs \in \text{listset } \?ss \)

then obtain \( a, b \) where \( a \in \text{cone } (h, U) \) and \( b \in \text{cone } (xx \ast h, \text{insert } x U) \)

and \( qs = [a, b] \) by (rule listset-doubletonE)

note this(1)

also have \( \text{cone } (h, U) \subseteq \text{cone } (h, \text{insert } x U) \) by (rule cone-mono-2) blast

finally have \( a: a \in \text{cone } (h, \text{insert } x U) \).

have \( \text{cone } (xx \ast h, \text{insert } x U) \subseteq \text{cone } (h, \text{insert } x U) \)

by (rule cone-mono-1) (simp add: xx-def Polys-def PP-closed-single)

with \( b \) have \( b \in \text{cone } (h, \text{insert } x U) \).

with \( a \) have \( a + b \in \text{cone } (h, \text{insert } x U) \) by (rule cone-closed-plus)

thus \( \text{sum-list } qs \in \text{cone } (h, \text{insert } x U) \) 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: except-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 ∈ keys q and t ∉ .[U] by (simp-all add: qx-def keys-except)
from (q ∈ P[insert x U]) have keys q ⊆ .[insert x U] by (rule PolysD)
with (t ∈ keys q) have t ∈ .[insert x U] ...
hence keys t ⊆ insert x U by (rule PPsD)
moreover from (t ∉ .[U]) have ¬ keys t ⊆ U by (simp add: PPs-def)
ultimately have x ∈ keys t by blast
thus ?thesis by (simp add: lookup-single True in-keys-iff)
next
case False
thus ?thesis by (simp add: lookup-single)
qed
qed
define qx' where qx' = Poly-Mapping.map-key ((+) ?x) qx
have lookup-qq': lookup qx' = (λt. lookup qx (?x + t))
  by (rule ext) (simp add: qx'-def map-key_rep-eq)
have qx' * xx = punit.monom-mult t ?x qx'
  by (simp only: xx-def mult.commute flip: times-monomial-left)
also have ... = qx
  by (auto simp: punit.lookup-monom-mult lookup-qq' add.commute[of ?x]
  simp flip: not-in-keys-iff-lookup-eq-zero dest: x-adds intro!: poly-mapping-eqI)
finally have qx: qx = qx' * xx by (rule sym)
have qx' ∈ P[insert x U]
proof (intro PolysI subsetI)
fix t
assume t ∈ keys qx'
hence t + ?x ∈ keys qx by (simp only: lookup-qq' in-keys-iff not-False-eq-True
add.commute)
also have ... ⊆ keys q by (auto simp: qx-def keys-except)
also from (q ∈ P[insert x U]) have ... ⊆ .[insert x U] by (rule PolysD)
finally have (t + ?x) − ?x ∈ .[insert x U] by (rule PPs-closed-minus)
thus t ∈ .[insert x U] by simp
qed
define qs where qs = [qU * h, qx' * (xx * h)]
show ∃!qs∈listset ?ss. a = sum-list qs
proof (intro ex1I conjI allI impI)
  from refl (\(qU \in P[U]\)) have \(qU \ast h \in \text{cone} (h, U)\) by (rule coneI)
  moreover from refl (\(qx' \in P[\text{insert} \ x \ U]\)) have \(qx' \ast (xx \ast h) \in \text{cone} (xx \ast h, \text{insert} \ x \ U)\)
  by (rule coneI)
  ultimately show \(qs \in \text{listset} \ ?ss\) using qs-def by (rule listset-doubletonI)

next
  fix \(qs0\)
  assume \(qs0 \in \text{listset} \ ?ss \land a = \text{sum-list} \ qs0\)
  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 \(p2: p2 \in \text{cone} (xx \ast h, \text{insert} \ x \ U)\)
  and \(qs0: qs0 = [p1, p2]\) by (rule listset-doubletonE)
  from this(1) obtain \(qU0\) where \(qU0 \in P[U]\) and \(p1 = qU0 \ast h\) by (rule coneE)
  from \(p2\) obtain \(qx0\) where \(p2 = qx0 \ast (xx \ast h)\) by (rule coneE)
  show \(qs0 = qs\)
  proof (cases \(h = 0\))
    case True
    thus \(?thesis\) by (simp add: qs-def qs0 p1 p2)
  next
    case False
    from \(a0\) have \((qU - qU0) \ast h = (qx0 - qx') \ast xx \ast h\)
    by (simp add: a qs0 p1 p2 q qx algebra-simps)
    hence eq: \(qU - qU0 = (qx0 - qx') \ast xx\) using False by (rule times-canc-right)
    have \(qx0 = qx'\)
    proof (rule ccontr)
      assume \(qx0 \neq qx'\)
      hence \(qx0 - qx' \neq 0\) by simp
      moreover have \(xx \neq 0\) by (simp add: xx-def monomial-0-iff)
      ultimately have \(lpp ((qx0 - qx') \ast xx) = lpp (qx0 - qx') + lpp xx\)
      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 \(?x\ \ x \leq\) lookup \((lpp (qU - qU0))\) \(x\) by (simp only: adds-poly-mapping le-fun-def)
      hence \(x \in \text{keys} \ (lpp (qU - qU0))\) by (simp add: in-keys-iff lookup-single)
      moreover have \(lpp (qU - qU0) \in \text{keys} \ (qU - qU0)\)
      proof (rule punit.lt-in-keys)
        from \(qx0 - qx' \neq 0\) \(\&\) \(xx \neq 0\) show \(qU - qU0 \neq 0\) unfolding eq by (rule times-not-zero)
      qed
      ultimately have \(x \in \text{indets} \ (qU - qU0)\) by (rule in-indetsI)
      from \(qU \in P[U]\), \(qU0 \in P[U]\), \(qU - qU0 \in P[U]\) by (rule Polys-closed-minus)
      hence \(\text{indets} \ (qU - qU0) \subseteq U\) by (rule PolysD)
      with \(x \in \text{indets} \ (qU - qU0)\) have \(x \in U\) ..
      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 \( \Rightarrow \) ((\( 'x \) \Rightarrow \) nat) \Rightarrow \) 'a::zero) \times \) 'x set) list \( \Rightarrow \) bool
where valid-decomp X ps \( \leftarrow\rightarrow \) ((\( \forall \) (h, U)\( \in\) set ps \( . \) h \( \in \) P[X] \( \wedge \) h \( \neq \) 0 \( \wedge \) U \( \subseteq \) X))

definition monomial-decomp :: ((\( 'x \Rightarrow \) nat) \Rightarrow \) 'a::\{one,zero\}) \times \) 'x set) list \( \Rightarrow \) bool
where monomial-decomp ps \( \leftarrow\rightarrow \) (\( \forall \) hU\( \in\) set ps \( . \) is-monomial (fst hU) \( \wedge \) punit.lc (fst hU) = 1)

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

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

lemma valid-decompI:
(\( \bigwedge \) h U. (h, U)\( \in\) set ps \( \Rightarrow \) h \( \in \) P[X]) \( \Rightarrow \) (\( \bigwedge \) h U. (h, U)\( \in\) set ps \( \Rightarrow \) h \( \neq \) 0)
\( \Rightarrow \)
(\( \bigwedge \) h U. (h, U)\( \in\) set ps \( \Rightarrow \) U \( \subseteq \) X) \( \Rightarrow \) 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 \( \forall \) ps. ps \( \in\) set pss \( \Rightarrow \) valid-decomp X ps
shows valid-decomp X (concat pss)
proof (rule valid-decompI)
fix $h, U$
assume $(h, U) \in \text{set } (\text{concat } pss)$
then obtain $ps$ where $ps \in \text{set } pss$ and $(h, U) \in \text{set } ps$ unfolding $\text{set-concat}$

from this(1) have $\text{valid-decomp } X \text{ ps}$ by (rule assms)
thus $h \in P[X]$ and $h \neq 0$ and $U \subseteq X$ using $(h, U) \in \text{set } ps$ by (rule $\text{valid-decompD}$)+

\[ \text{corollary } \text{valid-decomp-append:} \]
assumes $\text{valid-decomp } X \text{ ps}$ and $\text{valid-decomp } X \text{ qs}$
shows $\text{valid-decomp } X \ (\text{ps } \ominus \text{ qs})$
proof -
have $\text{valid-decomp } X \ (\text{concat } [\text{ps}, \text{qs}])$ by (rule $\text{valid-decomp-concat}$) (auto simp: assms)
thus $?\text{thesis}$ by simp

\[ \text{qed} \]

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

\[ \text{qed} \]

\[ \text{lemma } \text{monomial-decompI:} \]
$(\forall h U. \ (h, U) \in \text{set } ps \implies is-monomial \ h) \implies (\forall h U. \ (h, U) \in \text{set } ps \implies \text{punit.lc } h = 1) \implies$
$\text{monomial-decomp } ps$
by (auto simp: monomial-decomp-def)

\[ \text{lemma } \text{monomial-decompD:} \]
assumes $\text{monomial-decomp } ps$ and $(h, U) \in \text{set } ps$
shows is-monomial $h$ and $\text{punit.lc } h = 1$
using assms by (auto simp: monomial-decomp-def)

\[ \text{lemma } \text{monomial-decomp-append-iff:} \]
$\text{monomial-decomp } (\text{ps } \ominus \text{ qs}) \iff \text{monomial-decomp } ps \land \text{monomial-decomp } qs$
by (auto simp: monomial-decomp-def)
lemma monomial-decomp-concat:
  \( (\forall ps. \ ps \in \ set \ pss \Rightarrow \ monomial-decomp \ ps) \Rightarrow \ monomial-decomp \ (\text{concat} \ pss) \)
  by (induct pss) (auto simp: monomial-decomp-def)

lemma monomial-decomp-map-times:
  assumes monomial-decomp \( ps \) and is-monomial \( f \) and \( \text{punit}.\text{lc} f = (1::'a::semiring-1) \)
  shows monomial-decomp \( (\map \ (\text{apfst} \ ((*) \ f)) \ ps) \)
  proof (rule monomial-decompI)
    fix \( h \ U \)
    assume \( (h, U) \in \ set \ (\map \ (\text{apfst} \ ((*) \ f)) \ ps) \)
    then obtain \( x \) where \( x \in \ set \ ps \) and \( (h, U) = \map \ (\text{apfst} \ ((*) \ f)) \ x \) unfolding set-map ..
    moreover obtain \( a \ b \) where \( x = (a, b) \) using prod.exhaust by blast
    ultimately have \( h = f \times a \) and \( (a, U) \in \ set \ ps \) by simp-all
    from assms(1) this(2) have is-monomial \( a \) and \( \text{punit}.\text{lc} a = 1 \) by (rule monomial-decompD)+
    from this(1) have monomial \( (\text{punit}.\text{lc} a) \) \( (\text{lpp} a) = a \) by (rule punit.monomial-eq-itself)
    moreover define \( t \) where \( t = \text{lpp} a \)
    ultimately have \( a = \text{monomial} 1 \ t \) by (simp only: \( \langle \text{punit}.\text{lc} a = 1 \rangle \))
    from assms(2) have monomial \( (\text{punit}.\text{lc} f) \) \( (\text{lpp} f) = f \) by (rule punit.monomial-eq-itself)
    moreover define \( s \) where \( s = \text{lpp} f \)
    ultimately have \( f = \text{monomial} 1 \ s \) by (simp only: assms(3))
    show is-monomial \( h \) by (simp add: \( h \ a \times \text{monomial-monomial} \) monomial-is-monomial)
    show \( \text{punit}.\text{lc} h = 1 \) by (simp add: \( h \ a \times \text{times-monomial-monomial} \) monomial-is-monomial)
  qed

lemma monomial-decomp-monomial-in-cone:
  assumes monomial-decomp \( ps \) and \( hU \in \ set \ ps \) 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 \( hU = (h, U) \) using prod.exhaust by blast
    with assms(2) have \( (h, U) \in \ set \ ps \) by simp
    with assms(1) have is-monomial \( h \) by (rule monomial-decompD)
    then obtain \( c \ s \) where \( h = \text{monomial} c \ s \) by (rule is-monomial-monomial)
    from assms(3) obtain \( q \) where \( q \in \text{P}[U] \) and \( a = q \times h \) unfolding \( hU \) by (rule coneE)
    from this(2) have \( a = h \times q \) by (simp only: mult.commute)
    also have \( \ldots = \text{punit}.\text{monom-mult} c \ s \ q \) by (simp only: h times-monomial-left)
    finally have \( a = \text{punit}.\text{monom-mult} c \ s \ q \).
    with True have \( t \in \text{keys} (\text{punit}.\text{monom-mult} c \ s \ q) \) by simp
    hence \( t \in (+) \ s \times \text{keys} q \) using punit.keys-monom-mult-subset[simplified] ..
    then obtain \( u \) where \( u \in \text{keys} q \) and \( t = s + u \).
    note this(1)
    also from \( q \in \text{P}[U] \) have \( \text{keys} q \subseteq \{U\} \) by (rule PolysD)
    finally have \( u \in \{U\} \).
    have monomial \( (\text{lookup} a \ t) \) \( t = \text{monomial} (\text{lookup} q \ u) \ u \times h \)
    by (simp add: \( a \times \text{punit}.\text{lookup-monom-mult} h \times \text{monomial-monomial-mult}\times\text{commute} \))
    moreover from \( u \in \{U\} \) have monomial \( (\text{lookup} q \ u) \ u \in \text{P}[U] \) by (rule

ultimately show \( \text{thesis unfolding} \ h U \) by (rule coneI)

next

  case False

  thus \( \text{thesis by (simp add: zero-in-cone in-keys-iff)} \)

qed

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

  assumes monomial-decomp ps and \( a \in \text{sum-list ' listset (map cone ps)} \) and \( t \in \text{keys a} \)

  obtains \( c \in \text{set ps} \) where \( c \neq 0 \) and monomial \( c t \in \text{cone (h, U)} \)

proof

  from assms (2) obtain qs where qs-in: \( \text{qs \in listset (map cone ps)} \) and \( a: a = \text{sum-list qs} \).

  from assms (3) keys-sum-list-subset have \( t \in \text{Keys (set qs)} \) unfolding a ..

  then obtain q where \( q \in \text{set qs} \) and \( t \in \text{keys q} \) by (rule in-KeysE).

  show \( \text{thesis} \)

proof

  from \( \text{ps ! i \in set ps} \) show \( h U \in \text{set ps} \) by (simp only: psi)

next

  from \( t \in \text{keys q} \) show \( \text{lookup q t \neq 0} \) by (simp add: in-keys-iff)

next

  from * show monomial \( \text{lookup q t \in cone (h, U)} \) by (simp only: psi)

qed

lemma hom-decompI: \( (\forall h U. (h, U) \in \text{set ps} \Rightarrow \text{homogeneous h}) \Rightarrow \text{hom-decomp ps} \)

by (auto simp: hom-decomp-def)

lemma hom-decompD: hom-decomp ps \( \Rightarrow (h, U) \in \text{set ps} \Rightarrow \text{homogeneous h} \)

by (auto simp: hom-decomp-def)

lemma hom-decomp-append-iff: hom-decomp \( (\text{ps @ qs}) \) \( \iff \text{hom-decomp ps} \land \text{hom-decomp qs} \)

by (auto simp: hom-decomp-def)

lemma hom-decomp-concat: \( (\forall ps. ps \in \text{set pss} \Rightarrow \text{hom-decomp ps}) \Rightarrow \text{hom-decomp (concat pss)} \)
by (induct pss) (auto simp: hom-decomp-def)

lemma hom-decomp-map-times:
  assumes hom-decomp ps and homogeneous f
  shows hom-decomp (map (apfst ((*) f)) ps)
proof (rule hom-decompI)
  fix h U
  assume (h, U) ∈ set (map (apfst ((*) f)) ps)
  then obtain x where x ∈ set ps and (h, U) = apfst ((*) f) x unfolding set-map ..
  moreover obtain a b where x = (a, b) using prod.exhaust by blast
  ultimately have h: h = f * a and (a, U) ∈ set ps by simp-all
  from assms(1) this(2) have 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) ∈ set ps
  with assms have is-monomial h by (rule monomial-decompD)
  then obtain c t where h: h = monomial c t by (rule is-monomial-monomial)
  show homogeneous h unfolding h by (fact homogeneous-monomial)
qed

lemma cone-decompI: direct-decomp T (map cone ps) ⇒ cone-decomp T ps
unfoldning cone-decomp-def by blast

lemma cone-decompD: cone-decomp T ps ⇒ direct-decomp T (map cone ps)
unfoldning cone-decomp-def by blast

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

lemma cone-decomp-indets:
  assumes cone-decomp ![0] ps and ![0] ⊆ ![1] ![0] and ![0]oreach ![1] ![0] = ![0]̸ = (0:: ![1]) ![0]∈ ![0] ![0]̸ = 0 = ⇒ ![1] ⊆ ![2] proof
− from assms ![1] ![0] have cone ![0] ![1] ⊆ ![0] by (rule cone-decomp-cone-subset)
hence cone ![0] ![1] ⊆ ![1] using assms ![1] ![2] by (rule subset-trans)
thus ![0] ∈ ![1] and ![0]̸ = 0 = ⇒ ![1] ⊆ ![2] by (rule cone-subset-PolysD)+
qed

lemma cone-decomp-closed-plus:
  assumes cone-decomp ![0] ps and ![0] ∈ ![0] and ![1] ∈ ![0] shows ![0] + ![1] ∈ ![0] proof
− from assms ![1] ![0] have dd: direct-decomp ![0] (map cone ps) by (rule cone-decompD)
then obtain qsa where qsa: qsa ∈ listset (map cone ps) and ![0]: ![0] = sum-list qsa using assms ![1] ![2] by (rule direct-decompE)
from dd assms ![1] ![3] have ![0] = sum-list (map2 (+) qsa qsb) by (simp only: sum-list-map2-plus ![0] ![1])
also from dd have sum-list (map2 (+) qsa qsb) ∈ ![0] proof (rule direct-decompD)
  from qsa qsb show map2 (+) qsa qsb ∈ listset (map cone ps) proof (rule listset-closed-map2)
    fix c ![1] ![2]
    assume c ∈ set (map cone ps)
    then obtain hU where c: c = cone hU by auto
    assume ![1] ∈ c and ![2] ∈ c
    thus ![1] + ![2] ∈ c unfolding c by (rule cone-closed-plus)
  qed
  qed
finally show ![0] = thesis .
qed

lemma cone-decomp-closed-uminus:
  assumes cone-decomp ![0] ps and (a:: ⇒ ![0] : :: comm-ring) ∈ ![0]
shows $a \in T$

proof

from assms(1) have $dd$: direct-decomp $T$ (map cone $ps$) by (rule cone-decompD)
then obtain $qsa$ where $qsa$: $qsa \in \text{listset} \ (\text{map cone} \ ps)$ and $a$: $a = \text{sum-list} \ qsa \ \text{using} \ \text{assms}(2)$
by (rule direct-decompE)
from $qsa$ have $\text{length} \ qsa = \text{length} \ (\text{map cone} \ ps)$ by (rule listsetD)
have $-a = \text{sum-list} \ (\text{map uminus} \ qsa)$ unfolding $a$ by (induct $qsa$, simp-all)
also from $dd$ have ... $\in T$
proof (rule direct-decompD)
from $qsa$ show $\text{map uminus} \ qsa \in \text{listset} \ (\text{map cone} \ ps)$
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 $\text{?thesis}$.

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 $a \ominus b \in T$
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 $\text{?thesis}$ 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 $\text{cone} \ (t, U) \ [(t, U)]$
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 $S1$ $\ (\text{map cone} \ ps)$ 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 $S2 \ (\text{map cone} \ qs)$ by (rule cone-decompD)
ultimately have direct-decomp $T \ (\text{map cone} \ ps @ \text{map cone} \ qs)$ by (intro direct-decomp-direct-decomp)
thus direct-decomp $T \ (\text{map cone} \ (ps @ qs))$ by simp
lemma cone-decomp-concat:
assumes direct-decomp T ss and length pss = length ss
and \( i \in \{ i \mid i < \text{length } ss \} \rightarrow \text{cone-decomp } (ss ! i) (pss ! i) \)
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.prems)
hence cone-decomp s ps by simp
hence *: direct-decomp s \( (\text{map cone } ps) \) by (rule cone-decompD)
with Cons.prems(1) have direct-decomp T \( (ss @ \text{map cone } ps) \) by (rule direct-decomp-direct-decomp)
hence 1: direct-decomp T \[ \text{sum-list } ' \text{listset } ss, \text{sum-list } ' \text{listset } (\text{map cone } ps) \] and 2: direct-decomp \( (\text{sum-list } ' \text{listset } ss) ss \) by (auto dest: direct-decomp-appendD intro!: empty-not-in-map-cone)
note 1
moreover from 2 have cone-decomp \( \text{sum-list } ' \text{listset } ss ) (\text{concat } pss) \)
proof (rule cone-decompI)
fix i
assume i < length ss
hence cone-decomp \( ((s \# ss) ! Suc i) ((ps \# pss) ! Suc i) \) by (rule Cons.prems)
thus cone-decomp \( (ss ! i) (pss ! i) \) by simp
qed
moreover have cone-decomp \( \text{sum-list } ' \text{listset } (\text{map cone } ps) \) ps
proof (intro cone-decompI direct-decompI refl)
from * show inj-on sum-list \( \text{listset } (\text{map cone } ps) \)
by (simp only: direct-decomp-def bij_betw_def)
qed
ultimately have cone-decomp T \( (\text{concat } pss @ ps) \) by (rule cone-decomp-append)
hence direct-decomp T \( \text{map cone } (\text{concat } pss) @ \text{map cone } ps \) by (simp add: cone-decomp-def)
hence direct-decomp T \( \text{map cone } ps @ \text{map cone } (\text{concat } pss) \)
using perm-append-swap by (rule 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 \( ((\cdot) s \cdot T) (\text{map } (\text{apfst } ((\cdot})(s:::-0-::\{\text{comm-ring-1},\text{ring-no-zero-divisors}\})))) ps \)
proof (rule cone-decompI)
from assms have direct-decomp T \( \text{map cone } ps \) by (rule cone-decompD)
hence direct-decomp \( ((\cdot) s \cdot T) (\text{map } (\cdot) ((\cdot) s)) \) (map cone ps)
by (rule direct-decomp-image-times) (rule times-canc-left)
also have \( \text{map } ((\cdot)) ((\cdot) s) \text{ (map cone ps)} = \text{map cone } (\text{map apfst ((\cdot) s)} \text{ ps}) \)
by (simp add: cone-image-times')
finally show direct-decomp ((\cdot) s · T) (\text{map cone } (\text{map apfst ((\cdot) s)} \text{ ps})) .
qed

lemma cone-decomp-perm:
  assumes \( \text{cone-decomp T ps and perm ps qs} \)
  shows \( \text{cone-decomp T qs} \)
  using assms(1) unfolding cone-decomp-def
proof (rule direct-decomp-perm)
  from assms(2) show \( \text{perm (map cone ps)} \text{ (map cone qs)} \)
  by (induct ps qs rule: perm.induct) auto
qed

lemma valid-cone-decomp-subset-Polys:
  assumes \( \text{valid-decomp X ps and cone-decomp T ps} \)
  shows \( T \subseteq P[X] \)
proof
  fix \( p \)
  assume \( p \in T \)
  from assms(2) have \( \text{direct-decomp T (map cone ps)} \) by (rule cone-decompD)
  then obtain \( qs \) where \( qs \in \text{listset (map cone ps)} \) and \( p = \text{sum-list qs} \)
  using \( \langle p \in T \rangle \) by (rule direct-decompE)
  from Cons.prems(1) this(1) show \( \text{p \in P[X]} \) 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 \( a = (h, U) \) using prod.exhaust by blast
    hence \( (h, U) \in \text{set (a \# ps)} \) by simp
    with Cons.prems(1) have \( h \in P[X] \) and \( U \subseteq X \) by (rule valid-decompD)+
    hence \( \text{cone a \subseteq P[X]} \) unfolding a by (rule cone-subset-PolysI)
    from Cons.prems(1) have \( \text{valid-decomp X ps} \) by (simp add: valid-decomp-def)
    from Cons.prems(2) have \( qs \in \text{listset (cone a \# map cone ps)} \) by simp
    then obtain \( q \) \( qs' \) where \( q \in \text{cone a and qs': qs' \in listset (map cone ps)} \) and \( qs: qs = q \# qs' \)
    by (rule listset-ConsE)
    from this(1) :cone a \subseteq P[X]; have q \in P[X] ..
    moreover from \( \text{valid-decomp X ps}; qs' \) \( \text{have sum-list qs' \in P[X]} \) 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 ∈ set (map cone ps)
  then obtain hU where hU ∈ set ps and cn = cone hU unfolding set-map
  moreover obtain h U where hU = (h, U) using prod.exhaust by blast
  ultimately have (h, U) ∈ 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::(- ⇒₀ ::field) set)
proof (rule phull.subspace-direct-decomp)
  from assms show direct-decomp T (map cone ps) by (rule cone-decompD)
next
  fix cn
  assume cn ∈ set (map cone ps)
  then obtain hU where hU ∈ set ps and cn = cone hU unfolding set-map
  show phull.subspace cn unfolding cn by (rule subspace-cone)
qed

definition pos-decomp :: ((('x ⇒₀ nat) ⇒₀ 'a) × 'x set) list ⇒ ((('x ⇒₀ nat) ⇒₀ 'a) × 'x set) list
  ((-+) [1000] 999)
  where pos-decomp ps = filter (λp. snd p ≠ {}) ps

definition standard-decomp :: nat ⇒ (((('x ⇒₀ nat) ⇒₀ 'a::zero) × 'x set) list ⇒ bool
  where standard-decomp k ps ←→ (∀(h, U)∈set (ps+). k ≤ poly-deg h ∧
  (∀d. k ≤ d → d ≤ poly-deg h →
  (∃(h', U')∈set ps. poly-deg h' = d ∧ card U ≤
  card U')))

lemma pos-decomp- Nil [simp]; [] = []
  by (simp add: pos-decomp-def)

lemma pos-decomp-subset: set (ps+) ⊆ 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: \((\text{concat }\textit{ps})_+ = \text{concat } (\text{map pos-decomp }\textit{ps})\)
by (metis (mono-tags, lifting) filter-concat map-ev-cone pos-decomp-def)

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

lemma card-Diff-pos-decomp: \(\{ (h, U) \in \text{set } \textit{qs} - \text{set } (\textit{qs}_+) \mid P \textit{h} \} = \text{card } \{ h. (\{\}), \text{set } \textit{qs} \land P \textit{h} \}\)
proof
have \(\{ h. (\{\}), \text{set } \textit{qs} \land P \textit{h} \} = \text{fst } \{ (h, U) \in \text{set } \textit{qs} - \text{set } (\textit{qs}_+) \mid P \textit{h} \}\)
  by (auto simp: pos-decomp-def image-Collect)
also have \(\text{card } \ldots = \text{card } \{ (h, U) \in \text{set } \textit{qs} - \text{set } (\textit{qs}_+) \mid P \textit{h} \}\)
  by (rule card-image, auto simp: pos-decomp-def intro: inj-onI)
finally show \(\text{thesis} \text{ by (rule sym)}\)
qed

lemma standard-decompI: \(\forall h U. (h, U) \in \text{set } (\textit{ps}_+) \longrightarrow k \leq \text{poly-deg } h\)
  and \(\forall h U d. (h, U) \in \text{set } (\textit{ps}_+) \longrightarrow k \leq d \longrightarrow d \leq \text{poly-deg } h \longrightarrow (3\text{ }h' U', (h', U') \in \text{set } \textit{ps} \land \text{poly-deg } h' = d \land \text{card } U \leq \text{card } U')\)
shows standard-decomp k ps
unfolding standard-decomp-def using assms by blast

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

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

lemma standard-decomp-Nil: \(\textit{ps}_+ = [] \implies \text{standard-decomp } k \textit{ps}\)
by (simp add: standard-decomp-def)

lemma standard-decomp-singleton: standard-decomp \((\text{poly-deg } h) [\{ (h, U) \}]\)
by (simp add: standard-decomp-def pos-decomp-def)

lemma standard-decomp-concat: \(\forall \textit{ps}. \textit{ps} \in \text{set } \textit{ps} \Rightarrow \text{standard-decomp } k \textit{ps}\)
shows standard-decomp k (concat \textit{ps})
proof (rule standard-decompI)
  fix h U
  assume \((h, U) \in \text{set } ((\text{concat }\textit{ps})_+)\)
  then obtain \textit{ps} where \textit{ps} \in \text{set } \textit{ps} \text{ and } \ast: (h, U) \in \text{set } (\textit{ps}_+) \text{ by (auto simp: pos-decomp-concat)}
  from this(1) have standard-decomp k \textit{ps} by (rule assms)

118
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 $(\text{standard-decomp } k \text{ ps})$ * obtain $h' \ U' \text{ where } (h', U') \in \text{ set ps and }$
\text{poly-deg } h' = d
\text{ and card } U \leq \text{ card } U' \text{ by (rule standard-decompE)}
\text{note this(2, 3)}
moreover from $(h', U') \in \text{ set ps} \land \text{ ps } \in \text{ set pss} \Rightarrow (h', U') \in \text{ set (concat pss)}$ by auto
ultimately show $\exists h' \ U'. \ (h', U') \in \text{ set (concat pss)} \land \text{poly-deg } h' = d \land \text{ card } U \leq \text{ card } U'$
\text{by blast}

qed

corollary standard-decomp-append:
assumes standard-decomp $k \text{ ps}$ and standard-decomp $k \text{ qs}$
shows standard-decomp $k \ (\text{ps @ qs})$
proof
have standard-decomp $k \ \text{(concat [ps, qs])}$ by (rule standard-decomp-concat) (auto)
\text{simp: assms)
thus $\text{thesis}$ by simp

qed

lemma standard-decomp-map-times:
assumes standard-decomp $k \text{ ps}$ and valid-decomp $X \text{ ps}$ and $s \neq (0 :: - \Rightarrow 0 \text{ a::semiring-no-zero-divisors})$
shows standard-decomp $(k + \text{poly-deg } s) \ (\text{map (apfst } ((*) \ s) \text{) ps})$
proof (rule standard-decompI)
fix $h \ U$
assume $(h, U) \in \text{ set ((map (apfst } ((*) \ s) \text{) ps})_+)$
then obtain $h0$ where $1: \ (h0, U) \in \text{ set (ps +)} \land \text{ h: } h = s \ast h0 \text{ by (fastforce simp: pos-decomp-map)}$
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 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) I 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) I obtain $h' \ U'$ where $2: \ (h', U') \in \text{ set ps and } \text{poly-deg } h' = d - \text{poly-deg } s$
\text{ and card } U \leq \text{ card } U' \text{ by (rule standard-decompE)}
from assms(2) this(1) have $h' \neq 0$ by (rule valid-decompD)
with assms(3) have deg-h: $\text{poly-deg } (s \ast h') = \text{poly-deg } s + \text{poly-deg } h'$ by (rule
poly-deg-times)
  from 2 have \((s * h', U') \in \text{set (map (apfst ((*) s)) ps)}\) by force
moreover from \((k + \text{poly-deg} s \leq d); \text{poly-deg} h' = d - \text{poly-deg} s\) have poly-deg
\((s * h') = d\)
  by (simp add: deg-h')
ultimately show \(\exists h' \quad (h', U') \in \text{set (map (apfst ((*) s)) ps)} \land \text{poly-deg} h'
= d \land \text{card} U \leq \text{card} U'\)
  using \(\text{card} U \leq \text{card} U'\) by fastforce
qed

lemma standard-decomp-nonempty-unique:
  assumes finite X and valid-decomp X ps and standard-decomp k ps and ps+ \(\neq \emptyset\)
  shows \(k = \text{Min (poly-deg ' fst ' set (ps+)})\)
proof
  let \(?A = \text{poly-deg ' fst ' set (ps+)}\)
  define \(m\ where\ m = \text{Min} \ ?A\)
  have finite \(?A\) by simp
moreover from assms(4) have \(?A \neq \emptyset\) by simp
ultimately have \(m \in \ ?A\) unfolding m-def by (rule Min-in)
then obtain \(h \quad U\) where \((h, U) \in \text{set (ps+)}\) and \(m = \text{poly-deg} h\) 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 poly-deg (fst (h', U')) \(\in \ ?A\) by (intro imageI)
  with finite \(?A\): 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) (h, U) \(\in \text{set (ps+)})\) le-refl obtain \(h' \quad U'\)
    where \((h', U') \in \text{set ps and poly-deg} h' = k\) and \(\text{card} U \leq \text{card} U'\) by (rule standard-decompE)
from this(2) \(k < m\) have \(m < m \leq \text{poly-deg} h'\) by simp
with \(m\)-min have \((h', U') \notin \text{set (ps+})\) by blast
with \(\langle h', U' \rangle \in \text{set ps}\) have \(U' = \{\}\) by (simp add: pos-decomp-def)
with \(\text{card} U \leq \text{card} U'\) have \(U = \{\}\ \lor \text{infinite} U\) by (simp add: card-eq-0-iff)
thus \(?thesis\)
proof
  assume \(U = \{\}\)
  with \(\langle h', U' \rangle \in \text{set ps}\) show \(?thesis\) by (simp add: pos-decomp-def)
next
  assume infinite \(U\)
moreover from assms(1, 2) have \(\text{finite} U\)
proof (rule valid-decompD-finite)
from \(\langle h', U' \rangle \in \text{set (ps+)}\) show \(h, U) \in \text{set ps}\) by (simp add: pos-decomp-def)
qed
ultimately show \(?thesis\).

\quad qed

next

\quad assume \(m \leq k\)
\quad hence \(k \leq m\) by simp

moreover from assms(3) \((h, U) \in \text{set} (ps_+): \text{have} \ k \leq m\) unfolding \(m\) by (rule standard-decompD)

\quad ultimately show \(?thesis\).

\quad qed (simp only: \(m\)-def)

\quad qed

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

\quad from assms(2, 1) have finite \(U\) by (rule finite-subset)

\quad thus \(?thesis\) using assms(2) that

\quad proof (induct \(U\) arbitrary: \(\text{thesis rule: finite-induct)\)

\quad case empty

\quad from assms(3, 4) have valid-decomp \(X \ [(h, \{\})]\) by (simp add: valid-decomp-def)

\quad moreover note cone-decomp-singleton

\quad moreover have standard-decomp (Suc (poly-deg \(h\)) \([(h, \{\})]\)

\quad by (rule standard-decomp-\(\text{Nil}\) (simp add: \(\text{pos-decomp-def}\)

\quad ultimately show \(?case\) by (rule empty) (simp-all add: monomial-decomp-def hom-decomp-def)

\quad qed

\quad next

\quad case (insert \(x\) \(U\))

\quad from insert.prems(1) have \(x \in X\) and \(U \subseteq X\) by simp-all

\quad from this(2) obtain \(ps\) where \(0: \text{valid-decomp X \(ps\) and 1: cone-decomp (cone \((h, U)\) \(ps\)\)
and \(2: \text{standard-decomp (Suc (poly-deg \(h\)) \(ps\)\)
and \(3: \text{is-monomial \(h \Rightarrow \text{punit.lc \(h\)} = 1 \Rightarrow \text{monomial-decomp ps}\)
and \(4: \text{homogeneous \(h \Rightarrow \text{hom-decomp ps}\)} by (rule insert.hyps) blast

\quad let \(\exists x = \text{monomial (1::\(a\)) (Poly-Mapping.single x (Suc 0))\)

\quad have \(\exists x \neq 0\) by (simp add: monomial-\(\neq\)-iff)

\quad with assms(\(\_\)\) have \(\text{deg: poly-deg (\(\exists x \ast h\) = Suc (poly-deg \(h\))\)

\quad by (simp add: poly-deg-times poly-deg-monomial deg-pm-single)

\quad define \(qs\) where \(qs = [(\(\exists x \ast h\), insert \(x\) \(U)]\)

\quad show \(?case\)

\quad proof (rule insert.prems)

\quad from \(x \in X\) have \(\exists x \in P[X]\) by (intro Polys-closed-monomial PPs-closed-single)

\quad hence \(\exists x \ast h \in P[X]\) using assms(3) by (rule Polys-closed-times)

\quad moreover from \((\exists x \neq 0)\) assms(\(\_\)) have \(\exists x \ast h \neq 0\) by (rule times-not-zero)

\quad ultimately have \(\text{valid-decomp X \(qs\) using insert.hyps(1) \(x \in X\) \(U \subseteq X)\)

\quad by (simp add: qs-def valid-decomp-def)

\quad with 0 show \(\text{valid-decomp X (ps \(\oplus\) \(qs)\) by (rule valid-decomp-append)\)
next
  show cone-decomp (cone (h, insert x U)) (ps @ qs)
proof (rule cone-decomp-append)
    show direct-decomp (cone (h, insert x U)) [cone (h, U), cone (?x * h, insert x U)]
      using insert.hyps(2) by (rule direct-decomp-cone-insert)
next
  show cone-decomp (cone (?x * h, insert x U)) qs by (simp add: qs-def cone-decomp-singleton)
qed (fact 1)

next
from standard-decomp-singleton[of ?x * h insert x U]
  have standard-decomp (Suc (poly-deg h)) qs by (simp add: deg qs-def)
  with 2 show standard-decomp (Suc (poly-deg h)) (ps @ qs) by (rule standard-decomp-append)

next
  assume is-monomial h and punit.lc h = 1
  hence monomial-decomp ps by (rule 3)
  moreover have monomial-decomp qs proof
    have is-monomial (?x * h) by (metis ⟨is-monomial h⟩ is-monomial-monomial monomial-is-monomial mult.commute)
    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 proof
    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 ⇒ 0 nat) ⇒ 0 'a::{comm-ring-1,ring-no-zero-divisors}) set) ps and standard-decomp k ps and k ≤ d obtains qs where valid-decomp X qs and cone-decomp T qs and standard-decomp d qs and monomial-decomp ps ⇒ monomial-decomp qs and hom-decomp ps ⇒ hom-decomp qs
proof
  have ∃ qs. valid-decomp X qs ∧ cone-decomp T qs ∧ standard-decomp (k + i) qs

122
(monomial-decomp ps → monomial-decomp qs) ∧ (hom-decomp ps → hom-decomp qs) for i

proof (induct i)
case 0
  from assms(2, 3, 4) show ?case unfolding add-0-right by blast
next
case (Suc i)
then obtain qs where 0: valid-decomp X qs and 1: cone-decomp T qs
  and 2: standard-decomp (k + i) qs and 3: monomial-decomp ps ⇒ monomial-decomp qs
  and 4: hom-decomp ps ⇒ hom-decomp qs by blast
let ?P = λhU. poly-deg (fst hU) ≠ k + i
define rs where rs = filter (¬ ?P) qs
define ss where ss = filter ?P qs

have set rs ⊆ set qs by (auto simp: rs-def)
have set ss ⊆ set qs by (auto simp: ss-def)
define f where f = (λhU. SOME ps'. valid-decomp X ps' ∧ cone-decomp (cone hU) ps') ∧
  standard-decomp (Suc (poly-deg ((fst hU)::('a ⇒ 0 a))) ps') ∧
  (monomial-decomp ps → monomial-decomp ps') ∧
  (hom-decomp ps → hom-decomp ps')

have valid-decomp X (f hU) ∧ cone-decomp (cone hU) (f hU) ∧ standard-decomp
  (Suc (k + i)) (f hU) ∧
  (monomial-decomp ps → monomial-decomp (f hU)) ∧ (hom-decomp ps → hom-decomp (f hU))
if hU ∈ set rs for hU
proof –
  obtain h U where hU: hU = (h, U) using prod.exhaust by blast
  with that have eq: poly-deg (fst hU) = k + i by (simp add: rs-def)
  from that (set rs ⊆ set qs) have (h, U) ∈ set qs unfolding hU ..
  with 0 have U ⊆ X and h ∈ P[X] and h ≠ 0 by (rule valid-decompD)+
  with assms(1) obtain ps' where valid-decomp X ps' and cone-decomp (cone (h, U)) ps'
    and standard-decomp (Suc (poly-deg h)) ps'
    and md: is-monomial h ⇒ punit.le h = 1 ⇒ monomial-decomp ps'
    and hd: homogeneous h ⇒ hom-decomp ps' by (rule standard-decomp-SucE)
  blast

note this(1–3)
moreover have monomial-decomp ps' if monomial-decomp ps
proof –
  from that have monomial-decomp qs by (rule 3)
  hence is-monomial h and punit.le h = 1 using (h, U) ∈ set qs; by (rule monomial-decompD)+
  thus ?thesis by (rule md)
qed
moreover have hom-decomp ps' if hom-decomp ps
proof
  from that have hom-decomp qs by (rule 4)
  hence homogeneous h using (h, U) ∈ set qs by (rule hom-decompD)
  thus ?thesis by (rule hd)
qed
ultimately have valid-decomp X ps' ∧ cone-decomp (cone hU) ps' ∧
  standard-decomp (Suc (poly-deg (fst hU))) ps' ∧ (monomial-decomp ps →
  monomial-decomp ps') ∧
  (hom-decomp ps → hom-decomp ps') by (simp add: hU)
thus ?thesis unfolding f-def eq by (rule someI)
qed
hence f1: ∀ ps. ps ∈ set (map f rs) ⇒ valid-decomp X ps
  and f2: ∀ h U. h U ∈ set rs ⇒ cone-decomp (cone hU) (f hU)
  and f3: ∀ ps. ps ∈ set (map f rs) ⇒ standard-decomp (Suc (k + i)) ps
  and f4: ∀ ps'. monomial-decomp ps ⇒ ps' ∈ set (map f rs) ⇒ monomial-decomp
  ps'
  and f5: ∀ ps'. hom-decomp ps ⇒ ps' ∈ set (map f rs) ⇒ 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) ∈ set ss
    hence (h, U) ∈ set qs using (set ss ⊆ set qs) ..
    with 0 show h ∈ P[X] and h ≠ 0 and U ⊆ 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)) = unfolding rs'-def
proof (rule direct-decomp-concat)
  fix i

124
assume \(*\): \(i < \text{length (map cone } rs\)

\text{hence } rs ! i \in \text{set } rs \text{ by simp}

\text{hence cone-decomp (cone } (rs ! i)) (f (rs ! i)) \text{ by (rule f2)}

\text{with } \ast \text{ show cone-decomp (map cone } rs ! i) (map f rs ! i) \text{ by simp}

\text{qed simp}

\text{with } T \text{ ss show cone-decomp } T (ss @ rs') \text{ by (rule cone-decomp-append)}

\text{next}

\text{have standard-decomp } (Suc (k + i)) \text{ ss}

\text{proof (rule standard-decompI)}

\text{fix } h \text{ U}

\text{assume } (h, U) \in \text{set } (ss_+)

\text{hence } (h, U) \in \text{set } (qs_+) \text{ and poly-deg } h \neq k + i \text{ by (simp-all add: pos-decomp-def ss-def)}

\text{from 2 this(1) have } k + i \leq \text{poly-deg } h \text{ by (rule standard-decompD)}

\text{with } (\text{poly-deg } h \neq k + i) \text{ show } Suc (k + i) \leq \text{poly-deg } h \text{ by simp}

\text{fix } d'

\text{assume } Suc (k + i) \leq d' \text{ and } d' \leq \text{poly-deg } h

\text{from this(1) have } k + i \leq d' \text{ and } d' \neq k + i \text{ by simp-all}

\text{from 2 } (h, U) \in \text{set } (qs_+) \text{ this(1) obtain } h' U'

\text{where } (h', U') \in \text{set } qs \text{ and } \text{poly-deg } h' = d' \text{ and } \text{card } U \leq \text{card } U'

\text{using } d' \leq \text{poly-deg } h \text{ by (rule standard-decompE)}

\text{moreover from } d' \neq k + i \text{ this(1, 2) have } (h', U') \in \text{set } ss \text{ by (simp add: ss-def)}

\text{ultimately show } \exists h' U'. (h', U') \in \text{set } ss \land \text{poly-deg } h' = d' \land \text{card } U \leq \text{card } U' \text{ by blast}

\text{qed}

\text{moreover have standard-decomp } (Suc (k + i)) \text{ rs'}

\text{unfolding } rs'\text{-def using f3 by (rule standard-decomp-concat)}

\text{ultimately show standard-decomp } (Suc (k + i)) (ss @ rs') \text{ by (rule standard-decomp-append)}

