Universal Pairs for Diophantine Equations

Marco David and Théo André and Mathis Bouverot-Dupuis and Eva Brenner and Loïc Chevalier and Anna Danilkin and Charlotte Dorneich and Kevin Lee and Xavier Pigé and Timothé Ringeard and Quentin Vermande and Paul Wang and Annie Yao and Zhengkun Ye and Jonas Bayer

July 29, 2025

Abstract

We formalize a universal construction of Diophantine equations with bounded complexity. This is a formalization of our own work in number theory [2].

Hilbert's Tenth Problem was answered negatively by Yuri Matiyasevich, who showed that there is no general algorithm to decide whether an arbitrary Diophantine equation has a solution[4]. However, the problem remains open when generalized to the field of rational numbers, or contrarily, when restricted to Diophantine equations with bounded complexity, characterized by the number of variables ν and the degree δ. If every Diophantine set can be represented within the bounds (ν, δ), we say that this pair is *universal*, and it follows that the corresponding class of equations is undecidable. In a separate mathematics article, we have determined the first non-trivial universal pair for the case of integer unknowns.

This AFP entry contributes the main construction required to establish said universal pair. In doing so, we markedly extend previous work on multivariate polynomials [6], and develop classical theory on Diophantine equations [5]. Additionally, our work includes metaprogramming infrastructure designed to efficiently handle complex definitions of multivariate polynomials. Our mathematical draft has been formalized while the mathematical research was ongoing, and benefitted largely from the help of the theorem prover.

Contents

Overview We provide a detailed high-level overview of this formal proof in a forthcoming paper [1]. Here we just reference the various mathematical sources that we have formalized.

The main mathematical text is our preprint "Diophantine Equations over $\mathbb{Z}$: Universal Bounds and Parallel Formalization" [2]. It contains the majority of the proofs verified here. A lot of it is based on ideas by Zhi-Wei Sun [8, 7].

Moreover, we formalize classical theory on Diophantine Equations following an article by Matyivasevich and Robinson [5]. This material can be found in the section on relation combining.

We also formalize a variety of statements on multivariate polynomials adding to the current entry on multivariate polynomials [6]. Finally, our proof relies on the Three Squares Theorem [3] which we import.

Acknowledgements This project would not exist without Yuri Matiyasevich who proposed the formalization of Hilbert's Tenth Problem and encouraged our exploration of Universal Pairs. We are deeply grateful to Dierk Schleicher for his unwavering support throughout, both on a personal level and through his mathematical insight. We thank Malte Haßler, Thomas Serafini and Simon Dubischar for their mathematical contributions to the project. We gratefully acknowledge support from the Département de mathémathiques et applications at École Normale Supérieure de Paris.

theory *Notation*
imports *Polynomials.More-MPoly-Type Complex-Main HOL−Library.Rewrite*
begin

1 Multivariate Polynomials

1.1 Elementary properties

1.1.1 Notation

notation *smult* (**infixl** $*_s$ *70*)

definition *max-coeff* :: *'a*::{*zero*,*abs*,*linorder*} *mpoly* ⇒ *'a* **where**
max-coeff P ≡ *Max* (*abs* ‘ *MPoly-Type.coeffs P*)

lemma *coeffs-empty-iff*: *coeffs P* = {} ⟷ *P* = *0*
proof
assume *coeffs P* = {} **thus** *P* = *0*
unfolding *coeffs-def*
by *simp* (*metis emptyE equals0I in-keys-lookup-in-range keys-eq-empty mapping-of-inject*

zero-mpoly.rep-eq)
next
assume *P* = *0* **thus** *coeffs P* = {}
by (*simp add*: *coeffs.abs-eq zero-mpoly.abs-eq*)
qed

lemma *coeff-minus*: *coeff p m* − *coeff q m* = *coeff* (*p*−*q*) *m*
by (*simp add*: *coeff-def lookup-minus minus-mpoly.rep-eq*)

definition *nth0* :: *'a*::*zero list* ⇒ *nat* ⇒ *'a* (**infixl** $!_0$ *100*) **where**
xs $!_0$ *i* = (*xs* ! *i when i* < *length xs*)

lemma *nth0-nth* : *i* < *length xs* ⟹ *xs* $!_0$ *i* = *xs* ! *i*
unfolding *nth0-def* **by** *auto*

lemma *nth0-0*: *i* ≥ *length xs* ⟹ *xs* $!_0$ *i* = *0*
unfolding *nth0-def* **by** *auto*

lemma *nth0-Cons*: (*x* # *xs'*) $!_0$ *i* = (*case i of 0* ⇒ *x* | *Suc i'* ⇒ *xs'* $!_0$ *i'*)
unfolding *nth0-def nth-Cons* **by** (*cases i*; *auto*)

lemma *nth0-Cons-0* [*simp*, *code*]: (*x* # *xs*) $!_0$ *0* = *x*
unfolding *nth0-def* **by** *auto*

lemma *nth0-Cons-Suc* [*simp*, *code*]: (*x* # *xs*) $!_0$ (*Suc n*) = *xs* $!_0$ *n*
unfolding *nth0-def* **by** *auto*

lemma *nth0-finite*[*simp*]: *finite* {*i*. *xs* $!_0$ *i* ≠ *0*}

```
  unfolding nth0-def by auto

lemma nth0-inj: length xs = length ys ⟹ (!₀) xs = (!₀) ys ⟹ xs = ys
proof −
  have aux:
    length xs = n ⟹ length ys = n ⟹ (!₀) xs = (!₀) ys ⟹ xs = ys for n
  proof (induction n arbitrary: xs ys)
    case 0
    thus ?case by auto
  next
    case (Suc n)
    from Suc obtain x xs' where xs-def: xs = x # xs' by (metis Suc-length-conv)
    from Suc obtain y ys' where ys-def: ys = y # ys' by (metis Suc-length-conv)
    have 0: x = y using Suc unfolding xs-def ys-def nth0-Cons
      by (metis nat.case(1))
    have 1: length xs' = n length ys' = n using xs-def ys-def Suc by auto
    have 2: (!₀) xs' = (!₀) ys' using Suc
      unfolding xs-def ys-def nth0-Cons
      by (simp; metis nth0-nth nth-equalityI nat.case(2))
    from 1 2 have 3: xs' = ys' using Suc by auto
    from 0 3 show ?case unfolding xs-def ys-def by auto
  qed
  assume length xs = length ys (!₀) xs = (!₀) ys
  thus ?thesis using aux by auto
qed

lemma nth0-sum-list: sum-list i = (∑ v | i !₀ v ≠ 0. i !₀ v)
proof (induction i)
  case Nil
  thus ?case unfolding nth0-def by auto
next
  case (Cons a i)
  have 0: (∑ v | i !₀ v ≠ 0. i !₀ v) =
    (∑ v∈Suc ‘ {v. i !₀ v ≠ 0}. i !₀ (v − 1))
    by (metis (no-types, lifting) add-diff-cancel-left' inj-Suc plus-1-eq-Suc
      sum.reindex-cong)
  show ?case proof cases
    assume 1: a = 0
    have 2: {v. v ≠ 0 ∧ (v ≠ 0 ⟶ i !₀ (v − 1) ≠ 0)} ∩ − {0} =
      Suc ‘ {v. i !₀ v ≠ 0} unfolding image-def by auto
    show ?case using nth0-finite[of a # i] Cons 0 1 2
      unfolding nth0-Cons Nitpick.case-nat-unfold by (simp add: sum.If-cases)
  next
    assume 3: a ≠ 0
    have 4: {v. v ≠ 0 ⟶ i !₀ (v − 1) ≠ 0} ∩ − {0} =
      Suc ‘ {v. i !₀ v ≠ 0} unfolding image-def by auto
    show ?case using nth0-finite[of a # i] Cons 0 3 4
      unfolding nth0-Cons Nitpick.case-nat-unfold by (simp add: sum.If-cases)
  qed
```

qed

lemma *lookup-Abs-poly-mapping-nth0*[*simp*]:
lookup (*Abs-poly-mapping* ((!$_0$) *xs*)) = (!$_0$) *xs*
by *auto*

lemma *Abs-poly-mapping-nth0-single*[*simp*]:
Abs-poly-mapping ((!$_0$) [*x*]) = *Poly-Mapping.single 0 x*
unfolding *Poly-Mapping.single-def nth0-def*
by (*simp*; *metis* (*mono-tags*, *opaque-lifting*) *nth-Cons-0 when-simps*(*2*))

lemma *Abs-poly-mapping-nth0-append-single*[*simp*]:
Abs-poly-mapping ((!$_0$) (*xs* @ [*x*])) =
Abs-poly-mapping ((!$_0$) *xs*) + *Poly-Mapping.single* (*length xs*) *x*
unfolding *Poly-Mapping.single-def plus-poly-mapping-def*
nth0-def nth-append when-def
by (*simp*; *rule cong*[*of Abs-poly-mapping*, *OF refl*]; *rule*; *simp*)

lemma *Sum-any-rev-image*:
assumes *finite* {*x*. *f x* ≠ *0*}
shows *Sum-any* (λ*m*. *Sum-any* (λ*x*. *f x when g x* = *m*)) = *Sum-any f*
proof −
have *0*: (∑ *x* | (*f x when g x* = *m*) ≠ *0*. *f x when g x* = *m*) =
(∑ *x* | *f x* ≠ *0*. *f x when g x* = *m*) **for** *m*
apply (*rule sum.mono-neutral-left*)
using *assms* **apply** *auto*
done

have *1*: (∑ *x* | *f x* ≠ *0*. *f x when g x* = *m*) ≠ *0* ⟹ (∃ *x*∈{*x*. *f x* ≠ *0*}. *g x* = *m*) **for** *m*
by (*meson sum.neutral when-neq-zero*)

have *Sum-any* (λ*m*. *Sum-any* (λ*x*. *f x when g x* = *m*)) =
(∑ *m* | (∑ *x* | *f x* ≠ *0*. *f x when g x* = *m*) ≠ *0*. ∑ *x* | *f x* ≠ *0*. *f x when g x* = *m*)
unfolding *Sum-any.expand-set 0* **..**
also have ... = (∑ *m*∈*g* ‘ {*x*. *f x* ≠ *0*}. ∑ *x* | *f x* ≠ *0*. *f x when g x* = *m*)
apply (*rule sum.mono-neutral-left*)
subgoal using *assms* **by** *simp*
subgoal using *1* **by** *blast*
subgoal by *auto*
done
also have ... = (∑ *m*∈*g* ‘ {*x*. *f x* ≠ *0*}. *sum f* {*x* ∈ {*x*. *f x* ≠ *0*}. *g x* = *m*})
apply (*rule arg-cong2*[*of - - - - sum*, *OF - refl*])
apply (*rule HOL.ext*)
apply (*rule sum.mono-neutral-cong*)
subgoal using *assms* **by** *simp*
subgoal using *assms* **by** *simp*
subgoal by *simp*

subgoal by *auto*
subgoal by *simp*
done
also have ... = *sum f* {*x*. *f x* ≠ *0*}
using *sum.image-gen*[*OF assms*, *of f*, *symmetric*] **.**
also have ... = *Sum-any f*
unfolding *Sum-any.expand-set* **..**
finally show *?thesis* **.**
qed

lemma *Sum-any-rev-image-add*:
assumes *finite* {(m_1, m_2). *f* m_1 m_2 ≠ *0*}
shows *Sum-any* (λ*m*. (*Sum-any* (λm_1. *Sum-any* (λm_2. *f* m_1 m_2 *when* *m* = m_1 + m_2))))
= *Sum-any* (λm_1. (*Sum-any* (λm_2. *f* m_1 m_2)))
proof −
have *finite* (*fst* ‘ {(m_1, m_2). *f* m_1 m_2 ≠ *0*})
by (*rule finite-imageI*[*OF assms*])
hence *0*: *finite* {m_1. ∃ m_2. *f* m_1 m_2 ≠ *0*}
unfolding *image-def* **by** *auto*
have *finite* (*snd* ‘ {(m_1, m_2). *f* m_1 m_2 ≠ *0*})
by (*rule finite-imageI*[*OF assms*])
hence *1*: *finite* {m_2. ∃ m_1. *f* m_1 m_2 ≠ *0*}
unfolding *image-def* **by** *auto*

have *finite* ({m_1. ∃ m_2. *f* m_1 m_2 ≠ *0*} × {m_2. ∃ m_1. *f* m_1 m_2 ≠ *0*})
using *0 1* **by** *simp*
hence *3*: *finite* {(m_1, m_2). (∃ m_1. *f* m_1 m_2 ≠ *0*) ∧ (∃ m_2. *f* m_1 m_2 ≠ *0*)} (**is** *finite ?C*)
apply (*rule back-subst*)
apply (*rule arg-cong*[*of* - - *finite*])
apply *auto*
done

have *Sum-any* (λ*m*. (*Sum-any* (λm_1. *Sum-any* (λm_2. *f* m_1 m_2 *when* *m* = m_1 + m_2)))) =
Sum-any (λ*m*. *Sum-any* (λ(m_1, m_2). *f* m_1 m_2 *when* *m* = m_1 + m_2))
apply (*rule arg-cong*[*of* - - *Sum-any*])
apply (*rule ext*)
apply (*rule Sum-any.cartesian-product*[*of ?C*])
subgoal using *3* **.**
subgoal by *auto*
done
also have ... = *Sum-any* (λ(m_1, m_2). *f* m_1 m_2)
apply (*subst Sum-any-rev-image*[*of* λ(m_1, m_2). *f* m_1 m_2 λ(m_1, m_2). m_1 + m_2, *symmetric*])
subgoal using *assms* **by** (*metis* (*mono-tags*, *lifting*) *Collect-cong split-def*)
subgoal
apply (*rule arg-cong*[*of* - - *Sum-any*])

```
    apply (rule ext)
    apply (rule arg-cong[of - - Sum-any])
    apply (rule ext)
    by (smt (verit, best) case-prod-unfold)
  done
 also have ... = Sum-any (λm1. (Sum-any (λm2. f m1 m2)))
  apply (rule Sum-any.cartesian-product[of ?C, symmetric])
  subgoal using 3 .
  subgoal by auto
  done
 finally show ?thesis .
qed

lemma of-int-Sum-any:
 fixes f :: 'b ⇒ int
 assumes finite {a. f a ≠ 0}
 shows (of-int :: int ⇒ 'a::ring-1) (Sum-any f) = Sum-any (of-int ∘ f)
proof −
 have of-int (sum f {a. f a ≠ 0}) = sum ((of-int :: int ⇒ 'a) ∘ f) {a. f a ≠ 0}
  by simp
 also have ... = sum ((of-int :: int ⇒ 'a) ∘ f) {a. (of-int ∘ f) a ≠ (0 :: 'a)}
  apply (rule sum.mono-neutral-right)
  subgoal using assms by simp
  subgoal using Collect-mono-iff by fastforce
  subgoal by auto
  done
 finally show ?thesis
  unfolding Sum-any.expand-set
  .
qed

lemma of-int-Prod-any:
 fixes f :: 'b ⇒ int
 assumes finite {a. f a ≠ 1}
 shows (of-int :: int ⇒ 'a::comm-ring-1) (Prod-any f) = Prod-any (of-int ∘ f)
proof −
 have of-int (prod f {a. f a ≠ 1}) = prod ((of-int :: int ⇒ 'a) ∘ f) {a. f a ≠ 1}
  by simp
 also have ... = prod ((of-int :: int ⇒ 'a) ∘ f) {a. (of-int ∘ f) a ≠ (1 :: 'a)}
  apply (rule prod.mono-neutral-right)
  subgoal using assms by simp
  subgoal using Collect-mono-iff by fastforce
  subgoal by auto
  done
 finally show ?thesis
  unfolding Prod-any.expand-set
  .
qed
```