\text{next}

\text{assume } *: \text{monomial-decomp ps}

\text{hence monomial-decomp } qs \text{ by (rule 3)}

\text{hence monomial-decomp ss \text{ by (simp add: monomial-decomp-def ss-def)}}

\text{moreover have monomial-decomp } rs'

\text{unfolding } rs'\text{-def using f4[OF *] by (rule monomial-decomp-concat)}

\text{ultimately show monomial-decomp } (ss @ rs') \text{ by (simp only: monomial-decomp-append-iff)}

\text{next}

\text{assume } *: \text{hom-decomp ps}

\text{hence hom-decomp } qs \text{ by (rule 4)}

\text{hence hom-decomp ss \text{ by (simp add: hom-decomp-def ss-def)}}

\text{moreover have hom-decomp } rs' \text{ unfolding } rs'\text{-def using f5[OF *] by (rule hom-decomp-concat)}

\text{ultimately show hom-decomp } (ss @ rs') \text{ by (simp only: hom-decomp-append-iff)}

\text{qed}

\text{qed}

\text{then obtain } qs \text{ where } 1: \text{valid-decomp } X \text{ qs and } 2: \text{cone-decomp } T \text{ qs}

\text{and standard-decomp } (k + (d - k)) \text{ qs and } 4: \text{monomial-decomp ps} \implies
monomial-decomp qs
and 5: hom-decomp ps \Rightarrow hom-decomp qs by blast
from this(3) assms(5) have standard-decomp d qs by simp
with 1 2 show \textup{thesis} using 4 5 ..
qed

10.5 Splitting w.r.t. Ideals

context
fixes X :: 'x set
begin

definition splits-wrt :: (((('x \Rightarrow \textup{nat}) \Rightarrow 'a) \times 'x set) list \times (((('x \Rightarrow \textup{nat}) \Rightarrow 'a) \times 'x set) list) \Rightarrow
(('x \Rightarrow \textup{nat}) \Rightarrow 'a::comm-ring-1) set \Rightarrow ((('x \Rightarrow \textup{nat}) \Rightarrow 'a) set \Rightarrow bool
where
splits-wrt pqT F \leftrightarrow cone-decomp T (fst pqT @ snd pqT) \land
\left( \forall hU \in set (fst pqT). \text{cone } hU \subseteq \text{ideal } F \cap P[X] \right) \land
\left( \forall (h, U) \in set (snd pqT). \text{cone } (h, U) \subseteq P[X] \land \text{cone } (h, U) \cap \text{ideal } F = \{0\} \right)

lemma splits-wrtI:
assumes cone-decomp T (ps @ qs)
and \bigwedge hU. (h, U) \in set ps \Rightarrow cone (h, U) \subseteq P[X] and \bigwedge hU. (h, U) \in set
ps \Rightarrow h \in \text{ideal } F
and \bigwedge hU. (h, U) \in set qs \Rightarrow cone (h, U) \subseteq P[X]
and \bigwedge hU a. (h, U) \in set qs \Rightarrow a \in cone (h, U) \Rightarrow a \in \text{ideal } F \Rightarrow a = 0
shows splits-wrt (ps, qs) T F
unfolding splits-wrt-def fst-conv snd-conv
proof (intro conjI ballI)
fix hU
assume hU \in set ps
moreover obtain h U where hU: hU = (h, U) using prod.exhaust by blast
ultimately have (h, U) \in set ps by simp
hence cone (h, U) \subseteq P[X] and h \in \text{ideal } F by (rule assms)+
from - this(1) show cone hU \subseteq \text{ideal } F \cap P[X] unfolding hU
proof (rule Int-greatest)
show cone (h, U) \subseteq \text{ideal } F
proof
fix a
assume a \in cone (h, U)
then obtain a' where a' \in P[U] and a = a' \ast 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 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 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 assms)+

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 \((h, U) \Rightarrow \text{cone} (h, U) \subseteq P[X] \land \text{cone} (h, U) \cap \text{ideal } F = \{0\}\)

by (fastforce simp: hU)

qed (fact assms)+

lemma splits-wrtI-valid-decomp:

assumes valid-decomp \(X ps\) and valid-decomp \(X qs\) and cone-decomp \(T (ps \oplus qs)\)

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 cone-subset-PolysI)

fix \(h U\)

assume \((h, U) \in \text{set } ps\)

thus \(h \in \text{ideal } F\) by (rule assms(4))

from assms(1) \((h, U) \in \text{set } ps\) 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) \((h, U) \in \text{set } qs\) 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 \(T (ps \oplus qs)\) 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 \(\cdot\) listset \((\map \text{cone } ps)\) \subseteq \text{ideal } F \cap P[X]

proof

fix \(x\)

assume \(x\text{-in: } x \in \text{sum-list } \cdot\) listset \((\map \text{cone } ps)\)

have listset \((\map \text{cone } ps)\) \subseteq listset \((\map (\lambda. \text{ideal } F \cap P[X]) ps)\)

proof (rule listset-mono)

fix \(i\)

assume \(i\text{: } i < \text{length } (\map (\lambda. \text{ideal } F \cap P[X]) ps)\)
hence \( ps ! i \in \text{set } ps \) by simp
with assms(1) have cone \((ps ! i) \subseteq \text{ideal } F \cap P[X]\) by (rule splits-wrtD)
with \( i \) show \( \text{map cone } ps ! i \subseteq \text{map } (\lambda-. \text{ideal } F \cap P[X]) \) ps \( ! i \) by simp
qed simp
hence \( \text{sum-list } \cdot \text{listset } (\text{map cone } ps) \subseteq \text{sum-list } \cdot \text{listset } (\text{map } (\lambda-. \text{ideal } F \cap P[X]) \) ps)
by (rule image-mono)
with \( x \in \text{in } \) have \( x \in \text{sum-list } \cdot \text{listset } (\text{map } (\lambda-. \text{ideal } F \cap P[X]) \) ps) ..
then obtain \( xs \) where \( xs : xs \in \text{listset } (\text{map } (\lambda-. \text{ideal } F \cap P[X]) \) 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 \( y = xs ! i \) by (metis in-set-conv-nth)
moreover from \( xs \) have \( \text{length } xs = \text{length } (\text{map } (\lambda-. \text{ideal } F \cap P[X]) \) ps)
by (rule listsetD)
ultimately have \( i < \text{length } (\text{map } (\lambda-. \text{ideal } F \cap P[X]) \) ps) by simp
moreover from \( xs \) this have \( xs ! i \in (\text{map } (\lambda-. \text{ideal } F \cap P[X]) \) 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

lemma splits-wrt-image-sum-list-snd-subset:
assumes \( \text{splits-wrt } (ps, qs) \) \( T \) \( F \)
shows \( \text{sum-list } \cdot \text{listset } (\text{map cone } qs) \subseteq P[X] \)
proof
fix \( x \)
assume \( x \in \text{in } \) have \( \text{listset } (\text{map cone } qs) \subseteq \text{listset } (\text{map } (\lambda-. P[X]) \) qs)
proof (rule listset-mono)
fix \( i \)

**Proof**

\[
\begin{align*}
\text{assume } & i < \text{length } (\text{map } (\lambda -. P[X]) q) \\
\text{hence } & q! i \in \text{set } q \text{ by simp} \\
\text{with } & \text{assms}(1) \text{ have cone } (q! i) \subseteq P[X] \text{ by (rule splits-wrtD)} \\
\text{with } & i \text{ show map cone } q! i \subseteq \text{map } (\lambda -. P[X]) q! i \text{ by simp} \\
\text{qed simp} \\
\text{hence } & \text{sum-list ' listset } (\text{map cone } q) \subseteq \text{sum-list ' listset } (\text{map } (\lambda -. P[X]) q) \\
\text{by } & \text{(rule image-mono)} \\
\text{with } & \text{x-in have } x \in \text{sum-list ' listset } (\text{map } (\lambda -. P[X]) q) \ldots \\
\text{then } & \text{obtain } \text{xs where } \text{xs} \in \text{listset } (\text{map } (\lambda -. P[X]) q) \text{ and } x = \text{sum-list } \text{x} \ldots \\
\text{show } & x \in P[X] \text{ unfolding } x \\
\text{proof } & \text{(rule Polys-closed-sum-list)} \\
\text{fix } & y \\
\text{assume } & y \in \text{set } x \\
\text{then } & \text{obtain } i \text{ where } i < \text{length } x \text{ and } y = x! i \text{ by } \text{(metis in-set-conv-nth)} \\
\text{moreover from } & \text{xs have } \text{length } x = \text{length } (\text{map } (\lambda -. P[X]):(- \Rightarrow 0 \ 'a) \set) q \\
\text{by } & \text{(rule listsetD)} \\
\text{ultimately have } & i < \text{length } (\text{map } (\lambda -. P[X]) q) \text{ by simp} \\
\text{moreover from } & \text{xs this have } x! i \in (\text{map } (\lambda -. P[X]) q) \text{ by } \text{(rule listsetD)} \\
\text{ultimately show } & y \in P[X] \text{ by } \text{(simp add: y)} \\
\text{qed} \quad \text{qed} \\
\end{align*}
\]

**Lemma** \( \text{splits-wrt-cone-decomp-1} \):

**Assumes** \( \text{splits-wrt } (p, q) \, T \, F \text{ and monomial-decomp } q \text{ and is-monomial-set } (F::(- \Rightarrow 0 \ 'a::field) \set) \)

--- The last two assumptions are missing in the paper.

**Shows** \( \text{cone-decomp } (T \cap \text{ideal } F) \, p \)

**Proof**

\[
\begin{align*}
\text{from } & \text{assms}(1) \text{ have } s: \text{cone-decomp } T \, (p @ q) \text{ by } \text{(rule splits-wrtD)} \\
\text{hence } & \text{direct-decomp } T \, (\text{map cone } p @ \text{map cone } q) \text{ by } \text{(simp add: cone-decomp-def)} \\
\text{hence } & 1: \text{direct-decomp } (\text{sum-list ' listset } (\text{map cone } p)) \text{ (map cone } p) \\
\text{and } & 2: \text{direct-decomp } T \, ((\text{sum-list ' listset } (\text{map cone } p)) \, \text{sum-list ' listset } (\text{map cone } q)) \\
\text{by } & \text{(auto dest: direct-decomp-appendD intro!: empty-not-in-map-cone)} \\
\text{let } & \text{?ss = } \text{[sum-list ' listset } (\text{map cone } p), \text{sum-list ' listset } (\text{map cone } q)] \\
\text{show } & \text{?thesis} \\
\text{proof } & \text{(intro cone-decompI direct-decompI)} \\
\text{from } & 1 \text{ show inj-on sum-list } (\text{listset } (\text{map cone } p)) \text{ by } \text{(simp only: direct-decomp-def bij-betw-def)} \\
\text{next} & \text{from } \text{assms}(1) \text{ have sum-list ' listset } (\text{map cone } p) \subseteq \text{ideal } F \cap P[X] \\
\text{by } & \text{(rule splits-wrt-image-sum-list-fst-subset)} \\
\text{hence } & \text{sub: sum-list ' listset } (\text{map cone } p) \subseteq \text{ideal } F \text{ by simp} \\
\text{show } & \text{sum-list ' listset } (\text{map cone } p) = T \cap \text{ideal } F \\
\text{proof } & \text{(rule set-eqI)} \\
\text{fix } & x \\
\end{align*}
\]
show \( x \in \text{sum-list } \text{listset} (\text{map cone } ps) \Longleftrightarrow x \in T \cap \text{ideal } F \)

proof
assume \( x \in \text{sum-list } \text{listset} (\text{map cone } ps) \)
show \( x \in T \cap \text{ideal } F \)
proof (intro IntI)
  have \( \text{map } (\lambda x . 0) \) qs \( \in \text{listset} (\text{map cone } qs) \) (is \( \text{ys} \in -\))
  by (induct qs) (auto intro: listset-ConsI zero-in-cone simp del: list-set.simps(2))
  hence \( \text{sum-list } \text{ys} \in \text{sum-list } \text{listset} (\text{map cone } qs) \) by (rule imageI)
  hence \( 0 \in \text{sum-list } \text{listset} (\text{map cone } qs) \) by simp
next
from \( x \in \text{ideal } F \) have \( p + q \in \text{ideal } F \) by (simp add: x xs)
moreover have \( p \in \text{ideal } F \) ..
ultimately have \( q = 0 \) by (simp add: monomial-0-iff)
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 assms(2) have \( \text{monomial } c t \in \text{cone } (h, U) \) by (rule monomial-decomp-sum-list-monomial-in-cone)
  moreover 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 have \( c = 0 \) by (simp add: monomial-0-iff)
finally show \( \text{False} \) ..
qed
with \( p \) have \( x \in \text{sum-list } \text{listset} (\text{map cone } ps) \) by (simp add: x xs)
qed
qed

Together, Theorems splits-wrt-image-sum-list-fst-subset and splits-wrt-cone-decomp-1

130
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 $(T \cap \text{normal-form } F \setminus P[X]) qs$.

**Proof** –

- From `assms(2)` **have** $(\ast)$ cone-decomp $T (ps @ qs)$ **by** (rule `splits-wrtD`).

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

- **Hence** 1: direct-decomp $(\text{sum-list } \cdot \text{ listset } (\text{map cone qs})) (\text{map cone qs})$.

- **And** 2: direct-decomp $T (\text{sum-list } \cdot \text{ listset } (\text{map cone ps}, \text{sum-list } \cdot \text{ listset } (\text{map cone qs})).$

- **By** (auto dest: direct-decomp-appendD intro!: empty-not-in-map-cone)

- **Let** $\ ?ss = \{\text{sum-list } \cdot \text{ listset } (\text{map cone ps}), \text{sum-list } \cdot \text{ listset } (\text{map cone qs})\}$

- **Let** $?G = \text{punit.reduced-GB } F$

- **From** `assms(1, 5)` **have** $?G \subseteq P[X]$ and $G\text{-GB}: \text{punit.is-Groebner-basis } ?G$

- **And** ideal-$G$: ideal $?G = \text{ideal } F$

- **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 $(\text{listset } (\text{map cone qs}))$ **by** (simp only: direct-decomp-def bij-beta-def)

**Next**

- **From** `assms(2)` **have** sum-list $\cdot \text{ listset } (\text{map cone ps}) \subseteq \text{ideal } F \cap P[X]$

- **By** (rule `splits-wrt-image-sum-list-fst-subset`)

- **Hence** sub: sum-list $\cdot \text{ listset } (\text{map cone ps}) \subseteq \text{ideal } F$ **by** simp

- **Show** sum-list $\cdot \text{ listset } (\text{map cone qs}) = T \cap \text{normal-form } F \setminus P[X]$

- **Proof** (rule `set-eqI`)

- **Fix** $x$

- **Show** $x \in \text{sum-list } \cdot \text{ listset } (\text{map cone qs}) \iff x \in T \cap \text{normal-form } F \setminus P[X]$

- **Proof**

  - **Assume** $x$-in: $x \in \text{sum-list } \cdot \text{ listset } (\text{map cone qs})$

  - **Show** $x \in T \cap \text{normal-form } F \setminus P[X]$

  - **Proof** (intro `IntI`)

    - **Have** map $(\lambda\cdot \cdot \cdot) \text{ps} \in \text{listset } (\text{map cone ps})$ (is $?ys \in -$)

    - **By** (induct ps) (auto intro: listset-ConsI zero-in-cone simp del: listset.simps(2))

    - **Hence** sum-list $?ys \in \text{sum-list } \cdot \text{ listset } (\text{map cone ps})$ **by** (rule `imageI`)

    - **Hence** $?0 \in \text{sum-list } \cdot \text{ listset } (\text{map cone ps})$ **by** simp

    - **From** this $x$-in **have** $(0, x) \in \text{listset } ?ss$ **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 $\cdot \text{ listset } (\text{map cone qs}) \subseteq P[X]$

  - **By** (rule `splits-wrt-image-sum-list-snd-subset`)

  - **With** $x$-in **have** $x \in P[X]$ ..

  - **Moreover** **have** $\neg \text{punit.is-red } ?G \ x$

  - **Proof**

    - **Assume** punit.is-red $?G \ x$
then obtain \( g \) and \( t \) such that \( g \in \mathcal{G} \) and \( t \in \text{keys } x \) and \( g \neq 0 \) and adds:

\[
\text{lpp } g \text{ adds } t
\]

by (rule \text{punit.is-red-addsE}[simplified])

from \text{assms}(3) obtain \( c \) \( h \) \( U \) where \((h, U) \in \text{set } qs \) and

\( c \neq 0 \)

\( c \) and \( monomial c t \in \text{cone } (h, U) \) by (rule \text{monomial-decomp-sum-list-monomial-in-cone})

note \text{this}(3)

moreover have \( monomial c t \in \text{ideal } \mathcal{G} \)

proof (rule \text{punit.is-red-monomial-monomial-set-in-pmdl}[simplified])

from \( c \neq 0 \) show \( \text{is-monomial } (\text{monomial } c t) \) by (rule \text{monomial-is-monomial})

next

from \text{assms}(1, 5, 4) show \( \text{is-monomial-set } \mathcal{G} \) by (rule \text{reduced-GB-is-monomial-set-Polys})

next

from \( c \neq 0 \) have \( t \in \text{keys } (\text{monomial } c t) \) by \text{simp}

with \( g \in \mathcal{G} \) \( g \neq 0 \) show \( \text{punit.is-red } \mathcal{G} (\text{monomial } c t) \) using

adds

by (rule \text{punit.is-red-addsI}[simplified])

qed

ultimately have \( \text{monomial } c t \in \text{cone } (h, U) \cap \text{ideal } F \) by (simp add: \text{ideal-G})

also from \text{assms}(2) \((h, U) \in \text{set } qs\) have \( \ldots = \{0\} \) by (rule \text{splits-wrtD})

finally have \( c = 0 \) by (simp add: \text{monomial-0-iff})

with \( c \neq 0 \) show False ..

qed

ultimately show \( x \in \text{normal-form } F \cdot P[X] \)

using \text{assms}(1, 5) by (simp add: \text{image-normal-form-iff})

qed

next

assume \( x \in T \cap \text{normal-form } F \cdot P[X] \)

hence \( x \in T \) and \( x \in \text{normal-form } F \cdot P[X] \) by \text{simp-all}

from \text{this}(2) \text{assms}(1, 5) have \( x \in P[X] \) and \( \text{irred: } \neg \text{punit.is-red } \mathcal{G} x \)

by (simp-all add: \text{image-normal-form-iff})

from \( 2 \) \((x \in T)\) obtain \( xs \) where \( xs \in \text{listset } ?ss \) and \( x \in \text{sum-list } xs \)

by (rule \text{direct-decompE})

from \text{this}(1) obtain \( p \) \( q \) where \( p \in \text{sum-list } \text{listset } (\text{map cone } ps) \)

and \( q \in \text{sum-list } \text{listset } (\text{map cone } qs) \) and \( xs : xs = [p, q] \)

by (rule \text{listset-doubletonE})

have \( x = p + q \) by (simp add: \text{xss})

from \( \text{p sub have } p \in \text{ideal } F \) ..

have \( p = 0 \)

proof (rule \text{ccntr})

assume \( p \neq 0 \)

hence \( \text{keys } p \neq \{\} \) by \text{simp}

then obtain \( t \) where \( t \in \text{keys } p \) by \text{blast}

from \text{assms}(4) \((p \in \text{ideal } F) \) \((t \in \text{keys } p)\) have \( \exists: \text{monomial } c t \in \text{ideal } F \)

for \( c \)

by (rule \text{punit.monomial-pmdl-field}[simplified])

have \( t \notin \text{keys } q \)

proof
assume \( t \in \text{keys} q \)
with \( \text{assms(3)} q \) obtain \( c \ b \ U \) where \( (b, U) \in \text{set} qs \) and \( c \neq 0 \)
and \( \text{monomial} c \ t \in \text{cone} (h, U) \) by (rule \text{monomial-decomp-sum-list-monomial-in-cone})
from \( \text{this(3)} 3 \) have \( \text{monomial} c \ t \in \text{cone} (h, U) \cap \text{ideal} F \) by simp
also from \( \text{assms(2)} \) \((h, U) \in \text{set} qs \) have \( \ldots = \{0\} \) by (rule \text{splits-wrtD})
finally have \( c = 0 \) by (simp add: \text{monomial-0-iff})
with \( \langle c \neq 0 \rangle \) show \( \text{False} \).
qed
with \( t \in \text{keys} p \) have \( t \in \text{keys} x \) unfolding \( x = p + q \) by (rule \text{in-keys-plusI})

\begin{verbatim}
have punit.is-red \(?G\) x
proof
  note G-is-GB
  moreover from \( \exists \) have \( \text{monomial} t \in \text{ideal} \ ?G \) by (simp only: \text{ideal-G})
  moreover have \( \text{monomial} (1::'a) \ t \neq 0 \) by (simp add: \text{monomial-0-iff})
  ultimately obtain \( q \) where \( g \in \ ?G \) and \( g \neq 0 \)
  and \( \text{lpp} \ g \ \text{adds} \ \text{lpp} \ (\text{monomial} (1::'a) \ t) \) by (rule \text{punit.GB-adds-lt[simplified]})
  from \( \text{this(3)} \) have \( \text{lpp} \ g \ \text{adds} \ t \) by (simp add: \text{punit.ilt-monomial})
  with \( \langle g \in \ ?G \rangle \) \langle \ ?G \ neq 0 \rangle \) \langle \ t \in \text{keys} \ x \rangle \ show \ ?thesis \ by (rule \text{punit.is-red-addsI[simplified]})
qed
with \( \text{irred} \) show \( \text{False} \).
qed
with \( q \) show \( x \in \text{sum-list} \ (\text{listset} \ (\text{map} \ \text{cone} \ qs)) \) by (simp only: \text{add}) \langle x \ xs \rangle
qed
qed
qed

\begin{verbatim}

lemma \text{quot-monomial-ideal-monomial}:
  \( \text{ideal} (\text{monomial} 1 \cdot S) \div \text{ideal} (\text{monomial} (1::'a) \cdot (\lambda \cdot s \cdot \text{Poly-Mapping.single} x) \cdot (\lambda \cdot S)) \)
proof (rule \text{set-eqI})
  let \( ?x = \text{Poly-Mapping.single} x \) (\text{1::nat})
  fix \( a \)
  have \( a \in \text{ideal} (\text{monomial} 1 \cdot S) \div \text{ideal} (\text{monomial} (1::'a) \cdot S) \)
  by (simp only: \text{quot-set-iff times-monomial-left})
  also have \( \langle \text{case} \rangle \ a \in \text{ideal} (\text{monomial} 1 \cdot (\lambda \cdot s \cdot ?x) \cdot S) \)
  proof
    (case \( a \))
    show \( ?a \) by (simp add: \text{ideal.span-zero})
  next
  case \( 2 \ a \ c \ t \)
  let \( ?S = \text{monomial} (1::'a) \cdot (\lambda \cdot s \cdot ?x) \cdot S \)
  show \( ?a \) by proof
    assume \( 0 \) \text{punit.monom-mult} \ ?x \ (\text{monomial} c \ t + a) \in \text{ideal} (\text{monomial} 1 \cdot S) \)
    have \( \text{is-monomial-set} \ (\text{monomial} (1::'a) \cdot S) \)
end
end
\end{verbatim}

133
by (auto intro!: is-monomial-setI monomial-is-monomial)

moreover from \(0\) have \(1\): monomial \(c\) \((?x + t) + \text{punit.monom-mult} 1\) ?x
\[\begin{align*}
a &\in \text{ideal} (\text{monomial} 1 \cdot S), \\
&\text{by (simp add: punit.monom-mult-monomial punit.monom-mult-dist-right)}
\end{align*}\]
moreover have \(?x + t \in \text{keys} (\text{monomial} c) (\?x + t) + \text{punit.monom-mult} 1\) ?x a
\[\begin{align*}
\text{proof (intro in-keys-plusI1 notI)}
\end{align*}\]

\[\begin{align*}
&\text{from } 2(1) \text{ show } \?x + t \in \text{keys} (\text{monomial} c) (\?x + t) \text{ by simp} \\
&\text{next}
\end{align*}\]

\[\begin{align*}
&\text{assume } \?x + t \in \text{keys} (\text{punit.monom-mult} 1 \?x a)
\end{align*}\]
also have \(\ldots \subseteq (+) \?x \cdot \text{keys a} \text{ by (rule punit.keys-monom-mult-subset[simplified])}
\[\begin{align*}
&\text{finally obtain } s \text{ where } s \in \text{keys a} \text{ and } \?x + t = ?x + s \ldots
\end{align*}\]

\[\begin{align*}
&\text{from } \text{this}(2) \text{ have } t = s \text{ by simp}
\end{align*}\]

\[\begin{align*}
&\text{with } \langle s \in \text{keys a}; 2(2) \text{ show False by simp} \rangle
\end{align*}\]

\[\begin{align*}
&\text{qed}
\end{align*}\]

ultimately obtain \(f\) where \(f \in \text{monomial} (1 :: a) \cdot S\) and \(\text{adds}: \text{lpp f adds} \?x + t \ldots
\[\begin{align*}
&\text{by (rule punit.keys-monomial-pmdl[simplified])}
\end{align*}\]

\[\begin{align*}
&\text{from } \text{this}(1) \text{ obtain } s \text{ where } s \in S \text{ and } f = \text{monomial} 1 s \ldots
\end{align*}\]

\[\begin{align*}
&\text{from } \text{adds have } s \text{ adds } \?x + t \text{ by (simp add: f punit.lt-monomial)}
\end{align*}\]

\[\begin{align*}
&\text{hence } s - \?x \text{ adds } t
\end{align*}\]

\[\begin{align*}
&\text{by (metis (no-types, lifting) add-minus-2 adds-minus adds-triv-right plus-minus-assoc-pm-nat-1)}
\end{align*}\]

then obtain \(s'\) where \(t = (s - \?x) + s' \) by (rule addsE)
\[\begin{align*}
&\text{from } s \in S \text{ have } \text{monomial} 1 (s - \?x) \in ?S \text{ by (intro imageI)}
\end{align*}\]

\[\begin{align*}
&\text{also have } s' \subseteq \text{ideal } ?S \text{ by (rule ideal.span-superset)}
\end{align*}\]

\[\begin{align*}
&\text{finally have } \text{monomial c } s' \cdot \text{monomial } 1 \text{ (s - ?x) } \in \text{ideal } ?S
\end{align*}\]

\[\begin{align*}
&\text{by (rule ideal.span-scale)}
\end{align*}\]

\[\begin{align*}
&\text{hence } \text{monomial c } t \in \text{ideal } ?S \text{ by (simp add: times-monomial-monomial t add.commute)}
\end{align*}\]

\[\begin{align*}
&\text{moreover have } a \in \text{ideal } ?S
\end{align*}\]

\[\begin{align*}
&\text{proof}
\end{align*}\]

\[\begin{align*}
&\text{from } \text{f \in \text{monomial} 1 \cdot S} \text{ have } f \in \text{ideal } \text{(monomial} 1 \cdot S) \text{ by (rule ideal.span-base)}
\end{align*}\]

\[\begin{align*}
&\text{hence } \text{punit.monom-mult c } (\?x + t - s) \text{ f } \in \text{ideal } \text{(monomial} 1 \cdot S)
\end{align*}\]

\[\begin{align*}
&\text{by (rule punit.pmdl-closed-monom-mult[simplified])}
\end{align*}\]

\[\begin{align*}
&\text{with } \langle s \text{ adds } \?x + t \rangle \text{ have } \text{monomial c } (\?x + t) \text{ in ideal } \text{monomial} 1 \cdot S
\end{align*}\]

\[\begin{align*}
&\text{by (simp add: f punit.monom-mult-monomial adds-minus)}
\end{align*}\]

\[\begin{align*}
&\text{with } \text{f have } \text{monomial c } (\?x + t) + \text{punit.monom-mult} 1 \cdot ?x a - \text{monomial}
\end{align*}\]

\[\begin{align*}
&\text{c } (\?x + t) \text{ in ideal } \text{monomial} 1 \cdot S
\end{align*}\]

\[\begin{align*}
&\text{by (rule ideal.span-diff)}
\end{align*}\]

\[\begin{align*}
&\text{thus } \text{?thesis by (simp add: 2(3) del: One-nat-def)}
\end{align*}\]

\[\begin{align*}
&\text{qed}
\end{align*}\]

ultimately show \(\text{monomial c t + a } \in \text{ideal } ?S
\[\begin{align*}
&\text{by (rule ideal.span-add)}
\end{align*}\]

next

\[\begin{align*}
&\text{have is-monomial-set } ?S \text{ by (auto intro!: is-monomial-setI monomial-is-monomial)}
\end{align*}\]

moreover assume \(1\): monomial \(c t + a \in \text{ideal } ?S
\[\begin{align*}
&\text{moreover from } -2(2) \text{ have } t \in \text{keys } \text{(monomial} \ c t + a)
\end{align*}\]

\[\begin{align*}
&\text{proof (rule in-keys-plusI1)}
\end{align*}\]
from 2(1) show \( t \in \text{keys} \ (\text{monomial} \ c \ t) \) by simp

qed

ultimately obtain \( f \) where \( f \in ?S \) and \( \text{adds}: \text{lpp} \ f \) adds \( t \)

by (rule \text{punit}.\text{keys}-\text{monomial}-\text{pmdl}[\text{simplified}])

from this(1) obtain \( s \) where \( s \in S \) and \( f = \text{monomial} \ 1 \ (s - \ ?x) \) by blast

from \( \text{adds} \) have \( s - \ ?x \) adds \( t \) by (simp add: \text{f punit}.\text{lt}-\text{monomial})

hence \( s \) adds \( ?x + t \)

by (auto simp: \text{adds}-\text{poly}-\text{mapping} \ \text{le}-\text{def} \ \text{lookup}-\text{add} \ \text{lookup}-\text{minus}\)

\text{lookup}-\text{single when-def}

\text{split: if-splits})

then obtain \( s' \) where \( t: ?x + t = s + s' \) by (rule \text{addsE})

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

also have \( \ldots \subseteq \text{ideal} \ (\text{monomial} \ 1 \ ?S) \) by (rule \text{ideal}.\text{span}-\text{superset})

finally have \( \text{monomial} \ c \ s' * \text{monomial} \ 1 \ s \in \text{ideal} \ (\text{monomial} \ 1 \ ?S) \)

by (rule \text{ideal}.\text{span-scale})

hence \( \text{monomial} \ c \ (\ ?x + t) \in \text{ideal} \ (\text{monomial} \ 1 \ ?S) \)

by (simp only: \( t \)) (simp add: \text{times}-\text{monomial}-\text{monomial} \text{add}.\text{commute})

moreover have \( \text{punit}.\text{monom-mult} \ 1 \ ?x \ a \in \text{ideal} \ (\text{monomial} \ 1 \ ?S) \)

proof –

from \( \langle f \in \ ?S \rangle \) have \( f \in \text{ideal} \ ?S \) by (rule \text{ideal}.\text{span}-\text{base})

hence \( \text{punit}.\text{monom-mult} \ c \ (t - (s - \ ?x)) \ f \in \text{ideal} \ ?S \)

by (rule \text{punit}.\text{pmdl-closed-\text{monom-mult}}[\text{simplified}])

with \( (s - \ ?x) \) adds \( t \) have \( \text{monomial} \ c \ t \in \text{ideal} \ ?S \)

by (simp add: \text{f punit}.\text{monom-mult-\text{poly}-\text{mapping} \ \text{add}.\text{commute}})

with \( 1 \) have \( \text{monomial} \ c \ t + a - \text{monomial} \ c \ t \in \text{ideal} \ ?S \)

by (rule \text{ideal}.\text{span-diff})

thus \( ?\text{thesis} \) by (simp add: \text{\text{2}(3)} \ del: \text{One-nat-def})

qed

ultimately have \( \text{monomial} \ c \ (\ ?x + t) + \text{punit}.\text{monom-mult} \ 1 \ ?x \ a \in \text{ideal} \ (\text{monomial} \ 1 \ ?S) \)

by (rule \text{ideal}.\text{span-add})

thus \( \text{punit}.\text{monom-mult} \ 1 \ ?x \ (\text{monomial} \ c \ t + a) \in \text{ideal} \ (\text{monomial} \ 1 \ ?S) \)

by (simp add: \text{punit}.\text{monom-mult-\text{poly}-\text{mapping} \ \text{add}.\text{commute}})

qed

finally show \( a \in \text{ideal} \ (\text{monomial} \ 1 \ ?S) \div \text{monomial} \ 1 \ ?x \leftarrow a \in \text{ideal} \ (\text{monomial} \ 1 \ ?S) \) .

qed

lemma \text{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 \( \text{cone} \ (\text{monomial} \ 1 \ t, U) \subseteq \text{ideal} \ F \leftarrow 0 \in S \)

proof

have \( \text{monomial} \ 1 \ t \in \text{cone} \ (\text{monomial} \ (1::'a) \ t, U) \) by (rule \text{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 \( \text{is-monomial-set} \ (\text{monomial} \ (1::'a) \ ?S) \)

by (auto intro!: \text{is-monomial-set} \ \text{monomial}-\text{is-monomial})
moreover from * have 1 ∈ ideal (monomial (1::'a) * S) by (simp only: quot-set-iff flip: assms)
moreover have 0 ∈ keys (1::'a ⇒ 0 'a) by simp
ultimately obtain g where g ∈ monomial (1::'a) * S and adds: lpp g adds 0
  by (rule punit.keys-monomial-pmdl[simplified])
from this(1) obtain s where s ∈ monomial 1 s ..
from adds have s adds 0 by (simp add: g punit.lt-monomial flip: single-one)
with (s ∈ S) show 0 ∈ S by (simp only: adds-zero)
next
assume 0 ∈ S
hence monomial 1 0 ∈ monomial (1::'a) * S by (rule imageI)
hence 1 ∈ ideal (monomial (1::'a) * S) unfolding single-one by (rule ideal.span-base)
hence eq: ideal F ∩ monomial 1 t = UNIV (is - ÷ t = -)
  by (simp only: assms ideal-eq-UNIV-iff-contains-one)
show cone (monomial 1 t, U) ⊆ ideal F
proof
  fix a
  assume a ∈ cone (?t, U)
  then obtain q where a: a = q * ?t by (rule coneE)
  have q ∈ ideal F ÷ ?t by (simp add: eq)
  thus a ∈ ideal F by (simp only: a quot-set-iff mult.commute)
qed
qed

lemma lem-4-2-2:
assumes ideal F ÷ monomial 1 t = ideal (monomial (1::'a::comm-ring-1) * S)
shows cone (monomial 1 t, U) ∩ ideal F = {0} ↔ S ∩ .[U] = {}
proof
let ?t = monomial (1::'a) t
assume eq: cone (?t, U) ∩ ideal F = {0}
{}
  fix s
  assume s ∈ S
  hence monomial 1 s ∈ monomial (1::'a) * S (is ?s ∈ -) by (rule imageI)
  hence ?s ∈ ideal (monomial 1 * S) by (rule ideal.span-base)
  also have .. = ideal F ÷ ?t by (simp only: assms)
  finally have *: ?s * ?t ∈ ideal F by (simp only: quot-set-iff mult.commute)
  assume s ∈ .[U]
  hence ?s ∈ P[U] by (rule Polys-closed-monomial)
  with refl have ?s * ?t ∈ cone (?t, U) by (rule coneI)
  with * have ?s * ?t ∈ cone (?t, U) ∩ ideal F by simp
  hence False by (simp add: eq times-monomial-monomial monomial-0-iff)
}
thus S ∩ .[U] = {} by blast
next
let ?t = monomial (1::'a) t
assume eq: S ∩ .[U] = {}
{}
  fix a
assume \( a \in \text{cone}(\ldots, U) \)
then obtain \( q \) where \( q \in P[U] \) and \( a = q \cdot \ldots \) by (rule coneE)

assume \( a \in \text{ideal } F \)
have \( a = 0 \)
proof (rule ccontr)
assume \( a \neq 0 \) by (auto simp: a)
from \( (a \in \text{ideal } F) \) have \( \ast: q \in \text{ideal } F \) \( \ldots \) by (simp only: quot-set-iff a mult.commute)

have is-monomial-set (monomial \((1::') S\))
  by (auto intro!:
    is-monomial-setI monomial-is-monomial)
moreover from \( \ast \) have \( q\in \text{ideal } \) monomial \((1 :: ') S\)
ultimately have \( g \) where \( g \in \text{monomial } (1 :: ') S \)

ultimately obtain \( s \) where \( s \in S \) and \( g = \text{monomial } 1 \ldots \) by (rule punit. lt-monomial)

ultimately have \( s \in \) (rule PPs-closed-adds)
with eq (\( s \in S \)) show False by blast
qed

thus \( \text{cone}(\ldots, U) \cap \text{ideal } F = \{0\} \) using zero-in-cone ideal.span-zero by blast

10.6 Function split

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

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 \( \text{card} B \leq \text{card} (\text{max-subset } A \ P) \) (is ?thesis3)
proof –
from assms(2, 3) have \( B \subseteq A \land P \ B \) by simp
moreover have \( \forall C. \ C \subseteq A \land P \ C \longrightarrow \text{card} C < \text{Suc} \ (\text{card} A) \)
proof (intro allII impI, elim conjE)
  fix \( C \)
  assume \( C \subseteq A \)
  with assms(1) have \( \text{card } C \leq \text{card } A \) by (rule card-mono)
  thus \( \text{card } C < \text{Suc} \ (\text{card } A) \) by simp
qed
ultimately have ?thesis1 \land ?thesis2 and ?thesis3 unfolding max-subset-def

137
by (rule arg-max-natI, rule arg-max-nat-le)
thus \( ?\text{thesis1} \) and \( ?\text{thesis2} \) and \( ?\text{thesis3} \) by simp-all
qed

function (domintros) split :: ('x ⇒ nat) ⇒ 'x set ⇒ ('x ⇒ nat) set ⇒
\((((((\ '& 0 \text{ nat}) ⇒ (' '& 0 \text{ nat}) \set ⇒
\((((((\ '& 0 \text{ nat}) ⇒_0 'a) \times ('x \set)) \list) \times
\((((((\ '& 0 \text{ nat}) ⇒_0 'a::{zero, one}) \times ('x \set)) \list))\)

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

by auto

Function \text{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 \( ?m = \text{args}' \), card (fst (snd args')) + sum deg-pm (snd (snd args'))
from wf-measure[of \( ?m \)] assms(2, 3) show \( ?\text{thesis} \)
proof (induct args)
  case (less args)
  obtain t U F where args: args = (t, U, F) using prod.exhaust by metis
  from less.prems have U ⊆ X and finite F by (simp-all only: args fst-conv
  snd-cone)
  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 = max-subset U (\lambda V. F ∩ [V] = {})
define x where x = (SOME x'. x' ∈ U ∧ x' \∉ S)
show ?case unfolding args
proof (rule split.domintros, simp-all only: x-def[symmetric] S-def[symmetric])
fix f
assume 0 \∉ F and f ∈ F and f ∈ [U]
from this(1) have F ∩ [{}] = {} by simp
with (finite U) empty-subsetI have S ⊆ U and F ∩ [S] = {}
unfolding $S$-def by (rule max-subset)+
have $x \in U \land x \notin S$ unfolding $x$-def
proof (rule someI-ex)
  from $\{ f \in F. \{ f \in \cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\
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[sPoly-Mapping.single x (Suc 0)]
  have deg-pm (s - Poly-Mapping.single x (Suc 0)) = deg-pm s by simp
hence keys s ∩ keys (Poly-Mapping.single x (Suc 0)) = {} by (simp only: deg-pm-minus-id-iff)
hence x /∈ keys s by simp
moreover assume s ∈ .[insert x S]
ultimately have s ∈ .[S] by (fastforce simp: PPs-def)
with (s ∈ F) (F ∩ .[S] = {}): have False by blast
} thus ?thesis by blast
qed
ultimately have card (insert x S) ≤ card S unfolding S-def by (rule max-subset)
moreover from (S ⊆ U) (finite U) have finite S by (rule finite-subset)
ultimately show False using x /∈ S; by simp
qed
qed
finally show card U + sum deg-pm ((λf. f - monomial (Suc 0) x) ' F)
< card U + sum deg-pm F
  by simp
  ultimately show False ..
} qed
qed

corollary split-domI: finite X ⇒ U ⊆ X ⇒ finite S ⇒ 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 ⊆ X
  shows split t U \{\} = \{[], [(monomial (1::'a::{zero,one}) t, U)]\}
proof -
  have finite \{\} ..
  with assms have split-dom TYPE('a) (t, U, \{\}) by (rule split-domI)
  thus ?thesis by (simp add: split.psimps)
  qed

lemma split-induct [consumes 3, case-names base1 base2 step]:
  fixes P :: ('x ⇒0 nat) ⇒ -
  assumes finite X and U ⊆ X and finite S
  assumes \(\forall t U S. U ⊆ X ⇒ finite S ⇒ 0 ∈ S ⇒ P t U S ((monomial
\[(1':a:\{\text{zero,one}\}) t, U)]\]
assumes \(\text{[t U S. } U \subseteq X \implies \text{finite } S \implies 0 \notin S \implies S \cap [U] = \{\} \implies P t U S ([], [(\text{monomial } 1 t, U)])\)
assumes \(\text{[t U S V x ps0 ps1 qs0 qs1. } U \subseteq X \implies \text{finite } S \implies 0 \notin S \implies S \cap \cdot [U] \neq \{\} \implies V \subseteq U \implies S \cap \cdot [V] = \{\} \implies (\forall V'. V' \subseteq U \implies S \cap \cdot [V'] = \{\} \implies \text{card } V' \leq \text{card } V) \implies \)
\(x \in U \implies x \notin V \implies V = \text{max-subset } U (\forall V'. S \cap \cdot [V'] = \{\}) \implies \)
\(x = (\text{SOME } x'. x' \in U \setminus V) \implies \)
\((ps0, qs0) = \text{split } t (U \setminus \{x\}) S \implies \)
\((ps1, qs1) = \text{split } (\text{Poly-Mapping.single } x 1 + t) U ((\forall f - \text{Poly-Mapping.single } x 1') \cdot S) \implies \)
\(\text{split } t U S = (ps0 @ ps1, qs0 @ qs1) \implies \)
\(P t (U \setminus \{x\}) S (ps0, qs0) \implies \)
\(P (\text{Poly-Mapping.single } x 1 + t) U ((\forall f - \text{Poly-Mapping.single } x 1) \cdot S) (ps1, qs1) \implies \)
\(P t U S (ps0 @ ps1, qs0 @ qs1) \implies \)
\(\text{shows } P t U S (\text{split } t U S)\)
proof –
from \(\text{assms(1–3)}\) have \(\text{split-dom TYPE('a) (t, U, S) by (rule split-domI)}\)
thus \(\text{thesis using assms(2,3)}\)
proof (induct t U S rule: split.pinduct)
case step: \((1 t F)\)
from 
\(\text{step(4)}\) assms(1) have \(\text{finite } U\) by (rule finite-subset)
define \(S\) where \(S = \text{max-subset } U (\forall V. F \cap \cdot [V] = \{\})\)
define \(x\) where \(x = (\text{SOME } x'. x' \in U \land x' \notin S)\)
show \(\forall \text{case}\)
proof (simp add: split.psimp[OF step(1)] S-def[symmetric] x-def[symmetric] split: prod.split, intro allI conjI impI)
assume \(0 \notin F\)
with \(\text{step(4, 5)}\) show \(P t U F [[(\text{monomial } 1 t, U)], []] \implies (\text{monomial } 1 t, U)], [] \).
next
assume \(0 \notin F\) and \(F \cap \cdot [U] = \{\}\)
with \(\text{step(4, 5)}\) show \(P t U F ([], [(\text{monomial } 1 t, U)]) \implies \text{assms(5)}\)
next
fix \(ps0 qs0 ps1 : (\cdot \Rightarrow 'a) \times - \text{ list}\)
assume \(\text{split } (\text{Poly-Mapping.single } x (\text{Suc } 0) + t) U ((\forall f - \text{Poly-Mapping.single } x (\text{Suc } 0)) \cdot F) = (ps1, qs1)\)
hence \(\text{PQ1[symmetric]: split } (\text{Poly-Mapping.single } x 1 + t) U ((\forall f - \text{Poly-Mapping.single } x 1) \cdot F) = (ps1, qs1)\)
by simp
assume \(\text{PQ0[symmetric]: split } t (U \setminus \{x\}) F = (ps0, qs0)\)
assume \(F \cap \cdot [U] \neq \{\}\) and \(0 \notin F\)
from \(\text{this(2)}\) have \(F \cap \cdot [U] = \{\}\) by simp
with \(\text{finite } U\) empty-subsetI have \(S \subseteq U\) and \(F \cap \cdot [S] = \{\}\)
unfolding S-def by (rule max-subset+)
have \(\text{S-max: card } S' \leq \text{card } S\) if \(S' \subseteq U\) and \(F \cap \cdot [S'] = \{\}\) for \(S'\)
using \(\text{finite } U\) that unfolding S-def by (rule max-subset)
have \( x \in U \land x \notin S \) unfolding x-def

proof (rule someI-ex)
from \( \{F \cap [U] \neq \{\}\} \) have \( S \neq U \) by blast
with \( S \subseteq U \) show \( \exists y. y \in U \land y \notin S \) by blast
qed

hence \( x \in U \) and \( x \notin S \) by simp-all

from step(4, 5) \( \langle 0 \notin F \rangle \) have \( (S \subseteq U \land F \cap [U] = \{\}) \) S-max :\( x \in U \) \( \langle x \notin S \rangle \) S-def - PQ0 PQ1

show \( \exists x. \) split \( t U F \) (ps0 @ ps1, qs0 @ qs1)
proof (rule assms(6))
show \( \exists x. \) unfolding PQ0 using \( \langle 0 \notin F \rangle \) \( \langle F \cap [U] = \{\} \rangle \) - - step(5)
proof (rule step(3))
from \( \langle U \subseteq X \rangle \) show \( \exists x. \) fastforce
qed (simp add: x-def S-def)

next
show \( \exists x. \) (Poly-Mapping.single x 1 + t) U \( \langle (\lambda f. f = Poly-Mapping.single x 1) \rangle \) F) (ps1, qs1)
unfolding PQ1 using \( \langle 0 \notin F \rangle \) \( \langle F \cap [U] = \{\} \rangle \) refl PQ0 \( \langle U \subseteq X \rangle \)
proof (rule finite-imageI)
qed (simp add: x-def S-def)

next
show \( \exists x. \) split \( t U F \) (ps0 @ ps1, qs0 @ qs1) using \( \langle 0 \notin F \rangle \) \( \langle F \cap [U] = \{\} \rangle \) - refi PQ0 \( \langle U \subseteq X \rangle \)
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

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

proof
from assms have valid-decomp X (fst ?s) ∧ valid-decomp X (snd ?s)
proof (induct t U S rule: split-induct)
  case (base1 t U S)
  from base1(1, 4) show ?case by (simp add: valid-decomp-def monomial-0-iff Polys-closed-monomial)
next
  case (base2 t U S)
  from base2(1, 5) show ?case by (simp add: valid-decomp-def monomial-0-iff Polys-closed-monomial)

142
next
  case (step t U S V x ps0 ps1 qs0 qs1)
  from step.hyps(8, 1) have x ∈ X .
  hence Poly-Mapping.single x 1 ∈ .[X] by (rule PPs-closed-single)
  hence Poly-Mapping.single x 1 + t ∈ .[X] using step.prems by (rule PPs-closed-plus)
  with step.hyps(15, 16) step.prems show ?case by (simp add: valid-decomp-append)
qed
thus valid-decomp X (fst ?s) and valid-decomp X (snd ?s) by simp-all
qed

lemma monomial-decomp-split:
  assumes finite X and U ⊆ X and finite S
  shows monomial-decomp (fst ((split t U S)::((- × (((- ⇒ 0 'a::zero-neq-one) × -) list)))))
  and monomial-decomp (snd ((split t U S)::((- × (((- ⇒ 0 'a::zero-neq-one) × -) list)))))
  (is monomial-decomp (snd ?s))
proof –
  from assms have monomial-decomp (fst ?s) ∧ monomial-decomp (snd ?s)
proof (induct t U S rule: split-induct)
  case (base1 t U S)
  from base1(1) show ?case by (simp add: monomial-decomp-def monomial-is-monomial)
next
  case (base2 t U S)
  from base2(1) show ?case by (simp add: monomial-decomp-def monomial-is-monomial)
next
  case (step t U S V x ps0 ps1 qs0 qs1)
  from step.hyps(15, 16) show ?case by (auto simp: monomial-decomp-def)
qed
thus monomial-decomp (fst ?s) and monomial-decomp (snd ?s) by simp-all
qed

lemma split-splits-wrt:
  assumes finite X and U ⊆ X and finite S and t ∈ .[X]
  and ideal F ÷ monomial 1 t = ideal (monomial 1 ' S)
  shows splits-wrt (split t U S) (cone (monomial (1::'a::{comm-ring-1,ring-no-zero-divisors}))
  t, U)) F
using assms
proof (induct t U S rule: split-induct)
  case (base1 t U S)
  from base1(3) have cone (monomial 1 t, U) ⊆ ideal F by (simp only: lem-4-2-1
  base1(5))
  show ?case
proof (rule splits-wrtI)
  fix h0 U0
  assume (h0, U0) ∈ set [(monomial (1::'a) t, U)]
  hence h0: h0 = monomial 1 t and U0 = U by simp-all
  note this(1)
  also have monomial 1 t ∈ cone (monomial (1::'a) t, U) by (fact tip-in-cone)
also have ... ⊆ ideal F by fact
finally show h₀ ∈ ideal F.

from base1(4) have h₀ ∈ P[X] unfolding h₀ by (rule Polys-closed-monomial)
moreover from base1(1) have U₀ ⊆ X by (simp only: U₀ = U₁)
ultimately show cone (h₀, U₀) ⊆ P[X] by (rule cone-subset-PolysI)
qed (simp-all add: cone-decomp-singleton : U ⊆ X)

next

case (base2 t U S)

from base2(4) have cone (monomial 1 t, U) ∩ ideal F = {0} by (simp only: lem-4-2-2 base2(6))

show ?case
proof (rule splits-wrtI)
fix h₀ U₀
assume (h₀, U₀) ∈ set [(monomial (1::a) t, U)]
hence h₀: h₀ = monomial 1 t and U₀ = U by simp-all

note this(1)
also from base2(5) have monomial 1 t ∈ P[X] by (rule Polys-closed-monomial)
finally have h₀ ∈ P[X].
moreover from base2(1) have U₀ ⊆ X by (simp only: U₀ = U₁)
ultimately show cone (h₀, U₀) ⊆ P[X] by (rule cone-subset-PolysI)

next

fix h₀ U₀ a
assume (h₀, U₀) ∈ set [(monomial (1::a) t, U)] and a ∈ cone (h₀, U₀)
hence a ∈ cone (monomial 1 t, U) by simp
moreover assume a ∈ ideal F
ultimately have a ∈ cone (monomial 1 t, U) ∩ ideal F by (rule IntI)
also have ... = {0} by fact
finally show a = 0 by simp
qed (simp-all add: cone-decomp-singleton : U ⊆ X)

next

case (step t U S V x ps0 ps1 qs0 qs1)

let ?x = Poly-Mapping.single x 1
from step.prems have 0: splits-wrt (ps0, qs0) (cone (monomial 1 t, U = {x}))

F by (rule step.prems)
have 1: splits-wrt (ps1, qs1) (cone (monomial 1 (?x + t), U)) F
proof (rule step.prems)

from step.prems(8, 1) have x ∈ X ..
hence ?x ∈ [X] by (rule PPs-closed-single)
thus ?x + t ∈ [X] using step.prems(1) by (rule PPs-closed-plus)

next

have ideal F ÷ monomial 1 (?x + t) = ideal F ÷ monomial 1 t ÷ monomial 1 ?x

by (simp add: times-monomial-monomial add.commute)
also have ... = ideal (monomial 1 :: S) ÷ monomial 1 ?x by (simp only: step.prems)
finally show ideal F ÷ monomial 1 (?x + t) = ideal (monomial 1 :: (λs. s - ?x) :: S)

by (simp only: quot-monomial-ideal-monomial)
qed

show ?case
proof (rule splits-wrtI)
  from step.hyps(8) have U: insert x U = U by blast
  have direct-decomp (cone (monomial (1::'a) t, insert x (U - {x})))
    [cone (monomial 1 t, U - {x})],
    cone (monomial 1 (monomial (Suc 0) t) * monomial 1 t, insert x (U - {x}))]
    by (rule direct-decomp-cone-insert)
  hence direct-decomp (cone (monomial 1 (monomial (Suc 0) t) * monomial 1 t, insert x (U - {x})))
    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 (Suc 0) t) * monomial 1 t, U - {x}))
    (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) (metis append.assoc perm-append1 perm-append2 perm-append-swap)
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 \((h_0, U_0) \in \set{\text{qs1}}\)
  with 1 show \(?\text{thesis}\) by (rule splits-wrtD)
qed

from \((h_0, U_0) \in \set{\text{qs0}} \cup \set{\text{qs1}}\) have cone \((h_0, U_0) \cap \text{ideal} F = \{0\}\)
proof
  assume \((h_0, U_0) \in \set{\text{qs0}}\)
  with 0 show \(?\text{thesis}\) by (rule splits-wrtD)
next
  assume \((h_0, U_0) \in \set{\text{qs1}}\)
  with 1 show \(?\text{thesis}\) by (rule splits-wrtD)
qed
  thus \(\bigwedge a. \ a \in \text{cone} (h_0, U_0) \implies a \in \text{ideal} F \implies a = 0\) by blast
qed

lemma lem-4-5:
assumes finite \(X\) and \(U \subseteq X\) and \(t \in [X]\) and \(F \subseteq P[X]\)
and \(\text{ideal} F \div \text{monomial} \ 1 \ t = \text{ideal} \ (\text{monomial} (1::'a) \ :: S)\)
and cone \((\text{monomial} (1::'a::field) \ t', V) \subseteq \text{cone} \ (\text{monomial} \ 1 t, U) \cap \text{normal-form} F \cdot P[X]\)
shows \(V \subseteq U\) and \(S \cap [V] = \{\}\)
proof
  let \(\ ?t = \text{monomial} (1::'a) \ t\)
  let \(\ ?t' = \text{monomial} (1::'a) \ t'\)
  from \(\text{assms(6)}\) have 1: cone \((\ ?t', V) \subseteq \text{cone} \ (\ ?t, U)\) and 2: cone \((\ ?t', V) \subseteq \text{normal-form} F \cdot P[X]\)
  by blast+
  from \(\text{this(1)}\) show \(V \subseteq U\) by (rule cone-subsetD) (simp add: monomial-0-iff)

show \(S \cap [V] = \{\}\)
proof
  let \(\ ?t = \text{monomial} (1::'a) \ t\)
  let \(\ ?t' = \text{monomial} (1::'a) \ t'\)
  show \(S \cap [V] \subseteq \{\}\)
proof
    fix \(s\)
    assume \(s \in S \cap [V]\)
    hence \(s \in S\) and \(s \in [V]\) by simp-all
    from \(\text{this(2)}\) 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 \(\text{tip-in-cone} \ 1\) have \(\ ?t' \in \text{cone} \ (\ ?t, U)\)
    then obtain \(s' \in P[U]\) and \(t': \ ?t' = s' * \ ?t\) by (rule coneE)
    note \(\text{this(1)}\)
    also from \(\text{assms(2)}\) 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 \(\text{refl} (\ ?s \in P[V]\) have \(\ldots \in \text{cone} \ (\ ?t', V)\) by (rule coneI)
finally have \( s' \ast s \ast t \in \text{cone} (\langle t', V \rangle) \).

hence 1: \( s' \ast s \ast t \in \text{normal-form} F \vdash P[X] \) using 2 ...

from \( s \in S \) have \( s \in \text{monomial} 1 \vdash S \) by (rule imageI)

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

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

hence \( s' \ast s \in \text{ideal} F \vdash t \) by (simp only: assms(5))

hence \( s' \ast s \in \text{ideal} F \) by (simp only: quot-set-iff mult.commute)

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

also from \( \text{assms}(1, 4) \) have \( \ldots \subseteq \{0\} \)

by (auto simp: \text{normal-form-normal-form} simp flip: \text{normal-form-zero-iff})

finally have \( s \ast t' = 0 \) by (simp add: \text{t'}-ac-simps)

thus \( s \in \{\} \) by (simp add: \text{times-monomial-monomial} \text{monomial-0-iff})

qed

qed (fact empty-subsetI)

lemma lem-4-6:

assumes \( \text{finite} X \) and \( U \subseteq X \) and \( \text{finite} S \) and \( t \in \{X\} \) and \( F \subseteq P[X] \)

and \( \text{ideal} F \vdash \text{monomial} 1 t = \text{ideal} (\text{monomial} 1 \vdash S) \)

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

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

proof –

let \( \langle t \rangle = \text{monomial} (1::'a) t \)

let \( \langle t' \rangle = \text{monomial} (1::'a) t' \)

from \( \text{assms}(7) \) have \( \text{cone} (\langle t' \rangle, V) \subseteq \text{cone} (\langle t \rangle, U) \) and \( \text{cone} (\langle t' \rangle, V) \subseteq \text{normal-form} F \vdash P[X] \)

by blast+

from \( \text{assms}(1, 2, 4, 5, 6, 7) \) have \( V \subseteq U \) and \( S \cap \{V\} = \{\} \) by (rule lem-4-5)+

with \( \text{assms}(1, 2, 3) \) show \( ?\text{thesis} \) using that

proof (induct \( t U S \) arbitrary: \( V \) thesis rule: split-induct)

case \( \text{base}1 t U S \)

from \( \text{base1.hyps}(3) \) have \( 0 \in S \cap \{V\} \) using \text{zero-in-PPs} by (rule IntI)

thus \( ?\text{case} \) by (simp add: \text{base1.prems}(2))

next

case \( \text{base}2 t U S \)

show \( ?\text{case} \)

proof (rule \text{base}2.prems)

from \( \text{base2.hyps}(1) \) \( \text{assms}(1) \) have \( \text{finite} U \) by (rule \text{finite-subset})

thus \( \text{card} V \leq \text{card} U \) using \text{base2.prems}(1) by (rule \text{card-mono})

qed simp

next

case \( \text{step} t U S V0 x ps0 ps1 qs0 qs1 \)

from \( \text{step.prems}(1, 2) \) have \( \text{card} V \leq \text{card} V0 \) by (rule \text{step.prems})

from \( \text{step.prems}(5, 9) \) have \( V0 \subseteq U - \{x\} \) by blast

then obtain \( V' \) where \( 1: (\text{monomial} 1 t, V') \in \text{set} (\text{snd} (ps0, qs0)) \) and 2:

\( \text{card} V0 \leq \text{card} V' \)

using \( \text{step.prems}(6) \) by (rule \text{step.prems})
show \(?case\)
proof (rule step.prems)
  from \(\_\) show \((\text{monomial } 1 \ t, V') \in \text{ set} \ (\text{snd} \ (\text{ps0 @} \text{ps1}, \text{qs0 @} \text{qs1}))\) by simp
  next
  from \(0 \ 2\) show \(\text{card} \ V \leq \text{card} \ V'\) by (rule le-trans)
  qed
  qed
  qed

lemma lem-4-7:
  assumes \(\text{finite } X\) and \(S \subseteq [X]\) and \(g \in \text{punit.reduced-GB} \ (\text{monomial } (1::a) \ ^' S)\)
  and \(\text{cone-decomp} \ (P[X] \cap \text{ideal} \ (\text{monomial } (1::a::\text{field}) \ ^' S)) \text{ ps}\)
  and \(\text{monomial-decomp ps}\)
  obtains \(U \text{ where} \ (g, U) \in \text{ set} \ ps\)
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: \text{Polys-closed-monomial})
  moreover have \(\text{is-monomial-set} \ ?S\)
    by (auto intro: is-monomial-setI \text{monomial-is-monomial})
  ultimately have \(\text{is-monomial-set} \ ?G\) by (rule reduced-GB-is-monic-set-Polys)
  hence \(\text{is-monomial} \ g\) 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 \(\text{punit.is-monic-set} \ ?G\)
    by (rule reduced-GB-is-monic-set-Polys)
  ultimately have \(\text{punit.lc} \ g = 1\) using \(\text{assms}(3)\) by (simp add: \text{punit.is-monic-set-def})
  moreover define \(t\) where \(t = \text{lpp} \ g\)
  moreover from \(\text{is-monomial} \ g\) have \(\text{monomial} \ (\text{punit.lc} \ g) \ (\text{lpp} \ g) = g\)
    by (rule punit.monomial-eq-itself)
  ultimately have \(g \colon \text{monomial } 1 \ t\) by simp
  hence \(t \in \text{keys} \ g\) by simp
  from \(\text{assms}(3)\) have \(g \in \text{ideal} \ ?S\) by (rule ideal.span-base)
  also from \(\text{assms}(1)\) \(?S \subseteq P[X]\) have \(\text{ideal-G} : \ldots = \text{ideal} \ ?S\) 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)
  \assms(1) \(?S \subseteq P[X]\)
  ultimately have \(g \in P[X] \cap \text{ideal} \ ?S\) by simp
  with \(\text{assms}(4)\) have \(g \in \text{sum-list} \ \text{ listset} \ (\text{map} \ \text{cone} \ \text{ps})\)
    by (simp only: cone-decomp-def direct-decompD)
  with \(\text{assms}(5)\) obtain \(d \ h \ U \text{ where} \ : \ (h, U) \in \text{ set} \ ps\) and \(d \neq 0\) and \(\text{monomial} \ d \ t \in \text{cone} \ (h, U)\)
    using \(\{t \in \text{keys} \ g\}\) by (rule monomial-decomp-sum-list-monomial-in-cone)
  from \(\text{this}(3)\) \(\text{zero-in-PPs}\) have \(\text{punit.monom-mult} \ (1 / d) \ 0\) \(\text{ (monomial} \ d \ t) \in \text{cone} \ (h, U)\)
    by (rule cone-closed-monom-mult)
  with \(d \neq 0\) have \(g \in \text{cone} \ (h, U)\) by (simp add: \(g \ \text{punit.monom-mult-monomial}\)
then obtain \( q \) where \( q \in P[U] \) and \( g' : g = q \cdot h \) by (rule coneE)
from \( (g \neq 0) \) have \( q \neq 0 \) and \( h \neq 0 \) by (auto simp: \( g' \))
hence \( ltp-g' : lpp g = lpp q + lpp h \) 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 \( \text{punit}.\text{lc} h = 1 \) by (rule monomial-decompD)+
moreover from this(1) have monomial \( (\text{punit}.\text{lc} h) \) \( (lpp h) = h \)
by (rule punit.monomial-eq-itself)
moreover define \( s \) where \( s = lpp h \)
ultimately have \( h : h = \text{monomial} 1 s \) by simp

have \( \text{punit}.\text{lc} q = \text{punit}.\text{lc} g \) by (simp add: \( g'\)-lc-times \( \langle \text{punit}.\text{lc} h = 1 \rangle \))
hence \( \text{punit}.\text{lc} q = 1 \) by (simp only: \( \text{punit}.\text{lc} g = 1 \))
note tip-in-cone
also from \( \text{assms}(4) \) * have cone \( (h, U) \subseteq P[X] \cap \text{ideal } ?S \) by (rule cone-decomp-cone-subset)
also have \( \ldots \subseteq \text{ideal } ?G \) by (simp add: ideal-G)
finally have \( h \in \text{ideal } ?G \).

from \( \text{assms}(1) \) (?\( S \subseteq P[X] \)) have \( \text{punit}.\text{is-Groebner-basis } ?G \) by (rule reduced-GB-is-GB-Polys)
then obtain \( g' \) where \( g' \in ?G \) and \( g' \neq 0 \) and \( \text{adds2} : lpp g' \text{ adds } lpp h \)
using \( \langle h \in \text{ideal } ?G ; h \neq 0 \rangle \) by (rule punit.GB-adds-lt[simplified])
from this(3) \( \text{adds1} \) have \( lpp g' \text{ adds } t \) by (rule adds-trans)
with \( \langle g' \neq 0 \rangle ; \langle t \in \text{keys } g \rangle \) have \( \text{punit}.\text{is-red } \{g'\} \) \( g \)
by (rule punit.is-red-addsI[simplified]) simp

have \( g' = q \)
proof (rule ccontr)
  assume \( g' \neq g \)
  with \( \langle g' \in ?G \rangle \) have \( \{g'\} \subseteq ?G - \{g\} \) by simp
  with \( \langle \text{punit}.\text{is-red } \{g'\} \rangle \) \( g \) have red: \text{punit}.\text{is-red } \( ?G - \{g\} \) \( g \) by (rule punit.is-red-subset)
from \( \text{assms}(1) \) (?\( S \subseteq P[X] \)) have \( \text{punit}.\text{is-auto-reduced } ?G \) by (rule reduced-GB-is-auto-reduced-Polys)

  hence \( \neg \text{punit}.\text{is-red } \( ?G - \{g\} \) \( g \) using \( \text{assms}(3) \) by (rule punit.is-auto-reducedD)
  thus False using red ..

qed

with \( \text{adds2} \) have \( t \) \( \text{adds } lpp h \) by (simp only: \( t \)-def)
with \( \text{adds1} \) have \( lpp h = t \) by (rule adds-antisym)

hence \( lpp q = 0 \) using \( \langle ltp-g' \rangle \) by (simp add: \( t \)-def)

hence monomial \( (\text{punit}.\text{lc} q) \) \( 0 = q \) by (rule punit.lt-eq-min-term-monomial[simplified])

hence \( g = h \) by (simp add: \( \langle \text{punit}.\text{lc} q = 1 \rangle \) \( g' \))

with \( \ast \) have \( \langle g, U \rangle \in \text{set } ps \) by simp
thus \( \text{thesis} \) ..

qed

lemma snd-splitI:
  assumes \( \text{finite } X \) and \( U \subseteq X \) and \( \text{finite } S \) 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 \( \text{assms} \)
proof (induct \( t U S \) arbitrary; thesis rule: split-induct)
case \( \text{base1 } t U S \)
from base1.prems(2) base1.hyps(3) show \( \ast \)case ..
next
case \( \text{base2 } t U S \)

149
from subset-refl show ?case by (rule base2.prems) simp
next
  case (step t U S V0 x ps0 ps1 qs0 qs1)
from step.hyps(3) obtain V where I: V ⊆ U - {x} and 2: (monomial 1 t, V) ∈ set (snd (ps0, qs0))
  using step.hyps(15) by blast
show ?case
proof (rule step.prems)
  from I show V ⊆ U by blast
next
  from 2 show (monomial 1 t, V) ∈ set (snd (ps0 @ ps1, qs0 @ qs1)) by fastforce
qed
qed

lemma fst-splitE:
  assumes finite X and U ⊆ X and finite S and 0 ∉ S
  and (monomial (1::'a) s, V) ∈ set (fst (split t U S))
  obtains t' x where t' ∈ ⊥X and x ∈ X and V ⊆ U and 0 ∉ (λs. s - t') ⋵ S
  and s = t' + t + Poly-Mapping.single x 1
  and (monomial (1::'a::zero-neq-one) s, V) ∈ set (fst (split (t' + t) V ((λs. s - t') ⋵ S)))
  and set (snd (split (t' + t) V ((λs. s - t') ⋵ S))) ⊆ (set (snd (split t U S))) :: ((λ0 ⋵ 'a) ∗ -) set
  using assms
proof (induct t U S arbitrary: thesis rule: split-induct)
  case (base1 t U S)
from base1.prems(2) base1.hyps(3) show ?case ..
next
  case (base2 t U S)
from base2.prems(3) show ?case by simp
next
  case (step t U S V0 x ps0 ps1 qs0 qs1)
from step.prems(3) have (monomial 1 s, V) ∈ set ps0 ∪ set ps1 by simp
thus ?case
proof
  assume (monomial 1 s, V) ∈ set ps0
  hence (monomial (1::'a) s, V) ∈ set (fst (ps0, qs0)) by (simp only: fst-conv)
  with step.hyps(3) obtain t' x' where t' ∈ ⊥X and x' ∈ X and V ⊆ U - {x}
  and 0 ∉ (λs. s - t') ⋵ S and s = t' + t + Poly-Mapping.single x' 1
  and (monomial (1::'a) s, V) ∈ set (fst (split (t' + t) V ((λs. s - t') ⋵ S)))
  and set (snd (split (t' + t) V ((λs. s - t') ⋵ S))) ⊆ set (snd (ps0, qs0))
  using step.hyps(15) by blast
note this(7)
also have set (snd (ps0, qs0)) ⊆ set (snd (ps0 @ ps1, qs0 @ qs1)) by simp
finally have set (snd (split (t' + t) V ((λs. s - t') ⋵ S))) ⊆ set (snd (ps0 @ ps1, qs0 @ qs1)).
from :V ⊆ U - {x} have V ⊆ U by blast

150
show ?thesis by (rule step.prems) fact+

next

assume (monomial 1 s, V) ∈ set ps1

show ?thesis

proof (cases 0 ∈ (λf. f − Poly-Mapping.single x 1) ' S)

case True

from step.hyps(2) have fin: finite ((λf. f − Poly-Mapping.single x 1) ' S)

by (rule finite-imageI)

have split (Poly-Mapping.single x 1 + t) U ((λf. f − Poly-Mapping.single x 1 + t) ' 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 (monomial 1 s, V) ∈ set ps1; have s = Poly-Mapping.single x 1 + t and V = U

by (auto dest!: monomial-inj)

show ?thesis

proof (rule step.prems)

show 0 ∈ [X] by (fact zero-in-PPs)

next

from step.hyps(8, 1) show x ∈ X ..

next

show V ⊆ U by (simp add: V = U)

next

from step.hyps(3) show 0 ∉ (λs. s − 0) ' S by simp

next

show s = 0 + t + Poly-Mapping.single x 1 by (simp add: s add.commute)

next

show (monomial 1::'a) s, V) ∈ set (fst (split (0 + t) V ((λs. s − 0) ' S)))

using (monomial 1 s, V) ∈ set ps1 by (simp add: step.hyps(14) V = U)

next

show set (snd (split (0 + t) V ((λs. s − 0) ' S))) ⊆ set (snd (ps0 @ ps1, qs0 @ qs1))

by (simp add: step.hyps(14) V = U)

qed

next

case False

moreover from (monomial 1 s, V) ∈ set ps1; have (monomial 1 s, V) ∈ set (fst (ps1, qs1))

by (simp only: fst-conv)

ultimately obtain t' x' where t' ∈ [X] and x' ∈ X and V ⊆ U

and 1: 0 ∉ (λs. s − t') ' (λf. f − Poly-Mapping.single x 1) ' S

and s: s = t' + (Poly-Mapping.single x 1 + t) + Poly-Mapping.single x' 1

and 2: (monomial 1::'a) s, V) ∈ set (fst (t' + (Poly-Mapping.single x 1 + t)) V

151
\begin{align*}
((\lambda s. s - t') \cdot (\lambda f - \text{Poly-Mapping}.\text{single } x 1) \cdot S))
\end{align*}

\textbf{PPs-closed-plus}\: \textbf{simp only}:
\begin{align*}
t \cdot (\lambda f - \text{Poly-Mapping}.\text{single } x 1 + t)) V ((\lambda s - t') \cdot (\lambda f - \text{monomial } 1 x) \cdot S)) \subseteq
\end{align*}

\textbf{set} \: (\text{snd} (\text{ps}1, \text{qs}1))

\textbf{using} \: \text{step.hyps}(16) \: \text{by} \: \text{blast}

\textbf{have} \: \text{eq} \: (\lambda s - t') \cdot (\lambda f - \text{Poly-Mapping}.\text{single } x 1) \cdot S =
(\lambda s - (t' + \text{Poly-Mapping}.\text{single } x 1)) \cdot S

\textbf{by} \: (\text{simp add: image-image add.commute diff-diff-eq})

\textbf{show} \: \text{thesis}

\textbf{proof} \: (\text{rule step.prems})

\textbf{from} \: \text{step.hyps}(8, 1) \: \textbf{have} \: x \in X ..

\textbf{hence} \: \text{Poly-Mapping}.\text{single } x 1 \in [X] \: \text{by} \: (\text{rule PPs-closed-single})

\textbf{with} \: t' \in [X] \: \textbf{show} \: t' + \text{Poly-Mapping}.\text{single } x 1 \in [X] \: \text{by} \: (\text{rule PPs-closed-plus})

\textbf{next}

\textbf{from} \: 1 \: \textbf{show} \: 0 \notin (\lambda s - (t' + \text{Poly-Mapping}.\text{single } x 1)) \cdot S
\textbf{by} \: (\text{simp only: eq not-False-eq-True})

\textbf{next}

\textbf{show} \: (\text{monomial } (1::'a) \cdot s, V) \in \text{set} \: (\text{fst} (\text{split} (t' + \text{Poly-Mapping}.\text{single } x 1 + t) V)
((\lambda s - (t' + \text{Poly-Mapping}.\text{single } x 1)) \cdot S))

\textbf{using} \: 2 \: \textbf{by} \: (\text{simp only: eq add.assoc})

\textbf{next}

\textbf{have} \: \text{set} \: (\text{snd} (\text{split} (t' + \text{Poly-Mapping}.\text{single } x 1 + t) V ((\lambda s - (t' + \text{Poly-Mapping}.\text{single } x 1)) \cdot S)) \subseteq
\text{set} \: (\text{snd} (\text{ps}1, \text{qs}1)) \: (\text{is} \: (?x \subseteq -) \: \text{using} \: 3 \: \textbf{by} \: (\text{simp only: eq add.assoc})
\textbf{also} \: \text{have} \: \ldots \subseteq \text{set} \: (\text{snd} (\text{ps}0 @ \text{ps}1, \text{qs}0 @ \text{qs}1)) \: \textbf{by} \: \text{simp}
\textbf{finally} \: \textbf{show} \: ?x \subseteq \text{set} \: (\text{snd} (\text{ps}0 @ \text{ps}1, \text{qs}0 @ \text{qs}1)) \cdot \text{qed fact+}
\text{qed}
\text{qed}
\text{qed}

\textbf{lemma lem-4-8:}
\textbf{assumes} \: \text{finite } X \: \textbf{and} \: \text{finite } S \: \textbf{and} \: S \subseteq [X] \: \textbf{and} \: \emptyset \notin S
\textbf{and} \: g \in \text{punit.reduced-GB} \: (\text{monomial } (1::'a) \cdot S)
\textbf{obtains} \: t \: \textbf{where} \: U \subseteq X \: \textbf{and} \: (\text{monomial } (1::'a::field) \cdot t, U) \in \text{set} \: (\text{snd} \: (\text{split} \: 0 \: X \: S))
\textbf{and} \: \text{poly-deg } g = \text{Suc} \: (\text{deg-pm } t)
\textbf{proof} -
\textbf{let} \: ?S = \text{monomial } (1::'a) \cdot S
\textbf{let} \: ?G = \text{punit.reduced-GB} \: ?S
\textbf{have} \: \text{md}1: \: \text{monomial-decomp} \: (\text{fst} \: (\text{split} \: 0 \: X \: S)::(- \times ((((- \Rightarrow 0 \: 'a) \times -) \: \text{list})))
\textbf{and} \: \text{md}2: \: \text{monomial-decomp} \: (\text{snd} \: (\text{split} \: 0 \: X \: S)::(- \times ((((- \Rightarrow 0 \: 'a) \times -) \: \text{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
by (rule reduced-GB-is-auto-reduced-Polys, rule reduced-GB-is-monic-set-Polys,
rule reduced-GB-ideal-Polys, rule reduced-GB-nonzero-Polys)
from this(2, 4) assms(5) have punit.lc g = 1 by (auto simp: punit.is-monic-set-def)
have is-monomial-set ?S by (auto intro!: is-monomial-set1 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-monic-setD)
moreover define s where s = lpp g
ultimately have g: g = monomial 1 s using ⟨punit.lc g = 1⟩ by (metis punit.monic-set-def)

note assms(1) subset-refl assms(2) zero-in-PPs
moreover have ideal ?G ÷ monomial 1 0 = ideal ?S by (simp add: ideal-G)
ultimately have splits-wrt (split 0 X S) (cone (monomial (1::a) 0, X)) ?G by
(rule split-splits-wrt)
hence splits-wrt (fst (split 0 X S), snd (split 0 X S)) [P[X] ?G by simp
hence cone-decomp (P[X] ∩ ideal ?G) (fst (split 0 X S))
using md2 is-monomial-set ?G. by (rule splits-wrt-cone-decomp-1)
hence cone-decomp (P[X] ∩ ideal ?S) (fst (split 0 X S)) by (simp only: ideal-G)
with assms(1, 3, 5) obtain U where (g, U) ∈ set (fst (split 0 X S)) using md1 by (rule lcms-4-7)
with assms(1) subset-refl assms(2, 4) obtain t' x where t' ∈ .[X] and x ∈ X
and U ⊆ X
and 0 ∉ (λs. s − t') · S and s: s = t' + 0 + Poly-Mapping.single x 1
and (g, U) ∈ set (fst (split (t' + 0) U ((λs. s − t') · S)))
and set (snd (split (t' + 0) U ((λs. s − t') · S))) ⊆ (set (snd (split 0 X S)))
:: ((· ⇒0 'a) × ·) set

unfolding g by (rule fst-splitE)
let S = (λs. s − t') · S
from assms(2) have finite ?S by (rule finite-imageI)
with assms(1) :U ⊆ X obtain V where V ⊆ U
and (monomial (1::a) (t' + 0), V) ∈ set (snd (split (t' + 0) U ?S))
using 0 ∉ ?S by (rule snd-splitI)

note this(2).
also have . . . ⊆ set (snd (split 0 X S)) by fact
finally have (monomial (1::a) t', V) ∈ set (snd (split 0 X S)) by simp
have poly-deg g = Suc (deg-pm t') by (simp add: g s deg-pm-plus deg-pm-single
poly-deg-monomial)
from ‹V ⊆ U› :U ⊆ X obtain V ⊆ X by (rule subset-trans)
show ‹thesis› by rule fact+

qed

corollary cor-4-9:
assumes finite X and finite S and ‹S ⊆ .[X]›
and g ∈ punit.reduced-GB (monomial (1::a::field) · S)
shows poly-deg g ≤ Suc (Max (poly-deg · fst · (set (snd (split 0 X S)))) :: ((· ⇒0 'a) × ·) set)›
proof (cases $t \in S$)
case True
  have $\deg-pm \ t$ using $\assms(3)$ where
  thus $\assms(4)$ using $\assms(1)$
next
case False
have $\assms(3)$ using $\assms(1)$
with $\assms(4)$
qed

lemma standard-decomp-snd-split:
  assumes $finite \ X$ and $U \subseteq X$ and $finite \ S$ and $S \subseteq \{X\}$ and $t \in .\{X\}$
  shows $standard-decomp \ (\deg-pm \ t) \ (\snd \ (split \ t \ U \ S))$ :: $((\Rightarrow a::\text{field}) \times \cdot)$
proof (induct $t \ U \ S$ rule: split-induct)
case (base1 $t \ U \ S$)
  show $\assms$ by (simp add: standard-decomp-nil)
next
case (base2 $t \ U \ S$)
  have $\deg-pm \ t = \deg-pm \ (monomial \ (1::\text{a}) \ t)$ by (simp add: poly-deg-monomial)
  thus $\assms$ by (simp add: standard-decomp-singleton)
next
case (step $t \ U \ S \ V \ x \ ps0 \ ps1 \ qs0 \ qs1$)
from step.hyps(15) step.prems have $qs0$: $standard-decomp \ (\deg-pm \ t) \ qs0$ by
  (simp only: snd-conv)
  have $(\lambda s. \ s - \text{Poly-Mapping.single \ x \ 1}) \ (S \subseteq .\{X\})$
  proof
    fix $u$
    assume $u \in (\lambda s. \ s - \text{Poly-Mapping.single \ x \ 1}) \ (S$
    then obtain $s$ where $s \in S$ and $u \in (\lambda s. \ s - \text{Poly-Mapping.single \ x \ 1}$ ..
    from this(1) step.prems(1) have $s \in .\{X\}$ ..
    thus $u \in .\{X\}$ unfolding $u$ by (rule PPs-closed-minus)
  qed
next
case (step $t \ U \ S \ V \ x \ ps0 \ ps1 \ qs0 \ qs1$)
  have $\assms(2)$ have $\assms(1)$ by (simp add: poly-deg-monomial)
  thus $\assms$ by (simp add: poly-deg-monomial)
proof (rule PPs-closed-plus)
from \textit{step.hyps}(8, 1) \textbf{have} \( x \in X \) ..
thus \textit{Poly-Mapping.single} \( x \in \mathcal{X} \) \textbf{by} (rule \textit{PPs-closed-single})
\textbf{qed}
ultimately \textbf{have} \( qs1: \text{standard-decomp} (\text{Suc} (\text{deg-pm} t)) \) \textbf{using} \( \text{step.hyps}(16) \)
- \textbf{by} (simp add: \textit{deg-pm-plus deg-pm-single})
\textbf{show} \(?case\ unfold\ \textit{snd-conv}\)
\textbf{proof} (rule \textit{standard-decompI})
\textbf{fix} \( h \) \( U0 \)
\textbf{assume} \((h, U0) \in \text{set} (qs0 \ @ \ qs1+)\)
\textbf{hence} \(? \in \text{set} (qs0+) \cup \text{set} (qs1+)\) \textbf{by} (simp add: \textit{pos-decomp-append})
\textbf{thus} \textit{deg-pm} \( t \leq \text{poly-deg} \ h \) \textbf{proof}
\textbf{assume} \((h, U0) \in \text{set} (qs1+), with\ \textit{qs0}\ \textbf{show} \ ?\text{thesis}\ \textbf{by} (rule \textit{standard-decompD})\)
\textbf{next}
\textbf{assume} \((h, U0) \in \text{set} (qs1+), with\ \textit{qs1}\ \textbf{have} \ \text{Suc} (\text{deg-pm} t) \leq \text{poly-deg} \ h \) \textbf{by} (rule \textit{standard-decompD})
\textbf{thus} \ ?\text{thesis}\ \textbf{by simp}
\textbf{qed}
\textbf{fix} \( d \)
\textbf{assume} \( d1: \text{deg-pm} \ t \leq \ d \) and \( d2: \ d \leq \text{poly-deg} \ h \)
\textbf{from} \(?\ \text{show} \ \exists t' U'. (t', U') \in \text{set} (qs0 \ @ \ qs1) \land \text{poly-deg} \ t' = d \land \text{card} \ U0 \leq \text{card} \ U'\)
\textbf{proof}
\textbf{assume} \((h, U0) \in \text{set} (qs0+)
\textbf{with}\ \textit{qs0}\ \textbf{obtain} \ h' U' \ where\ \textbf{(}h', U'\textbf{)} \in \text{set} \ qs0\ \textbf{and} \ \text{poly-deg} \ h' = d\ \textbf{and} \ \text{card} \ U0 \leq \text{card} \ U'\)
\textbf{using} \( d1\ d2\ \textbf{by} (rule \textit{standard-decompE})\)
\textbf{moreover from} \( \textit{this}(1)\ \textbf{have} \ \langle h', U' \rangle \in \text{set} (qs0 \ @ \ qs1)\ \textbf{by simp}\)
\textbf{ultimately show} \ ?\text{thesis}\ \textbf{by blast}\)
\textbf{next}
\textbf{assume} \((h, U0) \in \text{set} (qs1+), with\ \textit{qs1}\ \textbf{have} \ \text{Suc} (\text{deg-pm} t) \leq \text{poly-deg} \ h \) \textbf{by} (rule \textit{standard-decompD})+
\textbf{moreover from} \( \textit{this}(1)\ \textbf{have} \ \text{is-monomial} \ h\ \textbf{and} \ \textit{punit.lc} \ h = 1\ \textbf{by} (rule \textit{punit.monomial-eq-itself})\)
\textbf{moreover define} \( s\ where\ s = \text{lpp} \ h\)
\textbf{ultimately have} \( h: h = \text{monomial} 1 s\ \textbf{by simp}\)
\textbf{from} \( d1\ \textbf{have} \ \text{deg-pm} \ t = d \lor \text{Suc} (\text{deg-pm} t) \leq d\ \textbf{by auto}\)
\textbf{thus} \ ?\text{thesis}\ \textbf{proof}
\textbf{assume} \( \text{deg-pm} \ t = d\)
\textbf{define} \( F\ where\ F = (\ast) (\text{monomial} 1 t) \ast \text{monomial} (1::'a) \ast S\)

155
have \( F \subseteq P[X] \)

proof

fix \( f \)

assume \( f \in F \)

then obtain \( u \) where \( u \in S \) and \( f : \text{monomial } 1 \( (t + u) \)

by (auto simp: F-def times-monomial-monomial)

from this(1) step.prems(1) have \( u \in \{X\} \)

with step.prems(2) have \( t + u \in \{X\} \) by (rule PPs-closed-plus)

thus \( f \in P[X] \) unfolding \( f \) by (rule Polys-closed-monomial)

qed

have \( \text{ideal } F = (\ast) \) (monomial \( 1 \ t \) \( \triangleleft \) ideal (monomial \( 1 \ t \) \( \triangleleft S \))

by (simp only: ideal.span-image-scale-eq-image-scale F-def)

moreover have \( \text{inj } (\ast) \) (monomial \( 1 \ a \ t \))

by (auto intro!: injI simp: times-monomial-left monomial-0-iff dest!:

punit.monom-mult-inj-3)

ultimately have \( \text{eq: ideal } F \div \text{monomial } 1 \ t = \text{ideal } (\text{monomial } 1 \ t \ \triangleleft S) \)

by (simp only: quot-set-image-times)

with assms(1) step.hyps(1, 2) step.prems(2)

have \( \text{splits-wrt } (\text{split } t \ U \ S) \) (cone (monomial \( 1 \ a \ t \), \( U \)) \( F \)) by (rule split-splits-wrt)

hence \( \text{splits-wrt } (qs0 @ qs1, qs0 @ qs1) \) (cone (monomial \( 1 \ a \ t \), \( U \)) \( F \)) by

(simp only: step.hyps(14))

with assms(1) have \( \text{cone-decomp } (\text{cone } (\text{monomial } 1 \ a \ t , \ U) \ \triangleleft \ \text{normal-form } F \div \text{P}[X]) \) (qs0 @ qs1)

using \( md \ \cdot f \subseteq \text{P}[X] \)

by (rule split-splits-wrt-cone-decomp-2)

(auto intro!: is-monomial-setI monomial-is-monomial simp: F-def

times-monomial-monomial)

hence \( \text{cone } (\text{monomial } 1 \ s , \ U0) \subseteq \text{cone } (\text{monomial } 1 \ a \ t , \ U) \ \triangleleft \ \text{normal-form } F \div \text{P}[X]) \)

using \( (h, U) \in \text{set } (qs0 @ qs1) \) unfolding \( h \) by (rule cone-decomp-cone-subset)

with assms(1) step.hyps(1, 2) step.prems(2) \( F \subseteq \text{P}[X] \)

obtain \( U' \) where \( (\text{monomial } 1 \ a \ t , \ U') \in \text{set } (\text{snd } (\text{split } t \ U \ S)) \) and