1.1.2 The constant polynomial

lemma *Const-zero*: *Const 0 = 0*
by (*simp add*: *Const.abs-eq Const$_0$-zero zero-mpoly-def*)
lemma *Const-one*: *Const 1 = 1*
by (*simp add*: *Const.abs-eq Const$_0$-one one-mpoly.abs-eq*)
lemma *Const-add*: *Const a + Const b = Const (a + b)*
using *Const$_0$-def monom.abs-eq monom-add* **by** (*metis Const.abs-eq*)
lemma *Const-mult*: *Const a * Const b = Const (a * b)*
using *Const$_0$-def monom.abs-eq mult-monom Const.abs-eq* **by** (*metis add-0*)
lemma *Const-power*: *Const c ^ i = Const (c ^ i)*
by (*induct i*; *auto simp add*: *Const-mult Const-one*)
lemma *Const-sub*: *Const a − Const b = Const (a − b)*
by (*metis Const.abs-eq Const$_0$-def monom.abs-eq monom-diff*)

lemma *Const-when*: *Const (a when P) = (Const a when P)*
using *Const-zero fun-when* **by** *auto*

lemma *coeff-Const-zero*: *MPoly-Type.coeff (Const c) 0 = c*
unfolding *MPoly-Type.coeff-def Const.rep-eq Const$_0$-def lookup-single*
by *simp*

lemma *Const-sum-Any*: *Const (Sum-any f) = Sum-any (Const ∘ f)*
proof −
have *0*: {*a*. *Const (f a) ≠ 0*} = {*a*. *f a ≠ 0*}
by (*metis Const-zero coeff-Const-zero*)
show *?thesis*
unfolding *Sum-any.expand-set comp-def 0*
apply (*rule infinite-finite-induct*)
apply (*auto simp add*: *Const-zero Const-add*[*symmetric*])
done
qed

lemma *Const-numeral*: *Const (numeral x) = numeral x*
unfolding *Const.abs-eq*
apply (*simp add*: *Const$_0$-numeral*)
by (*metis monom.abs-eq monom-numeral single-numeral*)

lemma *union-subset*:
fixes *A* :: *'a set*
and *B* :: *'b set*
and *f* :: *'a ⇒ 'b set*
assumes ⋀*x*. *f x ⊆ B*
shows ⋃ (*f*‘*A*) ⊆ *B*
using *assms* **by** *auto*

lemma *of-nat-Const*: *of-nat n = Const (int n)*
proof (*induction n*)
case *0* **show** *?case* **by** (*simp add*: *Const-def Const$_0$-def zero-mpoly-def*)
next

case (*Suc n*)
have *of-nat* (*Suc n*) = *1* + *of-nat n* **by** (*simp add*: *algebra-simps*)
also have ... = *1* + *Const* (*int n*) **by** (*simp add*: *Suc.IH*)
also have ... = *Const 1* + *Const* (*int n*) **by** (*simp add*: *Const-one*)
also have ... = *Const* (*1* + *int n*) **by** (*simp add*: *Const-add*)
finally show *?case* **by** (*simp add*: *algebra-simps*)
qed

lemma *of-int-Const*: *of-int x* = *Const x*
using *of-nat-Const Const-sub*
by (*metis int-diff-cases of-int-diff of-int-of-nat-eq*)

1.1.3 Finite sums and products

lemma *add-to-finite-sum*:
fixes *f*::*'b*::*comm-monoid-add*⇒*'a*::*comm-monoid-add* **and** *g*::*'c*⇒*'b*
assumes ⋀*x y*. *f* (*x*+*y*) = *f x* + *f y* **and** *f 0* = *0*
shows *finite S* ⟹ (∑ *x*∈*S*. *f* (*g x*)) = *f*(∑ *x*∈*S*. *g x*)
proof (*induction card S arbitrary*:*S*)
case *0*
then show *?case* **using** *assms*(*2*) **by** *force*
next
case (*Suc n*)
then obtain *y* **where** *y-prop*: *y*∈*S* **by** *fastforce*
define *S'* **where** *S'* = *S* − {*y*}
hence *card-S'*:*card S'* = *n* **using** *Suc* **by** (*simp add*: *y-prop*)
have *disj-un-S*: *S* = *S'*∪{*y*} ∧ *S'*∩{*y*}={}∧*finite S* **using** *y-prop S'-def Suc* **by** *force*
hence (∑ *x*∈*S*. *f* (*g x*)) = (∑ *x*∈*S'*. *f* (*g x*)) + (∑ *x*∈{*y*}. *f* (*g x*))
by (*meson finite-Un sum.union-disjoint*)
also have ... = *f*(∑ *x*∈*S'*. *g x*) + *f* (*g y*) **using** *Suc*(*1*)[*of S'*] *card-S'*
by (*metis S'-def add.commute calculation disj-un-S finite-Un sum.remove y-prop*)
also have ... = *f*((∑ *x*∈*S'*. *g x*) + *g y*) **using** *assms* **by** *simp*
also have ... = *f*(∑ *x*∈*S*. *g x*) **using** *disj-un-S*
by (*metis S'-def add.commute sum.remove y-prop*)
finally show (∑ *x*∈*S*. *f* (*g x*)) = *f* (*sum g S*) **by** *blast*
qed

lemma *mult-to-finite-prod*:
fixes *f*::*'b*::*comm-monoid-mult*⇒*'a*::*comm-monoid-mult* **and** *g*::*'c*⇒*'b*
assumes ⋀*x y*. *f* (*x*∗*y*) = *f x* ∗ *f y* **and** *f 1* = *1*
shows *finite S* ⟹ (∏ *x*∈*S*. *f* (*g x*)) = *f*(∏ *x*∈*S*. *g x*)
proof (*induction card S arbitrary*:*S*)
case *0*
then show *?case* **using** *assms*(*2*) **by** *force*
next
case (*Suc n*)
then obtain *y* **where** *y-prop*: *y*∈*S* **by** *fastforce*

define S' **where** $S' = S - \{y\}$
hence *card-S'*:*card* $S' = n$ **using** *Suc* **by** (*simp add*: *y-prop*)
have *disj-un-S*: $S = S' \cup \{y\} \wedge S' \cap \{y\} = \{\} \wedge$ *finite S* **using** *y-prop S'-def Suc* **by** *force*
hence $(\prod x \in S.\ f\ (g\ x)) = (\prod x \in S'.\ f\ (g\ x)) * (\prod x \in \{y\}.\ f\ (g\ x))$
by (*meson finite-Un prod.union-disjoint*)
also have ... $= f(\prod x \in S'.\ g\ x) * f\ (g\ y)$ **using** *Suc(1)*[*of S'*] *card-S'*
by (*metis S'-def calculation disj-un-S finite-Un mult.commute prod.remove y-prop*)
also have ... $= f((\prod x \in S'.\ g\ x) * g\ y)$ **using** *assms* **by** *simp*
also have ... $= f(\prod x \in S.\ g\ x)$ **using** *disj-un-S*
by (*metis S'-def mult.commute prod.remove y-prop*)
finally show $(\prod x \in S.\ f\ (g\ x)) = f\ (prod\ g\ S)$ **by** *blast*
qed

lemma *nat-sum-distrib*:
fixes *f*::*nat*⇒*int*
assumes *S-fin*: *finite S* **and** *nonneg*: $\bigwedge i.\ i \in S \Longrightarrow f\ i \geq 0$
shows *nat* $(\sum i \in S.\ f\ i) = (\sum i \in S.\ nat\ (f\ i))$
using *assms* **proof** (*induct rule*:*finite-induct*[*OF S-fin*])
case *1* **thus** *?case* **by** *simp*
next
case (*2 x S*)
have *nat* (*sum f* (*insert x S*)) = *nat* (*sum f S* + *f x*)
using *2* **by** *simp*
also have ... = *nat* (*sum f S*) + *nat* (*f x*)
using *2 nat-add-distrib sum-nonneg* **by** *blast*
also have ... $= (\sum i \in insert\ x\ S.\ nat\ (f\ i))$
using *2* **by** *auto*
finally show *?case* **.**
qed

1.1.4 Insertion

named-theorems *insertion-simps Lemmas about insertion*

lemma *pow-when*: $b \neq 0 \Longrightarrow a\ \hat{}\ (b\ when\ P) = (if\ P\ then\ a\ \hat{}\ b\ else\ 1)$
by *simp*

declare *insertion-add*[*simp*, *insertion-simps*]
declare *insertion-mult*[*simp*, *insertion-simps*]

lemma *insertion-Var*[*simp*, *insertion-simps*]: *insertion f* (*Var n*) = *f n*
apply (*simp add*: *insertion-def insertion-aux-def insertion-fun-def Var-def MPoly-inverse* Var_0*-def*)
apply (*simp add*: *Poly-Mapping.lookup-single when-mult pow-when*)
done

lemma *insertion-Const*[*simp*, *insertion-simps*]: *insertion f* (*Const c*) = *c*

apply (*simp add*: *insertion-def insertion-aux-def insertion-fun-def Const-def MPoly-inverse Const$_0$-def*)
apply (*simp add*: *Poly-Mapping.lookup-single when-mult*)
done

lemma *insertion-numeral*[*simp*, *insertion-simps*]: *insertion f* (*numeral n*) = *numeral n*
by (*metis insertion-single monom-numeral mult.right-neutral power-0 single-zero*)

lemma *Sum-any-neg*:
fixes *f* :: - ⇒ *'a*::*ring-1*
shows *Sum-any* (λ*a*. − *f a*) = − *Sum-any* (λ*a*. *f a*)
proof *cases*
assume *finite* {*a*. *f a* ≠ *0*}
thus *?thesis*
apply (*subst* (*1 2*) *mult-minus1*[*symmetric*])
apply (*metis Sum-any-right-distrib*)
done
next
assume *infinite* {*a*. *f a* ≠ *0*}
thus *?thesis* **by** *auto*
qed

lemma *insertion-neg*[*simp*, *insertion-simps*]:
fixes *f* :: - ⇒ *'a*::*comm-ring-1*
shows *insertion f* (− *p*) = − *insertion f p*
by (*metis insertion-add insertion-zero neg-eq-iff-add-eq-0*)

lemma *insertion-diff*[*simp*, *insertion-simps*]:
fixes *f* :: - ⇒ *'a*::*comm-ring-1*
shows *insertion f* (*p* − *q*) = *insertion f p* − *insertion f q*
by (*metis eq-diff-eq insertion-add*)

lemma *insertion-of-int*[*simp*, *insertion-simps*]:
fixes *f*::*nat* ⇒ *int* **and** *c*::*int*
shows *insertion f* (*of-int c*) = *c*
by (*simp add*: *of-int-Const*)

lemma *insertion-of-nat*[*simp*, *insertion-simps*]:
fixes *f*::*nat* ⇒ *int* **and** *n*::*nat*
shows *insertion f* (*of-nat n*) = *int n*
by (*simp add*: *of-nat-Const*)

lemma *insertion-pow*[*simp*, *insertion-simps*]: *insertion f* (*P* ^ *n*) = (*insertion f P*) ^ *n*
by (*induct n*) *simp-all*

lemma *insertion-sum*[*simp*, *insertion-simps*]:
finite S ⟹ *insertion f* (∑ *i*∈*S*. *P i*) = (∑ *i*∈*S*. *insertion f* (*P i*))

using *add-to-finite-sum insertion-add insertion-zero*
by (*metis* (*no-types*, *lifting*) *sum.cong*)

lemma *insertion-prod*[*simp*, *insertion-simps*]:
finite S $\Longrightarrow$ *insertion f* ($\prod i{\in}S.\ P\ i$) = ($\prod i{\in}S.$ *insertion f* (*P i*))
using *mult-to-finite-prod insertion-mult insertion-one*
by (*metis* (*no-types*, *lifting*) *prod.cong*)

lemma *insertion-monom*[*simp*, *insertion-simps*]:
insertion f (*monom m a*) = *a* $*$ ($\prod x.$ *f x* ^ *lookup m x*)
unfolding *insertion.rep-eq insertion-aux.rep-eq insertion-fun-def*
unfolding *mapping-of-monom lookup-single*
unfolding *when-mult Sum-any-when-equal'*
..

lemma *insertion-of-int-times* : *insertion f* (*of-int n* $*$ *P*) = *n* $*$ *insertion f P*
by *simp*

lemma *pow-positive*:
fixes *a* :: *'a*::*idom*
assumes *a* $\neq$ *0*
assumes *n* > *0*
shows *a* ^ *n* $\neq$ *0*
apply (*induction rule*: *nat-induct-non-zero*[*OF assms*(*2*)])
unfolding *power-Suc2* **using** *assms*(*1*)
by *auto*

One more typeclasses

instance *mpoly* :: (*semiring-no-zero-divisors*) *semiring-no-zero-divisors*
by *intro-classes* (*transfer*, *simp*)

end
theory *Degree*
imports *Notation*
begin

1.2 Degree of a given variable

lemma *degree-Const* [*simp*]: *degree* (*Const x*) *v* = *0*
unfolding *degree-def Const-def* $Const_0$*-def*
by (*simp add*: *MPoly-inverse*)

lemma *degree-Var* [*simp*]:
degree ((*Var v*):: *'a*::*comm-semiring-1 mpoly*) *v'* = (*if v* = *v' then 1 else 0*)
unfolding *degree-def Var-def* Var_0*-def*
by (*simp add*: *MPoly-inverse lookup-single*)

lemma *degree-neg*:
fixes $P :: 'a::ab\text{-}group\text{-}add\ mpoly$
shows $degree\ (-\ P) = degree\ P$
unfolding *degree.rep-eq uminus-mpoly.rep-eq keys.rep-eq*
by *auto*

lemma *degree-add*:
fixes $P\ Q :: 'a::ab\text{-}group\text{-}add\ mpoly$
shows $degree\ (P + Q)\ v \leq max\ (degree\ P\ v)\ (degree\ Q\ v)$
proof −
{ fix m **assume** $m \in keys\ (mapping\text{-}of\ (P + Q))$
hence $m \in keys\ (mapping\text{-}of\ P) \vee m \in keys\ (mapping\text{-}of\ Q)$
by (*metis add.right-neutral coeff-add coeff-keys*)
moreover have $m \in keys\ (mapping\text{-}of\ P) \Longrightarrow lookup\ m\ v \leq degree\ P\ v$
by (*simp add*: *degree-def*)
moreover have $m \in keys\ (mapping\text{-}of\ Q) \Longrightarrow lookup\ m\ v \leq degree\ Q\ v$
by (*simp add*: *degree-def*)
ultimately have $lookup\ m\ v \leq max\ (degree\ P\ v)\ (degree\ Q\ v)$
by *auto* **}**
thus *?thesis* **by** (*simp add*: *degree-def*)
qed

lemma *degree-add′*:
fixes $P\ Q :: 'a::ab\text{-}group\text{-}add\ mpoly$
shows $degree\ (P + Q)\ v \leq degree\ P\ v + degree\ Q\ v$
using *degree-add*
by (*metis max-def trans-le-add1 trans-le-add2*)