\( \text{card } U0 \leq \text{card } U' \)

by (rule lem-4-6)

from this(1) have \( \text{(monomial } 1 \ t , \ U') \in \text{set } (qs0 @ qs1) \) by (simp add: step.hyps(14))

show \( \text{thesis} \)

proof (intro ezI conjI)

show \( \text{poly-deg } (\text{monomial } 1 \ a \ t) = d \) by (simp add: poly-deg-monomial

deg-pm t = d)

qed fact+

next

assume \( \text{Suc } (\text{deg-pm } t) \leq d \)

with \( qs1 \ \langle h, U0 \rangle \in \text{set } (qs1+) \) obtain \( h' \ U' \) where \( (h', U') \in \text{set } qs1 \)

and \( \text{poly-deg } h' = d \)

and \( \text{card } U0 \leq \text{card } U' \) using \( d2 \) by (rule standard-decompE)

moreover from this(1) have \( (h', U') \in \text{set } (qs0 @ qs1) \) by simp

ultimately show \( \text{thesis} \) by blast

156
proof
qed
qed
qed

theorem standard-cone-decomp-snd-split:
  fixes F
  defines G ≡ punit.reduced-GB F
  defines ss ≡ (split 0 X (lpp ' G)) :: ((· ⇒0 'a::field) × ·) list × ·
  defines d ≡ Suc (Max (poly-deg ' fst ' set (snd ss)))
  assumes finite X and F ⊆ P[X]
  shows standard-decomp 0 (snd ss) (is ?thesis1)
    and cone-decomp (normal-form F : P[X]) (snd ss) (is ?thesis2)
    and (∀ f ∈ F ⇒ homogeneous f) "poly-deg g ≤ d"
proof –
  have ideal G = ideal F and punit.is-Groebner-basis G and finite G and 0 ∉ G
    and G ⊆ P[X] and punit.is-reduced-GB G using assms(4 , 5) unfolding G-def
    by (rule reduced-GB-ideal-Polys, rule reduced-GB-is-GB-Polys, rule finite-reduced-GB-Polys, rule reduced-GB-nonzero-Polys, rule reduced-GB-Polys, rule reduced-GB-is-reduced-GB-Polys)
  define S where S = lpp ' G
  note assms(4) subset-refl
  moreover from (finite G) have finite S unfolding S-def by (rule finite-imageI)
  moreover from (G ⊆ P[X]) have S ⊆ [X] unfolding S-def by (rule PPs-closed-image-lpp)
  ultimately have standard-decomp (deg-pm (0::'x ⇒0 nat)) (snd ss)
    using zero-in-PPs unfolding ss-def S-def by (rule standard-decomp-snd-split)
  thus ?thesis1 by simp

let ?S = monomial (1::'a) ' S
from (S ⊆ [X]) have ?S ⊆ P[X] by (auto intro: Polys-closed-monomial)
have splits-wrt ss (cone (monomial 1 0, X)) ?S
  using assms(4) subset-refl (finite S) zero-in-PPs unfolding ss-def S-def
  by (rule split-splits-wrt) simp
hence splits-wrt (fst ss, snd ss) P[X] ?S by simp
with assms(4) have cone-decomp (P[X] ∩ normal-form ?S ' P[X]) (snd ss)
  using - - (?S ⊆ P[X])
proof (rule splits-wrt-cone-decomp-2)
  from assms(4) subset-refl (finite S) show monomial-decomp (snd ss)
    unfolding ss-def S-def by (rule monomial-decomp-split)
qed (auto intro: is-monomial-setI monomial-is-monomial)
moreover have normal-form ?S ' P[X] = normal-form F ' P[X]
  by (rule set-eqI)
    (simp add: image-normal-form-iff[OF assms(4)] assms(5) : ?S ⊆ P[X],
    simp add: S-def is-reduced-GB-monomial-It-GB-Polys[OF assms(4)] (G ⊆ P[X] : 0 ∉ G) flip: G-def)
moreover from assms(4, 5) have normal-form F ' P[X] ⊆ P[X]
  by (auto intro: Polys-closed-normal-form)
ultimately show ?thesis2 by (simp only: Int-absorb1)
assume $\forall f. f \in F \Rightarrow \text{homogeneous } f$
moreover note $\text{punit.is-reduced-GB } G; \text{ideal } G = \text{ideal } F$
moreover assume $g \in G$
ultimately have $\text{homogeneous } g$ by (rule $\text{is-reduced-GB-homogeneous}$)
moreover have $\text{lpp } g \in \text{keys } g$
proof (rule $\text{punit.lt-in-keys}$)
  from $g \in G; (0) \notin G; \text{show } g \neq 0$ by blast
qede
ultimately have $\text{deg-lt: deg-pm } (\text{lpp } g) = \text{poly-deg } g$ by (rule $\text{homogeneousD-poly-deg}$)
from $g \in G$ have monomial $I$ $(\text{lpp } g) \in ?S$ unfolding $\text{S-def}$ by (intro imagef)
also have ... = $\text{punit.reduced-GB } ?S$ unfolding $\text{S-def } G$-def using $\text{assms(4, 5)}$
  by (rule $\text{reduced-GB-monomial-lt-reduced-GB-Polys[symmetric]}$
finally have monomial $I$ $(\text{lpp } g) \in \text{punit.reduced-GB } ?S$
with $\text{assms(4) } (\text{finite } S); (\text{S } \subseteq \{X\})$ have $\text{poly-deg } (\text{monomial } (1::'a) (\text{lpp } g)) \leq d$
  unfolding $\text{d-def ss-def } \text{S-def } \text{symmetric }$ by (rule $\text{cor-4-9}$)
thus $\text{poly-deg } g \leq d$ by (simp add: $\text{poly-deg-monomial } \text{deg-lt}$
qede

10.7 Splitting Ideals

qualified definition $\text{ideal-decomp-aux } :: (('x \Rightarrow 0 \text{ nat}) \Rightarrow 'a) \text{ set } \Rightarrow (('x \Rightarrow 0 \text{ nat}) \Rightarrow 0 'a) \Rightarrow ((('x \Rightarrow 0 \text{ nat}) \Rightarrow 0 'a::field) \text{ set } \times ((('x \Rightarrow 0 \text{ nat}) \Rightarrow 0 'a))$
where $\text{ideal-decomp-aux } F f =$
  $(\text{let } J = \text{ideal } F; L = (J \div f) \cap P[X]; L' = \text{lpp } f \text{' punit.reduced-GB } L \text{ in}$
  $(\ast) \text{' normal-form } L ' P[X], \text{ map } (\text{apfst } ((\ast) f)) (\text{snd } (\text{split } 0 X L' )))$
context
  assumes $\text{fin-X}: \text{finite } X$
begin

lemma $\text{ideal-decomp-aux}$:
  assumes $\text{finite } F \text{ and } F \subseteq P[X] \text{ and } f \in P[X]$
  shows $\text{fst } (\text{ideal-decomp-aux } F f) \subseteq \text{ideal } \{f\} \text{ (is } \text{?thesis1}$
  and $\text{ideal } F \cap \text{fst } (\text{ideal-decomp-aux } F f) = \{0\} \text{ (is } \text{?thesis2}$
  and $\text{direct-decomp } (\text{ideal } (\text{insert } f F) \cap P[X]) [\text{fst } (\text{ideal-decomp-aux } F f), \text{ideal } F \cap P[X]] \text{ (is } \text{?thesis3}$
  and $\text{cone-decomp } (\text{fst } (\text{ideal-decomp-aux } F f)) (\text{snd } (\text{ideal-decomp-aux } F f)) \text{ (is } \text{?thesis4}$
  and $f \neq 0 \Rightarrow \text{valid-decomp } X \text{ (snd } (\text{ideal-decomp-aux } F f)) \text{ (is } - \Rightarrow ?\text{thesis5}$
  and $f \neq 0 \Rightarrow \text{standard-decomp } (\text{poly-deg } f) \text{ (snd } (\text{ideal-decomp-aux } F f)) \text{ (is } - \Rightarrow ?\text{thesis6}$
  and $\text{homogeneous } f \Rightarrow \text{hom-decomp } (\text{snd } (\text{ideal-decomp-aux } F f)) \text{ (is } - \Rightarrow ?\text{thesis7}$
proof –
define \( J \) where \( J = \text{ideal } F \)
define \( L \) where \( L = (J \div f) \cap P[X] \)
define \( S \) where \( S = (\ast) f \cdot \text{normal-form } L \cdot P[X] \)
define \( L' \) where \( L' = \text{lpp} \cdot \text{punit.reduced-GB } L \)

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

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

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

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

unfolding eq fst-conv S-def L-def J-def using fin-X assms(2, 3) by (rule direct-decomp-ideal-insert)

thus \( ?\text{thesis3} \) using perm.swap by (rule direct-decomp-perm)

have \( \text{std: standard-decomp} 0 \ (\text{snd} (\text{split} 0 X L')) :: ((\Rightarrow_0 'a) \times \cdot) \text{ list} \)

and cone-decomp \( (\text{normal-form} L : P[X]) \ (\text{snd} (\text{split} 0 X L')) \)

unfolding \( L'-\text{def} \) using fin-X \( L \subseteq P[X] \) by (rule standard-cone-decomp-snd-split)+

from this(2) show \( ?\text{thesis4} \) unfolding eq fst-conv snd-conv S-def by (rule cone-decomp-map-times)

from fin-X \( L \subseteq P[X] \) have finite \( (\text{panit.reduced-GBL}) \) by (rule finite-reduced-GB-Polys)

hence finite \( L' \) unfolding \( L'-\text{def} \) by (rule finite-imageI)

\{
  have monomial-decomp \( (\text{snd} (\text{split} 0 X L')) :: ((\Rightarrow_0 'a) \times \cdot) \text{ list} \)
    using fin-X subset-refl \( \text{finite} L' \) by (rule monomial-decomp-split)

  hence hom-decomp \( (\text{snd} (\text{split} 0 X L')) :: ((\Rightarrow_0 'a) \times \cdot) \text{ list} \)
    by (rule monomial-decomp-imp-hom-decomp)

  moreover assume homogeneous \( f \)

  ultimately show \( ?\text{thesis7} \) unfolding eq snd-conv by (rule hom-decomp-map-times)
\}

have \( \text{vd: valid-decomp} X \ (\text{snd} (\text{split} 0 X L')) :: ((\Rightarrow_0 'a) \times \cdot) \text{ list} \)

using fin-X subset-refl \( \text{finite} L' \) zero-in-PPs by (rule valid-decomp-split)

moreover note assms(3)

moreover assume \( f \neq 0 \)

ultimately show \( ?\text{thesis5} \) unfolding eq snd-conv by (rule valid-decomp-map-times)

from std \( \text{vd} (f \neq 0) \) have standard-decomp \( (0 + \text{poly-deg} f) \ (\text{map} (\text{apfst} ((\ast) f)) \ (\text{snd} (\text{split} 0 X L'))) \)

by (rule standard-decomp-map-times)

thus \( ?\text{thesis6} \) by (simp add: eq)

qed

lemma ideal-decompE:

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

assumes finite \( F \) and \( F \subseteq P[X] \) and \( f0 \in P[X] \) and \( \forall f. \ f \in F \implies \text{poly-deg} f \leq \text{poly-deg} f0 \)

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

and cone-decomp \( T \ps \)

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

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

using assms(1, 2, 4)

proof (induct \( F \) arbitrary: \( \text{thesis} \))

case empty

show \( ?\text{case} \)

proof (rule empty.prems)

show \( \text{valid-decomp X} \ \[] \) by (rule valid-decompI) simp-all

qed
next
  show standard-decomp \((\text{poly-deg } f_0)\) [] 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 \([f_0] \cap P[X]\)) \([\text{ideal } \{f_0\} \cap P[X]]\)
    by (fact direct-decomp-singleton)
hence direct-decomp (ideal \([f_0] \cap P[X]\)) \([\{0\}, \text{ideal } \{f_0\} \cap P[X]]\) by (rule direct-decomp-Cons-zeroI)
  thus direct-decomp (ideal \([f_0] \cap P[X]\)) \([\text{ideal } \{f_0\} \cap P[X], \{0\}]\)
    using perm.swap by (rule direct-decomp-perm)
qed (simp add: hom-decomp-def)
next
case \((\text{insert } f \ F)\)
  from insert.prems\((2)\) have \(F \subseteq P[X]\) by simp
moreover have poly-deg \(f' \leq\) poly-deg \(f_0\) if \(f' \in F\) for \(f'\)
proof –
  from that have \(f' \in \text{insert } f \ F\) by simp
  thus ?thesis by (rule insert.prems)
qed
ultimately obtain \(T \ ps\) where valid-ps: valid-decomp \(X \ ps\) and std-ps: standard-decomp
  \((\text{poly-deg } f_0)\) \(ps\)
  and cn-ps: cone-decomp \(T \ ps\) and dd: direct-decomp (ideal (insert \(f_0 \ F\) \(\cap P[X]\)) \([\text{ideal } \{f_0\} \cap P[X], T]\)
  and hom-ps: \((\forall f. f \in F \Rightarrow \text{homogeneous } f) \Rightarrow \text{hom-decomp } ps\)
  using insert.hyps\((3)\) by metis
show ?case
proof (cases \(f = 0\))
case True
show ?thesis
proof (rule insert.prems)
  from dd show direct-decomp (ideal (insert \(f_0 \ (\text{insert } f \ F)\) \(\cap P[X]\)) \([\text{ideal } \{f_0\} \cap P[X], T]\)
    by (simp only: insert-commute[of \(f_0\) True ideal.span-insert-zero)
next
  assume \((\forall f'. \ f' \in \text{insert } f \ F \Rightarrow \text{homogeneous } f')\)
hence \((\forall f. f \in F \Rightarrow \text{homogeneous } f) \text{ by blast}\)
  thus hom-decomp ps by (rule hom-ps)
qed fact+
next
case False
let \(?D = \text{ideal-decomp-aux } (\text{insert } f_0 \ F)\) \(f\)
from insert.hyps\((1)\) have \(f0F-f\in: \text{finite } (\text{insert } f_0 \ F)\) by simp
moreover from \(\{F \subseteq P[X]\} \ assms\((3)\) have \(f0F-sub: \text{insert } f_0 \ F \subseteq P[X]\) by simp
moreover from insert.prems\((2)\) have \(f \in P[X]\) by simp
ultimately have eq: \(\text{ideal } (\text{insert } f_0 \ F) \cap \text{fst } ?D = \{0\}\) and valid-decomp \(X\)
  (snd \(?D\))
and \( cn-D; \) cone-decomp \((\text{fst } ?D) \) \((\text{snd } \, ?D)\)
and standard-decomp \((\text{poly-deg } f) \) \((\text{snd } \, ?D)\)
and \( dd': \) direct-decomp \((\text{ideal } (\text{insert } f \ (\text{insert } f0 \ F)) \cap P[X])\)
\[
\left[\text{fst } \, ?D, \ \text{ideal } (\text{insert } f0 \ F) \cap P[X]\right]
\]
and \( \text{hom-}D; \) homogeneous \( f \implies \text{hom-decomp } \) \((\text{snd } \, ?D)\)
by \((\text{rule ideal-decomp-aux, auto intro: ideal-decomp-aux simp: False})\)
\begin{itemize}
\item note \( \text{fin-}X \) this \((2-4)\)
\end{itemize}
moreover \( \text{have poly-deg } f \leq \text{poly-deg } f0 \) by \((\text{rule insert.prems})\)
simp
ultimately obtain \( \text{qs} \) where \( \text{valid-qs} : \text{valid-decomp } X \text{ qs and cn-qs: cone-decomp} \)
\((\text{fst } ?D)\) \(\text{qs}\)
and \( \text{std-qs: standard-decomp } (\text{poly-deg } f0) \) \(\text{qs}\)
and \( \text{hom-qs: hom-decomp } (\text{snd } \, ?D) \implies \text{hom-decomp } \text{qs by (rule standard-decomp-geE)}\)
blast
let \( ?T = \text{sum-list } ' \text{listset } [T, \text{fst } ?D]\)
let \( ?ps = ps \odot \text{qs}\)
show \( \text{thesis}\)
proof \((\text{rule insert.prems})\)
from \( \text{valid-ps valid-qs show valid-decomp } X \text{ ?ps by (rule valid-decomp-append)}\)
next
from \( \text{std-ps std-qs show standard-decomp } (\text{poly-deg } f0) \) \(\text{?ps by (rule standard-decomp-append)}\)
next
from \( \text{dd perm.swap have direct-decomp } (\text{ideal } (\text{insert } f0 \ F) \cap P[X]) \) \([T, \text{ideal} \{f0\} \cap P[X]]\)
by \((\text{rule direct-decomp-perm})\)
\begin{itemize}
\item hence \( T \subseteq \text{ideal } (\text{insert } f0 \ F) \cap P[X]\)
\end{itemize}
by \((\text{rule direct-decomp-Cons-subsetI})\) \((\text{simp add: ideal.span-zero zero-in-Polys})\)
\begin{itemize}
\item hence \( T \cap \text{fst } ?D \subseteq \text{ideal } (\text{insert } f0 \ F) \cap \text{fst } ?D\) by blast
\end{itemize}
\begin{itemize}
\item hence \( T \cap \text{fst } ?D \subseteq \{0\}\) by \((\text{simp only: eq})\)
\end{itemize}
from refl have \( \text{direct-decomp } ?T \) \([T, \text{fst } ?D]\)
proof \((\text{intro direct-decompI inj-onI})\)
fix \( xs \) \(ys\)
assume \( xs \in \text{listset } [T, \text{fst } ?D]\)
then obtain \( x1 \) \(x2 \) where \( x1 \in T \) \( and \) \( x2 \in \text{fst } ?D \) \( and \) \( xs: xs = [x1, x2]\)
by \((\text{rule listset-doubletonE})\)
assume \( ys \in \text{listset } [T, \text{fst } ?D]\)
then obtain \( y1 \) \(y2 \) where \( y1 \in T \) \( and \) \( y2 \in \text{fst } ?D \) \( and \) \( ys: ys = [y1, y2]\)
by \((\text{rule listset-doubletonE})\)
assume \( \text{sum-list } xs = \text{sum-list } ys\)
\begin{itemize}
\item hence \( x1 - y1 = y2 - x2 \) by \((\text{simp add: xs ys})\) \((\text{metis add-diff-cancel-left add-diff-cancel-right})\)
\end{itemize}
moreover from \( \text{cn-ps } (x1 \in T) \) \((y1 \in T)\) have \( x1 - y1 \in T \) by \((\text{rule cone-decomp-closed-minus})\)
moreover from \( \text{cn-D } (y2 \in \text{fst } ?D) \) \((x2 \in \text{fst } ?D)\) have \( y2 - x2 \in \text{fst } ?D\)
by \((\text{rule cone-decomp-closed-minus})\)
ultimately have \( y2 - x2 \in T \cap \text{fst } ?D\) by simp
also have \( \ldots \subseteq \{0\}\) by fact
finally have \( x2 = y2\) by simp
with \( (x1 - y1 = y2 - x2)\) show \( xs = ys\) by \((\text{simp add: xs ys})\)
qed

162
thus cone-decomp \( ?T \) using cn-ps cn-qs by (rule cone-decomp-append)

next

assume \( \forall \cdot f' \in \text{insert } f \ F \Rightarrow \text{homogeneous } f' \)

hence \( \text{homogeneous } f \) and \( \forall \cdot f' \in F \Rightarrow \text{homogeneous } f' \) by blast+

from this(2) have hom-decomp ps by (rule hom-ps)

moreover from \( \text{homogeneous } f \) have hom-decomp qs by (intro hom-qs hom-D)

ultimately show hom-decomp \( (ps @ qs) \) by (simp only: hom-decomp-append-iff)

next

from \( dd' \) have direct-decomp \( (\text{ideal } (\text{insert } f0 \ (\text{insert } f \ F)) \cap P[X]) \)

by (simp add: insert-commute direct-decomp-perm perm.swap)

hence direct-decomp \( (\text{ideal } (\text{insert } f0 \ (\text{insert } f \ F)) \cap P[X]) \)

\( ([\text{fst } ?D] @ [\text{ideal } \{f0\} \cap P[X], \ T]) \) using \( dd \) by (rule direct-decomp-direct-decomp)

hence direct-decomp \( (\text{ideal } (\text{insert } f0 \ (\text{insert } f \ F)) \cap P[X]) \)

\( ([\text{ideal } \{f0\} \cap P[X], \ T]) \) \( \Rightarrow \)

by (rule direct-decomp-perm auto)

hence direct-decomp \( (\text{ideal } (\text{insert } f0 \ (\text{insert } f \ F)) \cap P[X]) \)

\( \text{sam-list } \text{set}\) listset

\( [\text{ideal } \{f0\} \cap P[X], \ ?T] \)

by (rule direct-decomp-appendD)

thus direct-decomp \( (\text{ideal } (\text{insert } f0 \ (\text{insert } f \ F)) \cap P[X]) \)

\( [\text{ideal } \{f0\} \cap P[X], \ T] \)

by (simp add: image-image)

qed

qed

qed

10.8 Exact Cone Decompositions

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

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

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

\( h' \)

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

\( U' \))

lemma exact-decompI:

\( (\forall h. (h, U) \in \text{set } ps \Rightarrow h \in P[X]) \Rightarrow (\forall h. (h, U) \in \text{set } ps \Rightarrow U \subseteq X) \Rightarrow \)

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

\( h' \)

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

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 \( U \subseteq X \)

163
\begin{align*}
\text{and} \quad (h', U') \in \text{set } ps & \implies \text{poly-deg } h = \text{poly-deg } h' \implies m < \text{card } U \implies m < \text{card } U' \implies \\
(h, U) = (h', U')
\end{align*}

\textbf{using} \quad \text{assms} \quad \text{unfolding} \quad \text{exact-decomp-def by fastforce+}

\textbf{lemma} \quad \text{exact-decompI-zero:}

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

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

\begin{equation}
= \text{poly-deg } h' \implies \\
(h, U) = (h', U')
\end{equation}

\textbf{shows} \quad \text{exact-decomp } 0 \text{ } ps

\textbf{using} \quad \text{assms}(1, 2)

\textbf{proof} \quad (\text{rule } \text{exact-decompI})

\textbf{fix} \quad h \text{ } h' \quad \text{and} \quad U U' :: \quad \text{x set}

\textbf{assume} \quad 0 < \text{card } U

\text{hence} \quad U \neq \{\} \quad \text{by auto}

\textbf{moreover} \quad \text{assume} \quad (h, U) \in \text{set } ps

\textbf{ultimately have} \quad (h, U) \in \text{set } (ps_+) \quad \text{by} \quad (\text{simp add: pos-decomp-def})

\textbf{assume} \quad 0 < \text{card } U'

\text{hence} \quad U' \neq \{\} \quad \text{by auto}

\textbf{moreover} \quad \text{assume} \quad (h', U') \in \text{set } ps

\textbf{ultimately have} \quad (h', U') \in \text{set } (ps_+) \quad \text{by} \quad (\text{simp add: pos-decomp-def})

\textbf{assume} \quad \text{poly-deg } h = \text{poly-deg } h'

\textbf{with} \quad ⟨(h, U) \in \text{set } (ps_+)⟩ \quad ⟨(h', U') \in \text{set } (ps_+)⟩ \quad \text{show} \quad (h, U) = (h', U') \quad \text{by}

(\text{rule } \text{assms}(3))

\textbf{qed}

\textbf{lemma} \quad \text{exact-decompD-zero:}

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

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

\text{shows} \quad (h, U) = (h', U')

\textbf{proof} \quad --

\textbf{from} \quad \text{assms}(2) \quad \text{have} \quad (h, U) \in \text{set } ps \quad \text{and} \quad U \neq \{\} \quad \text{by} \quad (\text{simp-all add: pos-decomp-def})

\textbf{from} \quad \text{assms}(1) \quad \text{this}(1) \quad \text{have} \quad U \subseteq X \quad \text{by} \quad (\text{rule } \text{exact-decompD})

\text{hence} \quad \text{finite } U \quad \text{using} \quad \text{fin-X} \quad \text{by} \quad (\text{rule } \text{finite-subset})

\textbf{with} \quad ⟨U \neq \{\}⟩ \quad \text{have} \quad 0 < \text{card } U \quad \text{by} \quad (\text{simp add: card-gt-0-iff})

\textbf{from} \quad \text{assms}(3) \quad \text{have} \quad (h', U') \in \text{set } ps \quad \text{and} \quad U' \neq \{\} \quad \text{by} \quad (\text{simp-all add: pos-decomp-def})

\textbf{from} \quad \text{assms}(1) \quad \text{this}(1) \quad \text{have} \quad U' \subseteq X \quad \text{by} \quad (\text{rule } \text{exact-decompD})

\text{hence} \quad \text{finite } U' \quad \text{using} \quad \text{fin-X} \quad \text{by} \quad (\text{rule } \text{finite-subset})

\textbf{with} \quad ⟨U' \neq \{\}⟩ \quad \text{have} \quad 0 < \text{card } U' \quad \text{by} \quad (\text{simp add: card-gt-0-iff})

\textbf{show} \quad \text{?thesis} \quad \text{by} \quad (\text{rule } \text{exact-decompD}) \quad \text{fact+}

\textbf{qed}

\textbf{lemma} \quad \text{exact-decomp-imp-valid-decomp:}

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

\text{shows} \quad \text{valid-decomp } X \text{ } ps

\textbf{proof} \quad (\text{rule } \text{valid-decompI})

\textbf{fix} \quad h \text{ } U

164
assume *: \((h, U) \in \text{set} \ ps\)
with \text{assms}(1)\ show \(h \in P[X]\) and \(U \subseteq X\) by (rule exact-decompD)+
from * show \(h \neq 0\) by (rule \text{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 \text{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 \text{assms}(1)\ have \(U1 \subseteq X\) by (rule valid-decompD)
with \text{fin-X} have \(\text{card} U1 \leq \text{card} X\) by (rule card-mono)
also have \(\ldots \leq m\) by (fact \text{assms}(2))
also assume \(m < \text{card} U1\)
finally show \((h1, U1) = (h2, U2)\) by simp
qed

definition \(a::(((\'_x \Rightarrow \text{nat}) \Rightarrow \text{nat}) \times '_x \text{set})\)\ list \Rightarrow \text{nat}\)
where \(a ps = (\text{LEAST} k. \text{standard-decomp} k ps)\)
definition \(b::(((\'_x \Rightarrow \text{nat}) \Rightarrow \text{nat}) \times '_x \text{set})\)\ list \Rightarrow \text{nat} \Rightarrow \text{nat}\)
where \(b ps i = (\text{LEAST} d. a ps \leq d \land (\forall (h, U) \in \text{set} \ ps. i \leq \text{card} U \longrightarrow \text{poly-deg} h < d))\)

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

lemma a-nil:
assumes \(ps_+ = []\)
shows \(a ps = 0\)
proof --
from \text{assms} have \(\text{standard-decomp} 0 ps\) 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 \(\text{standard-decomp} k ps\) and \(ps_+ \neq []\)
shows \(a ps = \text{Min} (\text{poly-deg} ' \text{fst} ' \text{set} (ps_+))\)
using \text{fin-X} \text{assms}(1) - \text{assms}(3)
proof (rule standard-decomp-nonempty-unique)
from \text{assms}(2)\ show \(\text{standard-decomp} (a ps) ps\) by (rule a)
qed

lemma a-nonempty-unique:
assumes valid-decomp $X$ $ps$ and standard-decomp $k$ $ps$ and $ps_+ \neq \emptyset$
shows $a \leq k$
proof –
from assms have $a = \operatorname{Min} (\operatorname{poly-deg} \ (\operatorname{fst} \ (\operatorname{set} \ (ps_+))))$ by (rule a-nonempty)
moreover from fin-X assms have $k = \operatorname{Min} (\operatorname{poly-deg} \ (\operatorname{fst} \ (\operatorname{set} \ (ps_+))))$
by (rule standard-decomp-nonempty-unique)
ultimately show $?thesis$ by simp
qed

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

lemma b-le:
$a \leq d \implies (\forall h' \ U'. \ (h', U') \in \operatorname{set} \ ps \implies i \leq \operatorname{card} \ U' \implies \operatorname{poly-deg} h' < d)$

unfolding b-def by (intro Least-le) blast

lemma b-decreasing:
assumes $i \leq j$
shows $b$ $ps$ $j \leq b$ $ps$ i
proof (rule b-le)
fix $h$ $U$

166
assume \((h, U) \in \text{set } ps\)
assume \(j \leq \text{card } U\)
with \(\text{assms}(1)\) have \(i \leq \text{card } U\) by \(\text{rule le-trans}\)
with \((h, U) \in \text{set } ps\) show \(\text{poly-deg } h < b \text{ ps } i\) by \(\text{rule b}\)
qed \(\text{fact b}\)

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

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

corollary b-\text{zero-gr}:
assumes \((h, U) \in \text{set } ps\)
shows \(\text{poly-deg } h < b \text{ ps } 0\)
proof –
have \(\text{poly-deg } h \leq \text{Max } (\text{poly-deg } ' \text{ fst } ' \text{ set } ps)\)

proof (rule Max-ge)
  from finite-set show finite (poly-deg ' fst ' set ps) by (intro finite-imageI)
next
  from assms have poly-deg (fst (h, U)) ∈ poly-deg ' fst ' set ps by (intro imageI)
  thus poly-deg h ∈ poly-deg ' fst ' set ps by simp
qed
also have ... < Suc ... by simp
also have ... ≤ b ps 0
proof (rule b-zero)
  from assms show ps ≠ [] 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 bNil)
  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 ≠ {} by simp
  have b ps (Suc 0) = Suc (Max ?A) unfolding b-def
  proof (rule Least-equality)
    from finite ?A. (?A ≠ {}) have a ps ∈ ?A unfolding aP by (rule Min-in)
    with finite ?A. have a ps ≤ Max ?A by (rule Max-ge)
    hence a ps ≤ Suc (Max ?A) by simp
    moreover { fix h U
      assume (h, U) ∈ set ps
      with fin-X assms(1) have finite U by (rule valid-decompD-finite)
      moreover assume Suc 0 ≤ card U
      ultimately have U ≠ {} by auto
      with (h, U) ∈ set ps; have (h, U) ∈ set (ps+) by (simp add: pos-decomp-def)
      hence poly-deg (fst (h, U)) ∈ ?A by (intro imageI)
      hence poly-deg h ∈ ?A by (simp only: fst-conv)
      with finite ?A. have poly-deg h ≤ Max ?A by (rule Max-ge)
      hence poly-deg h < Suc (Max ?A) by simp
    }
    ultimately show a ps ≤ Suc (Max ?A) ∧ (∀ (h, U) ∈ set ps. Suc 0 ≤ card U → poly-deg h < Suc (Max ?A))
by blast

next

fix \( d \)

assume \( a \in ps \leq d \land (\forall (h, U) \in set ps. \text{Suc} \ 0 \leq \text{card} \ U \longrightarrow \text{poly-deg} \ h < d) \)

hence \( \text{rl}: \text{poly-deg} \ h < d \) if \( (h, U) \in set ps \) and \( 0 < \text{card} \ U \) for \( h \ U \) using that by auto

have \( \text{Max} \ ?A < d \) unfolding \( \text{Max-less-iff}[OF \ \text{finite} \ ?A; \ ?A \neq \{\}] \)

proof

fix \( d0 \)

assume \( d0 \in \text{poly-deg} \ \text{fst} \ \text{set} \ (ps_+) \)

then obtain \( h \ U \) where \( (h, U) \in \text{set} \ (ps_+) \) and \( d0: d0 = \text{poly-deg} \ h \) by auto

from this(1) have \( (h, U) \in \text{set} \ ps \) and \( \text{U} \neq \{\} \)

with \( \langle \text{U} \neq \{\} \rangle \) have \( 0 < \text{card} \ \text{U} \) by (simp add: card-gt-0-iff)

with \( \langle (h, U) \in \text{set} \ ps \rangle \) show \( d0 < d \) unfolding \( d0 \) by (rule rl)

qed

thus \( \text{Suc} \ \text{Max} \ ?A \leq d \) by simp

qed

with False show \( \text{?thesis} \) by simp

qed

corollary \( b\text{-one-gr} \):

assumes \( \text{valid-decomp} \ X \ ps \) and \( \text{standard-decomp} \ k \ ps \) and \( (h, U) \in \text{set} \ (ps_+) \)

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

proof

from assms(3) have \( ps_+ \neq [] \) by auto

with assms(1, 2) have eq: \( b \ ps \) (Suc 0) = \( \text{Suc} \ (\text{Max} \ (\text{poly-deg} \ \text{fst} \ \text{set} \ (ps_+))) \)

by (simp add: b-one)

have poly-deg \( h \leq \text{Max} \ (\text{poly-deg} \ \text{fst} \ \text{set} \ (ps_+)) \)

proof (rule Max-ge)

from finite-set show finite (poly-deg \( \text{fst} \ \text{set} \ (ps_+) \)) by (intro finite-imageI)

next

from assms(3) have poly-deg \( \text{fst} \ (h, U) \) \( \in \) poly-deg \( \text{fst} \ \text{set} \ (ps_+) \) by (intro imageI)

thus poly-deg \( h \in \text{poly-deg} \ \text{fst} \ \text{set} \ (ps_+) \) by simp

qed

also have \( \ldots < b \ ps \) (Suc 0) by (simp add: eq)

finally show \( \text{?thesis} \).

qed

lemma \( b\text{-card-X} \):

assumes \( \text{exact-decomp} \ m \ ps \) and \( \text{Suc} \ (\text{card} \ X) \leq i \)

shows \( b \ ps \ i = a \ ps \)

unfolding \( b\text{-def} \)

proof (rule Least-equality)

\{ 

fix \( h \ U \)

assume \( (h, U) \in \text{set} \ ps \)

\}

169
with `assms(1)` have \( U \subseteq X \) by (rule `exact-decompD`)

note `assms(2)`
also assume \( i \leq \text{card } U \)
finally have \( \text{card } X < \text{card } U \) by simp
with `fin-X` have \( \neg U \subseteq X \) by (auto dest: `card-mono leD`)
hence `False` using \( (U \subseteq X) \)
}

thus \( a \ ps \leq a \ ps \land (\forall (h, U) \in \text{set } ps. \ i \leq \text{card } U \rightarrow \text{poly-deg } h < a \ ps) \) by blast

```
``` lemma `lem-6-1-1`:
assumes `standard-decomp k ps` and `exact-decomp m ps` and `Suc 0 \leq i`
and \( i \leq \text{card } X \) and \( b \ ps \ (Suc i) \leq d \) and \( d < b \ ps i \)
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 []`
proof
  assume `ps_+ = []`
  hence \( b \ ps i = 0 \) using `assms(3)` by (rule `b-Nil`)
  with `assms(6)` show `False` by simp
```
``` qed

```
``` have `eq1`: \( b \ ps \ (Suc \ (\text{card } X)) = a \ ps \) using `assms(2)` `le-refl` by (rule `b-card-X`)
from `assms(1)` have `std`: `standard-decomp \( b \ ps \ (Suc \ (\text{card } X)) \) \ ps` unfolding `eq1` by (rule `a`)
from `assms(4)` have `Suc i \leq Suc \ (\text{card } X)` ..
hence \( b \ ps \ (Suc \ (\text{card } X)) \leq b \ ps \ (Suc i) \) by (rule `b-decreasing`)
hence `a \ ps \leq b \ ps \ (Suc i)` by (simp only: `eq1`)
```
``` have \( \exists h \ U. \ (h, U) \in \text{set } ps \land i \leq \text{card } U \land b \ ps i \leq Suc \ (\text{poly-deg } h) \)
```
``` proof (rule `ccontr`)
  assume `*: \exists h \ U. (h, U) \in \text{set } ps \land i \leq \text{card } U \land b \ ps i \leq Suc \ (\text{poly-deg } h)`
  note `a \ ps \leq b \ ps \ (Suc i)`
  also from `assms(5), 6` have `b \ ps \ (Suc i) < b \ ps i` by (rule `le-less-trans`)
  finally have `a \ ps \leq b \ ps i` .
hence `a \ ps \leq b \ ps i - 1` by simp
hence `b \ ps i \leq b \ ps i - 1`
proof (rule `b-le`)
```
``` fix `h \ U`
assume `(h, U) \in \text{set } ps \land i \leq \text{card } U`
show `\text{poly-deg } h < b \ ps i - 1`
proof (rule `ccontr`)
  assume `\neg \text{poly-deg } h < b \ ps i - 1`
  hence `b \ ps i \leq Suc \ (\text{poly-deg } h)` by simp
  with `* `(h, U) \in \text{set } ps \( i \leq \text{card } U\) show `False` by auto
```
``` qed
  ```
``` thus `False` using `(a \ ps < b \ ps i)` by `linarith`
```
``` qed
then obtain `h \ U` where `(h, U) \in \text{set } ps \land i \leq \text{card } U \land b \ ps i \leq Suc \ (\text{poly-deg } h)` by blast
```
``` 170
from `assms(3)` this(2) have $U \neq \{\}$ by `auto`
with $(h, U) \in \text{set } ps$ have $(h, U) \in \text{set } (ps_+)$ by `(simp add: pos-decomp-def)`
note std this
moreover have $b \in \text{set } ps$ $(\text{Suc } (\text{card } X)) \leq d$ unfolding eq1 using $(a \in \text{set } ps \leq b \in \text{set } ps$ $(\text{Suc } i))$ `assms(5)`
  by `(rule le-trans)`
moreover have $d \leq \text{poly-deg } h$
proof
  from `assms(6)` $d \leq \text{Suc } \text{(poly-deg } h)$: have $d < \text{Suc } \text{(poly-deg } h)$ by `(rule less-le-trans)`
  thus `?thesis` by `simp`
qed
ultimately obtain $h' \in \text{set } ps$ where $(h', U') \in \text{set } ps$ and $d$: `poly-deg h' = d` and `card U = \text{card } U'`
  by `(rule standard-decompE)`
from $(i \leq \text{card } U)$, this(3) have $i \leq \text{card } U'$ by `(rule le-trans)`
with `assms(3)` have $U' \neq \{\}$ by `auto`
with $(h', U') \in \text{set } ps$ have $(h', U') \in \text{set } (ps_+)$ by `(simp add: pos-decomp-def)`
moreover note $(\text{poly-deg } h' = d)$
moreover have $(\text{card } U' = i)$
moreover note $(\text{poly-deg } h' < b)$
moreover have $d < b$ by `(rule le-trans)`
proof (rule ccontr)
  assume $(\text{card } U' \neq i)$
  with $(i \leq \text{card } U')$ have `Suc i \leq \text{card } U'` by `simp`
  with $(h', U') \in \text{set } ps$ have $(h', U') \in \text{set } ps$ by `(rule le-trans)`
  with `assms(5)` show $(\text{False})$ by `(simp add: d)`
qed
ultimately show `?thesis` ..
qed

corollary `len-6-1-2`:
assumes standard-decomp $k$ `ps` and `exact-decomp` $0$ `ps` and `Suc` $0 \leq i$
  and $(i \leq \text{card } X)$ and $b \in \text{set } ps$ $(\text{Suc } i) \leq d$ and $d < b$ and $i$
obtains $h \in \text{set } ps$ where $(h', U') \in \text{set } (ps_+)$, $\text{poly-deg } h' = d$ = $(\{h, U\})$ and `card U = i`
proof
  from `assms` obtain $h \in \text{set } ps$ and $\text{poly-deg } h = d$ and `card U = i`
    by `(rule `len-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 $h' \in \text{set } ps$ where $(h', U') \in \text{set } (ps_+)$ and $\text{poly-deg } h' = d$ and $x$: `x = (h', U')`
    by `blast`
  note `assms(2)` $(h, U) \in \text{set } (ps_+)$: this(1)
  moreover have `poly-deg h = poly-deg h'` by `(simp only: `poly-deg h = d` `poly-deg h' = d)`
  qed

171
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 \(\text{card } U = \upsilon\))

qed

corollary lem-6-1-2:

assumes \(\text{standard-decomp } k \text{ ps and exact-decomp } 0 \text{ ps and Suc } 0 \leq i\)

and \(i \leq \text{card } X \text{ and } b \text{ ps } (\text{Suc } i) \leq d \text{ and } d < b \text{ ps } i\)

shows \(\text{card } \{(h', U') \in \text{set } (ps_+). \text{poly-deg } h' = d\} = 1 \text{ (is card } ?A = -)\)

and \(\{(h', U') \in \text{set } (ps_+). \text{poly-deg } h' = d \text{ and card } U' = i\} = \{(h', U') \in \text{set } (ps_+). \text{poly-deg } h' = d\}\)

(is \(?B = -)\)

and \(\text{card } \{(h', U') \in \text{set } (ps_+). \text{poly-deg } h' = d \text{ and card } U' = i\} = 1\)

proof –

from assms obtain \(h \text{ U where } ?A = \{(h, U)\}\) 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: \(?A = \{(h, U)\}\))

have \(?A = \{(h, U)\}\) by fact

also from \((h, U) \in ?A\) (\(\text{card } U = \upsilon\)) have \(\ldots \subseteq ?B\) by simp

finally show \(?A \subseteq ?B\)

qed blast

ultimately show \(\text{card } ?B = 1\) by simp

qed

corollary lem-6-1-3:

assumes \(\text{standard-decomp } k \text{ ps and exact-decomp } 0 \text{ ps and Suc } 0 \leq i\)

and \(i \leq \text{card } X \text{ and } (h, U) \in \text{set } (ps_+) \text{ and card } U = i\)

shows \(b \text{ ps } (\text{Suc } i) \leq \text{poly-deg } h\)

proof (rule ccontr)

define \(j\) where \(j = (\text{LEAST } j'. \ b \text{ ps } j' \leq \text{poly-deg } h)\)

assume \(\neg b \text{ ps } (\text{Suc } i) \leq \text{poly-deg } h\)

hence \(\text{poly-deg } h < b \text{ ps } (\text{Suc } i)\) by simp

from assms(2) le-refl have \(b \text{ ps } (\text{Suc } (\text{card } X)) = a \text{ ps}\) by (rule b-card-X)

also from - assms(5) have \(\ldots \leq \text{poly-deg } h\)

proof (rule standard-decompD)

from assms(1) show \(\text{standard-decomp } (a \text{ ps}) \text{ ps}\) by (rule a)

qed

finally have \(b \text{ ps } (\text{Suc } (\text{card } X)) \leq \text{poly-deg } h\)

hence \(j : b \text{ ps } j \leq \text{poly-deg } h\) unfolding \(j\)-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

172
hence $b \, ps (Suc \, i) \leq b \, ps \, j$ by (rule b-decreasing)

also have $\ldots \leq poly-deg \, h$ by fact

finally show False using $poly-deg \, h < b \, ps (Suc \, i)$ by simp

qed

hence eq: $Suc \, (j - 1) = j$ by simp

note assms(1, 2)

moreover from assms(3) have $Suc \, 0 \leq j - 1$

proof (rule le-trans)

from $(Suc \, i < j)$ show $i \leq j - 1$ by simp

qed

moreover have $j - 1 \leq card \, X$

proof

have $j \leq Suc (card \, X)$ unfolding $j$-def by (rule Least-le)

thus $\neg$thesis by simp

qed

ultimately obtain $h0 \, U0$

where eq1: $\{ (h', U') . \langle h', U' \rangle \in set (ps_+) \land poly-deg \, h' = poly-deg \, h \} = \{ (h0, U0) \}$

and $card \, U0 = j - 1$ by (rule lem-6-1-2)

from assms(5) have $(h, U) \in \{ (h', U') . \langle h', U' \rangle \in set (ps_+) \land poly-deg \, h' = poly-deg \, h \}$

by simp

hence $(h, U) \subseteq \{ (h0, U0) \}$ by (simp only: eq1)

hence $U = U0$ by simp

hence $card \, U = j - 1$ by (simp only: $\langle card \, U0 = j - 1 \rangle$)

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 :: $'a :: \{ \mbox{comm-ring-1}, \mbox{ring-no-zero-divisors} \} \times \{ x \ \mbox{set} \} \Rightarrow$ $'a :: \{ x \ \mbox{set} \}$

where shift-list $(h, \, U) \, x \, ps =$

$(\mbox{punit.monom-mult} \, 1 \, \langle \mbox{Poly-Mapping.single} \, x \, 1 \rangle \, h, \, U) \# (h, \, U - \{ x \}) \# \mbox{removeAll} \, (h, \, U) \, ps$

declare shift-list.simps[simp del]

lemma monomial-decomp-shift-list:

assumes monomial-decomp $ps$ and $hU \in set \, ps$

173
shows monomial-decomp (shift-list hU x ps)
proof –
  let ?x = Poly-Mapping.single x (1::nat)
obtain h U where hU: hU = (h, U) using prod.exhaust by blast
  with assms(2) have (h, U) ∈ set ps by simp
with assms(1) have 1: is-monomial h and 2: lcf h = 1 by (rule monomial-decompD)+
from this(1) have monomial (lcf h) (lpp h) = h by (rule punit.monomial-eq-itself)
moreover define t where t = lpp h
ultimately have h = monomial 1 t by (simp only: 2)
  by (simp-all add: punit.monom-mul-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 ∈ 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) ∈ set ps by simp
with assms(1) have 1: is-monomial h by (rule homogeneous-h)
hence homogeneous (punit.monom-mul 1 ?x h) by (simp only: homogeneous-monom-mul)
  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) ∈ set ps and x ∈ 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 ∈ P[X] and h ≠ 0 and U ⊆ X by (rule valid-decompD)+
moreover from this(1) have punit.monom-mul 1 ?x h ∈ P[X]
proof (intro Pows-closed-monom-mul PPs-closed-single)
  from x ∈ U | U ⊆ X show x ∈ X ..
qed
moreover from (U ⊆ X) have U - {x} ⊆ X by blast
ultimately show ?thesis
  using assms(1) | h ≠ 0 | by (simp add: valid-decomp-def punit.monom-mul-eq-zero-iff shift-list.simps)
qed

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

proof (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) ∈ set ((shift-list (h2, U2) x ps)+)
hence disj: (h, U) = ?p1 ∨ ((h, U) = ?p2 ∧ U2 - {x} ≠ {}) ∨ (h, U) ∈ set (ps+)
  by (auto simp: pos-decomp-def shift-list.simps split_if-split-asm)
from assms(7) have U2 ≠ {} by blast
with assms(3) have (h2, U2) ∈ set (ps+) by (simp add: pos-decomp-def)
with assms(1) have k-le: k ≤ poly-deg h2 by (rule standard-decompD)

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

fix d
assume k ≤ d and d ≤ poly-deg h
from disj obtain h' U' where 1: (h', U') ∈ set (?p1 # ps) and poly-deg h' = d
  and card U ≤ card U'
proof (elim disjE)
  assume (h, U) = ?p1
  hence h: h = punit.monom-mult 1 ?x h2 and U = U2 by simp-all
  from d ≤ poly-deg h have d ≤ poly-deg h2 ∨ poly-deg h = d
    by (cases h2 = 0) (auto simp: h poly-deg-monom-mult deg-pm-single)
  thus thesis
proof
  assume d ≤ poly-deg h2
  with assms(1) (h2, U2) ∈ set (ps+) have k ≤ d: obtain h' U'
    where (h', U') ∈ set ps and poly-deg h' = d and card U2 ≤ card U'
    by (rule standard-decompE)
  from this(1) have (h', U') ∈ set (?p1 # ps) by simp
  moreover note (poly-deg h' = d)
moreover from \( \text{card } U2 \leq \text{card } U \uparrow \) have \( \text{card } U \leq \text{card } U' \) by (simp only: \( U = U2 \))

ultimately show ?thesis ..

next

have \( (h, U) \in \text{set } (?p1 \neq ps) \) by (simp add: \( (h, U) = ?p1 \))

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 \( \{d \leq \text{poly-deg } h\} \) this(1) have \( d \leq \text{poly-deg } h2 \) by simp

with assms(1) \( (h2, U2) \in \text{set } (ps+) \) \( \langle k \leq d \rangle \) 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 \( (h', U') \in \text{set } (?p1 \neq ps) \) 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 \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 \( (h', U') \in \text{set } (?p1 \neq ps) \) by simp

thus ?thesis using \( \langle \text{poly-deg } h' = d \rangle \) \langle \text{card } U \leq \text{card } U' \rangle ..

qed

show \( \exists h' U'. (h', U') \in \text{set } (\text{shift-list } (h2, U2) x ps) \) and \( \text{poly-deg } h' = d \) and \( \text{card } U \leq \text{card } U' \)

proof (cases \( (h', U') = (h2, U2) \))

case True

hence \( h' = h2 \) and \( U' = U2 \) by simp-all

from assms(2, 6) have \( (h1, U1) \in \text{set } (\text{shift-list } (h2, U2) x ps) \) by (simp add: shift-list.simps)

moreover from \( \langle \text{poly-deg } h' = d \rangle \) have \( \text{poly-deg } h1 = d \) by (simp only: \( h' = h2 \) 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: \( U' = U2 \))

ultimately show ?thesis by blast

next

case False

with 1 have \( (h', U') \in \text{set } (\text{shift-list } (h2, U2) x ps) \) 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 (shift-list \((h, U) \times ps\))

proof
- let \(?p1 = (\text{punit} \cdot \text{monom-mult} \ 1 \ \text{Poly-Mapping.single} \ x \ 1) \ h, U\)
- 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 \(*: (h, U) \notin \text{set ps1}\)

by (meson split-list-first)

have count-list ps2 \((h, U) = 0\)

proof
(rule cocontr)
- from assms(1, 3) have \(h \neq 0\) by \((\text{rule valid-decompD})\)
  assume count-list ps2 \((h, U) \neq 0\)
  hence \(I < \text{count-list} \ ps \ (h, U)\) by \((\text{simp add: count-list-append})\)
  also have \(\ldots \leq \text{count-list} \ (\text{map cone} ps) \ (\text{cone} \ (h, U))\) by \((\text{fact count-list-map-ge})\)
  finally have \(I < \text{count-list} \ (\text{map cone} ps) \ (\text{cone} \ (h, U))\)
  with cone-decompD have cone \((h, U) = \{0\}\)

proof (rule direct-decomp-repeated-eq-zero)
- fix \(s\)
  assume \(s \in \text{set} \ (\text{map cone} ps)\)
  thus \(0 \in s\) by \((\text{auto intro: zero-in-cone})\)

qed (fact assms(2))

with tip-in-cone[of \(h\ \text{U}\)] have \(h = 0\) by simp

with \(h \neq 0\) show False ..

qed

hence \(**: (h, U) \notin \text{set ps2\ by \((\text{simp add: count-list-eq-0-iff})\)

have perm ps \((h, U) \# ps1 @ ps2\) \((\text{is perm - ps})\)
  by \((\text{rule perm-sym})\) \((\text{simp only: perm-append-Cons ps})\)

with assms(2) have cone-decomp T \(\text{ps by \((\text{rule cone-decomp-perm})\)}\)

hence direct-decomp T \((\text{map cone} \ ps)\) by \((\text{rule cone-decompD})\)

hence direct-decomp T \((\text{cone} \ (h, U) \# \text{map cone} \ (ps1 @ ps2))\) by simp

hence direct-decomp T \((\text{map cone} \ (ps1 @ ps2)) @ [\text{cone} \ ?p1, \text{cone} \ ?p2]\)

proof (rule direct-decomp-direct-decomp)

let \(?x = \text{Poly-Mapping.single} \ x \ (\text{Suc 0})\)

have direct-decomp \((\text{cone} \ (h, \text{insert} \ x \ (U - \{x\}))\)

[(\(\text{cone} \ (h, U - \{x\})\), \(\text{cone} \ (\text{monomial} \ (1::a) \ ?x \cdot h, \text{insert} \ x \ (U - \{x\}))\)]

by \((\text{rule direct-decomp-cone-insert})\) simp

with assms(4) show direct-decomp \((\text{cone} \ (h, U))\) \([\text{cone} \ ?p1, \text{cone} \ ?p2]\)

by \((\text{simp add: insert-absorb times-monomial-left direct-decomp-perm perm.swap})\)

qed

hence direct-decomp T \((\text{map cone} \ (ps1 @ ps2 @ [?p1, ?p2])\)) by simp

hence cone-decomp T \((ps1 @ ps2 @ [?p1, ?p2])\) by \((\text{rule cone-decompI})\)

moreover have perm \((ps1 @ ps2 @ [?p1, ?p2])\) \(\text{perm} \ (ps1 @ ps2)\)

proof
- have \(ps1 @ ps2 @ [?p1, ?p2] = (ps1 @ ps2) @ [?p1, ?p2]\) by simp
  also have perm \(\ldots (\text{ps1 @ ps2 @ [?p1, ?p2]) by \((\text{rule perm-append-swap})\)}\)
  also have \(\ldots = \ ?p1 \# \ ?p2 \# (ps1 @ ps2)\) by simp
ultimately have cone-decomp \( T(?p1 ≠ ?p2 (ps1 @ ps2)) \) by (rule cone-decomp-perm)
also from ∗∗∗ have \( ps1 @ ps2 = \text{removeAll}(h, U) ps \) by (simp add: remove1-append ps)
finally show \(?thesis\) by (simp only: shift-list.simps)
qed

10.9 Functions shift and exact

context
  fixes \( k \, m :: \text{nat} \)
begin

context
  fixes \( d :: \text{nat} \)
begin

definition shift2-inv :: ((('x ⇒₀ nat) ⇒₀ 'a::zero) × 'x set) list ⇒ bool where
  shift2-inv qs ←→ valid-decomp X qs ∧ standard-decomp k qs ∧ exact-decomp (Suc m) qs ∧
  (∀d0 < d. \( \{ q ∈ \text{set qs}. \text{poly-deg} (\text{fst q}) = d0 \land m < \text{card} (\text{snd q}) \} ≤ 1 \))

fun shift1-inv :: (((('x ⇒₀ nat) ⇒₀ 'a) × 'x set) list × ((('x ⇒₀ nat) ⇒₀ 'a::zero) × 'x set) set) ⇒ bool
  where shift1-inv (qs, B) ←→ B = \( \{ q ∈ \text{set qs}. \text{poly-deg} (\text{fst q}) = d \land m < \text{card} (\text{snd q}) \} \land \text{shift2-inv qs} \)

lemma shift2-invI:
  valid-decomp X qs ⇒ standard-decomp k qs ⇒ exact-decomp (Suc m) qs ⇒
  (∀d0. d0 < d ⇒ card \( \{ q ∈ \text{set qs}. \text{poly-deg} (\text{fst q}) = d0 \land m < \text{card} (\text{snd q}) \} \} ≤ 1 ⇒ shift2-inv qs
  by (simp add: shift2-inv-def)

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

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

declare shift1-inv.simps[simp del]

lemma shift1-inv-finite-snd:
  assumes shift1-inv (qs, B)
  shows finite B
  proof (rule finite-subset)
    from assms have B = {q ∈ set qs. poly-deg (fst q) = d ∧ m < card (snd q)}
    by (rule shift1-invD)
    also have ... ⊆ set qs by blast
    finally show B ⊆ set qs .
  qed (fact finite-set)

lemma shift1-inv-some-snd:
  assumes shift1-inv (qs, B) and 1 < card B and (h, U) = (SOME b. b ∈ B ∧ card (snd b) = Suc m)
  shows (h, U) ∈ B and (h, U) ∈ set qs and poly-deg h = d and card U = Suc m
  proof
    define A where A = {q ∈ B. card (snd q) = Suc m}
    define Y where Y = {q ∈ set qs. poly-deg (fst q) = d ∧ Suc m < card (snd q)}
    from assms(1) have B = {q ∈ set qs. poly-deg (fst q) = d ∧ m < card (snd q)}
      and inv2: shift2-inv qs by (rule shift1-invD)
    have B': B = A ∪ Y by (auto simp: B A-def Y-def)
    have finite A
      proof (rule finite-subset)
        show A ⊆ 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 ⊆ set qs unfolding Y-def by blast
      qed (fact finite-set)
    moreover have A ∩ Y = {} by (auto simp: A-def Y-def)
    ultimately have card (A ∪ Y) = card A + card Y by (rule card-Un-disjoint)
    with assms(2) have 1 < card A + card Y by (simp only: B')
    then card-le-Suc0-iff-eq[OF ⟨finite Y⟩]
    moreover have card Y ≤ 1 unfolding One-nat-def card-le-Suc0-iff-eq[OF finite Y]
    proof (intro ballI)
      fix q1 q2 :: (('x ⇒0 nat) ⇒0 'a) × 'x set
  179
obtain $h_1\ U_1$ where $q_1:q_1=(h_1,\ U_1)$ using prod.exhaust by blast
obtain $h_2\ U_2$ where $q_2:q_2=(h_2,\ U_2)$ using prod.exhaust by blast
assume $q_1\in Y$
  hence $(h_1,\ U_1)\in\ set\ qs$ and $\text{poly-deg}\ h_1=d$ and $\text{Suc}\ m<\ \text{card}\ U_1$ by 
  $(\text{simp-all add: } q_1\ Y\text{-def})$
assume $q_2\in Y$
  hence $(h_2,\ U_2)\in\ set\ qs$ and $\text{poly-deg}\ h_2=d$ and $\text{Suc}\ m<\ \text{card}\ U_2$ by 
  $(\text{simp-all add: } q_2\ Y\text{-def})$
from this(2) have $\text{poly-deg}\ h_1=\text{poly-deg}\ h_2$ by $(\text{simp only: } \text{poly-deg}\ h_1=d)$
from inv2 have exact-decomp $(\text{Suc}\ m)\ qs$ by (rule shift2-invD)
thus $q_1=q_2$ unfolding $q_1\ q_2$ 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 $(h,\ U)\in B\land\ \text{card}\ (\text{snd}\ (h,\ U))=\text{Suc}\ m$ unfolding assms(3)
proof (rule someI)
  from $\exists a\in A:\ \text{show}\ a\in B\land\ \text{card}\ (\text{snd}\ a)=\text{Suc}\ m$ 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 $(h,\ U)\in\ set\ qs$ and $\text{poly-deg}\ h=d$ by (simp-all add: $B$)
qed

lemma shift1-inv-preserved:
  assumes $\text{shift1-inv}\ (qs,\ B)$ and $1<\ \text{card}\ B$ and $(h,\ U)=(\text{SOME}\ b.\ b\in B\land\ \text{card}\ (\text{snd}\ b)=\text{Suc}\ m)$
  and $x=(\text{SOME}\ y.\ y\in U)$
  shows $\text{shift1-inv}\ (\text{shift-list}\ (h,\ U)\ x\ qs,\ B-\{(h,\ U)\})$
proof --
  let $?p_1=(\text{punit.monom-mult}\ 1\ (\text{Poly-Mapping}\ .single\ x\ 1)\ h,\ U)$
  let $?p_2=(h,\ U-\{x\})$
  let $?qs=\text{removeAll}(h,\ U)\ qs$
  let $?B=B-\{(h,\ U)\}$
from assms(1,\ 2,\ 3) have $(h,\ U)\in B$ and $(h,\ U)\in\ set\ qs$ and $\text{deg-h}\ h=d$
  and $\text{card-U}:\ \text{card}\ U=\text{Suc}\ m$ 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 $B: B=\{q\in\ set\ qs.\ \text{poly-deg}\ (\text{fst}\ q)=d\land m<\ \text{card}\ (\text{snd}\ q)\}$
  and inv2: $\text{shift2-inv}\ qs$ by (rule shift1-invD)+
from inv2 have valid-qs: valid-decomp $X\ qs$ by (rule shift2-invD)
hence $h\neq\ 0$ using $(h,\ U)\in\ set\ qs$ by (rule valid-decompD)
show $?thesis$
proof (intro shift1-invl shift2-invl)

180
show \( ?B = \{ q \in \text{set} (\text{shift-list} (h, U) x qs) \}. \text{poly-deg} (\text{fst} q) = d \land m < \text{card} (\text{snd} q) \} \) (is \(- = \) \( ?C \))
proof (rule Set.set-eqI)
fix \( b \)
show \( b \in ?B \rightleftharpoons b \in ?C \)
proof
assume \( b \in ?C \)
hence \( b \in \text{insert} ?p1 (\text{insert} ?p2 (\text{set} ?qs)) \) \( \land b1: \text{poly-deg} (\text{fst} b) = d \) 
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 \( h \neq 0 \) have \( \text{poly-deg} (\text{fst} b) = \text{Suc} d \) 
by (simp add: poly-deg-monom-malt deg-pm-single deg-h)
thus \( \text{thesis} \) by (simp add: \( b1 \))
next
assume \( b = ?p2 \)
hence \( \text{card} (\text{snd} b) = m \) by (simp add: card-Ux)
with \( b2 \) show \( \text{thesis} \) by simp
next
assume \( b \in \text{set} ?qs \)
with \( b1 \) \( b2 \) show \( \text{thesis} \) by (auto simp: \( B \))
qed
qed (auto simp: \( B \) shift-list.simps)
qed
next
from valid-qs \((h, U) \in \text{set} qs \) \( \forall x \in U \) show \( \text{valid-decomp} X (\text{shift-list} (h, U) x qs) \)
by (rule valid-decomp-shift-list)
next
from inv2 have \( \text{std} : \text{standard-decomp} k qs \) 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 \( \text{False} \) by simp
qed
then obtain \( h' \) \( U' \) where \( (h', U') \in B \) \( \land (h', U') \neq (h, U) \) by auto
from this(I) have \( (h', U') \in \text{set} qs \) \( \text{and} \) \( \text{poly-deg} h' = d \) \( \text{and} \) \( \text{Suc} m \leq \text{card} U' \)
by (simp-all add: \( B \))
note std this(I) \((h, U) \in \text{set} qs) \( \text{moreover from} : \text{poly-deg} h' = d \) have \( \text{poly-deg} h' = \text{poly-deg} h \) by (simp only: deg-h)
moreover from \( : \text{Suc} m \leq \text{card} U' \) 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 ext: exact-decomp (Suc m) qs by (rule shift2-invD)
show exact-decomp (Suc m) (shift-list (h, U) x qs)
proof (rule exact-decompI)
  fix h’ U’
  assume (h’, U’) ∈ set (shift-list (h, U) x qs)
thus h’ ∈ P[X]
proof (elim insertE)
  assume (h’, U’) = ?p1
  hence h’: h’ = punit.monom-mult 1 (Poly-Mapping.single x 1) h by simp
  from ext :(h, U) ∈ set qs have U ⊆ X by (rule exact-decompD)
  with ⟨x ∈ U⟩ have x ∈ X ..
  hence Poly-Mapping.single x 1 ∈ .[X] by (rule PPs-closed-single)
moreover from ext :(h, U) ∈ set qs have h ∈ 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 ext :(h, U) ∈ set qs have ... ∈ P[X] by (rule exact-decompD)
  finally show ?thesis .
next
  assume (h’, U’) ∈ set ?qs
  hence (h’, U’) ∈ set qs by simp
  with ext show ?thesis by (rule exact-decompD)
qed

from * show U’ ⊆ X
proof (elim insertE)
  assume (h’, U’) = ?p1
  hence U’ = U by simp
  also from ext :(h, U) ∈ set qs have ... ⊆ X by (rule exact-decompD)
  finally show ?thesis .
next
  assume (h’, U’) = ?p2
  hence U’ = U - {x} by simp
  also have ... ⊆ U by blast
  also from ext :(h, U) ∈ set qs have ... ⊆ X by (rule exact-decompD)
  finally show ?thesis .
next
  assume (h’, U’) ∈ set ?qs
  hence (h’, U’) ∈ set qs by simp
  with ext show ?thesis by (rule exact-decompD)
qed
next
fix h1 h2 U1 U2
assume (h1, U1) ∈ set (shift-list (h, U) x qs) and Suc m < card U1
hence (h1, U1) ∈ set qs using card-U card-Ux by (auto simp: shift-list.simps)
assume (h2, U2) ∈ set (shift-list (h, U) x qs) and Suc m < card U2
hence \((h_2, U_2) \in \text{set } qs\) using \(\text{card-}U\ \text{card-}Ux\) by (auto simp: shift-list.simps)

assume \(\text{poly-deg } h_1 = \text{poly-deg } h_2\)

from \(\text{exct}\) show \((h_1, U_1) = (h_2, U_2)\) by (rule exact-decompD) fact+

qed

next

fix \(d_0\)

assume \(d_0 < d\)

have finite \(\{ q \in \text{set } qs. \text{poly-deg } (\text{fst } q) = d_0 \land m < \text{card } (\text{snd } q)\}\) (is finite ?A) by auto

moreover have \(\{ q \in \text{set } (\text{shift-list } (h, U) \times qs). \text{poly-deg } (\text{fst } q) = d_0 \land m < \text{card } (\text{snd } q)\}\) \(\subseteq ?A\)

proof

fix \(q\)

assume \(q \in ?C\)

hence \(q = ?p_1 \lor q = ?p_2 \lor q \in \text{set } qs\) and \(1: \text{poly-deg } (\text{fst } q) = d_0\) and \(2: m < \text{card } (\text{snd } q)\)

by (simp-all add: shift-list.simps)

from this(1) show \(q \in ?A\) by (elim disjE)

assume \(q = ?p_1\)

with \(d \neq 0\) have \(d \leq \text{poly-deg } (\text{fst } q)\) by (simp add: poly-deg-monom-mult deg-h)

with \(d_0 < d\) show \(\text{thesis}\) by (simp only: 1)

next

assume \(q = ?p_2\)

hence \(d \leq \text{poly-deg } (\text{fst } q)\) by (simp add: deg-h)

with \(d_0 < d\) show \(\text{thesis}\) by (simp only: 1)

next

assume \(q \in \text{set } qs\)

with \(1\ \ 2\) show \(\text{thesis}\) by simp

qed

ultimately have \(\text{card } ?C \leq \text{card } ?A\) by (rule card-mono)

also from \(\text{inv2 \cdot } d_0 < d\) have \(\ldots \leq 1\) by (rule shift2-invD)

finally show \(\text{card } ?C \leq 1\).

qed

function \((\text{domintro})\) \(\text{shift1} :: \(((\text{'x } \Rightarrow_0 \text{nat}) \Rightarrow_0 \text{'a} \times \text{'x set}) \times ((\text{'x } \Rightarrow_0 \text{nat}) \Rightarrow_0 \text{'a} \times \text{'x set}) \Rightarrow \)

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

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

where \(\text{shift1 } (\text{qs}, B) = \)

(if \(I < \text{card } B\) then

let \((h, U) = \text{SOME } b. b \in B \land \text{card } (\text{snd } b) = \text{Suc } m; x = \text{SOME } y. y \in \)

183
lemma shift1-dom1:
  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 := (¬ ⇒ 'a) × set
    using - that(2)
  proof (rule less)
    from that(1) show ((qs0, B0), args) ∈ measure (card ∘ snd) by (simp add: args)
  qed
  from less(2) have inv: shift1-inv (qs, B) by (simp only: args)
  show ?case unfolding args
  proof (rule shift1-domintros)
    fix h U
    assume hU: (h, U) = (SOME b. b ∈ B ∧ card (snd b) = Suc m)
    define x where x = (SOME y. y ∈ U)
    assume Suc 0 < card B
    hence 1 < card B by simp
    have shift1-dom (shift-list (h, U) x qs, B − {(h, U)})
      proof (rule IH)
        from inv have finite B by (rule shift1-inv-finite-snd)
        moreover from inv (1 < card B) hU have (h, U) ∈ B by (rule shift1-inv-some-snd)
        ultimately show card (B − {(h, U)}) < card B by (rule card-Diff1-less)
      next
        from inv (1 < card B) hU x-def show shift1-inv (shift-list (h, U) x qs, (B
          − {(h, U)}))
          by (rule shift1-inv-preserved)
        qed
        thus shift1-dom (shift-list (SOME b. b ∈ B ∧ card (snd b) = Suc m) (SOME y. y ∈ U) qs,
          B − {SOME b. b ∈ B ∧ card (snd b) = Suc m}) by (simp add:
          hU x-def)
        qed
        qed
        qed

lemma shift1-induct [consumes 1, case-names base step]:
  assumes shift1-inv args
assumes $\exists y. y \in U \implies x \in U \implies \card(U - \{x\}) = m \implies P((\shift-list(h, U) x qs, B - \{(h, U)\})) (\shift1-shift-list(h, U) x qs, B - \{(h, U)\}))$) $$$ 
shows P args (\shift1-inv 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 \in B \land card(snd b) = Suc m)
  by (smt prod. exhaust)
define x where x = (SOME y. y \in U)
show ?case
proof (simp add: shift1-psimps[OF step.hyps(1)] flip: hU x-def del: One-nat-def, intro conjI impl)
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 \neq {} by auto
then obtain y where y \in U by blast
  hence x \in U unfolding x-def by (rule someI)
  with step.prems (1 < card B) hU x-def \langle finite U \rangle show P (qs, B) (\shift1 args)
proof (rule assms(3))
  from (finite U) x \in 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 \leq 1 by simp
  with step.prems show P (qs, B) (qs, B) by (rule assms(2))
qed
qed

lemma shift1-1:
assumes shift1-inv args and d0 \leq d
shows card \{q \in set (fst (\shift1 args))\}. poly-deg (fst q) = d0 \land m < card (snd
q\}) \leq 1
\text{using} \ \text{assms(1)}
\text{proof (induct args rule: shift1-induct)}
\text{case (base qs B)}
\text{from assms(2) have d0 < d \lor d0 = d by auto}
\text{thus ?case}
\text{proof}
\text{from base(1) have shift2-inv qs by (rule shift1-invD)}
\text{moreover assume d0 < d}
\text{ultimately show ?thesis unfolding fst-conv by (rule shift2-invD)}
\text{next}
\text{assume d0 = d}
\text{from base(1) have B = \{q \in set (fst (qs, B)). \text{poly-deg} (fst q) = d0 \land m < card (snd q)\}}
\text{unfolding fst-conv \cdot d0 = d by (rule shift1-invD)}
\text{with base(2) show ?thesis by simp}
\text{qed}
\text{qed}

\text{lemma shift1-2:}
\text{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)\}}
\text{proof (induct args rule: shift1-induct)}
\text{case (base qs B)}
\text{show ?case ..}
\text{next}
\text{case (step qs B h U x)}
\text{let ?x = Poly-Mapping.single x (1::nat)}
\text{let ?p1 = \{punit.monom-mult 1 ?x h\}}
\text{let ?A = \{q \in set qs. m < card (snd q)\}}
\text{from step(1-3) have card-U: \text{card} U = Suc m and (h, U) \in set qs by (rule shift1-inv-some-snd)}+
\text{from step(1) have shift2-inv qs by (rule shift1-invD)}
\text{hence valid-decomp X qs by (rule shift2-invD)}
\text{hence h \neq 0 using \\{h, U\} \in set qs by (rule valid-decompD)}
\text{have fin1: finite ?A by auto}
\text{hence fin2: finite (insert ?p1 ?A) by simp}
\text{from \\{h, U\} \in set qs have hU-in: (h, U) \in insert ?p1 ?A by (simp add: card-U)}
\text{have ?p1 \neq (h, U)}
\text{proof}
\text{assume ?p1 = (h, U)}
\text{hence lpp (punit.monom-mult 1 ?x h) = lpp h by simp}
\text{with \ \ h \neq 0 show False by (simp add: punit.lt-monom-mult monomial-0-iff)}
\text{qed}
\text{let ?qs = shift-list (h, U) x qs}
\text{have \{q \in set (fst (?qs, B - \{(h, U)\})) \land m < card (snd q)\} = (insert ?p1 ?A) - \{(h, U)\}}
using step(7) card-
U \{\?p1 \neq (h, U)\} by (fastforce simp: shift-list_simps)
also from fin2 h-U-in have card \ldots = card (insert \?p1 \?A) - 1 by (simp add: card-Diff-singleton-if)
also from fin1 have \ldots \leq Suc (card \?A) - 1 by (simp add: card-insert-if)
also have \ldots = 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.prems have cone-decomp T qs by (simp only: fst-conv)
moreover from step.hyps(1-3) have (h, U) \in set qs by (rule shift1-inv-some-snd)
ultimately have cone-decomp T (fst (shift-list (h, U) x qs, B - \{(h, U)\}))
unfolding fst-conv using step.hyps(6) by (rule cone-decomp-shift-list)
thus \?case by (rule step.hyps(8))
qed

lemma shift1-4:
shift1-inv args \implies
\begin{align*}
\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})))
\end{align*}
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 = \langle \text{punit.monom-mult} 1 \ ?x h, U \rangle
let \?qs = shift-list (h, U) x qs
from step(1) have B = \{q \in set qs. poly-deg (fst q) = d \land m < card (snd q)\}
and inv2: shift2-inv qs by (rule shift1-invD)+
from this(1) have B \subseteq set qs by auto
with step(2) have set qs \neq \{\} by auto
from finite-set have fin: finite (\text{poly-deg} \ ' \text{fst} \ ' \text{set} \ ?qs) by (intro finite-imageI)
have \text{Max} (\text{poly-deg} \ ' \text{fst} \ ' \text{set} \ (\text{fst} (\text{qs}, B))) \leq \text{Max} (\text{poly-deg} \ ' \text{fst} \ ' \text{set} \ (\text{fst} (\?qs, B - \{(h, U)\})))
unfolding fst-conv
proof (rule Max.boundedI)
from finite-set show finite (\text{poly-deg} \ ' \text{fst} \ ' \text{set} \ qs) by (intro finite-imageI)
nextrom \set qs \neq \{\} show poly-deg \ ' \text{fst} \ ' \text{set} qs \neq \{\} by simp

187
next
  fix a
  assume a ∈ poly-deg ' fst ' set qs
  then obtain q where q ∈ set qs and 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
finally show ?thesis 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 = (punst.monom-mult 1 (Poly-Mapping.single x 1) h, U)
  let ?qs = shift-list (h, U) x qs
  from step(1) have B = {q ∈ set qs. poly-deg (fst q) = d ∧ m < card (snd q)}
    and inv2: shift2-inv qs by (rule shift1-invD)+
  from this(1) have B ⊆ set qs by auto
  with step(2) have qs ≠ [] by auto
  moreover have fst (shift1 (?qs, B − {(h, U)})) ≠ []
    by (simp add: step-hyps(8) del: One-nat-def) (simp add: shift-list.simps)
  ultimately show ?case by simp
qed

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

lemma shift1-inv-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.prems have hom-decomp (fst (shift-list (h, U) x qs, B − {{h, U}}))
  unfolding fst-conv by (rule hom-decomp-shift-list)
  thus ?case by (rule step.hyps)
qed

end

lemma shift2-inv-preserved:
  assumes shift2-inv d qs
  shows shift2-inv (Suc d) (fst (shift1 (qs, {q ∈ 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

\[\text{shift2} \ c \ d \ qs =\]

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

by auto

termination proof

show \(\text{wf} \ (\text{measure} \ (\lambda (c, d). c - d))\) by (fact \(\text{wf-measure}\))

qed simp

lemma \text{shift2-1}: \text{shift2-inv} \ d \ qs \implies \text{shift2-inv} \ c \ (\text{shift2} \ c \ d \ qs)

proof (induct \ c \ d \ qs \ rule: \text{shift2.induct})

case IH: \((1 \ c \ d \ qs)\)

show \(?case\)

proof (subst \text{shift2.simps}, simp del: \text{shift2.simps}, intro conjI impI)

assume \(c \leq d\)

show \text{shift2-inv} \ c \ qs

proof (rule \text{shift2-invI})

from IH \((2)\) show \text{valid-decomp} \ X \ qs \ and \ \text{standard-decomp} \ k \ qs \ and \ \text{exact-decomp} \ (\text{Suc} \ m) \ qs

by (rule \text{shift2-invD})+

next

fix \(d0\)

assume \(d0 < c\)

hence \(d0 < d\) using \(c \leq d\) by (rule \text{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 \text{shift2-invD})

qed

next

assume \(\neg c \leq d\)

thus \text{shift2-inv} \ c \ (\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)})))))\)

proof (rule IH)

from IH \((2)\) show \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)})))))\)

by (rule \text{shift2-inv-preserved})

qed

qed

lemma \text{shift2-2}:

\text{shift2-inv} \ d \ qs \implies \text{card} \ \{q \in \text{set} \ (\text{shift2} \ c \ 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: \text{shift2.induct})

case IH: \((1 \ c \ d \ qs)\)

let \(?A = \{q \in \text{set} \ (\text{shift2} \ c \ (\text{Suc} \ d) \ (\text{fst} \ (\text{shift1} \ (qs, \ \{q \in \text{set} \ qs. \ \text{poly-deg} \ (\text{fst} q) = d \wedge m < \text{card} \ (\text{snd} q)})))))\). \(m < \text{card} \ (\text{snd} q)\)}
show ?case
proof (subst shift2.simps, simp del: shift2.simps, intro impI)
  assume \( \neg c \leq d \)
  hence card ?A \( \leq \) card \( \{ q \in \text{set} \ (\text{fst} \ (\text{shift1} (qs, \{ q \in \text{set} \ qs. \text{poly-deg} (\text{fst} q) = d \land m < \text{card} \ (\text{snd} q))))). \ m < \text{card} \ (\text{snd} q)\}\)
proof (rule IH)
  show shift2-inv (Suc d) (\{ q \in \text{set} \ (\text{fst} (\text{shift1} (qs, \{ q \in \text{set} \ qs. \text{poly-deg} (\text{fst} q) = d \land m < \text{card} \ (\text{snd} q))))). \ m < \text{card} \ (\text{snd} q)\})
    using IH(2) by (rule shift2-inv-preserved)
qed

also have \( \vdots \leq \) card \( \{ q \in \text{set} \ (\text{fst} (qs, \{ q \in \text{set} \ qs. \text{poly-deg} (\text{fst} q) = d \land m < \text{card} \ (\text{snd} q)))\}. \ m < \text{card} \ (\text{snd} q)\}
using refl IH(2) by (intro shift1-2 shift1-invI)
finally show card ?A \( \leq \) card \( \{ \text{fst-conv} q \in \text{set} \ qs. \ m < \text{card} \ (\text{snd} q)\}\)
by (simp only:)
qed

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

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

lemma shift2-5:
  \( \text{shift}2 \text{-inv} \ d \text{ d } \text{qs} \Rightarrow \text{shift}2 \ c \text{ d } \text{qs} = [] \\
\text{qs} = [] \)
proof (induct c d qs rule: shift2.induct)
  case IH: (1 c d qs)
  let ?args = (qs, \{ q \in \text{set} \text{ 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 \( \neg c \leq d \)
      hence \( \text{shift}2 \ c \ (\text{Suc} \ d) \ (\text{fst} \ (\text{shift}1 \ ?\text{args})) = [] \leftrightarrow \text{fst} \ (\text{shift}1 \ ?\text{args}) = [] \)
      proof (rule IH)
        from IH(2) show \( \text{shift}2 \text{-inv} \ (\text{Suc} \ d) \ (\text{fst} \ (\text{shift}1 \ ?\text{args})) \)
          by (rule shift2-inv-preserved)
      qed
      also from refl IH(2) have \( \ldots \leftrightarrow \text{fst} \ ?\text{args} = [] \) by (intro shift1-5 shift1-invI)
      finally show \( \text{shift}2 \ c \ (\text{Suc} \ d) \ (\text{fst} \ (\text{shift}1 \ ?\text{args})) = [] \leftrightarrow \text{qs} = [] \) by (simp only: fst-conv)
    qed
qed

lemma shift2-6:
  \( \text{shift}2 \text{-inv} \ d \text{ d } \text{qs} \Rightarrow \text{monomial-decomp} \text{ qs} \Rightarrow \text{monomial-decomp} \ (\text{shift}2 \ c \text{ d} \text{ qs}) \)
proof (induct c d qs rule: shift2.induct)
  case IH: (1 c d qs)
  let ?args = (qs, \{ q \in \text{set} \text{ 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 conjI impI)
      assume \( \neg c \leq d \)
      hence \( \text{shift}2 \ c \ (\text{Suc} \ d) \ (\text{fst} \ (\text{shift}1 \ ?\text{args})) = [] \leftrightarrow \text{fst} \ (\text{shift}1 \ ?\text{args}) = [] \)
      proof (rule IH)
        from IH(2) show \( \text{shift}2 \text{-inv} \ (\text{Suc} \ d) \ (\text{fst} \ (\text{shift}1 \ ?\text{args})) \)
          by (rule shift2-inv-preserved)
      qed
      moreover from refl IH(2) have \( \text{shift}1\text{-inv} \ d \ ?\text{args} \) by (rule shift1-invI)
      ultimately show \( \text{monomial-decomp} \ (\text{fst} \ (\text{shift}1 \ ?\text{args})) \) by simp
    qed
qed

192
lemma shift2-7:
shift2-inv d qs ⇒ hom-decomp qs ⇒ hom-decomp (shift2 c d qs)
proof (induct c d qs rule: shift2.induct)
case IH: (1 c d qs)
let ?args = (qs, {q ∈ set qs. poly-deg (fst q) = d ∧ m < card (snd q)})
show ?case
proof (subst shift2.simps, simp del: shift2.simps, intro 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 hom-decomp (fst ?args) by simp
ultimately show hom-decomp (fst (shift1 ?args)) by (rule shift1-7)
qed

definition shift :: ((('a ⇒ 0 nat) ⇒ 0 `'a × `'a set) list ⇒ ((('a ⇒ 0 nat) ⇒ 'a::{comm-ring-1,ring-no-zero-divisors}) × `'a set) list
where shift qs = shift2 (k + card {q ∈ set qs. m < card (snd q)}) k qs

lemma shift2-inv-init:
assumes valid-decomp X qs and standard-decomp k qs and exact-decomp (Suc m) qs
shows shift2-inv k qs
using assms
proof (rule shift2-invI)
fix d0
assume d0 < k
have {q ∈ set qs. poly-deg (fst q) = d0 ∧ m < card (snd q)} = {}
proof -
{ fix q
  assume q ∈ set qs
  obtain h U where q: q = (h, U) using prod.exhaust by blast
  assume poly-deg (fst q) = d0 and m < card (snd q)
  hence poly-deg h < k and m < card U using (d0 < k) by (simp-all add: q)
  from this(2) have U ≠ {} by auto
  with (q ∈ set qs) have (h, U) ∈ set (qs_4) by (simp add: q pos-decomp-def)
  with assms(2) have k ≤ poly-deg h by (rule standard-decompD)
  with (poly-deg h < k) have False by simp
  thus ?thesis by blast
qed
thus card {q ∈ set qs. poly-deg (fst q) = d0 ∧ m < card (snd q)} ≤ 1 by (simp only: card-empty)
qed
lemma shift:
  assumes valid-decomp X qs and standard-decomp k qs and exact-decomp (Suc m) qs
  shows valid-decomp X (shift qs) and standard-decomp k (shift qs) and exact-decomp m (shift qs)
proof –
  define c where c = card \{q \in set qs. m < card (snd q)\}
  define A where A = \{q \in set (shift qs). m < card (snd q)\}
  from assms have shift2-inv k qs by (rule shift2-inv-init)
  hence mv2: shift2-inv (k + c) (shift qs) and card A ≤ c
    unfolding shift-def c-def A-def by (rule shift2-1, rule shift2-2)
  from inv2 have fin: valid-decomp X (shift qs) and std: standard-decomp k (shift qs)
    and exct: exact-decomp (Suc m) (shift qs)
    by (rule shift2-invD)+
  show valid-decomp X (shift qs) and standard-decomp k (shift qs) by fact+
  have finite A by (auto simp: A-def)

  show exact-decomp m (shift qs)
proof (rule exact-decompI)
  fix h U
  assume (h, U) \in set (shift qs)
  with exct show h \in P[X] and U \subseteq X by (rule exact-decompD)+
  next
  fix h1 h2 U1 U2
  assume 1: (h1, U1) \in set (shift qs) and 2: (h2, U2) \in 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) \in set ((shift qs)+) by (simp add: pos-decomp-def)
  let ?C = \{q \in set (shift qs). poly-deg (fst q) = poly-deg h2 ∧ m < card (snd q)\}
  define B where B = \{q \in A. k ≤ poly-deg (fst q) ∧ poly-deg (fst q) ≤ poly-deg h2\}
    have Suc (poly-deg h2) - k ≤ card B
    proof –
      have B = (\bigcup d0∈\{k..poly-deg h2\}. \{q \in A. poly-deg (fst q) = d0\}) by (auto simp: B-def)
      also have card ... = (\sum d0=k..poly-deg h2. card \{q \in A. poly-deg (fst q) = d0\})
      proof (intro card-UN-disjoint ballI impI)
        fix d0
        from - finite A show finite \{q \in A. poly-deg (fst q) = d0\} by (rule finite-subset) blast
      next
      fix d0 d1 :: nat
      assume d0 ≠ d1
      thus \{q \in A. poly-deg (fst q) = d0\} ∩ \{q \in A. poly-deg (fst q) = d1\} = {} by blast
    by blast
qed (fact finite-atLeastAtMost)
also have \( \ldots \geq (\sum d_0 = k..\text{poly-deg } h_2. 1) \)
proof (rule sum-mono)
  fix d_0
  assume d_0 \in \{k..\text{poly-deg } h_2\}
  hence k \leq d_0 and d_0 \leq \text{poly-deg } h_2 by simp-all
with std \((h_2, U_2)\in\{ (\text{shift } qs) \}\):
  obtain h' \( U' \) where (h', U') \in \{ q \in A. \text{poly-deg } (\text{fst } q) = d_0 \}
from 5 this(3) have m < \text{card } U' by (rule less-le-trans)
with (h', U') \in \{ (\text{shift } qs) \}:
  have (h', U') \in \{ q \in A. \text{poly-deg } (\text{fst } q) = d_0 \} by simp-all
ultimately show 1 \leq \text{card } \{ q \in A. \text{poly-deg } (\text{fst } q) = d_0 \} by blast

also have \((\sum d_0 = k..\text{poly-deg } h_2. 1) = \text{Suc } (\text{poly-deg } h_2) - k by auto
finally show ?thesis .

also from \( \text{finite } A \) - have \( \ldots \leq \text{card } A \) by (rule card-mono) (auto simp: B-def)
also have \( \ldots \leq c \) by fact
finally have poly-deg h_2 < k + c by simp
with inv2 have card ?C \leq 1 by (rule shift2-ineqD)
have finite ?C by auto
moreover note (card ?C \leq 1)
moreover from 1 3 4 have \((h_1, U_1)\in ?C \) by simp
moreover from 2 5 have \((h_2, U_2)\in ?C \) by simp
ultimately show \((h_1, U_1) = (h_2, U_2) \) by (auto simp: card-Suc0-iff-eq)

lemma monomial-decomp-shift:
  assumes valid-decomp X qs and standard-decomp k qs and exact-decomp \((\text{Suc } m)\) 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 \((\text{Suc } m)\) qs
  and hom-decomp qs
shows hom-decomp (shift qs)
proof
  from assms(1, 2, 3) have shift2-inv k qs by (rule shift2-inv-init)
  thus ?thesis unfolding shift-def using assms(4) by (rule shift2-7)
qed

lemma cone-decomp-shift:
  assumes valid-decomp X qs and standard-decomp k qs and exact-decomp (Suc m) qs
  and cone-decomp T qs
  shows cone-decomp T (shift qs)
proof
  from assms(1, 2, 3) have shift2-inv k qs by (rule shift2-inv-init)
  thus ?thesis unfolding shift-def using assms(4) by (rule shift2-3)
qed

lemma Max-shift-ge:
  assumes valid-decomp X qs and standard-decomp k qs and exact-decomp (Suc m) qs
  shows Max (poly-deg ' fst ' set qs) ≤ Max (poly-deg ' fst ' set (shift qs))
proof
  from assms(1−3) have shift2-inv k qs by (rule shift2-inv-init)
  thus ?thesis unfolding shift-def by (rule shift2-4)
qed

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

end

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

lemma exact-aux:
  assumes valid-decomp X qs and standard-decomp k qs and exact-decomp m qs
  shows valid-decomp X (exact-aux k m qs) (is ?thesis1)
  and standard-decomp k (exact-aux k m qs) (is ?thesis2)
  and exact-decomp 0 (exact-aux k m qs) (is ?thesis3)
proof
  from assms have thesis1 ∧ thesis2 ∧ thesis3

196
proof (induct $m$ arbitrary: $qs$)
  case 0
  thus $?case$ by simp
next
  case (Suc $m$)
  let $?qs = \text{shift} k m \ qs$
  have valid-decomp $X$ (exact-aux $k \ m$ $?qs$) \and standard-decomp $k$ (exact-aux $k \ m$ $?qs$) \and exact-decomp 0 (exact-aux $k \ m$ $?qs$)
    proof (rule Suc)
      from Suc.prems show valid-decomp $X$ $?qs$ \and standard-decomp $k$ $?qs$ \and exact-decomp $m$ $?qs$
        by (rule shift)+
    qed
  thus $?case$ by simp
  qed
thus $?thesis1$ \and $?thesis2$ \and $?thesis3$ by simp-all
qed

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

lemma hom-decomp-exact-aux:
  assumes valid-decomp $X$ $qs$ \and standard-decomp $k$ $qs$ \and exact-decomp $m$ $qs$
  and hom-decomp $qs$
  shows hom-decomp (exact-aux $k \ m$ $qs$)
  using assms
proof (induct $m$ arbitrary: $qs$)
  case 0
  thus $?case$ by simp
next
\textbf{case} \((\text{Suc} \ m)\)
\begin{verbatim}
let \(?qs = \text{shift } m \ k \ qs\)
have \(\text{hom-decomp} (\text{exact-aux} \ k \ m \ ?qs)\)
proof (rule \text{Suc})
  show valid-decomp \(X \ ?qs\) \textbf{and} standard-decomp \(k \ ?qs\) \textbf{and} exact-decomp \(m \ ?qs\)
    using \text{Suc.prems}(1, 2, 3) by (rule \text{shift})+
next
from \text{Suc.prems} show hom-decomp \(?qs\) by (rule \text{hom-decomp-shift})
qed
\end{verbatim}
\textbf{thus} \(?\text{case}\) by simp
\textbf{qed}