lemma *degree-add-different-degree*:
fixes $P :: 'a::ab\text{-}group\text{-}add\ mpoly$
assumes $degree\ P\ v \neq degree\ Q\ v$
shows $degree\ (P + Q)\ v = max\ (degree\ P\ v)\ (degree\ Q\ v)$
proof −
have *0*: $Max\ (insert\ 0\ ((\lambda m.\ lookup\ m\ v)\ `\ keys\ (mapping\text{-}of\ P))) \neq$
$Max\ (insert\ 0\ ((\lambda m.\ lookup\ m\ v)\ `\ keys\ (mapping\text{-}of\ Q)))$
using *assms* **unfolding** *degree.rep-eq* **.**
have $Max\ (insert\ 0\ ((\lambda m.\ lookup\ m\ v)\ `\ keys\ (mapping\text{-}of\ P + mapping\text{-}of\ Q)))$
=
$max\ (Max\ (insert\ 0\ ((\lambda m.\ lookup\ m\ v)\ `\ keys\ (mapping\text{-}of\ P))))$
$(Max\ (insert\ 0\ ((\lambda m.\ lookup\ m\ v)\ `\ keys\ (mapping\text{-}of\ Q))))$
proof *cases*
assume *1*: $(\lambda m.\ lookup\ m\ v)\ `\ keys\ (mapping\text{-}of\ P) = \{\}$
show *?thesis*
proof *cases*
assume *2*: $(\lambda m.\ lookup\ m\ v)\ `\ keys\ (mapping\text{-}of\ Q) = \{\}$
show *?thesis* **using** *0 1 assms* **unfolding** *degree.rep-eq* **by** *simp*
next
assume *3*: $(\lambda m.\ lookup\ m\ v)\ `\ keys\ (mapping\text{-}of\ Q) \neq \{\}$
have $(\lambda m.\ lookup\ m\ v)\ `\ keys\ (mapping\text{-}of\ P + mapping\text{-}of\ Q) =$

```
        (λm. lookup m v) ‘ keys (mapping-of Q)
      using 1 3 by auto
    thus ?thesis using 1 by auto
  qed
 next
  assume 4: (λm. lookup m v) ‘ keys (mapping-of P) ≠ {}
  show ?thesis
  proof cases
    assume 5: (λm. lookup m v) ‘ keys (mapping-of Q) = {}
    have (λm. lookup m v) ‘ keys (mapping-of P + mapping-of Q) =
        (λm. lookup m v) ‘ keys (mapping-of P)
      using 4 5 by auto
    thus ?thesis using 5 by auto
  next
    assume 6: (λm. lookup m v) ‘ keys (mapping-of Q) ≠ {}
    have 7: (λm. lookup m v) ‘ keys (mapping-of P + mapping-of Q) ≠ {}
      using 0 eq-neg-iff-add-eq-0 by force
    obtain a where 8: lookup a v = (MAX m∈keys (mapping-of P). lookup m
v)
                    lookup (mapping-of P) a ≠ 0
                  ⋀m. lookup (mapping-of P) m ≠ 0 ⟹ lookup m v ≤ lookup a v
      using 4
        by (smt (verit, best) Max-ge Max-in finite-imageI finite-keys image-iff
in-keys-iff)
    obtain b where 9: lookup b v = (MAX m∈keys (mapping-of Q). lookup m
v)
                    lookup (mapping-of Q) b ≠ 0
                  ⋀m. lookup (mapping-of Q) m ≠ 0 ⟹ lookup m v ≤ lookup b v
      using 6
        by (smt (verit, best) Max-ge Max-in finite-imageI finite-keys image-iff
in-keys-iff)
    obtain c where 10: lookup c v = (MAX m∈keys (mapping-of P + mapping-of
Q). lookup m v)
                     lookup (mapping-of P) c + lookup (mapping-of Q) c ≠ 0
                   ⋀m. lookup (mapping-of P) m + lookup (mapping-of Q) m ≠
0
                     ⟹ lookup m v ≤ lookup c v
      using 7
      by (smt (verit, del-insts) Max-ge Max-in finite-imageI finite-keys image-iff
          lookup-not-eq-zero-eq-in-keys plus-poly-mapping.rep-eq)
    have 11: lookup a v ≠ lookup b v unfolding 8(1) 9(1) using 0 4 6 by simp
    have lookup c v = max (lookup a v) (lookup b v)
      using 8 9 10 11
    by (smt (verit, del-insts) add-cancel-right-right max.bounded-iff order-class.order-eq-iff)
    hence (MAX m∈keys (mapping-of P + mapping-of Q). lookup m v) =
        max (MAX m∈keys (mapping-of P). lookup m v)
            (MAX m∈keys (mapping-of Q). lookup m v)
      unfolding 8(1) 9(1) 10(1) .
    thus ?thesis using 4 6 7 by auto
```

```
    qed
  qed
  thus ?thesis unfolding degree.rep-eq plus-mpoly.rep-eq plus-poly-mapping.rep-eq
.
qed

lemma degree-diff:
  fixes P Q :: 'a::ab-group-add mpoly
  shows degree (P - Q) v ≤ max (degree P v) (degree Q v)
  unfolding diff-conv-add-uminus[of P] degree-neg[of Q, symmetric]
  by (rule degree-add)

lemma degree-diff':
  fixes P Q :: 'a::ab-group-add mpoly
  shows degree (P - Q) v ≤ degree P v + degree Q v
  unfolding diff-conv-add-uminus[of P] degree-neg[of Q, symmetric]
  by (rule degree-add')

lemma degree-diff-different-degree:
  fixes P :: 'a::ab-group-add mpoly
  assumes degree P v ≠ degree Q v
  shows degree (P - Q) v = max (degree P v) (degree Q v)
  unfolding diff-conv-add-uminus[of P] degree-neg[of Q, symmetric]
  apply (rule degree-add-different-degree)
  unfolding degree-neg apply (rule assms)
  done

lemma degree-sum:
  fixes P :: 'a ⇒ 'b::ab-group-add mpoly
  assumes S-fin: finite S
  shows degree (sum P S) v ≤ Max (insert 0 ((λi. degree (P i) v) ‘ S))
proof (induct rule:finite-induct[OF S-fin])
  case 1 show ?case by simp
next
  case (2 x S)
  have degree (sum P (insert x S)) v = degree (sum P S + P x) v
    by (simp add: 2.hyps add.commute)
  also have ... ≤ max (degree (sum P S) v) (degree (P x) v)
    using degree-add by auto
  also have ... ≤ max (Max (insert 0 ((λi. degree (P i) v) ‘ S))) (degree (P x) v)
    using 2.hyps by auto
  also have ... = Max (insert (degree (P x) v) (insert 0 ((λi. degree (P i) v) ‘
S)))
    by (simp add: 2.hyps)
  also have ... = Max (insert 0 ((λi. degree (P i) v) ‘ (insert x S)))
    by (simp add: insert-commute)
  finally show ?case .
qed
```

```
lemma degree-mult: degree (P * Q) v ≤ degree P v + degree Q v
proof −
  have m ∈ keys (mapping-of (P * Q)) ⟹
        lookup m v ≤ degree P v + degree Q v for m
  proof −
    assume m ∈ keys (mapping-of (P * Q))
    hence m ∈ keys (mapping-of P * mapping-of Q)
      by (simp add: times-mpoly.rep-eq)
    hence m ∈ {a + b | a b. a ∈ keys (mapping-of P) ∧ b ∈ keys (mapping-of Q)}
      using keys-mult by blast
    then obtain a b where m-def: m = a + b
                      and a-key: a ∈ keys (mapping-of P)
                      and b-key: b ∈ keys (mapping-of Q)
      by blast
    from a-key have a-bound: lookup a v ≤ degree P v
      unfolding degree.rep-eq by simp
    from b-key have b-bound: lookup b v ≤ degree Q v
      unfolding degree.rep-eq by simp
    have lookup m v = lookup a v + lookup b v
      unfolding m-def lookup-add by simp
    thus lookup m v ≤ degree P v + degree Q v
      using a-bound b-bound by simp
  qed
  thus ?thesis unfolding degree.rep-eq by simp
qed

lemma degree-mult-non-zero:
  fixes P Q :: 'a::idom mpoly
  assumes P ≠ 0 Q ≠ 0
  shows degree (P * Q) v = degree P v + degree Q v
proof (rule antisym)
  show degree (P * Q) v ≤ degree P v + degree Q v by (rule degree-mult)
next
  define a where a = Max ((λm. lookup m v) ` keys (mapping-of P))
  define b where b = Max ((λm. lookup m v) ` keys (mapping-of Q))
  define c where c = Max ((λm. lookup m v) ` keys (mapping-of (P * Q)))
  define p where p = Max {m ∈ keys (mapping-of P). lookup m v = a}
  define q where q = Max {m ∈ keys (mapping-of Q). lookup m v = b}
  define r where r = Max {m ∈ keys (mapping-of (P * Q)). lookup m v = c}
  have 0: P * Q ≠ 0 using assms by auto
  have 1: keys (mapping-of P) ≠ {}
          keys (mapping-of Q) ≠ {}
          keys (mapping-of (P * Q)) ≠ {}
    using assms 0 mapping-of-inject zero-mpoly.rep-eq
    by (metis keys-eq-empty)+
  have 2: a ∈ (λm. lookup m v) ` keys (mapping-of P)
          b ∈ (λm. lookup m v) ` keys (mapping-of Q)
          c ∈ (λm. lookup m v) ` keys (mapping-of (P * Q))
    unfolding a-def b-def c-def
```

```
  using 1 Max-in
  by simp+
have 3: p ∈ {m ∈ keys (mapping-of P). lookup m v = a}
       q ∈ {m ∈ keys (mapping-of Q). lookup m v = b}
       r ∈ {m ∈ keys (mapping-of (P * Q)). lookup m v = c}
  unfolding p-def q-def r-def
  using 2 Max-in[of {m ∈ keys (mapping-of -). lookup m v = -}]
  by auto
have 4: ⋀m. m ∈ keys (mapping-of P) ⟹ lookup m v ≤ a
       ⋀m. m ∈ keys (mapping-of Q) ⟹ lookup m v ≤ b
       ⋀m. m ∈ keys (mapping-of (P * Q)) ⟹ lookup m v ≤ c
  unfolding a-def b-def c-def
  by auto
have 5: ⋀m. m ∈ keys (mapping-of P) ⟹ lookup m v = a ⟹ m ≤ p
       ⋀m. m ∈ keys (mapping-of Q) ⟹ lookup m v = b ⟹ m ≤ q
       ⋀m. m ∈ keys (mapping-of (P * Q)) ⟹ lookup m v = c ⟹ m ≤ r
  unfolding p-def q-def r-def
  by auto
have p' + q' = p + q ⟹
     lookup (mapping-of P) p' ≠ 0 ⟹
     lookup (mapping-of Q) q' ≠ 0 ⟹
     p' = p ∧ q' = q for p' q'
proof −
  assume 6: p' + q' = p + q
  assume 7: lookup (mapping-of P) p' ≠ 0 lookup (mapping-of Q) q' ≠ 0
  have 8: lookup p' v ≤ lookup p v lookup q' v ≤ lookup q v
    using 3 4 7 unfolding in-keys-iff by auto
  have lookup p' v + lookup q' v = lookup p v + lookup q v
    using 6
    by (metis lookup-add)
  hence lookup p' v = lookup p v ∧ lookup q' v = lookup q v
    using 8
    by linarith
  hence p' ≤ p q' ≤ q
    using 3 5 7 unfolding in-keys-iff by blast+
  thus p' = p ∧ q' = q
    using 6
    by (metis add.commute add-le-cancel-left antisym)
qed
hence 9: (p + q = p' + q' ∧
          lookup (mapping-of P) p' ≠ 0 ∧
          lookup (mapping-of Q) q' ≠ 0) ⟷
         (p', q') = (p, q) for p' q'
  using 3 by auto
have lookup (mapping-of (P * Q)) (p + q) =
     (∑(p', q'). lookup (mapping-of P) p' * lookup (mapping-of Q) q' when p +
q = p' + q')
  unfolding times-mpoly.rep-eq times-poly-mapping.rep-eq
  by (rule prod-fun-unfold-prod; simp)
```

```
  also have ... = (∑(p', q'). lookup (mapping-of P) p' * lookup (mapping-of Q)
q'
                when p + q = p' + q' ∧ lookup (mapping-of P) p' ≠ 0 ∧ lookup
(mapping-of Q) q' ≠ 0)
    apply (rule Sum-any.cong)
    unfolding when-def apply auto
    done
  also have ... = Sum-any (λa. (case a of (p', q')
                  ⇒ lookup (mapping-of P) p' * lookup (mapping-of Q) q') when a
= (p, q))
    unfolding 9
    by (rule Sum-any.cong; auto)
  also have ... = lookup (mapping-of P) p * lookup (mapping-of Q) q
    unfolding Sum-any-when-equal
    by auto
  also have ... ≠ 0
    using 3 by auto
  finally have 10: lookup (mapping-of (P * Q)) (p + q) ≠ 0 .
  have a + b = lookup (p + q) v
    using 3
    unfolding plus-poly-mapping.rep-eq by simp
  also have ... ≤ c
    by (rule 4(3)[unfolded in-keys-iff]; rule 10)
  finally show degree P v + degree Q v ≤ degree (P * Q) v
    unfolding a-def b-def c-def degree.rep-eq
    using 1
    by simp
qed

lemma degree-pow: degree (P ^ n) v ≤ n * degree P v
proof (induct n)
  case 0
  show ?case by simp
next
  case (Suc n)
  have degree (P ^ Suc n) v ≤ degree P v + degree (P ^ n) v
    using degree-mult by auto
  also have ... ≤ degree P v + n * degree P v
    using Suc by auto
  also have ... = Suc n * degree P v
    by auto
  finally show ?case .
qed

lemma degree-pow-positive:
  fixes P :: 'a::idom mpoly
  assumes n > 0
  shows degree (P ^ n) v = n * degree P v
proof (induction rule: nat-induct-non-zero[OF assms])
```

```
  show degree (P ^ 1) v = 1 * degree P v by simp
next
  fix m :: nat
  assume 0: m > 0
  assume 1: degree (P ^ m) v = m * degree P v
  show degree (P ^ Suc m) v = Suc m * degree P v
  proof cases
    assume 2: P = 0
    show ?thesis unfolding 2 by simp
  next
    assume 3: P ≠ 0
    show ?thesis
      unfolding power-Suc2
      apply (subst degree-mult-non-zero)
      subgoal using 0 3 pow-positive by blast
      subgoal using 3 .
      subgoal using 1 by simp
      done
  qed
qed

lemma degree-prod:
  assumes S-fin: finite S
  shows degree (prod P S) v ≤ sum (λi. degree (P i) v) S
proof (induct rule:finite-induct[OF S-fin])
  case 1 show ?case by simp
next
  case (2 x S)
  have degree (prod P (insert x S)) v = degree (prod P S * P x) v
    by (simp add: 2.hyps mult.commute)
  also have ... ≤ (degree (prod P S) v) + (degree (P x) v)
    using degree-mult by auto
  also have ... ≤ (sum (λi. degree (P i) v) S) + (degree (P x) v)
    using 2.hyps by auto
  also have ... = sum (λi. degree (P i) v) (insert x S)
    by (simp add: 2.hyps)
  finally show ?case .
qed


end
theory Variables
  imports Degree HOL−Eisbach.Eisbach
begin
```

1.3 Variables

```
lemma Var-neq-zero: (Var v :: 'a::zero-neq-one mpoly) ≠ 0
proof −
```

```
  have lookup (mapping-of (Var v :: 'a mpoly)) (Poly-Mapping.single v 1) ≠ 0
    unfolding Var.rep-eq Var₀-def by simp
  thus ?thesis
    by (metis lookup-zero zero-mpoly.rep-eq)
qed

lemma in-vars-non-zero-degree: v ∈ vars P ⟷ degree P v ≠ 0
proof (rule iffI)
  have v ∉ vars P ⟹ degree P v = 0
  proof −
    assume v ∉ vars P
    hence (λm. lookup m v) ‘ keys (mapping-of P) ⊆ {0}
      unfolding vars-def image-def
      using in-keys-iff
      by fastforce
    hence 0: insert 0 ((λm. lookup m v) ‘ keys (mapping-of P)) = {0}
      by blast
    have Max (insert 0 ((λm. lookup m v) ‘ keys (mapping-of P))) = 0
      unfolding 0 by simp
    thus ?thesis unfolding degree.rep-eq .
  qed
  thus degree P v ≠ 0 ⟹ v ∈ vars P by auto
next
  assume v ∈ vars P
  thus degree P v ≠ 0
    unfolding vars-def degree.rep-eq
    by (simp; metis (no-types, lifting) Max-ge finite-imageI finite-insert finite-keys
gr0I image-subset-iff le-zero-eq lookup-eq-zero-in-keys-contradict subset-insertI)
qed

lemma vars-non-zero-degree: vars P = {v. degree P v ≠ 0}
  using in-vars-non-zero-degree by blast

lemma vars-Const [simp]: vars (Const x) = {}
  by (simp add: Const.rep-eq Const₀-def vars-def)

lemma vars-zero [simp]: vars 0 = {}
  by (metis Const-zero vars-Const)

lemma vars-Var [simp]: vars ((Var v) :: ('a::zero-neq-one) mpoly) = {v}
  using vars-monom-single-cases unfolding monom-def Var-def Var₀-def by force

lemma vars-neg:
  fixes P :: 'a::ab-group-add mpoly
  shows vars (− P) = vars P
proof −
  have keys (mapping-of (−P)) = keys (mapping-of P)
    unfolding keys.rep-eq uminus-mpoly.rep-eq
    by simp
```

thus *vars* $(-P) = vars\ P$
unfolding *vars-def*
by *simp*
qed

lemma *vars-add-different-degree*:
fixes $P :: 'a::ab\text{-}group\text{-}add\ mpoly$
assumes $\forall u \in vars\ P \cap vars\ Q.\ degree\ P\ u \neq degree\ Q\ u$
shows $vars\ (P + Q) = vars\ P \cup vars\ Q$
apply (*rule set-eqI*)
subgoal for *v*
using *assms*
unfolding *vars-non-zero-degree*
using *degree-add-different-degree*[*of P v Q*]
by (*simp*; *smt* (*verit*, *del-insts*) *IntI ab-semigroup-add-class.add-ac*(*1*) *add.commute add.right-inverse add.right-neutral degree-add-different-degree max.strict-coboundedI1 mem-Collect-eq*)
done

lemma *vars-diff*:
fixes $P :: 'a::ab\text{-}group\text{-}add\ mpoly$
shows $vars\ (P - Q) \subseteq vars\ P \cup vars\ Q$
unfolding *diff-conv-add-uminus*
using *vars-neg vars-add* **by** *blast*

lemma *vars-diff-different-degree*:
fixes $P :: 'a::ab\text{-}group\text{-}add\ mpoly$
assumes $\forall u \in vars\ P \cap vars\ Q.\ degree\ P\ u \neq degree\ Q\ u$
shows $vars\ (P - Q) = vars\ P \cup vars\ Q$
unfolding *diff-conv-add-uminus vars-neg*[*of Q*, *symmetric*]
apply (*rule vars-add-different-degree*)
unfolding *degree-neg vars-neg* **apply** (*rule assms*)
done

lemma *vars-mult-non-zero*:
fixes $P\ Q :: 'a::idom\ mpoly$
shows $P \neq 0 \Longrightarrow Q \neq 0 \Longrightarrow vars\ (P * Q) = vars\ P \cup vars\ Q$
unfolding *vars-non-zero-degree*
using *degree-mult-non-zero*[*of P Q*]
by *auto*

lemma *vars-pow*: $vars\ (P\hat{}n) \subseteq vars\ P$
proof (*induct n*)
case *0*
show *?case*
by (*metis bot-least monom-pow mult-is-0 power-eq-if vars-monom-single-cases*)
next
case (*Suc n*)
thus *?case*

```
    by (metis Un-absorb1 power-Suc2 vars-mult)
qed

lemma vars-pow-positive:
  fixes P :: 'a::idom mpoly
  assumes n > 0
  shows vars (P ^ n) = vars P
proof (induction rule: nat-induct-non-zero[OF assms])
  show vars (P ^ 1) = vars P by simp
next
  fix m :: nat
  assume 0: m > 0
  assume 1: vars (P ^ m) = vars P
  show vars (P ^ Suc m) = vars P
  proof cases
    assume 2: P = 0
    show ?thesis unfolding 2 by simp
  next
    assume 3: P ≠ 0
    show ?thesis
      unfolding power-Suc2
      apply (subst vars-mult-non-zero)
      subgoal using 0 3 pow-positive by blast
      subgoal using 3 .
      subgoal using vars-pow by blast
      done
  qed
qed

lemma vars-prod:
  fixes S :: 'a set
    and f :: - ⇒ (- :: zero-neq-one) mpoly
  shows vars (prod f S) ⊆ ⋃ (vars ‘ f ‘ S)
proof (cases finite S)
  case True
  then show ?thesis
  proof (induction S rule:finite-induct)
    case empty
    then show ?case
      apply simp
      unfolding vars-def one-mpoly.rep-eq
      by simp
  next
    case (insert s S)
     then have vars (prod f (insert s S)) = vars (f s * prod f S) by (metis
prod.insert)
    also have ... ⊆ vars (f s) ∪ vars (prod f S) by (simp add: vars-mult)
    also have ... ⊆ (⋃ m∈insert s S. vars (f m)) using insert.IH by auto
    finally show ?case by simp
```

qed
next
case *False*
then show *?thesis*
proof −
from ‹*infinite S*› **have** *prod f S = 1* **by** *simp*
hence *vars* (*prod f S*) = {}
by (*metis Const-one vars-Const*)
thus *?thesis* **by** *simp*
qed
qed

lemma *vars-empty*:
assumes *vars P* = {}
shows ∃ *c*. *P* = *Const c*
proof −
have *0*: ∀ *x*∈*keys* (*mapping-of P*). *x* = *0* **using** *assms* **unfolding** *vars-def* **by** *auto*
have *P* = *Const* (*lookup* (*mapping-of P*) *0*)
unfolding *mapping-of-inject*[*symmetric*] *Const.rep-eq Const*$_0$*-def*
unfolding *lookup-inject*[*of mapping-of P*, *symmetric*] *Poly-Mapping.single.rep-eq*
unfolding *when-def* **apply** (*rule ext*)
using *0* **unfolding** *keys-def* **apply** *auto*
done
thus *?thesis* **by** *blast*
qed

1.3.1 Maximum variable index

definition *max-vars* **where** *max-vars P* ≡ *Max* (*insert 0* (*vars P*))

lemma *after-max-vars*:
lookup (*mapping-of P*) *m* ≠ *0* ⟹ ∀ *v*≥*max-vars P* + *1*. *lookup m v* = *0*
proof (*rule allI*; *rule impI*)
fix *v*
assume *0*: *lookup* (*mapping-of P*) *m* ≠ *0*
assume *max-vars P* + *1* ≤ *v*
hence *v* ∉ *vars P* **unfolding** *max-vars-def*
by (*metis Max.insert Max-ge add.commute empty-iff max-nat.left-neutral not-less-eq-eq plus-1-eq-Suc vars-finite*)
thus *lookup m v* = *0* **using** *0* **by** (*simp add*: *in-keys-iff vars-def*)
qed

lemma *max-vars-of-nonempty*: *vars P* ≠ {} ⟹ *max-vars P* = *Max* (*vars P*)
unfolding *max-vars-def* **by** (*simp add*: *vars-finite*)

1.3.2 Simplification rules for maximum variable index

lemma *max-vars-Const* [*simp*]: *max-vars* (*Const x*) = *0*
unfolding *max-vars-def* **by** *simp*

lemma *max-vars-one* [*simp*]: *max-vars 1 = 0*
unfolding *max-vars-def*
by (*metis Const-one max-vars-def max-vars-Const*)

lemma *max-vars-Var* [*simp*]: *max-vars* ((*Var v*) :: ('*a*::*zero-neq-one*) *mpoly*) = *v*
unfolding *max-vars-def* **by** *simp*

lemma *max-vars-add*: *max-vars* (*P* + *Q*) ≤ *max* (*max-vars P*) (*max-vars Q*)
unfolding *max-vars-def* **using** *vars-add*
by (*smt* (*verit*, *ccfv-threshold*) *Max.coboundedI Max-in UnE finite.insertI insertCI insertE*
insert-not-empty le-max-iff-disj subset-iff vars-finite)

lemma *max-vars-neg*: *max-vars* (− *P*) = *max-vars P*
unfolding *max-vars-def* **using** *vars-neg* **by** *metis*

lemma *max-vars-diff*:
fixes *P* :: '*a*::*ab-group-add mpoly*
shows *max-vars* (*P* − *Q*) ≤ *max* (*max-vars P*) (*max-vars Q*)
unfolding *diff-conv-add-uminus*
by (*metis max-vars-add max-vars-neg*)

lemma *max-vars-diff'*:
fixes *P* :: *int mpoly*
shows *max-vars* (*P* − *Q*) ≤ *max* (*max-vars P*) (*max-vars Q*)
by (*rule max-vars-diff*)

lemma *max-vars-pow*: *max-vars* (*P* ^ *n*) ≤ *max-vars P*
unfolding *max-vars-def* **using** *vars-pow*
by (*metis Max-mono empty-not-insert finite-insert insert-mono vars-finite*)

lemma *max-vars-pow-positive*:
fixes *P* :: '*a*::*idom mpoly*
assumes *n* > *0*
shows *max-vars* (*P* ^ *n*) = *max-vars P*
unfolding *max-vars-def vars-pow-positive*[*of n P*, *OF* ‹*n* > *0*›] **..**

lemma *max-vars-mult*: *max-vars* (*P* ∗ *Q*) ≤ *max* (*max-vars P*) (*max-vars Q*)
unfolding *max-vars-def* **using** *vars-mult*
by (*smt* (*z3*) *Max-ge Max-in UnE finite.insertI in-mono insert-iff insert-not-empty max.bounded-iff*
max-def vars-finite)

lemmas *max-vars-simps* = *max-vars-add max-vars-neg max-vars-diff max-vars-pow*
max-vars-pow-positive max-vars-mult

method *mpoly-vars* =
(*rule subset-trans*[*OF vars-pow*]

| *rule subset-trans*[*OF vars-add Un-least*]
| *rule subset-trans*[*OF vars-diff Un-least*]
| *rule subset-trans*[*OF vars-mult Un-least*]
| *rule Set.empty-subsetI*
| *unfold vars-neg*
| *unfold Const-numeral*[*symmetric*]
| *unfold vars-Var*
| *unfold vars-Const*
| *simp-all*)+

end
theory *Total-Degree*
imports *Variables*
begin

1.4 Total degree

named-theorems *total-degree-simps Lemmas about the total-degree function*

lemma *total-degree-Const* [*simp*]: *total-degree* (*Const x*) = *0*
unfolding *total-degree-def Const-def* $Const_0$*-def*
by (*simp add*: *MPoly-inverse*)

lemma *total-degree-Var* [*simp*]:
total-degree ((*Var v*):: *'a*::*comm-semiring-1 mpoly*) = *1*
unfolding *total-degree-def Var-def* Var_0*-def*
by (*simp add*: *MPoly-inverse lookup-single*)

lemma *total-degree-zero-degree-zero*:
assumes *total-degree P* = *0*
shows *degree P v* = *0*
proof −
have *insert 0* ((λ*m*. *sum* (*lookup m*) (*keys m*)) ‘ *keys* (*mapping-of P*)) ⊆ {*0*}
using *assms* **unfolding** *total-degree.rep-eq*
by (*metis* (*no-types*, *lifting*) *Max-ge finite-imageI finite-insert finite-keys le-zero-eq singleton-iff subsetI*)
hence *m* ∈ *keys* (*mapping-of P*) ⟹ *sum* (*lookup m*) (*keys m*) = *0* **for** *m* **by** *blast*
hence *m* ∈ *keys* (*mapping-of P*) ⟹ (*lookup m*) *v* = *0* **for** *m*
unfolding *keys-def* **by** *simp*
thus *?thesis* **unfolding** *degree.rep-eq*
by (*metis* (*no-types*, *lifting*) *Max-insert2 finite-imageI finite-keys imageE zero-order*(*2*))
qed

lemma *total-degree-zero*:
assumes *total-degree P* = *0*
shows ∃ *c*. *P* = *Const c*
proof −

have *vars P* = {}
unfolding *vars-non-zero-degree*
using *total-degree-zero-degree-zero*[*OF assms*]
by *simp*
thus *?thesis* **by** (*rule vars-empty*)
qed

lemma *total-degree-neg*[*total-degree-simps*]:
fixes *P* :: *'a*::*ab-group-add mpoly*
shows *total-degree* (− *P*) = *total-degree P*
unfolding *total-degree.rep-eq uminus-mpoly.rep-eq keys.rep-eq*
by *auto*

lemma *total-degree-add*[*total-degree-simps*]:
shows *total-degree* (*P* + *Q*) ≤ *max* (*total-degree P*) (*total-degree Q*)
proof −
have *m* ∈ *keys* (*mapping-of* (*P* + *Q*)) ⟹
sum (*lookup m*) (*keys m*) ≤ *max* (*total-degree P*) (*total-degree Q*) **for** *m*
proof −
assume *m* ∈ *keys* (*mapping-of* (*P* + *Q*))
hence *m* ∈ *keys* (*mapping-of P* + *mapping-of Q*)
by (*simp add*: *plus-mpoly.rep-eq*)
hence *in-union*: *m* ∈ *keys* (*mapping-of P*) ∪ *keys* (*mapping-of Q*)
by (*meson Poly-Mapping.keys-add in-mono*)
have *m* ∈ *keys* (*mapping-of P*) ⟹ *sum* (*lookup m*) (*keys m*) ≤ *total-degree P*
unfolding *total-degree.rep-eq* **by** (*rule Max-ge*; *auto*)
moreover have *m* ∈ *keys* (*mapping-of Q*) ⟹ *sum* (*lookup m*) (*keys m*) ≤ *total-degree Q*
unfolding *total-degree.rep-eq* **by** (*rule Max-ge*; *auto*)
ultimately show *sum* (*lookup m*) (*keys m*) ≤ *max* (*total-degree P*) (*total-degree Q*)
using *in-union* **by** *force*
qed
thus *?thesis* **unfolding** *total-degree.rep-eq* **by** *simp*
qed

lemma *total-degree-add-different-total-degree*:
fixes *P* :: *'a*::*ab-group-add mpoly*
assumes *total-degree P* ≠ *total-degree Q*
shows *total-degree* (*P* + *Q*) = *max* (*total-degree P*) (*total-degree Q*)

proof(*cases total-degree P* > *total-degree Q*)
case *first*:*True*
hence *P-notzero*:*total-degree P* > *0* **by** *auto*
obtain *m* **where** *m-prop1*:*m* ∈ *keys* (*mapping-of P*) ∧ *Max* ((λ*l*. *sum* (*lookup l*) (*keys l*)) ' *keys* (*mapping-of P*)) = *sum* (*lookup m*) (*keys m*)
using *P-notzero*
by (*metis* (*mono-tags*, *lifting*) *MPoly-Type.total-degree-zero Max-in finite-imageI finite-keys imageE image-empty keys-zero not-less-zero total-degree.rep-eq zero-mpoly.rep-eq*)