\textbf{lemma} \textit{cone-decomp-exact-aux}: 
\begin{verbatim}
assumes valid-decomp \(X \ qs\) \textbf{and} standard-decomp \(k \ qs\) \textbf{and} exact-decomp \(m \ qs\)
and \(\text{cone-decomp} \ T \ qs\)
shows \(\text{cone-decomp} \ T (\text{exact-aux} \ k \ m \ qs)\)
using \text{assms}
proof (induct \(m\) arbitrary: \(qs\))
  case 0
  thus \(?\text{case}\) by simp
next
  case (\text{Suc} \(m\))
  let \(?qs = \text{shift } m \ k \ qs\)
  have \(\text{cone-decomp} \ T (\text{exact-aux} \ k \ m \ ?qs)\)
  proof (rule \text{Suc})
    show valid-decomp \(X \ ?qs\) \textbf{and} standard-decomp \(k \ ?qs\) \textbf{and} exact-decomp \(m \ ?qs\)
      using \text{Suc.prems}(1, 2, 3) by (rule \text{shift})+
    next
    from \text{Suc.prems} show \(\text{cone-decomp} \ T \ ?qs\) by (rule \text{cone-decomp-shift})
    qed
  thus \(?\text{case}\) by simp
  qed
\end{verbatim}
\textbf{lemma} \textit{Max-exact-aux-ge}: 
\begin{verbatim}
assumes valid-decomp \(X \ qs\) \textbf{and} standard-decomp \(k \ qs\) \textbf{and} exact-decomp \(m \ qs\)
shows \(\text{Max} (\text{poly-deg} \ ' \ \text{fst} \ ' \ \text{set} \ \text{qs}) \leq \text{Max} (\text{poly-deg} \ ' \ \text{fst} \ ' \ \text{set} (\text{exact-aux} \ k \ m \ qs))\)
using \text{assms}
proof (induct \(m\) arbitrary: \(qs\))
  case 0
  thus \(?\text{case}\) by simp
next
  case (\text{Suc} \(m\))
  let \(?qs = \text{shift } m \ k \ qs\)
  from \text{Suc.prems} have \(\text{Max} (\text{poly-deg} \ ' \ \text{fst} \ ' \ \text{set} \ \text{qs}) \leq \text{Max} (\text{poly-deg} \ ' \ \text{fst} \ ' \ \text{set} \ ?qs)\)
    by (rule \text{Max-shift-ge})
  also have \(\ldots \leq \text{Max} (\text{poly-deg} \ ' \ \text{fst} \ ' \ \text{set} (\text{exact-aux} \ k \ m \ ?qs))\)
  proof (rule \text{Suc})
\end{verbatim}

from Suc.prems show valid-decomp X qs and standard-decomp k qs and
exact-decomp m qs
by (rule shift)+
qed
finally show ?case by simp
qed

lemma exact-aux-Nil-iff:
assumes valid-decomp X qs and standard-decomp k qs and exact-decomp m qs
shows exact-aux k m qs = [] ↔ qs = []
using assms
proof (induct m arbitrary: qs)
case 0
thus ?case by simp
next
case (Suc m)
let ?qs = shift k m qs
have exact-aux k m ?qs = [] ↔ ?qs = []
proof (rule Suc)
from Suc.prems show valid-decomp X qs and standard-decomp k qs and
exact-decomp m qs
by (rule shift)+
qed
also from Suc.prems have ... qs = [] by (rule shift-Nil-iff)
finally show ?case by simp
qed

definition exact :: nat ⇒ ((’x ⇒₀ nat) ⇒₀ ’a) × ’x set) list ⇒
((’x ⇒₀ nat) ⇒₀ ’a::{comm-ring-1,ring-no-zero-divisors}) ×
’x set) list
where exact k qs = exact-aux k (card X) qs

lemma exact:
assumes valid-decomp X qs and standard-decomp k qs
shows valid-decomp X (exact k qs) (is ?thesis1)
and standard-decomp k (exact k qs) (is ?thesis2)
and exact-decomp 0 (exact k qs) (is ?thesis3)
proof −
from assms(1) le-refl have exact-decomp (card X) qs by (rule exact-decomp-card-X)
(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 \textit{assms}(1, 2) show \textit{thesis} unfolding \textit{exact-def} using \textit{assms}(3) by (rule monomial-decomp-exact-aux)

\textit{qed}

\textbf{lemma} \textit{hom-decomp-exact}:
\textit{assumes} \textit{valid-decomp} \textit{X qs} and \textit{standard-decomp} \textit{k qs} and \textit{hom-decomp} \textit{qs}
\textit{shows} \textit{hom-decomp} (\textit{exact k qs})
\textit{proof} –
\textit{from} \textit{assms}(1) \textit{le-refl} \textit{have} \textit{exact-decomp} (card \textit{X}) \textit{qs} by (rule \textit{exact-decomp-card-X})
\textit{with} \textit{assms}(1, 2) \textit{show} \textit{thesis} unfolding \textit{exact-def} using \textit{assms}(3) by (rule \textit{hom-decomp-exact-aux})

\textit{qed}

\textbf{lemma} \textit{cone-decomp-exact}:
\textit{assumes} \textit{valid-decomp} \textit{X qs} and \textit{standard-decomp} \textit{k qs} and \textit{cone-decomp} \textit{T qs}
\textit{shows} \textit{cone-decomp} \textit{T} (\textit{exact k qs})
\textit{proof} –
\textit{from} \textit{assms}(1) \textit{le-refl} \textit{have} \textit{exact-decomp} (card \textit{X}) \textit{qs} by (rule \textit{exact-decomp-card-X})
\textit{with} \textit{assms}(1, 2) \textit{show} \textit{thesis} unfolding \textit{exact-def} using \textit{assms}(3) by (rule \textit{cone-decomp-exact-aux})

\textit{qed}

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

\textit{qed}

\textbf{lemma} \textit{exact-Nil-iff}:
\textit{assumes} \textit{valid-decomp} \textit{X qs} and \textit{standard-decomp} \textit{k qs}
\textit{shows} \textit{exact k qs} = [] \iff \textit{qs} = []
\textit{proof} –
\textit{from} \textit{assms}(1) \textit{le-refl} \textit{have} \textit{exact-decomp} (card \textit{X}) \textit{qs} by (rule \textit{exact-decomp-card-X})
\textit{with} \textit{assms}(1, 2) \textit{show} \textit{thesis} unfolding \textit{exact-def} by (rule \textit{exact-aux-Nil-iff})

\textit{qed}

\textbf{corollary} \textit{b-zero-exact}:
\textit{assumes} \textit{valid-decomp} \textit{X qs} and \textit{standard-decomp} \textit{k qs} and \textit{qs} \neq []
\textit{shows} \textit{Suc (Max (poly-deg \ ' \fst\ ' \set \textit{qs}))} \leq b (\textit{exact k qs}) 0
\textit{proof} –
\textit{from} \textit{assms}(1, 2) \textit{have} \textit{Max (poly-deg \ ' \fst\ ' \set \textit{qs})} \leq \textit{Max (poly-deg \ ' \fst\ ' \set (\textit{exact k qs})}
\textit{by} (rule \textit{Max-exact-ge})
\textit{also have} \textit{Suc ...} \leq b (\textit{exact k qs}) 0
\textit{proof} (rule \textit{b-zero})
\textit{from} \textit{assms} \textit{show} \textit{exact k qs} \neq [] by (simp add: \textit{exact-Nil-iff})

\textit{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 \cdot P[X]) qs and exact-decomp 0 qs and \( \bigwedge g. (\bigwedge f. f \in F \implies \text{homogeneous } f) \implies g \in \text{punit.reduced-GB } F \implies \text{poly-deg } g \leq b \) qs 0

proof –
  let ?G = punit.reduced-GB F
  let ?S = lpp ' ?G
  let ?N = normal-form F \cdot P[X]
  define qs::((\cdot \Rightarrow ' a) \times \cdot) 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 \ldots = \{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-\text{-eq-UNIV-iff-reduced-GB-}\text{-eq-one-Polys}
       flip: ideal-\text{-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 \ldots \leq b \) qs 0 by (rule b-zero-exact)
    finally show ?thesis .
  qed
  ultimately show ?thesis ..
11 Dubé’s Degree-Bound for Homogeneous Gröbner Bases

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

context fixes n d :: nat
begin

function Dube-aux :: nat ⇒ nat where
  Dube-aux j = (if j + 2 < n then
                  2 + ((Dube-aux (j + 1)) choose 2) + (∑ i=j+3..n−1. (Dube-aux i) choose (Suc (i − j)))
                  else if j + 2 = n then d^2 + 2 * d else 2 * d)
    by pat-completeness auto

termination proof
  show wf (measure ((−)(−) n)) by (fact wf-measure)
  qed auto

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

lemma Dube-aux-ge-d: d ≤ Dube-aux j
proof (induct j rule: Dube-aux.induct)
  case step: (1 j)
  have j + 2 < n ∨ j + 2 = n ∨ n < j + 2 by auto
  show ?thesis
    proof (rule linorder-cases)
      assume *: j + 2 < n
      hence 1: d ≤ Dube-aux (j + 1)
        by (rule step.hyps)+
      show ?thesis
        proof (cases d ≤ 2)
          case True
          also from * have 2 ≤ Dube-aux j by simp
          finally show ?thesis.
        next
          case False

qed
hence $2 < d$ by simp
hence $2 < \text{Dube-aux} (j + 1)$ using 1 by (rule less-le-trans)
with - have Dube-aux $(j + 1) \leq \text{Dube-aux} (j + 1)$ choose 2 by (rule
upper-le-binomial) simp
also from * have ... $\leq \text{Dube-aux} j$ by simp
finally have Dube-aux $(j + 1) \leq \text{Dube-aux} j$.
with 1 show ?thesis by (rule le-trans)
qed
next
assume $j + 2 = n$
thus ?thesis by simp
next
assume $n < j + 2$
thus ?thesis by simp
qed
qed

**corollary** Dube-ge-d: $d \leq \text{Dube}$
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 $j = (1::a)$), rendering
the value inside the big brackets negative. This is also true without the
additional summand 2 we had to introduce in function local.Dube-aux to
correct another mistake found in [1]. Nonetheless, experiments carried out in
Mathematica still suggest that the short closed form is a valid upper bound
for local.Dube, even with the additional summand 2. So, with some effort it
might be possible to prove the theorem below; but in fact function local.Dube
gives typically much better (i.e. smaller) values for concrete values of $n$ and
d, so it is better to stick to local.Dube instead of the closed form anyway.
Asymptotically, as $n$ tends to infinity, local.Dube grows double exponentially,
too.

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

end

11.1 Hilbert Function and Hilbert Polynomial

context pm-powerprod
begin

context

fixes $X :: 'x set$
assumes fin-X: finite $X$

203
begin

lemma Hilbert-fun-cone-aux:
assumes \( h \in P[X] \) and \( h \neq 0 \) and \( U \subseteq X \) and homogeneous \( (h\cdot- \Rightarrow 0 \ 'a::field) \)
shows Hilbert-fun \( \langle \text{cone (} h, \ U \rangle \rangle \) \( z = \text{card \{t \in [U]. \ deg-pm t + poly-deg h = z\}} \)

proof
  from assms(2) have lpp h \in keys h by (rule punit.lt-in-keys)
  with assms(4) have degree[h][symmetric]: \( \text{deg-pm (} lpp h \rangle = \text{poly-deg h} \)
    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 \( \langle \text{cone (} h, \ U \rangle \rangle \) \( z = \text{card \{lpp ' (hom-deg-set z (cone (} h, \ U) \rangle) - \{0\})\}} \)
  using subspace-cone[of \( (h, \ U)\)] by (simp only: Hilbert-fun-alt)
also from assms(4) have \( lpp ' (\text{hom-deg-set z (cone (} h, \ U) \rangle) - \{0\}) = \{t \in lpp ' (cone (} h, \ U) \rangle - \{0\}. \ deg-pm t = z\} \)
    by (intro image-lt-hom-deg-set homomorphic-set-coneI)
also have \( \{t \in lpp ' (cone \( (h, \ U) \rangle) - \{0\}. \ deg-pm t = z\} = \langle \lambda t. t + lpp h \rangle \cdot \{t \in [U]. \ deg-pm t + poly-deg h = z\} \) (is \( \exists A \subseteq B\))
proof
  show \( \exists A \subseteq B\)
proof
    fix t
    assume t \in \( ?A\)
    hence t \in lpp ' (cone \( (h, \ U) \rangle) - \{0\}) \ and \ \text{deg-pm t = z} \ by simp-all
from this(1) obtain a where a \in cone \( (h, \ U) \rangle) - \{0\} \ and \ 2: t = lpp a ..
from this(1) have a \in cone \( (h, \ U) \rangle \ and \ a \neq 0 \ by simp-all
from this(1) obtain q where q \in P[U] \ and \ a = q \ast h \ by (rule coneE)
from \( a \neq 0\) have q \neq 0 \ by (auto simp: a)
  hence \( t = lpp q + lpp h \) using assms(2) unfolding 2 a by (rule lp-times)
    hence \( \text{deg-pm (} lpp q + poly-deg h = \text{deg-pm t by (simp add: deg-pm-plus degree-h)} \)
also have \( \ldots = z \) \ by fact
finally have \( \text{deg-pm (} lpp q + poly-deg h = z \).
moreover from \( q \in P[U] \) have \( lpp q \in [U] \) by (rule PPs-closed-lpp)
ultimately have \( lpp q \in \{t \in [U]. \ deg-pm t + poly-deg h = z\} \) by simp
moreover have \( t = lpp q + lpp h \) by (simp only: t)
ultimately show \( t \in \?B \) by (rule rev-image-eql)
qed
next
  show \( \exists B \subseteq \?A\)
proof
    fix t
    assume t \in \( ?B\)
    then obtain s where s \in \{t \in [U]. \ deg-pm t + poly-deg h = z\}
      and t1: \( t = s + lpp h \) ..
from this(1) have \( s \in [U] \) and \( 1: \deg-pm s + poly-deg h = z \) \ by simp-all
let \( \?q = \text{monomial \{1::'}a\} \rangle s \)
have \( \?q \neq 0 \) by (simp add: monomial-0-iff)
hence \( \forall q \neq 0 \) and \( \text{lpp} (\forall q \neq h) = \text{lpp} \forall q + \text{lpp} h \) using \( \forall h \neq 0 \)
by (rule times-not-zero, rule lp-times)
hence \( t = \text{lpp} (\forall q \neq h) \) by (simp add: \( t \) punit lt-monomial)
from \( s \in \{U\} \) have \( \forall q \in \text{P}[U] \) by (rule Polys-closed-monomial)
with refl have \( \forall q \neq h \in \text{cone} (h, U) \) by (rule coneI)
moreover from \( - \) assms(2) have \( \forall q \neq h \neq 0 \) by (rule times-not-zero) (simp add: monomial-0-iff)
ultimately have \( \forall q \neq h \in \text{cone} (h, U) - \{0\} \) by simp
hence \( t \in \text{lpp} \) (cone \( (h, U) - \{0\} \)) unfolding \( t \) by (rule imageI)
moreover have \( \text{deg-pm} t = \text{int} z \) by (simp add: \( t \) l) (simp add: \( \text{deg-pm-plus} \) \( \text{deg-h flip: 1} \))
ultimately show \( t \in \{A\} \) by simp
qed
also have \( \text{card} \ldots = \text{card} \{t \in \{U\}, \text{deg-pm} t + \text{poly-deg} h = z\} \) by (simp add: \( \text{card-image} \))
finally show \( \forall \text{thesis} . \)
qed

lemma Hilbert-fun-cone-empty:
assumes \( h \in \text{P}[X] \) and \( h \neq 0 \) and \( \text{homogeneous} \) (h::\( \Rightarrow \)0 'a::field)
shows Hilbert-fun (cone \( h, \{\}\)) \( z = (\text{if poly-deg} h = z \) then \( 1 \) 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 \( \ldots = (\text{if poly-deg} h = z \) then \( 1 \) else \( 0 \)) by simp
finally show \( \forall \text{thesis} . \)
qed

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

corollary Hilbert-fun-Polys:
  assumes X ≠ {} shows Hilbert-fun (P[X]::('- ⇒ 0 a::field) set) z = (z + (card X - 1)) choose (card X - 1)
proof –
  let ?one = 1::('a ⇒ 0 nat) ⇒ 0 'a
  have Hilbert-fun (P[X]::('- ⇒ 0 a) set) z = Hilbert-fun (cone (?one, X)) z by simp
  also have ... = (if poly-deg ?one ≤ 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)
  also have ... = (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 = (∑hU∈set ps. Hilbert-fun (cone hU) z)
proof –
  note fin-X
  moreover from assms(2, 1) have T ⊆ 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 = (∑s∈set (map cone ps). Hilbert-fun s z)
  proof (rule Hilbert-fun-direct-decomp)
    fix cn
    assume cn ∈ set (map cone ps)
    then obtain hU where hU ∈ set ps and cn: cn = cone hU unfolding set-map
    note this(1)
    moreover obtain h U where h U: hU = (h, U) using prod.exhaust by blast
    ultimately have (h, U) ∈ 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

206
also have \( \ldots = (\sum hU \in \text{set ps}. \ ((\lambda s. \text{Hilbert-fun } s \ z) \circ \text{cone}) \ hU) \) unfolding set-map using finite-set

proof (rule sum.reindex-nontrivial)
  fix \( hU1 \ hU2 \)
  assume \( hU1 \in \text{set ps} \) and \( hU2 \in \text{set ps} \) and \( hU1 \neq hU2 \)
  with dd have cone \( hU1 \cap \text{cone } hU2 = \{0\} \) using zero-in-cone by (rule direct-decomp-map-Int-zero)
  moreover assume \( \text{cone } hU1 = \text{cone } hU2 \)
  ultimately show \( \text{Hilbert-fun } (\text{cone } hU1) \ z = 0 \) by simp
qed

finally show \( \text{thesis} \) by simp
qed

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

lemma \( \text{poly-fun-Hilbert-poly}: \text{poly-fun } (\text{Hilbert-poly } b) \)
by (simp add: Hilbert-poly-def Let-def)

lemma \( \text{Hilbert-fun-eq-Hilbert-poly-plus-card}: \)
  assumes \( X \neq \{\} \) and valid-decomp \( X \ p s \) and hom-decomp \( p s \) and cone-decomp \( T \ p s \)
  and standard-decomp \( k \ p s \) and exact-decomp \( X \ 0 \ p s \) and \( b \ p s \ (\text{Suc } 0) \leq d \)
shows \( \text{int } (\text{Hilbert-fun } T \ d) = \text{card } \{h::\Rightarrow_0 \ a::\text{field}. \ (h, \{\}) \in \text{set } p s \ \land \ \text{poly-deg } h = d\} + \text{Hilbert-poly } (b \ p s) \ d \)
proof –
  define \( n \) where \( n = \text{card } X \)
  with assms(1) have \( \text{0 < } n \) using fin-X by (simp add: card-gt-0-iff)
  hence \( \text{1 \leq } n \) and \( \text{Suc } 0 \leq n \) by simp-all
  from pos-decomp-subset have eq0: \( \text{set } p s \ - \ \text{set } (p s_+) \cup \text{set } (p s_+) = \text{set } p s \) by blast
  have \( \text{set } p s - \ \text{set } (p s_+) \subseteq \text{set } p s \) by blast
  hence fin2: \( \text{finite } (\text{set } p s - \ \text{set } (p s_+)) \) using finite-set by (rule finite-subset)

have \( (\sum hU \in \text{set } p s - \ \text{set } (p s_+). \ \text{Hilbert-fun } (\text{cone } hU) \ d) = \)
  \( (\sum (h, U) \in \text{set } p s \ - \ \text{set } (p s_+). \ \text{if } \text{poly-deg } h = d \text{ then } 1 \text{ else } 0) \)
  using refl
proof (rule sum.cong)
  fix \( x \)
  assume \( x \in \text{set } p s \ - \ \text{set } (p s_+) \)
  moreover have \( U = \{\} \) and \( (h, U) \in \text{set } p s \) by (simp-all add: pos-decomp-def)
  ultimately have \( U = \{\} \) and \( (h, U) \in \text{set } p s \) by (rule valid-decompD)+
  moreover from assms(2) have \( h \in P[X] \) and \( h \neq 0 \) by (rule valid-decompD)+
  ultimately show \( \text{Hilbert-fun } (\text{cone } x) \ d = \text{case } x \text{ of } (h, U) \Rightarrow \text{if } \text{poly-deg } h = d \text{ then } 1 \text{ else } 0 \)
  using refl
\[ d = \text{then } 1 \text{ else } 0 \]

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

qed

also from fin2 have \( \ldots = (\sum (h, U) \in \{(h', U') \in \text{set } ps \setminus \text{set } (ps_+) \cdot \text{poly-deg } h' = d \}. 1) \)

by (rule sum_mono_neutral_cong_right) (auto split: if_splits)

also have \( \ldots = \text{card } \{ (h, U) \in \text{set } ps \setminus \text{set } (ps_+) \cdot \text{poly-deg } h = d \} \) by auto

also have \( \ldots = \text{card } \{ (h, U, \{ \}) \in \text{set } ps \land \text{poly-deg } h = d \} \) by (fact card_Diff_pos_decomp)

finally have eq1: \( (\sum h \in \text{set } ps \setminus \text{set } (ps_+) \cdot \text{Hilbert-fun } (\text{cone } h U) d) = \text{card } \{ (h, U, \{ \}) \in \text{set } ps \land \text{poly-deg } h = d \} \).

let \( \tilde{\phi} = \lambda a \cdot \delta \cdot (\text{int } d) - a + b \) choose \( b \)

have \( \text{int } (\sum h \in \text{set } (ps_+) \cdot \text{Hilbert-fun } (\text{cone } h U) d) = (\sum h \in \text{set } (ps_+) \cdot \text{int } (\text{Hilbert-fun } (\text{cone } h U) d)) \)

by (simp add: int_sum_prod_case_distrib)

also have \( \ldots = (\sum (h, U) \in (\bigcup i \in \{1..n\} \cdot \{(h, U) \in \text{set } (ps_+) \cdot \text{card } U = i\}). \tilde{\phi} \)

by (simp add: int_sum_prod_case_distrib)

(\text{poly-deg } h) (\text{card } U - 1))

proof (rule sum_cong)

show \( \text{set } (ps_+) = (\bigcup i \in \{1..n\} \cdot \{(h, U) \in \text{set } (ps_+) \land \text{card } U = i\}) \)

proof (rule Set.set_eqI, rule)

fix \( x \)

assume \( x \in \text{set } (ps_+) \)

moreover obtain \( h \in U \) where \( x = (h, U) \) using prod.exhaust by blast

ultimately have \( (h, U) \in \text{set } (ps_+) \) by simp

hence \( (h, U) \in \text{set } ps \land U \neq \{\} \) by (simp_all add: pos_decomp_def)

from fin_X assms(6) this(1) have \( U \subseteq X \) by (rule exact_decompD)

hence finite \( U \) using fin_X by (rule finite_subset)

with \( U \neq \{\} \) have \( 0 < \text{card } U \) by (simp add: card_gt_0_iff)

moreover from fin_X \( \langle U \subseteq X \rangle \) have \( \text{card } U \leq n \) unfolding n_def by (rule card_mono)

ultimately have \( \text{card } U \in \{1..n\} \) by simp

moreover from \( (h, U) \in \text{set } (ps_+) \) have \( (h, U) \in \{(h', U') \in \{h', U'\} \in \text{set } (ps_+) \land \text{card } U' = \text{card } U\} \)

by simp

ultimately show \( x \in (\bigcup i \in \{1..n\} \cdot \{(h, U) \in \text{set } (ps_+) \land \text{card } U = i\}) \) by (simp add: x)

qed blast

next

fix \( x \)

assume \( x \in (\bigcup i \in \{1..n\} \cdot \{(h, U) \in \text{set } (ps_+) \land \text{card } U = i\}) \)

then obtain \( j \) where \( j \in \{1..n\} \) and \( x \in \{(h, U) \in \text{set } (ps_+) \land \text{card } U = j\} \).

from this(2) obtain \( h \in U \) where \( (h, U) \in \text{set } (ps_+) \) and \( \text{card } U = j \) and \( x = (h, U) \) by blast

from fin_X assms(2, 5) this(1) have \( \text{poly-deg } h < b \) ps (Suc 0) by (rule b_one_gr)

also have \( \ldots \leq d \) by fact

finally have \( \text{poly-deg } h < d \).

hence int1: \( \text{int } (d - \text{poly-deg } h) = \text{int } d - \text{int } (\text{poly-deg } h) \) by simp

208
from \( \{ \text{card } U = j \} \) have \( 0 < \text{card } U \) by simp  

hence int: \( \{ \text{card } U - \text{Suc } 0 \} = \text{int} (\text{card } U) = 1 \) by simp  

from \((h, U) \in \text{set } (ps_{+})\) have \((h, U) \in \text{set } ps \) using pos-decomp-subset ..  

with \( \text{assms(2)} \) have \( h \in P[X]\) and \( h \neq 0 \) and \( U \subseteq X \) by (rule valid-decompD)+  

moreover from \( \text{assms(3)} \) \((h, U) \in \text{set } ps\) have homogeneous \( h \) by (rule hom-decompD)  

moreover from \( 0 < \text{card } U \) have \( U \neq \{ \} \) by auto  

ultimately have Hilbert-fun \( \langle \text{cone } (h, U) \rangle d \rangle = \)  

(if poly-deg \( h \leq d \) then \( d - \text{poly-deg } h + (\text{card } U - 1) \) choose (card \( U - 1 \) ) else \( 0 \) )  

by (rule Hilbert-fun-cone-nonempty)  

also from \( \text{poly-deg } h < d \) have \( \ldots = (d - \text{poly-deg } h + (\text{card } U - 1)) \choose \) \( (\text{card } U - 1) \) by simp  

finally  

have \( \text{int } \langle \text{Hilbert-fun } \langle \text{cone } (h, U) \rangle d \rangle \rangle = \)  

(\( \text{int } d - \text{int } (\text{poly-deg } h) + (\text{int } (\text{card } U - 1)) \) \choose (\text{card } U - 1))  

by (simp add: \( x \))  

qed  

also have \( \ldots = (\sum_{j=1..n}. \sum (h, U)\in\{ (h', U') \in \text{set } (ps_{+}) \ldots \} \)  

proof (intro sum.UNION-disjoint ballI)  

fix \( j \)  

have \( \{ (h, U)\in\{ (h', U') \in \text{set } (ps_{+})\} \text{and } \text{card } U = j \} \subseteq \text{set } (ps_{+}) \) by blast  

thus \( \text{finite } \{ (h, U). (h, U) \in \text{set } (ps_{+}) \text{and } \text{card } U = j \} \) using finite-set by (rule finite-subset)  

qed blast+  

also from \( \text{refl} \) have \( \ldots = (\sum_{j=1..n}. \text{?f } (b \text{ ps } (\text{Suc } j)) j - ?f (b \text{ ps } j) j) \)  

proof (rule sum.cong)  

fix \( j \)  

assume \( j \in \{ 1..n \} \)  

hence Suc \( 0 \leq j \text{ and } 0 < j \text{ and } j \leq n \) by simp-all  

from fin-X this(1) have \( b \text{ ps } j \leq b \text{ ps } (\text{Suc } 0) \) by (rule b-decreasing)  

also have \( \ldots \leq d \) by fact  

finally have \( b \text{ ps } j \leq d \).  

from fin-X have \( b \text{ ps } (\text{Suc } j) \leq b \text{ ps } j \) by (rule b-decreasing) simp  

hence \( b \text{ ps } (\text{Suc } j) \leq d \) using \( b \text{ ps } j \leq d \) by (rule le-trans)  

from \( 0 < j \) have \( \text{int-j: } \text{int } (j - \text{Suc } 0) = \text{int } j - 1 \) by simp  

have \( \{ (\sum (h, U)\in\{ (h', U') \in \text{set } (ps_{+}) \text{and } \text{card } U' = j \} \) \}  

proof - refl  

show \( \{ (h', U'). (h', U') \in \text{set } (ps_{+}) \text{and } \text{card } U' = j \} = \)
\[(\bigcup d0 \in \{b\; \text{ps}\; (Suc\; j)\}\; \text{int}\; (b\; \text{ps}\; j) - 1\}. \{(h', U') \cdot (h', U') \in \text{set}\; (ps_+)\} \wedge \text{int}\; (\text{poly-deg}\; h') = d0 \wedge \text{card}\; U' = j\}\]

proof (rule Set.set-eqI, rule)

\[
\text{fix } x
\]

\[
\text{assume } x \in \{(h', U'), (h', U') \in \text{set}\; (ps_+) \wedge \text{card}\; U' = j\}
\]

moreover obtain \(h\; U\) where \(x = (h, U)\) using prod.exhaust by blast

ultimately have \((h, U) \in \text{set}\; (ps_+) \wedge \text{card}\; U' = \text{card}\; U\)

using \((h, U) \in \text{set}\; (ps_+)\)

ultimately show \(x \in (\bigcup d0 \in \{b\; \text{ps}\; (Suc\; j)\}\; \text{int}\; (b\; \text{ps}\; 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 (card\; U = j))

qed

also have \(\ldots = (\sum d0 = b\; \text{ps}\; (Suc\; j)\; \text{int}\; (b\; \text{ps}\; j) - 1. \sum (h, U) \in \{(h', U'), (h', U') \in \text{set}\; (ps_+) \wedge \text{poly-deg}\; h' = d0 \wedge \text{card}\; U' = j\}) \subseteq\)

\(\text{set}\; (ps_+)\)

by blast

thus \(\text{finite}\; \{(h', U'), (h', U') \in \text{set}\; (ps_+) \wedge \text{poly-deg}\; h' = d0 \wedge \text{card}\; U' = j\})\)

using finite-set by (rule finite-subset)

qed blast+

also from refl have \(\ldots = (\sum d0 = b\; \text{ps}\; (Suc\; j)\; \text{int}\; (b\; \text{ps}\; j) - 1. \if\; \text{def}\; \text{(card}\; U - 1))\)

proof (intro sum.UNION-disjoint ballI)

fix d0::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 \(\text{finite}\; \{(h', U'), (h', U') \in \text{set}\; (ps_+) \wedge \text{poly-deg}\; h' = d0 \wedge \text{card}\; U' = j\})\)

using finite-set by (rule finite-subset)

qed blast+

proof (rule sum.cong)

fix d0

assume \(d0 \in \{b\; \text{ps}\; (Suc\; j)\}\; \text{int}\; (b\; \text{ps}\; j) - 1\}

hence \(b\; \text{ps}\; (Suc\; j) \leq d0\) and \(d0 < \text{int}\; (b\; \text{ps}\; j)\) by simp-all

hence \(b\; \text{ps}\; (Suc\; j) \leq \text{nat}\; d0\) and \(\text{nat}\; d0 < b\; \text{ps}\; j\) by simp-all

have \(\sum (h, U) \in \{(h', U'), (h', U') \in \text{set}\; (ps_+) \wedge \text{poly-deg}\; h' = d0 \wedge \text{card}\; U' = j\}. \if\; \text{(poly-deg}\; h)\; \text{(card}\; U - 1))\)

= \(\sum (h, U) \in \{(h', U'), (h', U') \in \text{set}\; (ps_+) \wedge \text{poly-deg}\; h' = d0 \wedge \text{card}\; U' = j\}. \if\; d0\; (j - 1))\)

using refl by (rule sum.cong) auto

210
also have ... = \card \{(h', U'). (h', U') \in \text{set} (\text{ps}_+) \land \text{poly-deg} h' = \text{nat} d0 \\
\land \card U' = j\} \ast \text{if} d0 (j - 1) \\
\text{using} \ \lnot b \ \text{ps} (\text{Suc} j) \leq d0; \text{by} (\text{simp add: int-eq-iff})
also have ... = \text{if} d0 (j - 1) \\
\text{using} \ \text{fin-X assms(5, 6)} (\text{Suc} 0 \leq j) \langle j \leq n \rangle \ \lnot b \ \text{ps} (\text{Suc} j) \leq \text{nat} d0 \langle \text{nat} d0 < b \ \text{ps} j\rangle \\
\text{by} (\text{simp only: n-def lem-6-1-2'(3)})
finally show \(\sum (h, U) \in \{(h', U'). (h', U') \in \text{set} (\text{ps}_+) \land \text{poly-deg} h' = d0 \\
\land \card U' = j\}.
\text{if} (\text{poly-deg} h) (\card U = 1)) = \text{if} d0 (j - 1) .
\text{qed}
also have ... = \(\sum d0 \in \{0..\text{int} d - \{b \ \text{ps} (\text{Suc} j)\}\} - \{0..\text{int} d - b \ \text{ps} j\}. \ \text{d0} \\
+ \ \text{int} (j - 1) \ \text{gchoose} (j - 1)) \\
\text{using} \ \text{refl}
\text{proof} (\text{rule sum.cong})
\text{have} (\neg) (\text{int} d) \langle \{b \ \text{ps} (\text{Suc} j)\} - \{0..\text{int} d - \{b \ \text{ps} j\}\} = \{0..\text{int} d - \{b \ \text{ps} (\text{Suc} j)\}\} \\
- \{0..\text{int} d - \{b \ \text{ps} j\}\} \\
\text{by} (\text{simp only: image-diff-atLeastAtMost})
\text{also have} ... = \{0..\text{int} d - \{b \ \text{ps} (\text{Suc} j)\}\} - \{0..\text{int} d - \{b \ \text{ps} j\}\} \\
\text{proof} - \\
\text{from} \ \lnot b \ \text{ps} j \leq d; \text{have} \ \text{int} (b \ \text{ps} j) - 1 \leq \text{int} d \ \text{by} \ \text{simp} \\
\text{thus} \ \text{?thesis} \ \text{by} \ \text{auto}
\text{qed}
\text{finally show} (\neg) (\text{int} d) \langle \{b \ \text{ps} (\text{Suc} j)\} - \{0..\text{int} d - \{b \ \text{ps} j\}\} = \{0..\text{int} d - \{b \ \text{ps} (\text{Suc} j)\}\} - \{0..\text{int} d - \{b \ \text{ps} j\}\} .
\text{qed}
also have ... = \(\sum d0 \in 0..\text{int} d - \{b \ \text{ps} (\text{Suc} j)\}. d0 + \ \text{int} (j - 1) \ \text{gchoose} (j - 1)) \\
\text{by} (\text{rule sum-diff}) (\text{auto simp:} \ \lnot b \ \text{ps} (\text{Suc} j) \leq b \ \text{ps} j) \\
\text{also from} \ \lnot b \ \text{ps} (\text{Suc} j) \leq d; \ \text{b ps j} \leq d; \text{have} ... = \text{if} (b \ \text{ps} (\text{Suc} j)) j - \text{if} (b \ \text{ps} j) j \\
\text{by} (\text{simp add: gchoose-rising-sum, simp add: int-j ac-simps} \langle 0 < j \rangle) \\
\text{finally show} \ \sum (h, U) \in \{(h', U'). (h', U') \in \text{set} (\text{ps}_+) \land \card U' = j\}. \ \text{if} (\text{poly-deg} h) (\card U = 1)) \\
\text{if} (b \ \text{ps} (\text{Suc} j)) j - \text{if} (b \ \text{ps} j) j .
\text{qed}
also have ... = \(\sum j = 1..n. \ \text{if} (b \ \text{ps} (\text{Suc} j)) j - (\sum j = 1..n. \ \text{if} (b \ \text{ps} (Suc j)) j) \\
\text{by} (\text{fact sum-subtractf})
also have ... = \text{if} (b \ \text{ps} (\text{Suc} n)) n + (\sum j = 1..n - 1. \ \text{if} (b \ \text{ps} (\text{Suc} j)) j - (\sum j = 1..n. \ \text{if} (b \ \text{ps} j) j) \\
\text{by} (\text{simp only: sum-tail-nat}[\text{OF}\ \langle 0 < n\rangle \langle 1 \leq n\rangle])
\text{211}
also have \( \ldots = \#f (b\ ps (Suc\ n)) n - \#f (b\ ps 1) 1 + \\
(\sum_{j=1..n-1} \#f (b\ ps (Suc\ j)) j) - (\sum_{j=1..n-1} \#f (b\ ps (Suc\ j)) (Suc\ j)) \)
by (simp only: sum.atLeast-Suc-atMost[OF \( f \leq n \) \( f \leq n \)])
also have \( \ldots = \#f (b\ ps (Suc\ n)) n - \#f (b\ ps 1) 1 - \\
(\sum_{j=1..n-1} \#f (b\ ps (Suc\ j)) (Suc\ j)) - (\#f (b\ ps (Suc\ j)) j) \)
by (simp only: sum-subtractf)
also have \( \ldots = \#f (b\ ps (Suc\ n)) n - 1 - ((int\ d - b\ ps (Suc\ 0))\ \text{gchoose}\ (Suc\ 0)) \)
proof
  have \( \#f (b\ ps 1) 1 = 1 + ((int\ d - b\ ps (Suc\ 0))\ \text{gchoose}\ (Suc\ 0)) \)
  by (simp add: plus-Suc-gbinomial)
moreover from refl have \( (\sum_{j=1..n-1} \#f (b\ ps (Suc\ j)) (Suc\ j)) - \#f (b\ ps (Suc\ j)) j = \\
(\sum_{j=1..n-1} (int\ d - b\ ps (Suc\ j) + j)\ \text{gchoose}\ (Suc\ j)) \)
by (rule sum.cong) (simp add: plus-Suc-gbinomial)
ultimately show \#thesis by (simp only:)
qed
also have \( \ldots = \#f (b\ ps (Suc\ n)) n - 1 - (\sum_{j=0..n-1} (int\ d - b\ ps (Suc\ j) + j)\ \text{gchoose}\ (Suc\ j)) \)
by (simp only: sum.atLeast-Suc-atMost[OF \( f = 0 \) \( f = 0 \)], simp)
also have \( \ldots = \#f (b\ ps (Suc\ n)) n - 1 - (\sum_{j=Suc\ 0..n-1} (int\ d - b\ ps\ j + j - 1)\ \text{gchoose}\ j) \)
by (simp only: sum.shift-bounds-cl-Suc-ivl, simp add: ac-simps)
also have \( \ldots = \text{Hilbert-poly}\ (b\ ps)\ d\ \text{using}\ \emptyset < n\ \text{by}\ (\text{simp add: Hilbert-poly-def Let-def n-def}) \)
finally have eq2: \( \text{int} \ (\sum h\Uset (ps_+)) \cdot \text{Hilbert-fun}\ (\text{cone}\ h\Uset)\ d = \text{Hilbert-poly}\ (b\ ps)\ (\text{int}\ d) \).
from assms(4, 2, 3) have Hilbert-fun T d = \( \sum h\Uset \in\ set\ ps.\ \text{Hilbert-fun}\ (\text{cone}\ h\Uset)\ d \)
by (rule Hilbert-fun-cone-decomp)
also have \( \ldots = (\sum h\Uset \in\ set\ ps - \text{set}\ (ps_+))\cup\text{set}\ (ps_+).\ \text{Hilbert-fun}\ (\text{cone}\ h\Uset)\ d \)
by (simp only: eq1)
also have \( \ldots = (\sum h\Uset \in\ set\ ps - \text{set}\ (ps_+).\ \text{Hilbert-fun}\ (\text{cone}\ h\Uset)\ d + (\sum h\Uset \in\ set\ (ps_+).\ \text{Hilbert-fun}\ (\text{cone}\ h\Uset)\ d) \)
using fin2 finite-set by (rule sum.union-disjoint) blast
also have \( \ldots = \text{card}\ \{h. (\{\}) \in\ set\ ps\wedge\text{poly-deg}\ h = d\} + (\sum h\Uset \in\ set\ (ps_+).\ \text{Hilbert-fun}\ (\text{cone}\ h\Uset)\ d) \)
by (simp only: eq1)
also have \( \text{int} \ldots = \text{card}\ \{h. (\{\}) \in\ set\ ps\wedge\text{poly-deg}\ h = d\} + \text{Hilbert-poly}\ (b\ ps)\ d \)
by (simp only: eq2 int-plus)
finally show \#thesis .
qed

corollary Hilbert-fun-eq-Hilbert-poly:
assumes $X \neq \{\}$ and valid-decomp $X$ $ps$ and hom-decomp $ps$ and cone-decomp $T$ $ps$

and standard-decomp $k$ $ps$ and exact-decomp $X$ $0$ $ps$ and $b$ $ps$ $0 \leq d$

shows $\text{int} (\text{Hilbert-fun} (T :: (- \Rightarrow 0) \ a ::= \text{field}) \ \text{set}) \ d) = \text{Hilbert-poly} (b \ ps) \ d$

proof –

from fin-X have $b$ $ps$ $(\text{Suc} \ 0) \leq b$ $ps$ $0$ using $\text{le0}$ by (rule $b$-decreasing)
also have $\ldots \leq d$ by fact

finally have $b$ $ps$ $(\text{Suc} \ 0) \leq d$.

with assms($1-6$) have $\text{int} (\text{Hilbert-fun} T \ d) =$

$\text{int} (\text{card} \ \{(h, \{\}) \in \text{set} \ ps \land \text{poly-deg} \ h = d\}) + \text{Hilbert-poly} (b \ ps) (\text{int} \ d)$

by (rule Hilbert-fun-eq-Hilbert-poly-plus-card)

also have $\ldots = \text{Hilbert-poly} (b \ ps) (\text{int} \ d)$

proof –

have $eq: \{(h, \{\}) \in \text{set} \ ps \land \text{poly-deg} \ h = d\} = \{\}$

proof –

\{ 
fix $h$
assume $(h, \{\}) \in \text{set} \ ps$ and $\text{poly-deg} \ h = d$
from fin-X this(1) $\text{le0}$ have $\text{poly-deg} \ h < b$ $ps$ $0$ by (rule $b$)

with assms($7$) have $\text{False}$ by (simp add: $\langle$poly-deg $h = d\rangle$)
\}

thus $?\text{thesis}$ by blast
qed

show $?\text{thesis}$ by (simp add: $eq$)

qed

finally show $?\text{thesis}$.

qed

11.2 Dubé’s Bound

context

fixes $f :: (\ex 0 \text{nat}) \Rightarrow 0 \ a ::= \text{field}$
fixes $F$

assumes $n$-gr-1: $1 < \text{card} \ X$ and fin-F: finite $F$ and F-sub: $F \subseteq P[X]$ and

f-in: $f \in F$

and hom-F: $\bigwedge f' . f' \in F \Rightarrow \text{homogeneous} \ f'$ and

f-max: $\bigwedge f' . f' \in F \Rightarrow \text{poly-deg} \ f' \leq \text{poly-deg} \ f$

and d-gr-0: $0 < \text{poly-deg} \ f$ and ideal-f-neq: ideal $\{f\} \neq \text{ideal} \ F$

begin

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

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

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

213
lemma f-not-0: \( f \neq 0 \)
using d-gr-0 by auto

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

lemma n-gr-0: \( 0 < n \)
using \( 1 < n \) by simp

corollary int-n-minus-1 [simp]: \( \text{int } (n - \text{Suc } 0) = \text{int } n - 1 \)
using n-gr-0 by simp

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

lemma cone-f-X-sub: cone \((f, X)\) \(\subseteq \) P\[X]\]
proof
  have cone \((f, X)\) = cone \((f \ast 1, X)\) by simp
  also from f-in-Polys have \(\ldots \subseteq \) cone \((1, X)\) by (rule cone-mono-1)
  finally show \(\?\text{thesis}\) 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 \(p \in \text{ideal } \{f\} \cap P[X]\)
  hence \(p \in \text{ideal } \{f\}\) and \(p \in P[X]\) by simp-all
  have finite \(\{f\}\) by simp
  then obtain \(q\) where \(q \in P[X]\) and \(p = q \ast f\) using \(p \in \text{ideal } \{f\}\)
    by (rule ideal.span-finiteE)
  hence \(p = q \ast f\) by simp
  with \(q \in P[X]\) have \(f \ast q \in P[X]\) by (simp only: mult.commute)
  hence \(q \in P[X]\) using f-in-Polys f-not-0 by (rule times-in-PolysD)
  with \(p\) show \(p \in \text{cone } (f, X)\) by (rule coneI)
next
  fix \(p\)
  assume \(p \in \text{cone } (f, X)\)
  then obtain \(q\) where \(q \in P[X]\) and \(p = q \ast f\) by (rule coneE)
  have \(f \in \text{ideal } \{f\}\) by (rule ideal.span-base) simp
  with \(q \in P[X]\) f-in-Polys show \(p \in \text{ideal } \{f\} \cap P[X]\)
    unfolding \(p\) by (intro IntI ideal.span-scale Polys-closed-times)
qed