hence *m-prop2*:*total-degree P* = *sum* (*lookup m*) (*keys m*)
by (*metis* (*no-types*, *lifting*) *Max.insert empty-iff finite-imageI finite-keys image-is-empty max-nat.left-neutral total-degree.rep-eq*)

have *sum-pres-map*:*mapping-of P* + *mapping-of Q* = *mapping-of* (*P*+*Q*) **by** (*simp add*: *plus-mpoly.rep-eq*)
have *m* ∉ *keys* (*mapping-of Q*) **using** *m-prop2 first*
by (*metis* (*no-types*, *lifting*) *Max.insert Max-ge empty-iff finite-imageI finite-keys image-eqI linorder-not-le max-nat.left-neutral total-degree.rep-eq*)
hence *m* ∈ *keys* (*mapping-of* (*P*+*Q*)) **using** *m-prop1 sum-pres-map*
by (*metis* (*no-types*, *lifting*) *add.right-neutral in-keys-iff plus-poly-mapping.rep-eq*)
hence *geq*:*total-degree* (*P*+*Q*) ≥ *total-degree P* **using** *m-prop2* **by** (*simp add*: *total-degree.rep-eq*)

have *total-degree* (*P*+*Q*) ≤ *max* (*total-degree P*) (*total-degree Q*) **using** *total-degree-add* **by** *auto*
hence *total-degree* (*P*+*Q*) = *total-degree P* **using** *first geq* **by** *auto*
thus *?thesis* **using** *first* **by** *auto*

next
case *False*
hence *second*:*total-degree Q* > *total-degree P* **using** *assms* **by** *auto*
hence *Q-notzero*:*total-degree Q* > *0* **by** *auto*
obtain *n* **where** *n-prop1*:*n* ∈ *keys* (*mapping-of Q*) ∧ *Max* ((λ*l*. *sum* (*lookup l*) (*keys l*)) ‘ *keys* (*mapping-of Q*)) = *sum* (*lookup n*) (*keys n*)
using *Q-notzero*
by (*metis* (*mono-tags*, *lifting*) *MPoly-Type.total-degree-zero Max-in finite-imageI finite-keys imageE image-empty keys-zero not-less-zero total-degree.rep-eq zero-mpoly.rep-eq*)
hence *n-prop2*:*total-degree Q* = *sum* (*lookup n*) (*keys n*)
by (*metis* (*no-types*, *lifting*) *Max.insert empty-iff finite-imageI finite-keys image-is-empty max-nat.left-neutral total-degree.rep-eq*)

have *sum-pres-map*:*mapping-of P* + *mapping-of Q* = *mapping-of* (*P*+*Q*) **by** (*simp add*: *plus-mpoly.rep-eq*)
have *n* ∉ *keys* (*mapping-of P*) **using** *n-prop2 second*
by (*metis* (*no-types*, *lifting*) *Max.insert Max-ge empty-iff finite-imageI finite-keys image-eqI linorder-not-le max-nat.left-neutral total-degree.rep-eq*)
hence *n* ∈ *keys* (*mapping-of* (*P*+*Q*)) **using** *n-prop1 sum-pres-map*
by (*metis* (*no-types*, *lifting*) *add.commute add.right-neutral coeff-def coeff-keys plus-poly-mapping.rep-eq*)
hence *geq*:*total-degree* (*P*+*Q*) ≥ *total-degree Q* **using** *n-prop2* **by** (*simp add*: *total-degree.rep-eq*)

have *total-degree* (*P*+*Q*) ≤ *max* (*total-degree P*) (*total-degree Q*) **using** *total-degree-add* **by** *auto*
hence *total-degree* (*P*+*Q*) = *total-degree Q* **using** *second geq* **by** *auto*
thus *?thesis* **using** *second* **by** *auto*
qed

lemma *total-degree-diff*[*total-degree-simps*]:
fixes $P :: 'a::ab\text{-}group\text{-}add\ mpoly$
shows *total-degree* $(P - Q) \leq max$ (*total-degree* P) (*total-degree* Q)
unfolding *diff-conv-add-uminus*[*of* P] *total-degree-neg*[*of* Q, *symmetric*]
by (*rule total-degree-add*)

lemma *total-degree-diff-different-total-degree*:
fixes $P :: 'a::ab\text{-}group\text{-}add\ mpoly$
assumes *total-degree* $P \neq$ *total-degree* Q
shows *total-degree* $(P - Q) = max$ (*total-degree* P) (*total-degree* Q)
unfolding *diff-conv-add-uminus*[*of* P] *total-degree-neg*[*of* Q, *symmetric*]
apply (*rule total-degree-add-different-total-degree*)
unfolding *total-degree-neg* **apply** (*rule assms*)
done

lemma *total-degree-mult*[*total-degree-simps*]:
total-degree $(P * Q) \leq$ *total-degree* P + *total-degree* Q
proof −
have $m \in keys\ (mapping\text{-}of\ (P * Q)) \Longrightarrow$
$sum\ (lookup\ m)\ (keys\ m) \leq$ *total-degree* P + *total-degree* Q **for** m
proof −
assume $m \in keys\ (mapping\text{-}of\ (P * Q))$
hence $m \in keys\ (mapping\text{-}of\ P * mapping\text{-}of\ Q)$
by (*simp add*: *times-mpoly.rep-eq*)
hence $m \in \{a + b \mid a\ b.\ a \in keys\ (mapping\text{-}of\ P) \wedge b \in keys\ (mapping\text{-}of\ Q)\}$
using *keys-mult* **by** *blast*
then obtain $a\ b$ **where** *m-def*: $m = a + b$
and *a-key*: $a \in keys\ (mapping\text{-}of\ P)$
and *b-key*: $b \in keys\ (mapping\text{-}of\ Q)$
by *blast*
have *a-bound*: $sum\ (lookup\ a)\ (keys\ a) \leq$ *total-degree* P
unfolding *total-degree.rep-eq*
apply (*rule Max-ge*)
subgoal by *simp*
subgoal using *a-key* **by** *auto*
done
have *b-bound*: $sum\ (lookup\ b)\ (keys\ b) \leq$ *total-degree* Q
unfolding *total-degree.rep-eq*
apply (*rule Max-ge*)
subgoal by *simp*
subgoal using *b-key* **by** *auto*
done
have $sum\ (lookup\ m)\ (keys\ m) = sum\ (lookup\ a)\ (keys\ a) + sum\ (lookup\ b)\ (keys\ b)$
unfolding *m-def*
by (*rule setsum-keys-plus-distrib*; *auto*)
thus $sum\ (lookup\ m)\ (keys\ m) \leq$ *total-degree* P + *total-degree* Q
using *a-bound b-bound* **by** *simp*
qed

thus *?thesis* **unfolding** *total-degree.rep-eq* **by** *simp*
qed

lemma *total-degree-mult-non-zero*:
fixes $P\ Q :: 'a::idom\ mpoly$
assumes $P \neq 0\ Q \neq 0$
shows $total\text{-}degree\ (P * Q) = total\text{-}degree\ P + total\text{-}degree\ Q$
proof (*rule antisym*)
show $total\text{-}degree\ (P * Q) \leq total\text{-}degree\ P + total\text{-}degree\ Q$ **by** (*rule total-degree-mult*)
next
define a **where** $a = Max\ ((\lambda m.\ sum\ (lookup\ m)\ (keys\ m))\ `\ keys\ (mapping\text{-}of\ P))$
define b **where** $b = Max\ ((\lambda m.\ sum\ (lookup\ m)\ (keys\ m))\ `\ keys\ (mapping\text{-}of\ Q))$
define c **where** $c = Max\ ((\lambda m.\ sum\ (lookup\ m)\ (keys\ m))\ `\ keys\ (mapping\text{-}of\ (P * Q)))$
define p **where** $p = Max\ \{m \in keys\ (mapping\text{-}of\ P).\ sum\ (lookup\ m)\ (keys\ m) = a\}$
define q **where** $q = Max\ \{m \in keys\ (mapping\text{-}of\ Q).\ sum\ (lookup\ m)\ (keys\ m) = b\}$
define r **where** $r = Max\ \{m \in keys\ (mapping\text{-}of\ (P * Q)).\ sum\ (lookup\ m)\ (keys\ m) = c\}$
have *0*: $P * Q \neq 0$ **using** *assms* **by** *auto*
have *1*: $keys\ (mapping\text{-}of\ P) \neq \{\}$
$keys\ (mapping\text{-}of\ Q) \neq \{\}$
$keys\ (mapping\text{-}of\ (P * Q)) \neq \{\}$
using *assms 0 mapping-of-inject zero-mpoly.rep-eq*
apply *fastforce*
apply (*metis assms(2) keys-eq-empty mapping-of-inverse zero-mpoly.abs-eq*)
by (*metis 0 keys-eq-empty mapping-of-inverse zero-mpoly.abs-eq*)
have *2*: $a \in (\lambda m.\ sum\ (lookup\ m)\ (keys\ m))\ `\ keys\ (mapping\text{-}of\ P)$
$b \in (\lambda m.\ sum\ (lookup\ m)\ (keys\ m))\ `\ keys\ (mapping\text{-}of\ Q)$
$c \in (\lambda m.\ sum\ (lookup\ m)\ (keys\ m))\ `\ keys\ (mapping\text{-}of\ (P * Q))$
unfolding *a-def b-def c-def*
using *1 Max-in*
by *simp+*
have *3*: $p \in \{m \in keys\ (mapping\text{-}of\ P).\ sum\ (lookup\ m)\ (keys\ m) = a\}$
$q \in \{m \in keys\ (mapping\text{-}of\ Q).\ sum\ (lookup\ m)\ (keys\ m) = b\}$
$r \in \{m \in keys\ (mapping\text{-}of\ (P * Q)).\ sum\ (lookup\ m)\ (keys\ m) = c\}$
unfolding *p-def q-def r-def*
using *2 Max-in*[*of* $\{m \in keys\ (mapping\text{-}of\ \text{-}).\ sum\ (lookup\ m)\ (keys\ m) = \text{-}\}$]
by *auto*
have *4*: $\bigwedge m.\ m \in keys\ (mapping\text{-}of\ P) \Longrightarrow sum\ (lookup\ m)\ (keys\ m) \leq a$
$\bigwedge m.\ m \in keys\ (mapping\text{-}of\ Q) \Longrightarrow sum\ (lookup\ m)\ (keys\ m) \leq b$
$\bigwedge m.\ m \in keys\ (mapping\text{-}of\ (P * Q)) \Longrightarrow sum\ (lookup\ m)\ (keys\ m) \leq c$
unfolding *a-def b-def c-def*
by *auto*
have *5*: $\bigwedge m.\ m \in keys\ (mapping\text{-}of\ P) \Longrightarrow sum\ (lookup\ m)\ (keys\ m) = a \Longrightarrow m \leq p$

$\bigwedge m.\ m \in keys\ (mapping\text{-}of\ Q) \Longrightarrow sum\ (lookup\ m)\ (keys\ m) = b \Longrightarrow m \leq q$

$\bigwedge m.\ m \in keys\ (mapping\text{-}of\ (P * Q)) \Longrightarrow sum\ (lookup\ m)\ (keys\ m) = c \Longrightarrow m \leq r$

unfolding *p-def q-def r-def*

by *auto*

have $p' + q' = p + q \Longrightarrow$
$lookup\ (mapping\text{-}of\ P)\ p' \neq 0 \Longrightarrow$
$lookup\ (mapping\text{-}of\ Q)\ q' \neq 0 \Longrightarrow$
$p' = p \wedge q' = q$ **for** $p'\ q'$

proof −

assume *6*: $p' + q' = p + q$

assume *7*: $lookup\ (mapping\text{-}of\ P)\ p' \neq 0\ lookup\ (mapping\text{-}of\ Q)\ q' \neq 0$

have *8*: $sum\ (lookup\ p')\ (keys\ p') \leq sum\ (lookup\ p)\ (keys\ p)\ sum\ (lookup\ q')\ (keys\ q') \leq sum\ (lookup\ q)\ (keys\ q)$

using *3 4 7* **unfolding** *in-keys-iff* **by** *auto*

have $sum\ (lookup\ p')\ (keys\ p') + sum\ (lookup\ q')\ (keys\ q') = sum\ (lookup\ p)\ (keys\ p) + sum\ (lookup\ q)\ (keys\ q)$

using *6 setsum-keys-plus-distrib*[*of* λ- *x. x p q*] *setsum-keys-plus-distrib*[*of* λ- *x. x p' q'*]

by *auto*

hence $sum\ (lookup\ p')\ (keys\ p') = sum\ (lookup\ p)\ (keys\ p) \wedge sum\ (lookup\ q')\ (keys\ q') = sum\ (lookup\ q)\ (keys\ q)$

using *8*

by *linarith*

hence $p' \leq p\ q' \leq q$

using *3 5 7* **unfolding** *in-keys-iff* **by** *blast+*

thus $p' = p \wedge q' = q$

using *6*

by (*metis add.commute add-le-cancel-left antisym*)

qed

hence *9*: $(p + q = p' + q' \wedge$
$lookup\ (mapping\text{-}of\ P)\ p' \neq 0 \wedge$
$lookup\ (mapping\text{-}of\ Q)\ q' \neq 0) \longleftrightarrow$
$(p', q') = (p, q)$ **for** $p'\ q'$

using *3* **by** *auto*

have $lookup\ (mapping\text{-}of\ (P * Q))\ (p + q) =$
$(\sum (p', q').\ lookup\ (mapping\text{-}of\ P)\ p' * lookup\ (mapping\text{-}of\ Q)\ q'\ when\ p + q = p' + q')$

unfolding *times-mpoly.rep-eq times-poly-mapping.rep-eq*

by (*rule prod-fun-unfold-prod*; *simp*)

also have ... = $(\sum (p', q').\ lookup\ (mapping\text{-}of\ P)\ p' * lookup\ (mapping\text{-}of\ Q)\ q'\ when\ p + q = p' + q' \wedge lookup\ (mapping\text{-}of\ P)\ p' \neq 0 \wedge lookup\ (mapping\text{-}of\ Q)\ q' \neq 0)$

apply (*rule Sum-any.cong*)

unfolding *when-def* **apply** *auto*

done

also have ... = $Sum\text{-}any\ (\lambda a.\ (case\ a\ of\ (p', q') \Rightarrow lookup\ (mapping\text{-}of\ P)\ p' * lookup\ (mapping\text{-}of\ Q)\ q')\ when\ a = (p, q))$

```
    unfolding 9
    by (rule Sum-any.cong; auto)
  also have ... = lookup (mapping-of P) p * lookup (mapping-of Q) q
    unfolding Sum-any-when-equal
    by auto
  also have ... ≠ 0
    using 3 by auto
  finally have 10: lookup (mapping-of (P * Q)) (p + q) ≠ 0 .
  have a + b = sum (lookup (p + q)) (keys (p + q))
    using 3 setsum-keys-plus-distrib[of λ- x. x p q]
    by simp
  also have ... ≤ c
    by (rule 4(3)[unfolded in-keys-iff]; rule 10)
  finally show total-degree P + total-degree Q ≤ total-degree (P * Q)
    unfolding a-def b-def c-def total-degree.rep-eq
    using 1
    by simp
qed

lemma total-degree-pow: total-degree (P ^ n) ≤ n * total-degree P
  apply (induction n, auto simp: degree-mult)
  by (meson nat-add-left-cancel-le order-trans total-degree-mult)

lemma total-degree-pow-positive[total-degree-simps]:
  fixes P :: 'a::idom mpoly
  assumes n > 0
  shows total-degree (P ^ n) = n * total-degree P
proof (induction rule: nat-induct-non-zero[OF assms])
  show total-degree (P ^ 1) = 1 * total-degree P by simp
next
  fix m :: nat
  assume 0: m > 0
  assume 1: total-degree (P ^ m) = m * total-degree P
  show total-degree (P ^ Suc m) = Suc m * total-degree P
  proof cases
    assume 2: P = 0
    show ?thesis unfolding 2 by simp
  next
    assume 3: P ≠ 0
    show ?thesis
      unfolding power-Suc2
      apply (subst total-degree-mult-non-zero)
      subgoal using 0 3 pow-positive by blast
      subgoal using 3 .
      subgoal using 1 by simp
      done
  qed
qed
```

lemma *total-degree-sum*:
fixes $P :: 'a \Rightarrow 'b$::*comm-monoid-add mpoly*
assumes *S-fin*: *finite S*
shows *total-degree* (*sum P S*) $\leq$ *Max* (*insert 0* ((λ*i*. *total-degree* (*P i*)) ‘ *S*))
apply (*induct rule*: *finite-induct*[*OF S-fin*], *auto*)
apply (*rule le-trans*[*OF total-degree-add*], *simp*)
by (*smt* (*verit*) *Max-insert bot-nat-def finite.insertI finite-imageI insert-absorb2 le-trans*
insert-not-empty le-cases3 le-max-iff-disj max-bot2 insert-commute)

lemma *total-degree-prod*:
assumes *S-fin*: *finite S*
shows *total-degree* (*prod P S*) $\leq$ *sum* (λ*i*. *total-degree* (*P i*)) *S*
apply (*induct rule*: *finite-induct*[*OF S-fin*], *auto*)
by (*rule le-trans*[*OF total-degree-mult*]) *simp*

lemma *Max-function-mono*:
fixes *f g* :: $'a \Rightarrow$ *nat*
assumes *finite A*
assumes $A \neq \{\}$
assumes $\forall\, a{\in}A.\ f\ a \leq g\ a$
shows *Max* (*f* ‘ *A*) $\leq$ *Max* (*g* ‘ *A*)
apply (*subst Max.boundedI*, *auto simp*: *assms*)
by (*rule le-trans*[*of f - g -*, *OF - Max-ge*], *auto simp*: *assms*)

lemma *degree-total-degree-bound*:
degree P v $\leq$ *total-degree P*
proof (*cases v* $\in$ *vars P*)
case *True*
have *m* $\in$ *keys* (*mapping-of P*) $\Longrightarrow$ *lookup m v* $\leq$ *sum* (*lookup m*) (*keys m*) **for** *m*
apply (*cases v* $\in$ *keys m*)
apply (*subst member-le-sum*, *auto*)
using *lookup-not-eq-zero-eq-in-keys* **by** *fastforce*
hence *Max* ((λ*m*. *lookup m v*) ‘ *keys* (*mapping-of P*))
$\leq$ *Max* ((λ*m*. *sum* (*lookup m*) (*keys m*)) ‘ *keys* (*mapping-of P*))
using *True*[*unfolded vars-def*] **by** (*subst Max-function-mono*, *auto*)
then show *?thesis*
using *True*[*unfolded vars-def*]
unfolding *total-degree.rep-eq degree.rep-eq*
by (*smt* (*verit*, *ccfv-threshold*) *Max-ge Max-in dual-order.trans empty-not-insert finite-imageI*
finite-insert finite-keys image-is-empty insert-iff)
next
case *False*
then show *?thesis*
by (*simp add*: *in-vars-non-zero-degree*)
qed

lemma *total-degree-bound*:
  total-degree P ≤ *sum* (*degree P*) (*vars P*)
  unfolding *total-degree.rep-eq degree.rep-eq*
  apply (*subst Max.boundedI*, *auto simp*: *vars-def*)
  by (*subst sum-le-included*[*of - - - λx. x*], *auto*)

end
theory *Poly-Expansions*
  imports *Total-Degree*
begin

1.5 Explicit expansions

lemma *mpoly-keys-subset*: *keys* (*mapping-of P*) ⊆ *Abs-poly-mapping* ‘ (!$_0$) ‘
  {*i*. *length i* = *max-vars P* + *1* ∧ *sum-list i* ≤ *total-degree P*}
proof
  fix *k* **assume** *assm*: *k* ∈ *keys* (*mapping-of P*)
  define *l*::*nat list* **where** *l* ≡ *map* (*lookup k* ∘ *nat*) [*0*..*max-vars P*]

  have *0*: *lookup k i* = *l*!$_0$*i* **for** *i*
  proof (*cases i* ≤ *max-vars P*)
    case *True*
    have *l*!$_0$*i* = (*lookup k* ∘ *nat*) ([*0*..*max-vars P*]!*i*)
    by (*smt* (*verit*, *best*) *Suc-as-int True l-def le-imp-less-Suc length-map length-upto nth0-nth*
        nth-map)
    also have ... = *lookup k i*
      by (*simp add*: *True*)
    finally show *?thesis*
      by *simp*
  next
    case *False*
    hence *lookup k i* = *0*
      unfolding *max-vars-def vars-def*
    by (*metis Suc-eq-plus1 after-max-vars assm in-keys-iff max-vars-def not-less-eq-eq vars-def*)
    thus *?thesis*
      using *False* **by** (*simp add*: *l-def nth0-0*)
  qed

  have *1*: *k* = *Abs-poly-mapping* ((!$_0$) *l*)
    by (*simp add*: *0 poly-mapping-eqI*)

  have *sum-list l* = (∑ *i*=*0*..<*length l*. *l*!$_0$*i*)
    using *sum-list-sum-nth* **by** (*metis nth0-nth sum.ivl-cong*)
  also have ... = *sum* (*lookup k*) (*keys k*)
   by (*metis 1 calculation keys.rep-eq lookup-Abs-poly-mapping-nth0 nth0-sum-list*)
  also have ... ≤ *total-degree P*
    unfolding *total-degree-def* **using** *assm* **by** *simp*

finally have *2*: *sum-list l* ≤ *total-degree P* **.**

show *k* ∈ *Abs-poly-mapping* ‘ (!$_0$) ‘ {*i*. *length i* = *max-vars P* + *1* ∧ *sum-list i* ≤ *total-degree P*}
using *1 2 l-def* **by** *simp*
qed

lemma *monom-single*: *monom* (*Poly-Mapping.single v p*) *a* = *Const a* ∗ (*Var v*) ^ *p*
proof (*induction p*)
case *0*
show *?case* **by** (*simp add*: *Const.abs-eq Const*$_0$*-def monom.abs-eq*)
next
case (*Suc p*)
show *?case*
unfolding *power-Suc2 semigroup-mult-class.mult.assoc*[*symmetric*] *Suc*[*symmetric*]
by (*metis Suc-eq-plus1 Var.abs-eq Var*$_0$*-def monom.abs-eq mult.right-neutral mult-monom single-add*)
qed

lemma *coeff-monom* :
coeff (*monom m a*) *m′* = (*if m*=*m′ then a else 0*)
unfolding *coeff-def monom-def* **using** *lookup-single-eq lookup-single-not-eq*
by (*metis monom.rep-eq monom-def*)

lemma *monom-eq-var*:
monom (*Abs-poly-mapping* (λ*v′*. (*Suc 0*) *when v*=*v′*)) *1* = *MPoly* (*Var*$_0$ *v*)
proof −
have *0*: *Poly-Mapping.single* (*Abs-poly-mapping* (λ*v′*. *Suc 0 when v*=*v′*)) *1* = *Var*$_0$ *v*
unfolding *Var*$_0$*-def single-def* **by** *simp*
hence *MPoly* (*Poly-Mapping.single* (*Abs-poly-mapping* (λ*v′*. *Suc 0 when v*=*v′*)) *1*) = *MPoly* (*Var*$_0$ *v*)
by *metis*
thus *?thesis* **unfolding** *monom-def* **by** *simp*
qed

lemma *monom-eq-power-var*:
monom (*Abs-poly-mapping* (λ*v′*. *n when v* = *v′*)) *1* = *MPoly* (*Var*$_0$ *v*) ^*n*
proof (*induction n*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc n*)
note *t*=*this*
have *0*: *lookup* (*Abs-poly-mapping* (λ*v′*. *n when v* = *v′*)) = (λ*v′*. *n when v* = *v′*)
∧ *lookup* (*Abs-poly-mapping* (λ*v′*. *Suc 0 when v* = *v′*)) = (λ*v′*. *Suc 0 when v* = *v′*)
using *lookup-Abs-poly-mapping* **by** (*simp add*: *when-def*)

hence (λk. *lookup* (*Abs-poly-mapping* ($\lambda v'$. *n when* $v = v'$)) *k*
+ *lookup* (*Abs-poly-mapping* ($\lambda v'$. *Suc 0 when* $v = v'$)) *k*)
= (λk. ($\lambda v'$. *n when* $v = v'$) *k* + ($\lambda v'$. *Suc 0 when* $v = v'$) *k*)
by *simp*
also have ... = (λk. *Suc n when* $v = k$)
by (*metis when*(*1*) *when*(*2*) *add-0-iff add-Suc-right*)
finally have *0*: (λk. *lookup* (*Abs-poly-mapping* ($\lambda v'$. *n when* $v = v'$)) *k*
+ *lookup* (*Abs-poly-mapping* ($\lambda v'$. *Suc 0 when* $v = v'$)) *k*)
= (λk. *Suc n when* $v = k$) **by** *blast*
have *Abs-poly-mapping* ($\lambda v'$. *n when* $v = v'$) + *Abs-poly-mapping* ($\lambda v'$. *Suc 0 when* $v = v'$)
= *Abs-poly-mapping* ($\lambda v'$. *Suc n when* $v = v'$)
unfolding *plus-poly-mapping-def* **apply** *simp* **using** *0* **by** *simp*
then show *?case* **using** *monom-eq-var*[*of v*] *mult-monom*[*of Abs-poly-mapping* ($\lambda v'$. *n when* $v = v'$) *1*
Abs-poly-mapping ($\lambda v'$. *Suc 0 when* $v = v'$) *1*] *power-Suc2 lambda-one t* **by** *metis*
qed

lemma *coeff-prod-monom-not-enough*:
fixes *m m' a*
assumes $\exists k$. *lookup m k* < *lookup m' k*
shows *coeff* (*monom m' a* ∗ *Q*) *m* = *0*
proof −
from *assms* **obtain** *k* **where** *lookup m k* < *lookup m' k* **by** *auto*
hence *0*: *lookup m k* ≠ *lookup m' k* + *v* **for** *v*
by *simp*
have *m* ≠ *m'* + *m''* **for** *m''*
unfolding *lookup-inject*[*symmetric*]
using *0*[*of lookup m'' k*]
by (*auto simp*: *lookup-add*)
thus *?thesis*
unfolding *coeff-def times-mpoly.rep-eq times-poly-mapping.rep-eq prod-fun-def*
unfolding *coeff-def*[*symmetric*] *coeff-monom*
unfolding *when-def*[*symmetric*] *when-mult Sum-any-when-equal'*
apply (*subst zero-class.when*(*2*))
by *auto*
qed

lemma *finite-sum-mpoly-commute*:
finite S ⟹ (∑ *m*∈*S*. *MPoly* (*f m*)) = *MPoly* (∑ *m*∈*S*. *f m*)
by (*simp add*: *add-to-finite-sum plus-mpoly.abs-eq zero-mpoly-def*)

lemma *finite-prod-mpoly-commute*:
finite S ⟹ (∏ *m*∈*S*. *MPoly* (*f m*)) = *MPoly* (∏ *m*∈*S*. *f m*)
by (*simp add*: *mult-to-finite-prod one-mpoly-def times-mpoly.abs-eq*)

lemma *power-mpoly-commute*: *MPoly a* ^ *p* = *MPoly* (*a* ^ *p*)
by (*transfer*; *auto*)

lemma *finite-sum-poly-mapping-commute* :
finite S $\Longrightarrow$ ($\bigwedge m$. *finite* $\{x.\ f\ m\ x \neq 0\}$) $\Longrightarrow$
$(\sum m \in S.\ Abs\text{-}poly\text{-}mapping\ (f\ m)) = Abs\text{-}poly\text{-}mapping\ (\lambda x.\ \sum m \in S.\ f\ m\ x)$
proof (*induction card S arbitrary*:*S*)
case *0*
then show *?case* **by** *force*
next
case (*Suc n*)
then obtain *y* **where** *y-prop*: $y \in S$ **by** *fastforce*
define S' **where** $S' = S - \{y\}$
hence *card-S'*:*card* $S' = n$ **using** *Suc* **by** (*simp add*: *y-prop*)
hence *IH*: $(\sum m \in S'.\ Abs\text{-}poly\text{-}mapping\ (f\ m)) = Abs\text{-}poly\text{-}mapping\ (\lambda x.\ \sum m \in S'.\ f\ m\ x)$
using *Suc* **by** *force*
have *sum-disj*: $(\sum m \in S'.\ f\ m\ x) + f\ y\ x = (\sum m \in S.\ f\ m\ x)$ **for** *x* **using** *S'-def*
by (*metis Suc.prems*(*1*) *add.commute sum.remove y-prop*)
have *finite S'* **by** (*simp add*: *S'-def Suc.prems*(*1*))
hence $(\forall m \in S'.\ f\ m\ x = 0) \Longrightarrow (\sum m \in S'.\ f\ m\ x) = 0$ **for** *x* **by** *simp*
hence $\{x.\ (\sum m \in S'.\ f\ m\ x) \neq 0\} \subseteq (\bigcup m \in S'.\ \{x.\ f\ m\ x \neq 0\})$
by (*smt* (*verit*, *best*) *UN-I mem-Collect-eq subsetI*)
hence *fin-sum*: *finite* $\{x.\ (\sum m \in S'.\ f\ m\ x) \neq 0\}$ **using** *Suc*(*4*) ‹*finite S'*› *finite-subset* **by** *blast*
have $(\sum m \in S.\ Abs\text{-}poly\text{-}mapping\ (f\ m)) = (\sum m \in S'.\ Abs\text{-}poly\text{-}mapping\ (f\ m)) + Abs\text{-}poly\text{-}mapping\ (f\ y)$
using *S'-def* **by** (*metis Suc.prems*(*1*) *add.commute sum.remove y-prop*)
also have ... = $Abs\text{-}poly\text{-}mapping\ (\lambda x.\ \sum m \in S'.\ f\ m\ x) + Abs\text{-}poly\text{-}mapping\ (f\ y)$ **using** *IH* **by** *argo*
also have ... = $Abs\text{-}poly\text{-}mapping\ (\lambda x.\ (\sum m \in S'.\ f\ m\ x) + f\ y\ x)$
unfolding *plus-poly-mapping-def* **apply** *simp*
using *lookup-Abs-poly-mapping fin-sum Suc*(*4*)[*of y*] **by** *force*
also have ... = $Abs\text{-}poly\text{-}mapping\ (\lambda x.\ (\sum m \in S.\ f\ m\ x))$ **using** *sum-disj* **by** *presburger*
finally show *?case* **by** *blast*
qed