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

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

lemma shows valid-ps: valid-decomp $X$ $ps$ (is $\text{thesis1}$)
and std-ps: standard-decomp $d$ $ps$ (is $\text{thesis2}$)
and ext-ps: exact-decomp $X$ $0$ $ps$ (is $\text{thesis3}$)
and cn-ps: cone-decomp $P$ $ps$ (is $\text{thesis4}$)
and hom-ps: hom-decomp $ps$ (is $\text{thesis5}$)
and decomp-F: direct-decomp $\text{(ideal } F \cap P[X]) \cap \text{ideal } \{f\} \cap P[X], P$ (is $\text{thesis6}$)

proof –

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

lemma $P$-sub: $P \subseteq P[X]$
using valid-ps cn-ps by (rule valid-cone-decomp-subset-Polys)
lemma ps-not-Nil: \( ps_+ \neq [] \)
proof
  assume \( ps_+ = [] \)
  have Keys \( P \subseteq (\bigcup h \in \text{set } ps \cdot \text{keys } (\text{fst } hU)) \) \( (\text{is } - \subseteq ?A) \)
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 \( P \) (map cone ps) by (rule cone-decompD)
    then obtain \( qs \) where \( qs \cdot qs \in \text{listset } (\text{map cone } ps) \) and \( p \cdot p = \text{sum-list } qs \)
using \( \cdot p \in P \cdot 
by (\text{rule direct-decompE})
from \( t \in \text{keys } p \) \( \text{keys-sum-list-subset} \) 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 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 \( h \cdot U \) where \( eq \cdot ps ! i = (h, U) \) using prod.exhaust by blast
from \( i < \text{length } ps \) this[symmetric] have \( (h, U) \in \text{set } ps \) by simp
have \( U = [] \)
proof (rule ccontr)
  assume \( U \neq [] \)
with \((h, U) \in \text{set } ps\) have \((h, U) \in \text{set } (ps_+)\) by (simp add: pos-decomp-def)
with \( ps_+ = [] \) show False by simp
qed
with \( q \in \text{cone } (ps ! i) \) have \( q \in \text{range } (\lambda c \cdot h \cdot \text{by } (\text{simp only: eq cone-empty})
then obtain \( c \) where \( q = c \cdot h \) ..
also have \( \text{keys } .. \subseteq \text{keys } h \) by (fact keys-map-scale-subset)
finally have \( t \in \text{keys } h \) using \( t \in \text{keys } q \) ..
  hence \( t \in \text{keys } (\text{fst } (h, U)) \) by simp
with \((h, U) \in \text{set } ps\) 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 \( \exists q \in \text{ideal } F \cdot q \in P[X] \land q \neq 0 \land \neg \text{ lpp } f \text{ adds } lpp \cdot q \)
proof (rule ccontr)
  assume \( \neg (\exists q \in \text{ideal } F \cdot q \in P[X] \land q \neq 0 \land \neg \text{ lpp } f \text{ adds } lpp \cdot q) \)
  hence adds: \( lpp \cdot f \text{ adds } lpp \cdot q \) 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 \text{ideal } \{f\} = \text{ideal } F
proof (rule punit.pmdl-eql-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 \( \text{lpp } f \) adds \( \text{lpp } q \) by (rule adds)
with \( f \neq 0 \) show \( \exists y \in \{ f \} . y \neq 0 \land \text{lpp } g \) adds \( \text{lpp } q \) by blast
qed
with \( \text{ideal } f \neq 0 \) show False ..
qed

then obtain \( q_0 \) where \( q_0 \in \text{ideal } F \) and \( q_0 \in P[X] \) and \( q_0 \neq 0 \)
and \( \text{nadds-q0} \): \( \neg \text{lpp } f \) adds \( \text{lpp } q_0 \) by blast
define \( q \) where \( q = \text{hom-component } q_0 \) (\( \text{deg-pm } (\text{lpp } q_0) \))
from \( \text{hom-F } (q_0 \in \text{ideal } F) \) have \( q \in \text{ideal } F \) unfolding \( \text{q-def} \) by (rule homogeneous-ideal)
from \( \text{homogeneous-set-Polys } (q_0 \in P[X]) \) have \( q \in P[X] \) unfolding \( \text{q-def} \) by (rule homogeneous-setD)
from \( \langle q_0 \neq 0 \rangle \) have \( q \neq 0 \) and \( \text{lpp } q = \text{lpp } q_0 \) unfolding \( \text{q-def} \) by (rule homogeneous-set-lpp)
from \( \text{nadds-q0 this(2)} \) have \( \text{nadds-q} \): \( \neg \text{lpp } f \) adds \( \text{lpp } q \) by simp
have \( \text{hom-q} \): \( \text{homogeneous } q \) by (simp only: \( \text{q-def homogeneous-hom-component} \))
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 \( \text{n-qr-1} \) 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 \( \text{this obtain } y \) where \( y \in Y \) and \( y \neq x \) by blast
from \( \text{this(1)} (Y \subseteq X) \) have \( y \in X \) ..
thus \( \text{thesis} \) using \( y \neq x \) ..
qed
define \( q' \) where \( q' = (\lambda k . \text{punit.monom-mul} \ 1 \ \text{(Poly-Mapping.single } y k) \ q) \)
have \( \text{inj1} \): \( \text{inj } q' \) by (auto intro!: injI simp: \( \text{q'-def} \) \( \langle q \neq 0 \rangle \) dest: \( \text{punit.monom-mul-inj-2 homomorph-mul} \))
have \( \text{q'-in} \): \( q' k \in \text{ideal } F \cap P[X] \) for \( k \) unfolding \( \text{q'-def} \) using \( \langle q \in \text{ideal } F \rangle \)
\( \langle q \in P[X] \rangle \) \( \langle y \in X \rangle \)
by (intro IntI \( \text{punit.plmdl-closed-monom-mul[simplified]} \) \( \text{Polys-closed-monom-mul} \) 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: \( \text{q'-def} \) \( \text{punit.lt-monom-mul} \))
have \( \text{inj2} \): \( \text{inj-on } (\text{deg-pm } \circ \text{lpp}) \) (range \( q' \))
by (auto intro!: inj-onI simp: \( \text{lpp-q' deg-pm-plus deg-pm-single dest: monomorph-inj} \))
have \( (\text{deg-pm } \circ \text{lpp}) \) \( \forall q' \subseteq \text{deg-pm} \) \( \forall \text{Keys } P \)
proof (~)
fix \( d \)
assume \( d \in (\text{deg-pm } \circ \text{lpp}) \) \( \forall q' \)
then obtain \( k \) where \( d = \text{deg-pm } (\text{lpp } (q' k)) \) (is - = \( \text{deg-pm } ?t \)) by auto
from \( \text{hom-q} \) have \( \text{hom-q} \): \( \text{homogeneous } (q' k) \) by (simp add: \( \text{q'-def homogeneous-monom-mul} \))
from \( \langle q \neq 0 \rangle \) have \( q' k \neq 0 \) by (simp add: \( \text{q'-def} \) \( \text{punit.monom-mul-eq-zero-iiff} \))
hence \( ?t \in \text{keys } (q' k) \) by (rule \( \text{punit.lt-in-keys} \))
with \( \text{hom-q} \) have \( \text{deg-q} \): \( d = \text{poly-deg } (q' k) \) unfolding \( d \) by (rule homogeneousD-poly-deg)
from decomp-F $q'$-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 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 $f1 = \text{hom-component } f0$

define $p1$ where $p1 = \text{hom-component } p0$

from hom-q have homogeneous ($q'k$) by (simp add: $q'$-def homogeneous-monom-mul)

hence $q'k = \text{hom-component } (q'k)$ $d$ by (simp add: hom-component-of-homogeneous deg-q)

also have \ldots = $f1 + p1$ by (simp only: $q'$ hom-component-plus $f1$-def $p1$-def)

finally have $q'k = f1 + p1$.

have keys $p1 \neq \{\}$

proof

assume keys $p1 = \{\}$

with ($q'k = f1 + p1$) ($q'k \neq 0$) have $t$: $\exists t. lpp f1$ and $f1 \neq 0$ by simp-all

from $f0$ have $f0 \in \text{ideal } \{f\}$ by simp

with - have $f1 \in \text{ideal } \{f\}$ unfolding $f1$-def by (rule homogeneous-ideal)

(simp add: hom-f)

with puni.is-Grobner-basis-singleton obtain $g$ where $g \in \{f\}$ and $lpp g$

adds $lpp f1$

using ($f1 \neq 0$) by (rule puni.GB-adds-lt[simplified])

hence $lpp f$ adds $\exists t. lpp f$ $f1$ by (simp add: $t$)

hence $\exists$ lookup ($lpp f$) $x \leq$ lookup $\exists t. lpp f$ $f1$ $x$ by (simp add: adds-poly-mapping le-fun-def)

also have \ldots = $\exists$ lookup ($lpp f$) $x$ by (simp add: $lppq$-q lookup-add lookup-single $y \neq x$)

finally have lookup ($lpp f$) $x \leq$ lookup ($lpp f$) $x$.

with $x$ show False ..

qed

then obtain $t$ where $t \in$ 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 $p1 \in P$ using $\exists t. lpp f$ $f1$ unfolding $p1$-def by (rule homogeneous-setD)

with $t \in$ keys $p1$ have $t \in Keys P$ by (rule in-KeysI)

with $d = \deg-pm t$ show $d \in \deg-pm' \ Keys P$ by (rule image-eqI)

qed

moreover from inj1 inj2 have infinite ((deg-pm $\circ$ $lpp$) $'$ 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 $N = \text{normal-form } F \ ' P[X]$

private definition $qs$ where $qs = (\text{SOME } qs', \text{valid-decomp } X qs' \land \text{standard-decomp}$
\[ 0 \text{ qs'} \land \text{monomial-decomp qs'} \land \text{cone-decomp N qs'} \land \text{exact-decomp X 0 qs'} \land (\forall g \in \text{punit.reduced-GB F}. \text{poly-deg g} \leq b \text{ qs'} 0) \]

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

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

proof –
from \(\text{fin-X F-sub}\) obtain \(qs'\) where 1: \(\text{valid-decomp X qs'}\) and 2: \(\text{standard-decomp 0 qs'}\)
and 3: \(\text{monomial-decomp qs'}\) and 4: \(\text{cone-decomp (normal-form F \[ P[X]\]} qs')
and 5: \(\text{exact-decomp X 0 qs'}\)
and 60: \(\forall g. (\forall i. f \in F \Rightarrow \text{homogeneous f}) \Rightarrow g \in \text{punit.reduced-GB F} \Rightarrow \text{poly-deg g} \leq b \text{ qs'} 0\)
by (rule normal-form-exact-decompE) blast
from \(\text{hom-F}\) have \(\forall g. g \in \text{punit.reduced-GB F} \Rightarrow \text{poly-deg g} \leq b \text{ qs'} 0\) by (rule 60)
with 1 2 3 4 5 have \(\text{valid-decomp X qs'}\) \(\land \text{standard-decomp 0 qs'}\)
\(\land \text{monomial-decomp qs'}\) \(\land \text{cone-decomp N qs'}\) \(\land \text{exact-decomp X 0 qs'}\)
\(\forall g \in \text{punit.reduced-GB F}. \text{poly-deg g} \leq bb 0\) by (simp add: \(N\)-def)

hence \(?\text{thesis1}\) \(\land ?\text{thesis2}\) \(\land ?\text{thesis3}\) \(\land ?\text{thesis4}\) \(\land ?\text{thesis5}\) \(\land (\forall g \in \text{punit.reduced-GB F}. \text{poly-deg g} \leq bb 0)\) by (simp add: \(N\)-def)

unfolding \(qs\)-def \(bb\)-def by (rule \(\text{someI}\))

thus \(?\text{thesis1}\) \(\land ?\text{thesis2}\) \(\land ?\text{thesis3}\) \(\land ?\text{thesis4}\) \(\land ?\text{thesis5}\)
and \(g \in \text{punit.reduced-GB F} \Rightarrow \text{poly-deg g} \leq bb 0\) by simp-all
from \(?\text{thesis3}\) show \(?\text{thesis6}\) by (rule monomial-decomp-imp-hom-decomp)

qed

lemma \(N\)-sub: \(N \subseteq P[X]\)
using \(\text{valid-qs cc-qs}\) by (rule valid-cone-decomp-subset-Polys)

lemma \(\text{decomp-Polys}\): \(\text{direct-decomp P[X][ideal \{f\} \cap P[X], P, N]}\)

proof –
from \(\text{fin-X F-sub}\) have \(\text{direct-decomp P[X][ideal F \cap P[X], N]}\) unfolding \(N\)-def
by (rule direct-decomp-ideal-normal-form)
hence \(\text{direct-decomp P[X][[N] @ [ideal \{f\} \cap P[X], P]}\) using \(\text{decomp-F}\)
by (rule direct-decomp-direct-decomp)
hence direct-decomp $P[X] \ (\{f\} \cap P[X], P \ @ [N])$ using perm-append-swap
by (rule direct-decomp-perm)
thus ?thesis by simp
qed

lemma aa-Suc-n [simp]: $aa \ (\text{Suc} \ n) = d$
proof –
from fin-$X$ ext-ps le-refl have $aa \ (\text{Suc} \ n) = a \ ps$ unfolding $aa$-def by (rule b-card-$X$)
also from fin-$X$ valid-ps std-ps ps-not-Nil have $\ldots = d$ by (rule a-nonempty-unique)
finally show ?thesis .
qed

lemma bb-Suc-n [simp]: $bb \ (\text{Suc} \ n) = 0$
proof –
from fin-$X$ ext-qs le-refl have $bb \ (\text{Suc} \ n) = a \ qs$ unfolding $bb$-def by (rule b-card-$X$)
also from std-qs have $\ldots = 0$ 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 $(P[X]:: (- \Rightarrow 'a) \ set) \ z = ((z - d) + (n - 1))$ choose $(n - 1) + \text{Hilbert-fun} \ P \ z + \text{Hilbert-fun} \ N \ z$
proof –
define ss where ss = $[\text{ideal} \ {f} \cap P[X], P, N]$
have homogeneous-set $A$ ∧ phull.subspace $A$ if $A \in \text{set ss}$ for $A$
proof –
from that have $A = \text{ideal} \ {f} \cap P[X] \lor A = P \lor A = N$ by (simp add: ss-def)
thus ?thesis
proof (elim disjE)
assume $A: A = \text{ideal} \ {f} \cap P[X]$
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: $\forall A. \ A \in \text{set ss} \implies \text{homogeneous-set} \ A$ and 2: $\forall A. \ A \in \text{set ss} \implies \text{phull.subspace} \ A$
by simp-all
have Hilbert-fun \((P[X]:\{f \mapsto a\} \text{ set})\) \(z = (\sum p \in \text{set ss}. \text{Hilbert-fun } p \, z)\)
using fin-X subset-refl decomp-Polys unfolding ss-def
proof (rule Hilbert-fun-direct-decomp)
  fix \(A\)
  assume \(A \in \text{set } \{f\} \cap P[X], P, N\)
  hence \(A \in \text{set ss}\) by (simp only: ss-def)
  thus homogeneous-set \(A\) and phull.subspace \(A\) by (rule 1, rule 2)
qed
also have \(\ldots = (\sum p \in \text{set ss}. \text{count-list ss } p \ast \text{Hilbert-fun } p \, z)\)
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 \(\text{count-list ss } p = 1 \lor 1 < \text{count-list ss } p\) by auto
  hence \(\text{Hilbert-fun } p \, z = \text{count-list ss } p \ast \text{Hilbert-fun } p \, z\)
  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)
    hence \(?\)thesis by simp
  qed simp
qed
also have \(\ldots = \text{sum-list } (\lambda p. \text{Hilbert-fun } p \, z) \, ss\)
by (rule sym) (rule sum-list-map-eq-sum-count)
also have \(\ldots = \text{Hilbert-fun } (\text{cone } (f, X)) \, z + \text{Hilbert-fun } P \, z + \text{Hilbert-fun } N \, z\)
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 \(z::\text{int}\)
  let \(?z = \text{nat } z\)
  assume \(\text{max } (aa \, 0) \, (bb \, 0) \leq z\)
  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 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 \((aa \, 0 \leq \text{nat } z)\) by (rule le-trans)
hence \( \text{int- zd; int } (?z - d) = z - \text{int } d \) using \( \text{int-z by linarith} \)

from \( \langle d \leq ?z \rangle \) have \( \text{Hilbert-fun } (P[X];:(- \Rightarrow a') \text{ set} ) \) ?z = 

\( ((?z - d) + (n - 1)) \) choose \( (n - 1) + \text{Hilbert-fun } P \) ?z + 

\( \text{Hilbert-fun } N ?z \)

by (rule \( \text{Hilbert-fun-X} \))

also have \( \text{int} \ldots = (z - d + (n - 1)) \) choose \( (n - 1) + \text{Hilbert-poly } aa \) z + 

\( \text{Hilbert-poly } bb \) z

using \( \text{X-not-empty valid-ps hom-ps cn-ps std-ps ext-ps } \langle aa \rangle \leq \text{nat } z \)

valid-ps hom-ps cn-ps std-ps ext-ps \( \langle bb \rangle \leq \text{nat } z \rangle \langle 0 \leq z \rangle \)

by (simp add: \( \text{Hilbert-fun-eq-Hilbert-poly } \) int-z aa-def bb-def int-binomial int-zd)

finally show \( ?f z = ?g z \) using \( \text{fin-X } \text{not-empty } \langle 0 \leq z \rangle \)

by (simp add: \( \text{Hilbert-fun-Polys } \) int-binomial) smt

qed (simp-all add: poly-fun-Hilbert-poly)

corollary \( \text{dube-eq-1} \):

\( \langle \lambda z::\text{int}. (z + \text{int } n - 1) \rangle \) choose \( (n - 1) \)

\( \langle \lambda z::\text{int}. ((z - d + n - 1)) \rangle \) choose \( (n - 1) \) + \( ((z - d + n) \) choose \( n) + ((z + n) \) choose \( n) - 2 - 

\( \langle \sum i=1\ldots n. ((z - aa i + i - 1) \) choose \( i) + ((z - bb i + i - 1) \) choose \( i) \rangle \)

by (simp only: \( \text{dube-eq-0} \) ) (auto simp: Hilbert-poly-def Let-def sum.distrib)

lemma \( \text{dube-eq-2} \):

assumes \( j < n \)

shows \( \langle \lambda z::\text{int}. (z + \text{int } n - int j - 1) \rangle \) choose \( (n - j - 1) \)

\( \langle \lambda z::\text{int}. ((z - d + n - int j - 1)) \rangle \) choose \( (n - j - 1) \) + \( ((z - d + n - j) \) choose \( (n - j) \) + 

\( \langle \sum i=\text{Suc } j\ldots n. ((z - aa i + i - j) \) choose \( (i - j) \) + ((z - bb i + i - j - 1) \) choose \( (i - j) \rangle ) \)

is \( \langle \text{if } ?f = ?g \rangle \)

proof –

let \( ?h = \lambda z. ((z + (\text{int } i - aa i - 1)) \) choose \( i) + ((z + (\text{int } i - bb i - 1)) \) choose \( i) \)

let \( ?h j = \lambda z. ((z + (\text{int } i - aa i - 1) - j) \) choose \( (i - j) \) + ((z + (\text{int } i - bb i - 1) - j) \) choose \( (i - j) \) \)

from assms have \( 1: j \leq n - \text{Suc } 0 \) and \( 2: j \leq n \) by simp-all

have eq1: \( (\text{bw-diff } \sim j) \) \( \langle \lambda z. \sum i=1\ldots j. ?h z i \rangle = (\lambda-. \text{ if } j = 0 \text{ then } 0 \text{ else } 2) \)

proof (cases \( j \))

case \( 0 \)

thus \( \text{thesis by simp} \)

next

case \( \text{Suc } j0 \)

hence \( j \neq 0 \) by simp

have \( \langle \lambda z::\text{int}. \sum i = 1\ldots j. ?h z i \rangle = (\lambda z::\text{int}. \langle \sum i = 1\ldots j0. ?h z i \rangle + ?h z j) \)

by (simp add: \( \text{Suc } j0 \))

moreover have \( (\text{bw-diff } \sim j) \) \( \ldots = (\lambda z::\text{int}. \langle \sum i = 1\ldots j0. (\text{bw-diff } \sim j) \rangle \) (\( \lambda z. ?h z i) z \rangle + 2) \)
by (simp add: bw-diff-gbinomial-pow)
moreover have \((\sum i = 1..j_0. (bw-diff ^ j) (\lambda z. ?h z i) z) = (\sum i = 1..j_0. 0)\) for \(z::\text{int}\)
using refl
proof (rule sum.cong)
fix i
assume \(i \in \{1..j_0\}\)
hence \(\neg j \leq i\) by (simp add: \(\{j = Suc j_0\}\))
thus \((bw-diff ^ j) (\lambda z. ?h z i) z = 0\) by (simp add: bw-diff-gbinomial-pow)
qed
ultimately show \(?thesis by (simp add: \(j \neq 0\))\)
qed

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

from 1 have \(?f = (bw-diff ^ j) (\lambda z::\text{int}. (z + (int n - 1)) \text{choose} (n - 1))\)
  by (simp add: bw-diff-gbinomial-pow) (simp only: algebra-simps)
also have \(\ldots = (bw-diff ^ j) (\lambda z::\text{int}. (z + int n - 1) \text{choose} (n - 1))\)
  by (simp only: algebra-simps)
also have \(\ldots = (bw-diff ^ j) (\lambda z::\text{int}. \sum (\lambda z::\text{int}. (z + int n - 1) \text{choose} (n - 1)) + ((z + int n - 1) \text{choose} n) + ((z + n) \text{choose} n) - 2 - \sum i=1..n. ((z - aa i + i - 1) \text{choose} i) + ((z - bb i + i - 1) \text{choose} i))\)
  by (simp only: dube-eq-1)
also have \(\ldots = (bw-diff ^ j) (\lambda z::\text{int}. (z + (int n - d - 1)) \text{choose} (n - 1)) + ((z + (int n - d)) \text{choose} n) + ((z + n) \text{choose} n) - 2 - \sum i=1..n. ?h z i)\)
  by (simp only: algebra-simps)
also have \(\ldots = (\lambda z::\text{int}. (z + (int n - d - 1)) \text{choose} (n - 1 - j)) + ((z + (int n - d - j)) \text{choose} (n - j)) + ((z + n - j) \text{choose} (n - j)) - (if \(j = 0\) then 2 else 0) - \sum i=1..n. ?h z i)\)
using 1 2 by (simp add: bw-diff-const-pow bw-diff-gbinomial-pow del: bw-diff-sum-pow)
also from \( j \leq n \) have \( (\lambda z. \sum i=1..n. ?h z i) = (\lambda z. (\sum i=1..j. ?h z i)) + (\sum i=Suc j..n. ?h z i)) \)
by \( \text{simp add: sum-split-nat-iel} \)
also have \( (\text{bv-diff} ^-^- j) \ldots = (\lambda z. (\text{bv-diff} ^-^- j) (\lambda z. \sum i=Suc j..n. ?h z i)) z + (\text{bv-diff} ^-^- j) (\lambda z. \sum i=Suc j..n. ?h z i) z \)
by \( \text{simp only: bv-diff-plas-pow} \)
also have \( \ldots = (\lambda z. (\text{if } j = 0 \text{ then } 0 \text{ else } 2) + (\sum i=Suc j..n. ?h j z i)) \)
by \( \text{simp only: eq1 eq2} \)
finally show \( ?\text{thesis} \) by \( \text{simp add: algebra-simps} \)
qed

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

224
proof

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 \(?S = \lambda A.\ \{h, (h, \{\}) \in A \land \text{poly-deg } h = z\}\)

have \(\text{fin: finite } (?S\ A)\ \text{if finite } A\ \text{for } A::((\langle x \Rightarrow_0 \text{nat} \rangle \Rightarrow_0 'a) \times 'x set)\ \text{set}\)
proof -
  have \((\lambda t.\ (\{\}))\ ' ?S A \subseteq A\ \text{by blast}\)
  hence finite \((\lambda t.\ (\{\}::'x set))\ ' ?S A\ \text{using that by (rule finite-subset)}\)
  moreover have inj-on \((\lambda t.\ (\{\}::'x set))\ (\?S\ A)\ \text{by (rule inj-on) simp}\)
  ultimately show \(?\text{thesis}\ \text{by (rule finite-imageD)}\)
qed

from \(\text{finite-set have } 1::\ (\?S\ \text{set ps})\ \text{by (rule fin)}\)
from \(\text{finite-set have } 2::\ (\?S\ \text{set qs})\ \text{by (rule fin)}\)

from \(\neg\ z < \max (aa\ (1)\ (bb\ 1))\) have \(aa\ 1 \leq z\ \text{and bb } 1 \leq z\ \text{by simp-all}\)
have \(d \leq aa\ 1\ \text{unfolding aa-Suc-n[symmetric]}\ \text{aa-def}\ \text{using fin-X}\ \text{by (rule)}\)

b-decreasing simp
hence \(d \leq z\ \text{using } (aa\ 1 \leq z);\ \text{by (rule le-trans)}\)
hence eq: \(\text{int } (z - d) = \text{int } z - \text{int } d\ \text{by simp}\)
from \(d \leq z\) have \(\text{Hilbert-fun } (P[X]):(\Rightarrow_0 'a)\ \text{set}\) = 
\(\langle (z - d) + (n - 1)\rangle\ \text{choose } (n - 1) + \text{Hilbert-fun } P z + \text{Hilbert-fun } N z\)
by (rule Hilbert-fun-X)
also have \(\text{int } \ldots = (\langle \text{int } z - d + (n - 1)\rangle)\ \text{gchoose } (n - 1) + \text{Hilbert-poly } aa\ z + \text{Hilbert-poly } bb\ z\) + 
\(\langle \text{int } (\langle ?S\ \text{set ps} \rangle)\rangle + \text{int } (\langle ?S\ \text{set qs} \rangle)\rangle = \text{int } z + n - 1\ \text{gchoose } (n - 1)\)
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 \(\langle \text{int } z - d + n - 1\rangle\ \text{gchoose } (n - 1) + \text{Hilbert-poly } aa\ z + \text{Hilbert-poly } bb\ z\) + 
\(\text{int } (\langle ?S\ \text{set ps} \rangle)\rangle + \text{int } (\langle ?S\ \text{set qs} \rangle)\rangle = \text{int } z + n - 1\ \text{gchoose } (n - 1)\)
using fin-X X-not-empty by (simp add: Hilbert-fun-Polys int-binomial algebra-simps)
also have \(\ldots = \langle \text{int } z - d + n - 1\rangle\ \text{gchoose } (n - 1) + \text{Hilbert-poly } aa\ z + \text{Hilbert-poly } bb\ z\)
by (fact dube-eq-0[THEN fun-cong])
finally have \(\text{int } (\langle ?S\ \text{set ps} \rangle)\rangle + \text{int } (\langle ?S\ \text{set qs} \rangle)\rangle = 0\ \text{by simp}\)
hence \(\text{card } (\langle ?S\ \text{set ps} \rangle) = 0\ \text{and card } (\langle ?S\ \text{set qs} \rangle) = 0\ \text{by simp-all}\)
with \(1\ 2\ \text{have } ?S\ \text{set ps} \cup \text{set qs} = \{\}\ \text{by auto}\)
moreover from \(\text{assms have } h \in ?S\ \text{set ps} \cup \text{set qs}\) by (simp add: z-def)
ultimately have \(h \in \{\}\ \text{by (rule subst)}\)

225
thus False by simp

qed

lemma
shows aa-n: aa n = d and bb-n: bb n = 0 and bb-0: bb 0 ≤ max (aa 1) (bb 1)
proof –
let ?j = n – Suc 0
from n-gr-0 have ?j < n and eq1: Suc ?j = n and eq2: n = ?j + 1 by simp-all
from this(1) have (1::int) = (- 1) * ((int d - 1) gchoose (n - Suc ?j)) +
(−1)^(n - ?j) * ((int d - 1) gchoose (n - ?j)) - 1 -
(∑ i=Suc ?j..n. (−1)^(i - ?j) * ((int (aa i) gchoose (i - ?j)))
by (rule dube-eq-3)
hence eq: aa n + bb n = d by (simp add: eq1 eq2)
hence aa n ≤ d by simp
moreover have d ≤ aa n unfolding aa-Suc-n[symmetric] aa-def using fin-X
by (rule b-decreasing) simp
ultimately show aa n = d by (rule antisym)
with eq show bb n = 0 by simp
have bb 0 = b qs 0 by (simp only: bb-def)
also from fin-X have ... ≤ max (aa 1) (bb 1) (is - ≤ ?m)
proof (rule b-le)
from fin-X ext qs have a qs = bb (Suc n) by (simp add: b-card-X bb-def)
also have ... ≤ bb 1 unfolding bb-def using fin-X by (rule b-decreasing)
simp
also have ... ≤ ?m by (rule max.cobounded2)
finally show a qs ≤ ?m.
next
fix h U
assume (h, U) ∈ set qs
show poly-deg h < ?m
proof (cases card U = 0)
case True
from fin-X valid-qs (h, U) ∈ set qs have finite U by (rule valid-decompD-finite)
with True have U = {} by simp
with ⟨(h, U) ∈ set qs⟩ have ⟨h, {}⟩ ∈ set ps ∪ set qs by simp
thus ?thesis by (rule dube-aux-1)
next
case False
hence 1 ≤ card U by simp
with fin-X ⟨(h, U) ∈ set qs⟩ have poly-deg h < bb 1 unfolding bb-def by (rule b)
also have ... ≤ ?m by (rule max.cobounded2)
finally show ?thesis .
qed
qed
finally show bb 0 ≤ ?m.
lemma dube-eq-4:
assumes $j < n$
shows $1'\cdot\text{int} = 2 \cdot (-1)\cdot(n - \text{Suc j}) \cdot ((\text{int d - 1}) \text{gchoose (n - Suc j)}) - 1 -
(\sum i= Suc j..n-1. (-1)\cdot(i - j) \cdot ((\text{aa i}) \text{gchoose (i - j)})
+ (\text{int (bb i) gchoose (i - j)}))$
proof -
from assms have $\text{Suc j} \leq n$ and $0 < n$ and $1: \text{Suc (n - Suc j)} = n - j$ by simp-all
have $2: (-1) \cdot (n - \text{Suc j}) = -(((-1)\cdot \text{int}) \cdot (n - j))$ by simp flip: 1
from assms have $(1'\cdot\text{int}) = (-1)\cdot(n - \text{Suc j}) \cdot ((\text{int d - 1}) \text{gchoose (n - Suc j)}) +
(\sum i= Suc j..n. (-1)\cdot(i - j) \cdot ((\text{aa i}) \text{gchoose (i - j)})
+ (\text{int (bb i) gchoose (i - j)}))$
by (rule dube-eq-3)
also have \ldots = $(-1)\cdot(n - \text{Suc j}) \cdot ((\text{int d - 1}) \text{gchoose (n - Suc j)}) +
(-1)\cdot(n - j) \cdot ((\text{int (aa n) gchoose (n - j)}) + (\text{int (bb n)})
\text{gchoose (n - j)}) -
(\sum i= Suc j..n-1. (-1)\cdot(i - j) \cdot ((\text{aa i}) \text{gchoose (i - j)})
+ (\text{int (bb i) gchoose (i - j)}))$
using $0 < n'$: $\text{Suc j} \leq n'$ by (simp only: sum-tail-nat)
also have \ldots = $(-1)\cdot(n - \text{Suc j}) \cdot ((\text{int d - 1}) \text{gchoose (n - Suc j)}) +
(-1)\cdot(n - j) \cdot ((\text{int d gchoose (n - j)}) + (\text{int d gchoose (n - j)}) -
(\sum i= Suc j..n-1. (-1)\cdot(i - j) \cdot ((\text{aa i}) \text{gchoose (i - j)})
+ (\text{int (bb i) gchoose (i - j)}))$
using assms by (simp add: aa-n bb-n gbinomial-0-left right-diff-distrib)
also have $(-1)\cdot(n - j) \cdot ((\text{int d - 1}) \text{gchoose (n - j)}) - (\text{int d gchoose (n - j)}) =
(-1)\cdot(n - \text{Suc j}) \cdot ((\text{int d - 1}) \text{gchoose (Suc (n - Suc j)}) -
((\text{int d - 1}) \text{gchoose (Suc (n - Suc j)))})$
by (simp add: 1' 2 flip: mult-minus-right)
also have \ldots = $(-1)\cdot(n - \text{Suc j}) \cdot ((\text{int d - 1}) \text{gchoose (n - Suc j)})$
by (simp only: gbinomial-int-Suc-Suc, simp)
finally show $\text{thesis}$ by simp
qed

lemma cc-Suc:
assumes $j < n - 1$
shows $\text{int (cc (Suc j))} = 2 + 2 \cdot (-1)\cdot(n - j) \cdot ((\text{int d - 1}) \text{gchoose (n - Suc j)}) +
(\sum i=j+2..n-1. (-1)\cdot(i - j) \cdot ((\text{aa i}) \text{gchoose (i - j)})
+ (\text{int (bb i) gchoose (i - j)}))$
proof -
from assms have $j < n$ and $\text{Suc j} \leq n - 1$ by simp-all
hence $n - j = \text{Suc} \ (n - \text{Suc} \ j)$ by simp
hence eq: $((-1)^n - ((n - \text{Suc} \ j) - \text{Suc} \ j) - n - j))$ by simp
from $j < n$ have $((n - \text{Suc} \ j) - ((n - \text{Suc} \ j) - \text{Suc} \ j) - n - j) = 2 * (-1) * ((n - \text{Suc} \ j) - \text{Suc} \ j) - n - j) - 1 -
(\sum_{i=\text{Suc} \ j} \cdot n - 1) \cdot (-1) * (i - j) * ((\text{int} (\text{aa} \ i) \ \text{gchoose} \ (i - j)) + (\text{int} (\text{bb} \ j) \ \text{gchoose} \ (i - j)))$
  by (rule dube-eq-4)
also have $\ldots = \text{cc} \ (\text{Suc} \ j) - 2 * (-1) * ((\text{int} d - 1) \ \text{gchoose} \ (n - \text{Suc} \ j)) - 1 -
(\sum_{i=\text{Suc} \ j} \cdot n - 1) \cdot (-1) * (i - j) * ((\text{int} (\text{aa} \ i) \ \text{gchoose} \ (i - j)) + (\text{int} (\text{bb} \ i) \ \text{gchoose} \ (i - j)))$
  using $(\text{Suc} \ j \leq n - 1)$ by (simp add: sumLeast-Suc-atMost eq)
  finally show $?\text{thesis}$ 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: $n - (n - \text{Suc} \ 0) = \text{Suc} \ 0$ and 4: $n - $?j $= 2$
  by simp-all
  have $\text{int} (cc (n - 1)) = \text{int} (cc (Suc $?j))$ by (simp only: 1)
  also from $?j < n - 1$ have $\ldots = 2 + 2 * (-1) ^ \cdot (n - $?j) * ((\text{int} d - 1) \ \text{gchoose} \ (n - \text{Suc} $?j)) +
(\sum_{i=\text{Suc} \ j} \cdot n - 1) \cdot (-1) ^ \cdot (i - j) * ((\text{int} (\text{aa} \ i) \ \text{gchoose} \ (i - j)) + (\text{int} (\text{bb} \ i) \ \text{gchoose} \ (i - j)))$
  by (rule cc-Suc)
  also have $\ldots = \text{int} (2 * d)$ by (simp add: 1 2 3 4)
  finally show $?\text{thesis}$ by (simp only: int-int-eq)
qed

Since the case 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 $(n - 2) \leq d^2 + 2 * d$
proof
  let $?j = n - 3$
from n-gr-2 have 1: Suc $?j = n - 2$ and $?j < n - 1$ and 2: Suc $(n - 2) = n - \text{Suc} \ 0$
  and 3: $n - (n - 2) = 2$ and 4: $n - $?j $= 3$
  by simp-all
  have $\text{int} (cc (n - 2)) = \text{int} (cc (Suc $?j))$ by (simp only: 1)
  also from $?j < n - 1$ have $\ldots = 2 + 2 * (-1) ^ \cdot (n - $?j) * ((\text{int} d - 1) \ \text{gchoose} \ (n - \text{Suc} $?j)) +
(\sum_{i=\text{Suc} \ j} \cdot n - 1) \cdot (-1) ^ \cdot (i - j) * ((\text{int} (\text{aa} \ i) \ \text{gchoose} \ (i - j)) + (\text{int} (\text{bb} \ i) \ \text{gchoose} \ (i - j)))$

228
by (rule cc-Suc)
also have ... = (2 - 2 * (int d - 1) choose 2) + ((int (aa (n - 1)) choose 2) + (int (bb (n - 1)) choose 2))
by (simp add: 1 2 3 4)
also have ... ≤ (2 - 2 * (int d - 1) choose 2) + (2 * int d choose 2)
proof (rule add-left-mono)
  have int (aa (n - 1)) choose 2 + (int (bb (n - 1)) choose 2) ≤ int (aa (n - 1)) + int (bb (n - 1)) choose 2
  by (rule gbinomial-int-plus-le) simp-all
  also have ... = int (2 * d) choose 2 by (simp flip: cc-n-minus-1)
  also have ... = 2 * int d choose 2 by (simp add: int-ops (simp only: algebra-simps))
finally show int (aa (n - 1)) choose 2 + (int (bb (n - 1)) choose 2) ≤ 2 * int d choose 2 .
qed
also have ... = 2 - fact 2 * (int d - 1) choose 2 + (2 * int d choose 2) by
(simp only: fact-2)
also have ... = 2 - (int d - 1) * (int d - 2) + (2 * int d choose 2)
by (simp only: gbinomial-int-mult-fact) (simp add: numeral-2-eq-2 prod.atLeast0-lessThan-Suc)
also have ... = 2 - (int d - 1) * (int d - 2) + int d * (2 * int d - 1)
by (simp add: gbinomial-prod-rev numeral-2-eq-2 prod.atLeast0-lessThan-Suc)
also have ... = int (d^2 + 2 * d) by (simp add: power2-eq-square) (simp only: int-int-eq)
finally show ?thesis by (simp only: int-int-eq)


lemma cc-Suc-le:
  assumes j ≤ n - 3
  shows int (cc (Suc j)) ≤ 2 + (int (cc (j + 2)) choose 2) + (\sum i=j+4..n-1. int (cc i) choose (i - j))

— Could be proved without coercing to int, because everything is non-negative.

proof —
  let \( \lambda i. (\text{int } (aa i) \text{ choose } (i - j)) + (\text{int } (bb i) \text{ choose } (i - j)) \)
  let \( \lambda S = \lambda x y. (\sum i=j+x..n-y. (-1)^{(i - j)} \cdot \text{int } \text{if } i \text{ is } \text{int } (cc i) \text{ choose } (i - j)) \)
  have iel: int (aa i) choose k + (int (bb i) choose k) ≤ int (cc i) choose k if 0 < k for i k
    proof —
      from that have int (aa i) choose k + (int (bb i) choose k) ≤ int (aa i) + int (bb i) choose k
    by (rule gbinomial-int-plus-le) simp-all
    also have ... = int (cc i) choose k by simp
    finally show ?thesis .
  qed
  from d-gr-0 have 0 ≤ int d - 1 by simp
  from assms have 0 < n - Suc j by simp
  have f-nonneg: 0 ≤ \text{if } i \text{ is } \text{int } (cc i) \text{ choose } (i - j) if 0 < k for i k
    by (simp add: gbinomial-int-nonneg)
  show ?thesis


229
proof (cases \( n = j + 4 \))
  
case True
    
  hence \( j: j = n - j \) by simp
  
  have 1: \( n - Suc j = 3 \) and \( j < n - 1 \) and 2: \( Suc (n - 3) = Suc (Suc j) \)
  
  and 3: \( n - (n - 3) = 3 \)
  
  and 4: \( n - j = 4 \) and 5: \( n - Suc 0 = Suc (Suc (Suc j)) \) and 6: \( n - 2 = Suc (Suc j) \)
  
  by (simp-all add: True)
  
  from \( j < n - 1 \) have \( int (cc (Suc j)) = 2 + 2 \ast ( - 1) \ast (n - j) \ast (int d - 1) \ast (n - Suc j) \ast (\sum i = j + 2..n - 1. (- 1) \ast (i - j) \ast (int (aa i) \ast (int (bb (i) \ast (i - j)))) \)
  
  by (rule cc-Suc)
  
  also have \( \ldots = (2 + ((int (aa (n - 2)) \ast (int (bb (n - 2))) \ast (int gchoose 2))) + (2 \ast (int (d - 1) \ast (int gchoose 3)) - ((int (aa (n - 1)) \ast (int gchoose 3)) + (int (bb (n - 1)) \ast (int gchoose 3)))) \)
  
  by (simp add: 1 2 3 4 5 6)
  
  also have \( \ldots \leq (2 + ((int (aa (n - 2)) \ast (int (bb (n - 2))) \ast (int gchoose 2)) + (int (bb (n - 2)) \ast (int gchoose 2)))) + 0 \)
  
  proof (rule add-left-mono)
  
  from cc-n-minus-1 have \( eqI: int (aa (n - 1)) + int (bb (n - 1)) = 2 \ast int d \) by simp
    
  hence \( ie2: int (aa (n - 1)) \leq 2 \ast int d \) by simp
  
  from \( \emptyset \leq int d - 1 \) have \( int d - 1 \ast gchoose 3 \leq int d \ast gchoose 3 \) by (rule gbinomial-int-mono simp)
    
  hence \( 2 \ast (int d - 1) \ast gchoose 3 \leq 2 \ast (int d \ast gchoose 3) \) by simp
    
  also from \( - ie2 \) have \( \ldots \leq int (aa (n - 1)) \ast gchoose 3 + (2 \ast int d - int (aa (n - 1)) \ast gchoose 3) \)
    
  by (rule binomial-int-ineq-3 simp)
    
  also have \( \ldots \leq int (aa (n - 1)) \ast gchoose 3 + (int (bb (n - 1)) \ast gchoose 3) \)
    
  by (simp flip: eqI)
    
  finally show \( 2 \ast (int d - 1) \ast gchoose 3 - (int (aa (n - 1)) \ast gchoose 3 + (int (bb (n - 1)) \ast gchoose 3)) \leq 0 \)
    
  by simp
    
  qed
    
  also have \( \ldots = 2 + ((int (aa (n - 2)) \ast (int (bb (n - 2)) \ast (int gchoose 2)) + (int (bb (n - 1)) \ast (int gchoose 3))) \)
    
  by simp
    
  also from \( ie1 \) have \( \ldots \leq 2 + (int (cc (n - 2)) \ast (int gchoose 2)) \) by (rule add-left-mono simp)
    
  also have \( \ldots = 2 + (int (cc (j + 2)) \ast (int gchoose 2)) + ?S3 4 1 \) 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 \( 0 \leq n - 1 \) by simp
  
  from assms have \( j + 2 \leq n - 1 \) and \( j + 2 \leq n - 2 \) by simp-all
    
  hence \( n - j = Suc (n - Suc j) \) by simp

230
hence \(1: (-1) \cdot (n - Suc \ j) = -((-1::\text{int}) \cdot (n - j))\) by simp
from \text{assms have} \(j < n - 1\) by simp
hence \(\text{int (cc (Suc \ j))} = 2 + 2 \cdot (-1) \cdot (n - j) \cdot ((\text{int } d - 1) \text{ choose } (n - Suc \ j)) + ?S 2 1\)
by (rule cc-Suc)
also have \(\ldots = 2 \cdot (-1) \cdot (n - j) \cdot ((\text{int } d - 1) \text{ choose } (n - Suc \ j)) +\)
\((-1) \cdot (n - Suc j) \cdot ((\text{int } aa (n - 1)) \text{ choose } (n - Suc j)) +\)
\((\text{int } (bb (n - 1)) \text{ choose } (n - Suc j))) +\)
\((2 + ?S 2 2)\)
using \((0 < n - 1) \cdot j + 2 \leq n - 1)\) by (simp only: sum-tail-nat) (simp flip:
\text{numeral-2-eq-2})
also have \(\ldots \leq (\text{int (cc (n - 1)) \text{ choose } (n - Suc \ j)) + (2 + ?S 2 2)\)
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 \(2 \cdot (-1) \cdot (n - j) \cdot ((\text{int } d - 1) \text{ choose } (n - Suc j)) +\)
\((-1) \cdot (n - Suc j) \cdot ((\text{int } aa (n - 1)) \text{ choose } (n - Suc j)) +\)
\((\text{int } (bb (n - 1)) \text{ choose } (n - Suc j))) =\)
\((-1) \cdot (n - j) \cdot (2 \cdot ((\text{int } d - 1) \text{ choose } (n - Suc j)) -\)
\((\text{int } (aa (n - 1)) \text{ choose } (n - Suc j)) - (\text{int } (bb (n - 1)))\)
\text{choose } (n - Suc j)))
by (simp only: 1 \text{algebra-simps})
also have \(\ldots \leq (\text{int (cc (n - 1)) \text{ choose } (n - Suc \ j))\)
proof (cases even \((n - j))\)
case True
\begin{align*}
\text{hence } (-1) \cdot (n - j) \cdot (2 \cdot (\text{int } d - 1) \text{ choose } (n - Suc j)) - (\text{int } aa (n - 1)) \text{ choose } (n - Suc j)) = \end{align*}
\begin{align*}
& (\text{int } (bb (n - 1)) \text{ choose } (n - Suc j)) = \end{align*}
\begin{align*}
& 2 \cdot (\text{int } d - 1 \text{ choose } (n - Suc j)) - ((\text{int } (aa (n - 1)) \text{ choose } (n - Suc j)) + \end{align*}
\begin{align*}
& (\text{int } (bb (n - 1)) \text{ choose } (n - Suc j))\)
by simp
also have \(\ldots \leq 2 \cdot (\text{int } d - 1 \text{ choose } (n - Suc \ j))\) by (rule rl) (simp
add: gbinomial-int-nonneg)
also have \(\ldots = (\text{int } d - 1 \text{ choose } (n - Suc \ j)) + (\text{int } d - 1 \text{ choose } (n - Suc \ j))\) by simp
also have \(\ldots \leq (\text{int } d - 1) + (\text{int } d - 1) \text{ choose } (n - Suc \ j)\)
using \((0 < n - Suc \ j) \cdot 0 \leq int d - 1 \cdot 0 \leq int d - 1)\) by (rule
gbinomial-int-plus-le)
also have \(\ldots \leq 2 \cdot \text{int } d \text{ choose } (n - Suc \ j)\)
proof (rule gbinomial-int-mono)
from \(0 \leq int d - 1\) show \(0 \leq int d - 1 + (\text{int } d - 1)\) by simp
qed simp
also have \(\ldots = \text{int (cc (n - 1)) \text{ choose } (n - Suc \ j)}\) by (simp only:
cc-n-minus-1) simp
finally show \(\text{thesis}\).
next
case False
hence \((-1) \cdot (n - j) \cdot (2 \cdot (\text{int } d - 1) \text{ choose } (n - Suc \ j)) - (\text{int } aa (n - 1)) \text{ choose } (n - Suc \ j)) = \)
\[(\text{int } (bb \ (n - 1)) \ gchoose \ (n - \text{Suc } \ j))) =
(\text{int } (aa \ (n - 1)) \ gchoose \ (n - \text{Suc } \ j)) + (\text{int } (bb \ (n - 1)) \ gchoose \ (n - \text{Suc } \ j)) -
2 \ast (\text{int } d - 1 \ gchoose \ (n - \text{Suc } \ j))\]

by \text{simp}
also have \ldots \leq \text{int } (aa \ (n - 1)) \ gchoose \ (n - \text{Suc } \ j)) + (\text{int } (bb \ (n - 1)) \ gchoose \ (n - \text{Suc } \ j))
by \text{(rule if)} \ (\text{simp add: gbinomial-int-nonneg d-gr-0})
also \text{from } (0 < n - \text{Suc } \ j) \text{ have } \ldots \leq \text{int } (cc \ (n - 1)) \ gchoose \ (n - \text{Suc } \ j)) \text{ by } \text{(rule ie1)}
finally \text{ show } ?\text{thesis} .
\text{qed}
finally \text{ show } 2 \ast (-1)^\text{/(n - j)} \ast ((\text{int } d - 1) \ gchoose \ (n - \text{Suc } \ j)) +
(-1)^\text{/(n - Suc } \ j) \ast ((\text{int } (aa \ (n - 1)) \ gchoose \ (n - \text{Suc } \ j)) +
(\text{int } (bb \ (n - 1)) \ gchoose \ (n - \text{Suc } \ j))) \leq
(\text{int } (cc \ (n - 1))) \ gchoose \ (n - \text{Suc } \ j) .
\text{qed}
also have \ldots = 2 + (\text{int } (cc \ (n - 1)) \ gchoose \ ((n - 1) - j)) + ((\text{int } (aa \ (j + 2))) \ gchoose \ 2) +
(\text{int } (bb \ (j + 2)) \ gchoose \ 2)) + ?\text{S }3 \ 2
\text{using } (j + 2 \leq n - 2) \text{ by } \text{(simp add: sum.atLeast-Suc-atMost numeral-3- eq-3)}
also have \ldots \leq 2 + (\text{int } (cc \ (n - 1)) \ gchoose \ ((n - 1) - j)) + ((\text{int } (aa \ (j + 2))) \ gchoose \ 2) +
(\text{int } (bb \ (j + 2)) \ gchoose \ 2)) + ?\text{S }3 \ 4 \ 2
\text{proof } \text{(rule add-left-mono)}
\text{from } (j + 4 \leq n - 1) \text{ have } j + 3 \leq n - 2 \text{ by } \text{simp}
hence ?\text{S }3 \ 2 = ?\text{S }4 \ 2 - \text{if } (j + 3) \text{ j by } \text{(simp add: sum.atLeast-Suc-atMost add.commute)}
hence ?\text{S }3 \ 2 \leq ?\text{S }4 \ 2 \text{ using } \text{f-nonneg[of j + 3]} \text{ by } \text{simp}
also have \ldots \leq ?\text{S }3 \ 4 \ 2
\text{proof } \text{(rule sum-mono)}
fix \ i
assume \ i \in \{j + 4 \ldots n - 2\}
hence 0 < i - j \text{ by } \text{simp}
\text{from } \text{f-nonneg[of i]} \text{ have } (-1)^\text{/(i - j)} \ast \text{if } i \leq j \text{ by } \text{(smt minus-one-mult-self mult-cancel-right1 pos-mul eq-1-iff-lemma)}
\text{zero-less-mult-iff)}
also \text{from } (0 < i - j) \text{ have } \ldots \leq \text{int } (cc \ i) \ gchoose \ (i - j) \text{ by } \text{(rule ie1)}
finally \text{ show } (-1)^\text{/(i - j)} \ast \text{if } i \leq j \text{ \leq \text{int } (cc \ i) \ gchoose \ (i - j) .}
\text{qed}
finally \text{ show } ?\text{S }3 \ 2 \leq ?\text{S }3 \ 4 \ 2 .
\text{qed}
also have \ldots = ((\text{int } (aa \ (j + 2)) \ gchoose \ 2) + (\text{int } (bb \ (j + 2)) \ gchoose \ 2))
+ (2 + ?\text{S }3 \ 4 \ 1)
\text{using } (0 < n - 1) \ \\text{\& } j + 4 \leq n - 1) \text{ by } \text{(simp only: sum-tail-nat) (simp flip: numeral-2-eq-2)}
also \text{from } \text{ie1 have } \ldots \leq \text{int } (cc \ (j + 2)) \ gchoose \ 2 + (2 + ?\text{S }3 \ 4 \ 1)
\text{by } \text{(rule add-right-mono) simp}
also have \ldots = 2 + (\text{int } (cc \ (j + 2)) \ gchoose \ 2) + ?\text{S }3 \ 4 \ 1 \text{ by } \text{(simp only:}
corollary cc-le:
  assumes 0 < j and j < n - 2
  shows cc j ≤ 2 + (cc (j + 1) choose 2) + (∑ i=j+3..n-1. cc i choose (Suc (i - j)))
proof -
  define j0 where j0 = j - 1
  with assms have j = Suc j0 and j0 < n - 3 by simp-all
  have int (cc j) = int (cc (Suc j0)) by (simp only: j)
  also have ... ≤ 2 + (int (cc (j0 + 2)) gchoose 2) + (∑ i=j0+4..n-1. int (cc i) gchoose (i - j0))
  using (j0 < n - 3) by (rule cc-Suc-le)
  also have ... = 2 + (int (cc (j + 1)) gchoose 2) + (∑ i=j0+4..n-1. int (cc i) gchoose (i - j0))
  by (simp add: j)
  also have (∑ i=j0+4..n-1. int (cc i) gchoose (i - j0)) = int (∑ i=j+3..n-1. cc i choose (Suc (i - j)))
  unfolding int-sum
proof (rule int-sum)
  fix i
  assume i ∈ {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) ≤ int (2 + (cc (j + 1) choose 2) + (∑ 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 ≤ n implies cc j ≤ Dube-aux n d j
proof (induct j rule: Dube-aux.induct[where n=n])
case step: (1 j)
  from step.prems(2) have j + 2 < n ∨ j + 2 = n ∨ 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 ≤ n by simp
  ultimately have cc (j + 1) ≤ Dube-aux n d (j + 1) by (rule step.hyps)
  hence 1: cc (j + 1) choose 2 ≤ Dube-aux n d (j + 1) choose 2
    by (rule Binomial-Int.binomial-mono)
  have 2: (∑ i = j + 3..n - 1. cc i choose Suc (i - j)) ≤
\[
\sum i = j + 3..n - 1. \text{ Dube-aux } n \ d \ i \ \text{choose} \ Suc \ (i - j)
\]

proof (rule sum-mono)
fix \(i::\text{nat}\)

note *
moreover assume \(i \in \{j + 3..n - 1\}\)
mOREVER from this \(\langle 2 < n \rangle \) have \(0 < i \ \text{and} \ i + 1 \leq n \) by auto
ultimately have \(cc \ i \leq \text{Dube-aux } n \ d \ i \) by (rule step.hyps)
thus \(cc \ i \ \text{choose} \ Suc \ (i - j) \leq \text{Dube-aux } n \ d \ i \ \text{choose} \ Suc \ (i - j)\)
by (rule Binomial-Int.binomial-mono)

qed

from * have \(j < n - 2 \) by simp
with step.prems(1) have \(cc \ j \leq 2 + (cc \ (j + 1) \ \text{choose} \ 2) + \sum i = j + 3..n - 1. \ \text{cc } i \ \text{choose} \ Suc \ (i - j)\)
by (rule cc-le-Dube-aux)
also from * 1 2 have \(\ldots \leq \text{Dube-aux } n \ d \ j \) by simp

finally show ?thesis.

next
assume \(j + 2 = n\)
hence \(j = n - 2 \) and \(\text{Dube-aux } n \ d \ j = d^2 + 2 \ast d \) by simp-all
thus ?thesis by (simp only: cc-n-minus-2)

next
assume \(j + 1 = n\)
hence \(j = n - 1 \) and \(\text{Dube-aux } n \ d \ j = 2 \ast d \) by simp-all
thus ?thesis by (simp only: cc-n-minus-1)

qed

qed

end

lemma Dube-aux:
assumes \(g \in \text{punit.reduced-GB } F\)
shows \(\text{poly-deg } g \leq \text{Dube-aux } n \ d \ 1\)

proof (cases \(n = 2\))
case True
from assms have \(\text{poly-deg } g \leq bb \ 0 \) by (rule deg-RGB)
also have \(\ldots \leq \max \ (aa \ 1) \ (bb \ 1) \) by (fact bb-0)
also have \(\ldots \leq cc \ (n - 1) \) by (simp add: True)
also have \(\ldots = 2 * d \) by (fact cc-n-minus-1)
also have \(\ldots = \text{Dube-aux } n \ d \ 1 \) by (simp add: True)
finally show ?thesis.

case False
with \(\langle 1 < n \rangle \) have \(2 < n \) and \(1 + 1 \leq n \) by simp-all
from assms have \(\text{poly-deg } g \leq bb \ 0 \) by (rule deg-RGB)
also have \(\ldots \leq \max \ (aa \ 1) \ (bb \ 1) \) by (fact bb-0)
also have \(\ldots \leq cc \ 1 \) by simp
also from \(\langle 2 < n \rangle \ - \ (1 + 1 \leq n) \) have \(\ldots \leq \text{Dube-aux } n \ d \ 1 \) by (rule cc-le-Dube-aux)
simp
finally show ?thesis.
qed
end

theorem Dube:
  assumes finite F and F ⊆ P[X] and ∀f ∈ F ⇒ homogeneous f and g ∈ punit.reduced-GB F
  shows poly-deg g ≤ Dube (card X) (maxdeg F)
proof (cases F ⊆ {0})
  case True
  hence F = {} ∨ 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} ≠ {} by simp
  hence F ≠ {} by blast
  from assms(1) have fin1: finite (poly-deg 'F) by (rule finite-imageI)
  from assms(1) have finite (F − {0}) by simp
  hence fin: finite (poly-deg ' (F − {0})) by (rule finite-imageI)
  moreover from (F − {0}) ≠ {} have *: poly-deg ' (F − {0}) ≠ {} by simp
  ultimately have maxdeg (F − {0}) ∈ poly-deg ' (F − {0}) unfolding maxdeg-def
  by (rule Max-in)
  then obtain f where f ∈ F − {0} and md1: maxdeg (F − {0}) = poly-deg f
  then obtain f_max where f ∈ F − {0} and md1: maxdeg (F − {0}) = poly-deg f
  ..
  note this(2)
  moreover have maxdeg (F − {0}) ≤ maxdeg F
  unfolding maxdeg-def using image-monotone * fin1 by (rule Max-monotone) blast
  ultimately have poly-deg 'F ≤ maxdeg F by simp
  from f ∈ F − {0} have f ∈ F and f ≠ 0 by simp
  from this(1) assms(2) have f ∈ P[X] ..
  have f-max: poly-deg f' ≤ poly-deg f if f' ∈ F for f'
  proof (cases f' = 0)
    case True
    thus ?thesis by simp
  next
    case False
    with that have f' ∈ F − {0} by simp
    hence poly-deg f' ∈ poly-deg ' (F − {0}) by (rule imageI)
    with fin show poly-deg f' ≤ poly-deg f unfolding md1[symmetric] maxdeg-def
    by (rule Max-ge)
  qed
  have maxdeg F ≤ poly-deg f unfolding maxdeg-def using fin1 (poly-deg 'F ≠ {})
  proof (rule Max.boundedI)
    fix d
    assume d ∈ poly-deg 'F
    then obtain f' where f' ∈ F and d = poly-deg f'
    note this(2)
  ..

235
also from \( f' \in F \) have \( \text{poly-deg} \ f' \leq \text{poly-deg} \ f \) by (rule f-max)
finally show \( d \leq \text{poly-deg} \ f \).

qed

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

finally show \(?thesis\).

next

assume \(1 < \text{card} X\)

hence \(\text{poly-deg} g \leq \text{Dube-aux} \((\text{card} X)\) \((\text{poly-deg} f)\) 1\)

using \(\text{assms}(1, 2) \cdot f \in F\) \(\text{assms}(3) \cdot f\text{-max} :0 < \text{poly-deg} f\) \(<\text{ideal}\ \{f\} \neq \text{ideal}\ F\) \(\text{assms}(4)\)

by \((\text{rule} \text{Dube-aux})\)

also from \(\langle 1 < \text{card} X \rangle :0 < \text{poly-deg} f \rangle\) have \(\ldots = \text{Dube} \((\text{card} X)\) \((\maxdeg F)\)\)

finally show \(?thesis\).

qed

qed

qed

qed

\textbf{corollary} Dube-is-hom-GB-bound:

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

by \((\text{intro} \text{is-hom-GB-boundI} \text{Dube})\)

end

\textbf{corollary} Dube-indets:

\textbf{assumes} finite \(F\) and \(\bigwedge f. f \in F \implies \text{homogeneous} f\) and \(g \in \text{punit}\text{-reduced-GB} F\)

\textbf{shows} poly-deg \(g \leq \text{Dube} \((\text{card} \bigcup F \text{indets})\) \((\maxdeg F)\)\)

using \(-\ assms(1) -\ assms(2, 3)\)

\textbf{proof} \((\text{rule} \text{Dube})\)

from assms show finite \((\bigcup F \text{indets})\) by \((\text{simp add: finite-indets})\)

next

show \(F \subseteq P[\bigcup F \text{indets}]\ by \(\text{auto simp: Polys-alt})\)

qed

\textbf{corollary} Dube-is-hom-GB-bound-indets:

\(\text{finite} F \implies \text{is-hom-GB-bound} F \((\text{Dube} \((\text{card} \bigcup F \text{indets})\) \((\maxdeg F)\))\)\)

by \((\text{intro} \text{is-hom-GB-boundI} \text{Dube-indets})\)

end

\textbf{hide-const} (open) \(\text{pm-powerprod.a}\ \text{pm-powerprod.b}\)

\textbf{context} extended-ord-\textbf{pm-powerprod}\n
begin

\textbf{lemma} Dube-is-GB-cofactor-bound:

\textbf{assumes} finite \(X\) and \(\text{finite} F\) and \(F \subseteq P[X]\)

\textbf{shows} is-GB-cofactor-bound \(F \((\text{Dube} \left(\text{Suc} \left(\text{card} X\right)\right) \((\maxdeg F)\))\)\)

\textbf{end}
using assms(1, 3)

proof (rule hom-GB-bound-is-GB-cofactor-bound)
let ?F = homogenize None ' extend-indets ' F
let ?X = insert None (Some ' X)
from assms(1) have finite ?X by simp
moreover from assms(2) have finite ?F by (intro finite-imageI)
multiply have ?F ⊆ P[?X]
proof
  fix f'
  assume f' ∈ ?F
  then obtain f where f ∈ F and f' = homogenize None (extend-indets f)
  by blast
from this(1) assms(3) have f ∈ P[X] ..
  hence extend-indets f ∈ P[Some ' X] by (auto simp: Polys-alt indets-extend-indets)
  thus f' ∈ 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)
multiply have maxdeg ?F = maxdeg F
proof
  have maxdeg ?F = maxdeg (extend-indets ' F)
  by (auto simp: indets-extend-indets intro: maxdeg-homogenize)
  also have .. = maxdeg F by (simp add: maxdeg-def image-image)
  finally show maxdeg ?F = maxdeg F.
qed

moreover from assms(1) have card ?X = card X + 1 by (simp add: card-image)
ultiultimately 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 ⊆ P[X]
obtains G where punt.iss-Groebner-basis G and ideal G = ideal F and G ⊆ P[X]
  and ∀g. g ∈ G = (3 f ∈ F. q f * f) ∧ (∀f. q f ∈ P[X] ∧ poly-deg (q f * f) ≤ Dube (Suc (card X)) (maxdeg F)) ∧
  (f ′ ∉ F → 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 punt.iss-Groebner-basis G and ideal G = ideal F
  and G ⊆ P[X]
  and 1: ∀g. g ∈ G = (3 f'. q. finite F' ∧ F' ⊆ F ∧ q = (3 f ∈ F'. q f * f') ∧
    (∀f. q f ∈ P[X] ∧ poly-deg (q f * f) ≤ ?b ∧ (f ′ ∉ F' → q f = 0))
  by (rule is-GB-cofactor-boundE-Polys) blast
from this(1−3) show \( ? \)thesis

proof
fix \( g \)
assume \( g \in G \)
hence \( \exists F', q. \ finite \ F' \land F' \subseteq F \land \ g = (\sum_{f \in F'} q \ f \ f) \land \ 
(\forall f. \ q \ f \in P[X] \land poly-deg (q \ f \ f) \leq ?b \land (f \notin F' \rightarrow \ q f = 0)) \)
by (rule 1)
then obtain \( F', q \) where \( F' \subseteq F \) and \( g = (\sum_{f \in F'} q \ f \ f) \land \bigwedge f. \ q f \)
\( \in P[X] \)
and \( \bigwedge f. \ poly-deg (q \ f \ f) \leq ?b \) and \( 2: \ \bigwedge f. \ f \notin F' \rightarrow q f = 0 \) by blast
show \( \exists q. \ g = (\sum f \in F. q f \ f) \land (\forall f. \ q f \in P[X] \land poly-deg (q f \ f) \leq ?b \land (f \notin F \rightarrow q f = 0)) \)
proof (intro exI allI conjI impI)
from assms(2) (\( F' \subseteq F \)) have \( (\sum f \in F'. q f \ f) = (\sum f \in F. q f \ f) \)
proof (intro sum_mono_neutral_left ballI)
fix \( f \)
assume \( f \in F \land \neg f \in F' \)
hence \( f \notin F' \) by simp
hence \( q f = 0 \) by (rule 2)
thus \( q f = 0 \) by simp
thus \( g = (\sum f \in F. q f \ f) \) by (simp only: g)
next
fix \( f \)
assume \( f \notin F \)
with \( F' \subseteq F \) have \( f \notin F' \) by blast
thus \( q f = 0 \) by (rule 2)
qed fact+
qed

next
fix \( f \)
assume \( f \notin F \)
with \( F' \subseteq F \) have \( f \notin F' \) by blast
thus \( q f = 0 \) by (rule 2)
qed

end

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

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


context extended-ord-pm-powerprod
begin

theorem thm-2-3-6-Dube:
  assumes finite X and set fs ⊆ P[X]
  shows punit.is-Groebner-basis (set (punit.Macaulay-list
    (deg-shifts X (Dube (Suc (card X)) (maxdeg (set
     fs)))) fs)))
  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 ⊆ P[X]
  shows 1 ∈ ideal (set fs) ⟷ 1 ∈ set (punit.Macaulay-list (deg-shifts X (Dube (Suc (card X)) (maxdeg (set
    fs)))) fs))
  using assms Dube-is-GB-cofactor-bound by (rule thm-2-3-7) (simp-all add: assms)

theorem thm-2-3-6-indets-Dube:
  fixes fs
  defines X ≡ UNION (set fs) indets
  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 ≡ UNION (set fs) indets
  shows 1 ∈ ideal (set fs) ⟷ 1 ∈ set (punit.Macaulay-list (deg-shifts X (Dube (Suc (card X)) (maxdeg (set
     fs)))) fs))
  unfolding X-def using Dube-is-GB-cofactor-bound-indets by (rule thm-2-3-7-indets) (fact finite-set)
12.2 Preparations

primrec remdups-wrt-rev :: ('a ⇒ 'b) ⇒ 'a list ⇒ 'b list ⇒ 'a list where
  remdups-wrt-rev f [] vs = [] |
  remdups-wrt-rev f (x # xs) vs =
    (let fx = f x in if List.member vs fx then remdups-wrt-rev f xs vs else x #
     (remdups-wrt-rev f xs (fx # vs))))

lemma remdups-wrt-rev-notin: v ∈ set vs ⟷ v ∉ f ' set (remdups-wrt-rev f xs vs)
proof (induct xs arbitrary: vs)
  case Nil
  show ?case by simp
next
  case (Cons x xs)
  from Cons(2) have 1: v ∉ f ' set (remdups-wrt-rev f xs vs) by (rule Cons(1))
  from Cons(2) have v ∈ set (f x # vs) by simp
  hence 2: v ∉ f ' set (remdups-wrt-rev f xs (f x # vs)) by (rule Cons(1))
  from Cons(2) show ?case by (auto simp: Let-def 1 2 List.member-def)
qed

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

lemma map-of-remdups-wrt-rev':
  map-of (remdups-wrt-rev f (fst vs xs)) k = map-of (filter (λx. fst x ∉ set vs) xs) k
proof (induct xs arbitrary: vs)
  case Nil
  show ?case by simp
next
  case (Cons x xs)
  show ?case
proof (simp add: Let-def List.member-def Cons, intro impI)
  assume k ≠ fst x
  have map-of (filter (λy. fst y ≠ fst x ∧ fst y ∉ set vs) xs) =
    map-of (filter (λy. fst y ≠ fst x) (filter (λy. fst y ∉ set vs) xs))
    by (simp only: filter-filter conj-commute)
  also have ... = map-of (filter (λy. fst y ∉ set vs) xs) |' {y. y ≠ fst x} by (rule map-of-filter)
  finally show map-of (filter (λy. fst y ≠ fst x ∧ fst y ∉ set vs) xs) k =
    map-of (filter (λy. fst y ∉ set vs) xs) k
### 12.2.1 Connection between \((x \Rightarrow_0 a) \Rightarrow_0 b\) and \((x, a) \text{pp} \Rightarrow_0 b\)

**Definition**

\[ \text{keys-pp-to-list :: ('}x::\text{linorder}, a::\text{zero}) \text{pp} \Rightarrow_0 'x \text{ list} \]

**Where**

\[ \text{keys-pp-to-list t} = \text{sorted-list-of-set} (\text{keys-pp} t) \]

**Lemma**

\[ \text{inj-PP: inj PP} \]

**By**

\[ \text{(simp add: PP-inject inj-def)} \]

**Lemma**

\[ \text{inj-mapping-of: inj mapping-of} \]

**By**

\[ \text{(simp add: mapping-of-inject inj-def)} \]

**Lemma**

\[ \text{mapping-of-comp-PP [simp]:} \]

\[ \text{mapping-of} \circ \text{PP} = (\lambda x. x) \]

\[ \text{PP} \circ \text{mapping-of} = (\lambda x. x) \]

**By**

\[ \text{(simp-all add: comp-def PP-inverse mapping-of-inverse)} \]
lemma map-key-PP-mapping-of [simp]: Poly-Mapping.map-key PP (Poly-Mapping.map-key mapping-of p) = p
  by (simp add: map-key-compose[of inj-PP inj-mapping-of] comp-def PP-inverse map-key-id)

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

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

lemma lookup-map-key-PP: lookup (Poly-Mapping.map-key PP p) t = lookup p (PP t)
  by (simp add: map-key.rep-eq inj-PP)

lemma keys-map-key-PP: keys (Poly-Mapping.map-key PP p) = mapping-of ' keys p
  by (simp add: keys-map-key inj-PP)

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

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

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

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

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

lemma map-key-PP-monom-mult-punit:
  Poly-Mapping.map-key PP (monom-mult-punit c t p) =
monom-mult-punit (c (mapping-of t)) (Poly-Mapping.map-key PP p)
by (rule poly-mapping-eqI)
(simp add: punit.lookup-monom-mult monom-mult-punit-def adds-pp-iff PP-inverse
lookup-map-key-PP
mapping-of-inverse flip: minus-pp.abs-eq)

lemma map-key-PP-times:
Poly-Mapping.map-key PP (p * q) =
Poly-Mapping.map-key PP p * Poly-Mapping.map-key PP (q:(-, --::add-linorder)
pp ⇒₀ -)
by (induct p rule: poly-mapping-plus-induct)
(simp-all add: distrib-right map-key-PP-plus times-monomial-left map-key-PP-monom-mult-punit
flip: monom-mult-punit-def)

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

lemma map-key-PP-ideal:
Poly-Mapping.map-key PP ' ideal F = ideal (Poly-Mapping.map-key PP ' (F::((-,
-::add-linorder) pp ⇒₀ -) set))
proof −
from map-key-PP-mapping-of have surj (Poly-Mapping.map-key PP) by (rule
surj)
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 develop-
ment, and

• the executable representation by pp for code generation.

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

sublocale gd-powerprod ..

sublocale pp-pm: extended-ord-pm-powerprod λs t. ord (PP s) (PP t) λs t. ord-strict
(PP s) (PP t)

244
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 ⇒ 'a::zero ⇒ nat)
where poly-deg-pp p = (if p = 0 then 0 else max-list (map deg-pp (punit.keys-to-list p)))

primrec deg-le-sect-pp-aux :: 'x list ⇒ nat ⇒ (('x, nat) pp ⇒ 0 nat) ⇒ 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 (λx. (+) (monom-mult-punit 1 (single-pp x 1) p)) xs 0)

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

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

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

lemma map-PP-insort:
map PP (pp-pm.ordered-powerprod-lin.insort x xs) = ordered-powerprod-lin.insort (PP x) (map PP xs)
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 i T)
proof (induct T rule: infinite-finite-induct)
case (infinite T)
moreover from inj-PP subset-UNIV have inj-on PP T by (rule inj-on-subset)
ultimately show ?case by (simp add: inj-PP finite-image-iff)
next
case empty
show ?case by simp
next
case (insert t T)
moreover from insert(2) have PP t ∉ PP i T by (simp add: PP-inject image-iff)
ultimately show \(\text{case}\) by (simp add: map-PP-insort)
qed

lemma \text{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 \text{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 \(\text{map mapping-of (punit.pps-to-list T)} = \text{map mapping-of (punit.pps-to-list (PP ' mapping-of ' T))}\)
    by (simp add: image-comp)
  also have \(\ldots = \text{map mapping-of (map PP (pp-pm.punit.pps-to-list (mapping-of ' T))}\})
    by (simp only: map-PP-pps-to-list)
  also have \(\ldots = pp-pm.punit.pps-to-list (mapping-of ' T)\) by simp
finally show \(?\text{thesis}\).
qed

lemma \text{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 \text{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 \text{poly-deg-map-key-PP}: poly-deg (Poly-Mapping.map-key PP p) = poly-deg-pp p
proof –
  \{
  assume p \neq 0
  hence \(\text{map deg-pp (punit.keys-to-list p)} \neq []\)
    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 \(?\text{thesis}\).
  by (simp add: poly-deg-def poly-deg-pp-def keys-map-key-PP image-image flip: deg-pp.rep-eq)
qed

lemma \text{deg-le-sect-pp-aux-1}:
assumes $t \in \text{keys} \ (\text{deg-le-sect-pp-aux} \ \text{xs} \ \text{n})$

shows $\text{deg-pp} \ t \leq n$ and $\text{keys-pp} \ t \subseteq \text{set} \ \text{xs}$

proof –

from assms have $\text{deg-pp} \ t \leq n \land \text{keys-pp} \ t \subseteq \text{set} \ \text{xs}$

proof (induct $n$ arbitrary: $t$)

case 0

thus $\text{?case by} \ (\text{simp-all add:} \ \text{keys-pp}\text{-eq} \ \text{zero-pp}\text{-rep-eq})$

next

case $(\text{Suc} \ n)$

define $X$ where $X = \text{set} \ \text{xs}$

define $q$ where $q = \text{deg-le-sect-pp-aux} \ \text{xs} \ \text{n}$

have 1: $s \in \text{keys} \ q \implies \text{deg-pp} \ s \leq n \land \text{keys-pp} \ s \subseteq X$ for $s$ unfolding $q$-def

X-def by (fact Suc.hyps)

note Suc.prems

also have $\text{keys} \ (\text{deg-le-sect-pp-aux} \ \text{xs} \ (\text{Suc} \ n)) \subseteq \text{keys} \ q \cup$

keys $(\text{foldr} \ (\text{xx.} \ (+) \ (\text{monom-mult-punit} \ 1 \ (\text{single-pp} \ x \ 1) \ q)) \ \text{xs} \ \text{0})$

(is - $\subseteq$ - keys $(\text{foldr} \ ?r \ \text{xs} \ \text{0})$) by (simp add: Let-def Poly-Mapping.keys-add

finally show $\text{?case}$

proof

assume $t \in \text{keys} \ q$

hence $\text{deg-pp} \ t \leq n \land \text{keys-pp} \ t \subseteq \text{set} \ \text{xs}$ unfolding $q$-def by (rule Suc.hyps)

thus $\text{?thesis by} \ \text{simp}$

next

assume $t \in \text{keys} \ (\text{foldr} \ ?r \ \text{xs} \ \text{0})$

moreover have $\text{set} \ \text{xs} \subseteq X$ by (simp add: X-def)

ultimately have $\text{deg-pp} \ t \leq \text{Suc} \ n \land \text{keys-pp} \ t \subseteq X$

proof (induct $\text{xs}$ arbitrary: $t$)

case Nil

thus $\text{?case by} \ \text{simp}$

next

case $(\text{Cons} \ x \ \text{xs})$

from Cons.prems(2) have $x \in X \land \text{set} \ \text{xs} \subseteq X$ by simp-all

note Cons.prems(1)

also have $\text{keys} \ (\text{foldr} \ ?r \ (x \ # \ \text{xs}) \ \text{0}) \subseteq \text{keys} \ (\text{?r} \ x \ \text{0}) \cup \text{keys} \ (\text{foldr} \ ?r \ \text{xs} \ \text{0})$

by (simp add: Poly-Mapping.keys-add)

finally show $\text{?case}$

proof

assume $t \in \text{keys} \ (\text{?r} \ x \ \text{0})$

also have $\ldots = (+) \ (\text{single-pp} \ x \ 1) \ \text{'} \ \text{keys} \ q$

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 $\text{deg-pp} \ s \leq n \land \text{keys-pp} \ s \subseteq X$ by (rule 1)

with $x \in X$ show $\text{?thesis}$

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 $t \in \text{keys} \ (\text{foldr} \ ?r \ \text{xs} \ \text{0})$

thus $\text{deg-pp} \ t \leq \text{Suc} \ n \land \text{keys-pp} \ t \subseteq X$ using $\text{set} \ \text{xs} \subseteq X$ by (rule
Cons.hyps)

qed

thus ?thesis by (simp only: X-def)

qed

thus deg-pp t ≤ n and keys-pp t ⊆ set xs by simp-all

qed

lemma deg-le-sect-pp-aux-2:

assumes deg-pp t ≤ n and keys-pp t ⊆ set xs

shows t ∈ keys (deg-le-sect-pp-aux xs n)

using assms

proof (induct n arbitrary: t)

case 0

thus ?thesis by refl

next

case (Suc n)

have foldr: foldr (λx. (+) (f x)) ys 0 + y = foldr (λx. (+) (f x)) ys y

for f ys and y::'z::monoid-add by (induct ys) (simp-all add: ac-simps)

define q where q = deg-le-sect-pp-aux xs n

from Suc.prems(1) have deg-pp t ≤ n ∨ deg-pp t = Suc n by auto

thus ?case

proof

assume deg-pp t ≤ n

hence t ∈ keys q unfolding q-def using Suc.prems(2) by (rule Suc.hyps)

hence 0 < lookup q t by (simp add: in-keys-iff)

also have ... ≤ lookup (deg-le-sect-pp-aux xs (Suc n)) t

by (simp add: Let-def lookup-add flip: q-def)

finally show ?thesis by (simp add: in-keys-iff)

next

assume eq: deg-pp t = Suc n

hence keys-pp t ≠ {} by (auto simp: keys-pp.rep-eq deg-pp.rep-eq)

then obtain x where x ∈ keys-pp t by blast

with Suc.prems(2) have x ∈ set xs ..

then obtain xs1 xs2 where xs: xs = xs1 @ x # xs2 by (meson split-list)

define s where s = t − single-pp x 1

from x ∈ keys-pp t have single-pp x 1 adds t


le-fun-def

lookup-single when-def in-keys-iff)

hence s + single-pp x 1 = (t + single-pp x 1) − single-pp x 1

unfolding s-def by (rule minus-plus)

hence t: t = single-pp x 1 + s by (simp add: add.commute)

with eq have deg-pp s ≤ n by (simp add: deg-pp-plus deg-pp-single)

moreover have keys-pp s ⊆ set xs

proof (rule subset-trans)

from Suc.prems(2) (x ∈ set xs) show keys-pp t ∪ keys-pp (single-pp x (Suc 0)) ⊆ set xs

248
by (simp add: keys-pp.rep-eq single-pp.rep-eq)

ultimately have \( s \in \text{keys } q \) unfolding q-def by (rule Suc.hyps)

hence \( t \in \text{keys } (\text{monom-mul-unit } 1 \ (\text{single-pp } x \ 1) \ q) \)

by (simp add: monom-mul-unit-def punit.keys-monom-mul t)

hence \( 0 < \text{lookup } (\text{monom-mul-unit } 1 \ (\text{single-pp } x \ 1) \ q) \ t \) by (simp add: in-keys-iff)

also have \( \ldots \leq \text{lookup } (q + (\text{foldr } (\lambda x. (+) (\text{monom-mul-unit } 1 \ (\text{single-pp } x \ 1) \ q)) \ xs1 \ 0 + (\text{monom-mul-unit } 1 \ (\text{single-pp } x \ 1) \ q) + \text{foldr } (\lambda x. (+) (\text{monom-mul-unit } 1 \ (\text{single-pp } x \ 1) \ q)) \ xs2 \ 0)) \ t \)

by (simp add: lookup-add)

also have \( \ldots = \text{lookup } (\text{deg-sect-pp-aux } xs \ (\text{Suc } n)) \ t \)

by (simp add: Let-def foldr flip: q-def, simp add: xs)

finally show \( ?\text{thesis} \) by (simp add: in-keys-iff)

qed

lemma keys-deg-le-sect-pp-aux:

\[
\text{keys } (\text{deg-le-sect-pp-aux } xs \ n) = \{ t. \ \text{deg-pp } t \leq n \land \text{keys-pp } t \subseteq \text{set } xs \}
\]

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

lemma deg-le-sect-deg-le-sect-pp:

\[
\text{map } PP \ (\text{pp-pm.punit.pps-to-list } (\text{deg-le-sect } (\text{set } xs) \ d)) = \text{deg-le-sect-pp } xs \ d
\]

proof –

have \( PP' \cdot \{ t. \ \text{deg-pm } t \leq d \land \text{keys } t \subseteq \text{set } xs \} = PP' \cdot \{ t. \ \text{deg-pp } (PP \ t) \leq d \land \text{keys-pp } (PP \ t) \subseteq \text{set } xs \} \)

by (simp only: keys-pp.abs-eq deg-pp.abs-eq)

also have \( \ldots = \{ t. \ \text{deg-pp } t \leq d \land \text{keys-pp } t \subseteq \text{set } xs \} \)

proof (intro subset-antisym subsetI)

fix \( t \)

assume \( t \in \{ t. \ \text{deg-pp } t \leq d \land \text{keys-pp } t \subseteq \text{set } xs \} \)

moreover have \( t = PP' (\text{mapping-of } t) \) by (simp only: mapping-of-inverse)

ultimately show \( t \in PP' \cdot \{ t. \ \text{deg-pp } (PP \ t) \leq d \land \text{keys-pp } (PP \ t) \subseteq \text{set } xs \} \)

by auto

qed auto

finally show \( ?\text{thesis} \)

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:

\[
\text{pp-pm.deg-shifts } (\text{set } xs) \ d \ (\text{map } (\text{Poly-Mapping.map-key } PP) \ fs) = \text{map } (\text{Poly-Mapping.map-key } PP) \ (\text{deg-shifts-pp } xs \ d \ fs)
\]


249
lemma ideal-deg-shifts-pp: ideal (set (deg-shifts-pp xs d fs)) = ideal (set fs)
proof
  have ideal (set (deg-shifts-pp xs d fs)) =
    Poly-Mapping.map-key mapping-of ' Poly-Mapping.map-key PP ' ideal (set (deg-shifts-pp xs d fs))
    by (simp add: image-comp)
  also have ... = Poly-Mapping.map-key mapping-of ' ideal (set (map (Poly-Mapping.map-key PP) (deg-shifts-pp xs d fs))))
    by (simp add: map-key-PP-ideal)
  also have ... = Poly-Mapping.map-key mapping-of ' ideal (Poly-Mapping.map-key PP ' set fs)
    by (simp flip: deg-shifts-deg-shifts-pp)
  also have ... = Poly-Mapping.map-key mapping-of ' Poly-Mapping.map-key PP ' ideal (set fs)
    by (simp only: map-key-PP-ideal)
  also have ... = ideal (set fs) by (simp add: image-comp)
  finally show ?thesis .
qed

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

lemma poly-to-row-map-key-PP: poly-to-row (map pp.mapping-of xs) (Poly-Mapping.map-key PP p) = poly-to-row xs p
by (simp add: 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 $i1$: $i < \text{length } \text{ts}$ and $t1$: $t = \text{ts}!_i$ by (metis in-set-conv-nth)

hence $i2$: $i < \text{length } \text{ts}$ and $t2$: $PP \ t = ts!_i$ by (simp-all add: mapping-of-inverse)

have unfolding $t1$ using $1 \ 2$ $i1$ by (rule punit.lookup-row-to-poly)

moreover have unfolding $t1$ using $assms$ $i2$ by (rule punit.lookup-row-to-poly)

ultimately show ?thesis by simp

next

case False

have $PP \ t \notin set \ ts$

proof

  assume $PP \ t \in set \ ts$

  hence mapping-of $(PP \ t) \in mapping-of \ set \ ts$ by (rule imageI)

  with False show False by (simp add: PP-inverse)

  qed

  with punit.keys-row-to-poly have unfolding $t1$ using $PP \ t \in set \ ts$

  with $pp-pm$.punit.keys-row-to-poly $t2$ using $assms$ $i2$ by (rule punit.lookup-row-to-poly)

  moreover from False punit.keys-row-to-poly have unfolding $t1$ using $assms$ $i2$

  ultimately show ?thesis by simp

  qed

lemma mat-to-polys-mapping-of:

  assumes distinct ts and dim-col m = length ts

  shows mat-to-polys $(map \ pp.mapping-of \ ts) \ m = map \ (Poly-Mapping.map-key PP) \ (mat-to-polys \ ts \ m)$

proof

{ fix $r$

  assume $r \in set \ (rows \ m)$

  then obtain $i$ where $r = \text{row } m \ i$ by (auto simp: rows-def)

  hence dim-vec $r = \text{length } \text{ts}$ by (simp add: assms(1))

  with $assms$ $1$ have unfolding $t1$ using $pp-pm$.punit.keys-row-to-poly $r = Poly-Mapping.map-key PP \ (row-to-poly \ ts \ r)$

  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

$punit.distinct-Keys-to-list \ punit.length-Keys-to-list)$

251
lemma \text{lpp-map-key-PP}: \text{pp-pm.lpp} (\text{Poly-Mapping.map-key PP p}) = \text{mapping-of} (\text{lpp p})

proof (cases \( p = 0 \))
  case True
  thus \( ?\text{thesis} \) by (simp add: zero-pp.rep-eq)
next
  case False
  show \( ?\text{thesis} \) proof (rule pp-pm.punit.lt-eqI-keys)
    show \( \text{pp.mapping-of} (\text{lpp p}) \in \text{keys} (\text{Poly-Mapping.map-key PP p}) \) unfolding keys-map-key-PP
      by (intro imageI punit.lt-in-keys False)
  next
  fix s
  assume \( s \in \text{keys} (\text{Poly-Mapping.map-key PP p}) \)
  then obtain \( t \) where \( t \in \text{keys} p \) and \( s = \text{mapping-of} t \) unfolding keys-map-key-PP ..
  thus \( \text{ord} (PP s) (PP (\text{pp.mapping-of} (\text{lpp p}))) \) by (simp add: mapping-of-inverse punit.lt-max-keys)
next
qed

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

lemma \text{Dube-is-GB-cofactor-bound-pp}:
  \( \text{pp-pm.is-GB-cofactor-bound} (\text{Poly-Mapping.map-key PP ' set fs}) \)
proof (cases fs = [])

  case True
  show ?thesis by (rule pp-pm.is-GB-cofactor-boundI-subset-zero) (simp add: True)

next

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

qed

definition GB-Macaulay-Dube :: (('x, nat) pp ⇒ 0 'a) list ⇒ (('x, nat) pp ⇒ 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)

end

  cmp-term
  rewrites punit.adds-term = (adds)
  and punit.pp-of-term = (\x. x)
  and punit.component-of-term = (\- ())
  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.rand-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

253
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 * Y^2 + 3
* X^2 * Y, Y ^ 3 - X ^ 3]) =
[X ^ 5, X ^ 3 * Y - C_0 (1 / 9) * X ^ 4, Y ^ 3 - X ^ 3, X * Y^2 + 3 * X^2
* Y]
  by eval

end
References