lemma *coeff-sum-monom*:
assumes *finite* $\{m.\ f\ m \neq 0\}$
shows *coeff* ($Sum\text{-}any\ (\lambda m.\ monom\ m\ (f\ m))) = f$
proof −
define *S* **where** $S = \{m.\ f\ m \neq 0\}$
have *fin-S*: *finite S* **unfolding** *S-def* **using** *assms* **by** *simp*
have $Sum\text{-}any\ (\lambda m.\ monom\ m\ (f\ m)) = Sum\text{-}any\ (\lambda m.\ MPoly\ (Poly\text{-}Mapping.single\ m\ (f\ m)))$
unfolding *monom-def* **by** *simp*
also have ... = $(\sum m \in S.\ MPoly\ (Poly\text{-}Mapping.single\ m\ (f\ m)))$
unfolding *S-def*
by (*metis* (*mono-tags*, *lifting*) *Collect-mono Sum-any.expand-superset assms single-zero zero-mpoly-def*)

also have ... = *MPoly* ($\sum m{\in}S.$ (*Poly-Mapping.single m* (*f m*)))
using *finite-sum-mpoly-commute fin-S* **by** *fastforce*
finally have *0*: *Sum-any* ($\lambda m.$ *monom m* (*f m*)) = (*MPoly* ($\sum a{\in}S.$ (*Poly-Mapping.single a* (*f a*))))
by *simp*

have *1*:($\sum a{\in}S.$ (*Poly-Mapping.single a* (*f a*))) = *Abs-poly-mapping* ($\lambda m.$ $\sum a{\in}S.$ *f a when a = m*)
unfolding *single-def* **apply** *simp*
using *finite-sum-poly-mapping-commute*[*of S* ($\lambda a.$ ($\lambda k'.$ *f a when a = k'*))] *fin-S* **by** *fastforce*
have ($\sum a{\in}S.$ *f a when a = m*) = *f m* **for** *m*
proof (*cases m*$\in$*S*)
case *True*
hence *finite S* $\wedge$ *S* = (*S*$-${*m*})$\cup${*m*} $\wedge$ (*S*$-${*m*})$\cap${*m*}={} **using** *fin-S* **by** *blast*
hence ($\sum a{\in}S.$ *f a when a = m*) = ($\sum a{\in}$(*S*$-${*m*}). *f a when a = m*) + ($\sum a{\in}${*m*}. *f a when a = m*)
by (*meson True bot.extremum insert-subset sum.subset-diff*)
also have ... = *f m* **by** *auto*
finally show *?thesis* **using** ‹*m*$\in$*S*› **by** *presburger*
next
case *False*
hence *a*$\in$*S* $\Longrightarrow$ (*f a when a = m*) = *0* **for** *a* **unfolding** *when-def* **by** *force*
hence ($\sum a{\in}S.$ *f a when a = m*) = *0* **by** *simp*
then show *?thesis* **using** ‹*m*$\notin$*S*› *S-def* **by** *force*
qed
hence ($\sum a{\in}S.$ (*Poly-Mapping.single a* (*f a*))) = *Abs-poly-mapping f*
using *1* **by** *force*
hence *coeff* (*Sum-any* ($\lambda m.$ *monom m* (*f m*))) = *lookup* (*mapping-of* (*MPoly* (*Abs-poly-mapping f*)))
using *0 coeff-def* **by** *force*
thus *?thesis* **using** *MPoly-inverse UNIV-I lookup-Abs-poly-mapping*[*of f*] *assms* **by** *metis*
qed

lemma *coeff-sum-monom-bis*:
assumes *finite* {*m*. *f m*$\neq$*0*} **and** *finite S*
shows *coeff* ($\sum m{\in}S.$ *monom m* (*f m*)) *m'*= (*if m'*$\in$ *S then f m' else 0*)
proof −
define *g* **where** *g* = ($\lambda m.$ *if m*$\in$*S then f m else 0*)
hence {*m*. *g m* $\neq$ *0*} $\subseteq$ {*m*. *f m* $\neq$ *0*} **by** *auto*
hence *fin-g*: *finite* {*m*. *g m* $\neq$ *0*} **using** *assms finite-subset* **by** *blast*
hence ($\sum m{\in}S.$ *monom m* (*f m*)) = *Sum-any* ($\lambda m.$ *monom m* (*g m*)) **using** *assms*(*2*)
by (*smt* (*verit*, *best*) *Sum-any.conditionalize Sum-any.cong g-def monom.abs-eq monom-zero*
single-zero)
hence *coeff* ($\sum m{\in}S.$ *monom m* (*f m*)) *m'* = *coeff* (*Sum-any* ($\lambda m.$ *monom m*

$(g\ m)))\ m'$ **by** *simp*
also have $... = g\ m'$ **using** *fin-g coeff-sum-monom* **by** *fastforce*
finally show *?thesis* **unfolding** *g-def* **by** *blast*
qed

lemma *cst-poly-times-monom*: $MPoly\ (Const_0\ (a::('a::semiring\text{-}1)))\ *\ monom\ m\ b = monom\ m\ (a*b)$
proof −
have $MPoly\ (Const_0\ a)\ *\ monom\ m\ b = monom\ 0\ a\ *\ monom\ m\ b$
using $Const_0\text{-}def$ **by** (*metis monom.abs-eq*)
also have $... = monom\ m\ (a*b)$
using *mult-monom*[*of 0 a m b*] **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma *cst-poly-times-monom-one*: $MPoly\ (Const_0\ (a::('a::semiring\text{-}1)))\ *\ monom\ m\ 1 = monom\ m\ a$
using *cst-poly-times-monom*[*of a m 1*] **by** *simp*

lemma *poly-eq-sum-monom*: $P = Sum\text{-}any\ (\lambda m.\ monom\ m\ (coeff\ P\ m))$
proof −
have $\bigwedge m'.\ coeff\ P\ m' = coeff\ (Sum\text{-}any\ (\lambda m.\ monom\ m\ (coeff\ P\ m)))\ m'$
using *coeff-sum-monom coeff-def*
by (*metis* (*full-types*) *finite-lookup*)
thus *?thesis* **using** *coeff-eq* **by** *blast*
qed

lemma *poly-eq-sum-monom-alt*: $P = (\sum m{\in}(keys\ (mapping\text{-}of\ P)).\ monom\ m\ (coeff\ P\ m))$
using *poly-eq-sum-monom*[*of P*]
by (*smt* (*verit*) *Sum-any.conditionalize Sum-any.cong coeff-keys finite-keys monom.abs-eq monom-zero single-zero*)

lemma *coeff-sum*:
fixes *P*::- ⇒ *'a*::*comm-monoid-add mpoly*
assumes *S-fin*: *finite S*
shows $coeff\ (sum\ P\ S)\ m = (\sum i{\in}S.\ coeff\ (P\ i)\ m)$
proof (*induct rule*: *finite-induct*[*OF S-fin*])
case *1* **thus** *?case* **by** (*simp add*: *coeff-def zero-mpoly.rep-eq*)
next
case (*2 x S*)
have $coeff\ (sum\ P\ (insert\ x\ S))\ m = coeff\ (sum\ P\ S + P\ x)\ m$
by (*simp add*: *2.hyps*(*1*) *2.hyps*(*2*) *add.commute*)
also have $... = coeff\ (sum\ P\ S)\ m + coeff\ (P\ x)\ m$
by (*simp add*: *coeff-add*)
also have $... = (\sum i{\in}S.\ coeff\ (P\ i)\ m) + coeff\ (P\ x)\ m$
by (*simp add*: *2*)
finally show *?case*

by (*simp add*: *2.hyps*(*1*) *2.hyps*(*2*) *add.commute*)
qed

lemma *coeff-var-power-le*:
j ≤ *i* ⟹ *MPoly-Type.coeff* (*Var v* ^ *j* ∗ *P*) (*Poly-Mapping.single v i*)
= *MPoly-Type.coeff P* (*Poly-Mapping.single v* (*i* − *j*))
proof −
assume *j* ≤ *i*
then have *eqi*: *i* = *j* + (*i* − *j*)
by *auto*

show *?thesis*
apply (*subst eqi*)
unfolding *Var.abs-eq monom-eq-power-var*[*symmetric*] *Poly-Mapping.single-add*
Poly-Mapping.single.abs-eq[*symmetric*]
apply (*subst More-MPoly-Type.coeff-monom-mult*)
by *simp*
qed

lemma *coeff-var-power-eq*: *MPoly-Type.coeff* (*Var v* ^ *i*) (*Poly-Mapping.single v i*) = *1*
using *coeff-var-power-le*[*of i i v 1*]
unfolding *insertion-trivial*[*symmetric*] *MPoly-Type.coeff-def one-mpoly.rep-eq*
by *auto*

lemma *coeff-const*: *i* > *0* ⟹ *MPoly-Type.coeff* (*Const c*) (*Poly-Mapping.single v i*) = *0*
unfolding *MPoly-Type.coeff-def Const.rep-eq Const*$_0$*-def lookup-single*
unfolding *single-zero*[*of v*, *symmetric*]
apply (*rule when*(*2*))
by (*metis lookup-single-eq nat-less-le*)

lemma *mpoly-univariate-expansion*:
fixes *P*::′*a*::*comm-semiring-1 mpoly* **and** *v*::*nat*
assumes *univariate*: *vars P* ⊆ {*v*}
shows *P* = *Sum-any* (λ*i*. *monom* (*Poly-Mapping.single v i*) (*coeff P* (*Poly-Mapping.single v i*)))
proof −
define *K* **where** *K* = *keys* (*mapping-of P*)
define *f* **where** *f* = (λ*m*::*nat*⇒$_0$*nat*. *lookup m v*)

have *rewrite-monom*: *m*∈*K* ⟹ *m* = *Poly-Mapping.single v* (*f m*) **for** *m*
proof −
assume *m*∈*K*
hence *keys m* ⊆ *vars P*
unfolding *vars-def K-def* **by** *auto*
hence *keys m* ⊆ {*v*}
using *univariate* **by** *auto*
thus *?thesis*

```
    unfolding f-def
    by (metis diff-shunt-var keys-eq-empty lookup-single-eq remove-key-keys
        remove-key-single remove-key-sum)
  qed

  have f-inj: inj-on f K
    using rewrite-monom by (metis inj-onI)

  have coeff-null: i ∉ f ‘ K ⟹
    monom (Poly-Mapping.single v i) (coeff P (Poly-Mapping.single v i)) = 0 for
i
  proof −
    assume i ∉ f ‘ K
    hence Poly-Mapping.single v i ∉ K
      unfolding f-def by force
    hence coeff P (Poly-Mapping.single v i) = 0
      unfolding K-def coeff-def by (simp add: in-keys-iff)
    thus ?thesis by (metis mult-zero-left smult-0 smult-monom)
  qed

  have P = (∑ m∈K. monom m (coeff P m))
    by (simp add: K-def poly-eq-sum-monom-alt)
  also have ... = (∑ m∈K. monom (Poly-Mapping.single v (f m)) (coeff P (Poly-Mapping.single
v (f m))))
    using rewrite-monom by simp
  also have ... = (∑ i∈f ‘ K. monom (Poly-Mapping.single v i) (coeff P (Poly-Mapping.single
v i)))
    by (metis (mono-tags, lifting) f-inj sum.reindex-cong)
  also have ... = Sum-any (λi. monom (Poly-Mapping.single v i) (coeff P (Poly-Mapping.single
v i)))
    using coeff-null
   by (smt (verit) K-def Sum-any.conditionalize Sum-any.cong finite-imageI finite-keys)
  finally show ?thesis .
qed


lemma term-expansion-lemma-1: i ≠ [] ⟹
  Poly-Mapping.single (Abs-poly-mapping ((!0) i)) (1 :: 'a::comm-semiring-1) =
  (∏ s = 0..(length i − 1). Var0 s ^ (i ! s))
proof (induction i rule: length-induct)
  case (1 i)
  show ?case proof (cases i rule: rev-exhaust)
    case Nil
    thus ?thesis using 1 by auto
  next
    case (snoc i' x)
    show ?thesis proof cases
      assume i'-def: i' = []
      have Poly-Mapping.single (Abs-poly-mapping ((!0) [x])) 1 = Var0 0 ^ x
```

```
      unfolding MPoly-Type.Var_0-power by auto
    thus ?thesis unfolding snoc i'-def by auto
  next
    assume i'-non-empty: i' ≠ []
    have (∏ s = 0..length i'. Var_0 s ^ ((i' @ [x]) ! s)) =
      Poly-Mapping.single (Abs-poly-mapping ((!_0) (i' @ [x]))) (1 :: 'a) proof −
      have (∏ s = 0..length i'. Var_0 s ^ ((i' @ [x]) ! s)) =
        (∏ s = 0..Suc (length i' − 1). Var_0 s ^ ((i' @ [x]) ! s))
        using i'-non-empty by auto
      also have ... = (∏ s = 0..length i' − 1. Var_0 s ^ ((i' @ [x]) ! s)) *
        Var_0 (Suc (length i' − 1)) ^ ((i' @ [x]) ! (Suc (length i' − 1)))
        using prod.atLeast0-atMost-Suc by auto
      also have ... = (∏ s = 0..length i' − 1. Var_0 s ^ ((i' @ [x]) ! s)) *
        Var_0 (length i') ^ ((i' @ [x]) ! (length i'))
        using i'-non-empty by auto
      also have ... = (∏ s = 0..length i' − 1. Var_0 s ^ ((i' @ [x]) ! s)) *
        Var_0 (length i') ^ x by auto
      also have ... = (∏ s = 0..length i' − 1. Var_0 s ^ (i' ! s)) *
        Var_0 (length i') ^ x
        by (auto simp add: List.nth-append i'-non-empty order-le-less-trans)
      also have ... = Poly-Mapping.single (Abs-poly-mapping ((!_0) i'))
        (1 :: 'a) * Var_0 (length i') ^ x using 1 snoc i'-non-empty by auto
      also have ... = Poly-Mapping.single (Abs-poly-mapping ((!_0) i')) 1 *
        Poly-Mapping.single (Poly-Mapping.single (length i') x) 1
        unfolding MPoly-Type.Var_0-power by simp
      also have ... = Poly-Mapping.single (Abs-poly-mapping ((!_0) i))
        (1 :: 'a) proof −
        have Abs-poly-mapping ((!_0) i') + Poly-Mapping.single (length i') x =
          Abs-poly-mapping ((!_0) i) unfolding snoc by auto
        thus ?thesis by (simp add: Poly-Mapping.mult-single)
      qed
      finally show ?thesis unfolding snoc .
    qed
    thus ?thesis unfolding snoc by auto
  qed
 qed
qed

lemma term-expansion-lemma-2: i ≠ [] ⟹
  monom (Abs-poly-mapping ((!_0) i)) c =
  MPoly (Const_0 c) * MPoly (∏ s = 0..length i − 1. Var_0 s ^ (i ! s))
  unfolding term-expansion-lemma-1[symmetric] monom.abs-eq[symmetric]
   cst-poly-times-monom-one by simp

lemma term-expansion: i ≠ [] ⟹
  monom (Abs-poly-mapping ((!_0) i)) c =
  Const c * (∏ s = 0..length i − 1. Var s ^ (i ! s))
 unfolding MPoly-Type.Const.abs-eq MPoly-Type.Var.abs-eq term-expansion-lemma-2
  by (simp add: finite-prod-mpoly-commute power-mpoly-commute)
```

fun *sample-prefix* :: *nat* ⇒ (*nat* ⇒ *'a*) ⇒ *'a list* **where**
  sample-prefix 0 f = [] |
  sample-prefix (*Suc l*) *f* = *sample-prefix l f* @ [*f l*]

lemma *sample-prefix-length*[*simp*]: *length* (*sample-prefix l f*) = *l*
  by (*induction l*; *auto*)

lemma *sample-prefix-cong*:
  (∀ *x*<*n*. *f x* = *g x*) ⟹ *sample-prefix n f* = *sample-prefix n g*
proof −
  assume ∀ *x*<*n*. *f x* = *g x*
  hence *l* ≤ *n* ⟹ *sample-prefix l f* = *sample-prefix l g* **for** *l*
    by (*induction l*; *auto*)
  thus *?thesis* **by** *auto*
qed

lemma *sample-prefix-inv-nth0*: (∀ *i*≥*n*. *f i* = *0*) ⟹ *f* = (!$_0$) (*sample-prefix n f*)
proof (*induction n arbitrary*: *f*)
  case *0*
  thus *?case* **unfolding** *nth0-def* **by** *simp*
next
  case (*Suc n*)
  let *?g* = λ*x*. *f x when x* < *n*
  have *?g* = (!$_0$) (*sample-prefix n ?g*) **using** *Suc* **by** *auto*
  also have ... = (!$_0$) (*sample-prefix n f*)
    using *sample-prefix-cong*[*of n f ?g*] **by** *auto*
  finally have *0*: *?g* = (!$_0$) (*sample-prefix n f*) **.**
  have *f* = (λ*x*. *if x* = *n then f n else ?g x*) **unfolding** *when-def* **using** *Suc* **by** *auto*
  also have ... = (λ*x*. *if x* = *n then f n else* (!$_0$) (*sample-prefix n f*) *x*)
    unfolding *0*[*symmetric*] **by** *simp*
  also have ... = (!$_0$) (*sample-prefix* (*Suc n*) *f*)
    apply (*rule ext*)
    subgoal for *x*
      apply (*cases x* = *n*)
      unfolding *nth0-def* **apply** *simp-all*
      unfolding *List.nth-append* **apply** *simp-all*
      done
    done
  finally show *?case* **.**
qed

lemma *sample-prefix-inj*:
  inj-on (λ*f*. *sample-prefix n f*) {*f*. ∀ *i*≥*n*. (*f i* :: *'a*::*zero*) = *0*}
proof
  fix *g h*
  assume *g* ∈ {*f*. ∀ *i*≥*n*. (*f i* :: *'a*::*zero*) = *0*}
  hence *0*: ∀ *i*≥*n*. *g i* = *0* **by** *auto*

assume $h \in \{f.\ \forall i{\geq}n.\ (f\ i :: 'a{::}zero) = 0\}$
hence *1*: $\forall i{\geq}n.\ h\ i = 0$ **by** *auto*
assume *sample-prefix n g = sample-prefix n h*
hence $(!_0)$ *(sample-prefix n g) =* $(!_0)$ *(sample-prefix n h)* **by** *auto*
thus *g = h* **using** *0 1* **by** (*simp add*: *sample-prefix-inv-nth0*[*symmetric*])
qed

lemma *lookup-nth0-total-degree*:
lookup (mapping-of P) (Abs-poly-mapping $((!_0)\ i)) \neq 0 \Longrightarrow$
sum-list i $\leq$ *total-degree P*
apply *transfer*
apply (*rule Max-ge*)
subgoal for *P i*
apply *simp*
done
subgoal for *P i*
unfolding *image-def keys-def* **apply** *simp*
using *nth0-sum-list*[*of i*] **apply** *auto*
done
done

lemma *prod-monom*: *finite S* $\Longrightarrow$ *prod* $(\lambda s.\ monom\ (x\ s)\ (a\ s))\ S = monom\ (sum\ x\ S)\ (prod\ a\ S)$
proof (*induction card S arbitrary*:*S*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc n*)
then obtain *y* **where** *y-prop*: $y{\in}S$ **by** *fastforce*
define S' **where** $S' = S - \{y\}$
with *Suc*(*3*) **have** *finite S'* **by** *auto*
from *S'-def* **have** *card-S'*: *card* $S' = n$ **using** *Suc* **by** (*simp add*: *y-prop*)
have *disj-un-S*: $S = S'{\cup}\{y\} \wedge S'{\cap}\{y\}{=}\{\} \wedge$*finite S* **using** *y-prop S'-def Suc* **by** *force*
hence $(\prod s{\in}S.\ monom\ (x\ s)\ (a\ s)) = (\prod s{\in}S'.\ monom\ (x\ s)\ (a\ s)) * (\prod s{\in}\{y\}.\ monom\ (x\ s)\ (a\ s))$
by (*meson finite-Un prod.union-disjoint*)
also have *... = monom* $(sum\ x\ S')\ (prod\ a\ S') * monom\ (x\ y)\ (a\ y)$
using *Suc*(*1*)[*of S'*] *card-S'* ‹*finite S'*› **by** *auto*
also have *... = monom* $(sum\ x\ S' + x\ y)\ (prod\ a\ S' * a\ y)$
using *mult-monom* **by** *auto*
finally show $(\prod s{\in}S.\ monom\ (x\ s)\ (a\ s)) = monom\ (sum\ x\ S)\ (prod\ a\ S)$
using *S'-def Suc*
by (*smt* (*verit*, *ccfv-threshold*) *add.commute mult.commute prod.remove sum.remove y-prop*)
qed

lemma *poly-mapping-expansion*: $x = (\sum s{\in}keys\ x.\ Abs\text{-}poly\text{-}mapping\ (\lambda v'.\ lookup\ x\ s\ when\ s = v'))$

proof −
have *h0*: $(\sum xa{\in}keys\ x.\ lookup\ (Abs\text{-}poly\text{-}mapping\ (\lambda v'.\ lookup\ x\ xa\ when\ xa = v'))\ i)$
$= (\sum xa{\in}keys\ x.\ (\lambda v'.\ lookup\ x\ xa\ when\ xa = v')\ i)$ **for** *i*
by (*rule sum.cong*, *auto*)
define *x-fun* **where** *x-fun* ≡ *lookup x*
have *h1*: *lookup x i* = $(\sum xa{\in}keys\ x.\ lookup\ x\ xa\ when\ xa = i)$ **for** *i*
apply (*cases i* ∈ *keys x*, *simp add*: *sum.remove*[*OF finite-keys*])
unfolding *keys.rep-eq x-fun-def*[*symmetric*]
by (*smt* (*verit*, *ccfv-SIG*) *mem-Collect-eq sum.neutral when-simps*(*2*))

thus *?thesis*
unfolding *lookup-inject*[*symmetric*] *lookup-sum h0* **by** *blast*
qed

lemma *monom-expansion*:
shows *monom x c* = *Const c* ∗ $(\prod s{\in}keys\ x.\ Var\ s\ \hat{}\ (lookup\ x\ s))$
unfolding *Var.abs-eq monom-eq-power-var*[*symmetric*]
apply (*subst prod-monom*[*OF finite-keys*, **where** *?a* = λ-. *1*])
apply (*simp add*: *Const.abs-eq cst-poly-times-monom*)
by (*subst poly-mapping-expansion*) *simp*

lemma *monom-expansion′*:
fixes *P* :: *′a*::{*ring-no-zero-divisors*,*comm-semiring-1*} *mpoly*
assumes *x* ∈ *keys* (*mapping-of P*)
shows *monom x* (*coeff P x*) = *Const* (*coeff P x*) ∗ $(\prod s = 0..max\text{-}vars\ P.\ Var\ s\ \hat{}\ (lookup\ x\ s))$
proof (*cases vars P* = {})
case *True*
then show *?thesis*
by (*metis* (*no-types*, *lifting*) *Poly-Expansions.coeff-monom assms atLeastAtMost-singleton coeff-keys empty-iff finite.emptyI keys-zero lookup-zero max-vars-Const monom-expansion mult.commute mult-1 power-0 prod.empty prod.insert vars-empty*)
next
case *False*
then show *?thesis*
unfolding *max-vars-of-nonempty*[*OF False*]
apply (*subst monom-expansion*)
apply (*subst mult-cancel-left*, *rule disjI2*)
apply (*rule prod.mono-neutral-cong-left*)
using *assms* **apply** *simp-all*
subgoal
by (*metis Max-ge UN-I atLeastAtMost-iff le0 subsetI vars-def vars-finite*)
subgoal
by (*simp add*: *in-keys-iff*)
done
qed

lemma *mpoly-multivariate-expansion'*:
fixes *P*::*'a*::{*ring-no-zero-divisors*,*comm-semiring-1*} *mpoly*
shows $P = (\sum m{\in}keys\ (mapping\text{-}of\ P).$
$Const\ (coeff\ P\ m) * (\prod s = 0..max\text{-}vars\ P.\ (Var\ s)\ \hat{}\ (lookup\ m\ s)))$
apply (*subst poly-eq-sum-monom-alt*)
by (*rule sum.cong*, *simp-all add*: *monom-expansion'*)

lemma *mpoly-multivariate-expansion*:
fixes *P*::*'a*::*comm-semiring-1 mpoly*
shows $P = (\sum i \mid length\ i = max\text{-}vars\ P + 1 \wedge sum\text{-}list\ i \leq total\text{-}degree\ P.$
$Const\ (coeff\ P\ (Abs\text{-}poly\text{-}mapping\ ((!_0)\ i))) *$
$(\prod s = 0..max\text{-}vars\ P.\ (Var\ s)\ \hat{}\ (i\ !\ s)))$
proof −
let *?t* = $\lambda m.\ (sample\text{-}prefix\ (max\text{-}vars\ P + 1)\ (lookup\ m))$
have $P = (\sum m{\in}keys\ (mapping\text{-}of\ P).\ monom\ m\ (coeff\ P\ m))$
by (*rule poly-eq-sum-monom-alt*)
also have ... = $(\sum m \mid lookup\ (mapping\text{-}of\ P)\ m \neq 0.\ monom\ m\ (coeff\ P\ m))$
unfolding *Poly-Mapping.in-keys-iff*[*symmetric*] **by** *auto*
also have ... = $(\sum m \mid lookup\ (mapping\text{-}of\ P)\ m \neq 0.$
monom (*Abs-poly-mapping* (*lookup m*)) (*coeff P* (*Abs-poly-mapping* (*lookup m*))))
by *auto*
also have ... = $(\sum i \mid length\ i = max\text{-}vars\ P + 1 \wedge$
$lookup\ (mapping\text{-}of\ P)\ (Abs\text{-}poly\text{-}mapping\ ((!_0)\ i)) \neq 0.$
$monom\ (Abs\text{-}poly\text{-}mapping\ ((!_0)\ i))\ (coeff\ P\ (Abs\text{-}poly\text{-}mapping\ ((!_0)\ i))))$
proof (*rule sum.reindex-cong*[*symmetric*, *of ?t*])
let *?S* = {*m*. *lookup* (*mapping-of P*) *m* ≠ *0*}
let *?T* = {*i*. *length i* = *max-vars P* + *1* ∧
lookup (*mapping-of P*) (*Abs-poly-mapping* ((!$_0$) *i*)) ≠ *0*}
show *inj-on ?t ?S*
proof
fix *m1 m2*
assume *m1* ∈ {*m*. *lookup* (*mapping-of P*) *m* ≠ *0*}
hence *1*: ∀ *v*≥(*max-vars P* + *1*). *lookup m1 v* = *0* **using** *after-max-vars*
by *auto*
assume *m2* ∈ {*m*. *lookup* (*mapping-of P*) *m* ≠ *0*}
hence *2*: ∀ *v*≥(*max-vars P* + *1*). *lookup m2 v* = *0* **using** *after-max-vars*
by *auto*
assume *sample-prefix* (*max-vars P* + *1*) (*lookup m1*) =
sample-prefix (*max-vars P* + *1*) (*lookup m2*)
hence *lookup m1* = *lookup m2*
using *sample-prefix-inj*[*of max-vars P* + *1*] *1 2*
unfolding *inj-on-def* **by** *blast*
thus *m1* = *m2* **by** *auto*
qed
show *?T* = *?t* ‘ *?S*
proof (*rule set-eqI*; *rule iffI*)
fix *i*

```
    assume 3: i ∈ ?T
    let ?m = Abs-poly-mapping ((!0) i)
    from 3 have 4: lookup (mapping-of P) ?m ≠ 0 by auto
    have lookup ?m = (!0) (sample-prefix (max-vars P + 1) (lookup ?m))
      apply (rule sample-prefix-inv-nth0)
      using after-max-vars[of P ?m] 4 by auto
    hence (!0) i = (!0) (sample-prefix (max-vars P + 1) (lookup ?m)) by auto
    hence 5: i = (sample-prefix (max-vars P + 1) (lookup ?m))
      using nth0-inj[of i] 3 by auto
    from 4 5 have ∃ m. lookup (mapping-of P) m ≠ 0 ∧
      i = sample-prefix (max-vars P + 1) (lookup m) by blast
    thus i ∈ ?t ‘ ?S unfolding image-def by auto
  next
    fix i
    assume i ∈ ?t ‘ ?S
    then obtain m where 6: lookup (mapping-of P) m ≠ 0
      i = sample-prefix (max-vars P + 1) (lookup m) by blast
    have 7: length i = max-vars P + 1 using 6 by auto
    have lookup m = (!0) (sample-prefix (max-vars P + 1) (lookup m))
      apply (rule sample-prefix-inv-nth0)
      using after-max-vars[of P m] 6 apply auto
      done
    hence 8: lookup (mapping-of P) (Abs-poly-mapping ((!0) i)) ≠ 0
      using 6 by auto
    from 7 8 show i ∈ ?T by blast
  qed
  show m ∈ ?S ⟹
    monom (Abs-poly-mapping ((!0) (?t m)))
      (coeff P (Abs-poly-mapping ((!0) (?t m)))) =
    monom (Abs-poly-mapping (lookup m))
      (coeff P (Abs-poly-mapping (lookup m))) for m
  proof −
    assume 9: m ∈ ?S
    have lookup m = (!0) (sample-prefix (max-vars P + 1) (lookup m))
      apply (rule sample-prefix-inv-nth0)
      using after-max-vars[of P m] 9 apply auto
      done
    thus ?thesis by auto
  qed
qed
also have ... = (∑ i | length i = max-vars P + 1 ∧ sum-list i ≤ total-degree P.
  monom (Abs-poly-mapping ((!0) i)) (coeff P (Abs-poly-mapping ((!0) i))))
proof (rule sum.mono-neutral-left)
  let ?A = {i. length i = max-vars P + 1 ∧
   lookup (mapping-of P) (Abs-poly-mapping ((!0) i)) ≠ 0}
  let ?B = {i. length i = max-vars P + 1 ∧ sum-list i ≤ total-degree P}
  have ?B ⊆ {i. set i ⊆ {0..total-degree P} ∧ length i = max-vars P + 1}
    using member-le-sum-list by fastforce
  also have finite ... using List.finite-lists-length-eq by auto
```

```
    ultimately show finite ?B using Finite-Set.finite-subset by auto
    show ?A ⊆ ?B using lookup-nth0-total-degree by auto
    show ∀ i∈?B − ?A. monom (Abs-poly-mapping ((!0) i))
      (coeff P (Abs-poly-mapping ((!0) i))) = 0
      by (simp add: coeff-all-0 coeff-def)
  qed
  also have ... = (∑ i | length i = max-vars P + 1 ∧ sum-list i ≤ total-degree P.
    Const (coeff P (Abs-poly-mapping ((!0) i))) *
    (∏ s = 0..max-vars P. (Var s) ^ (i ! s)))
  proof (rule sum.cong[OF refl])
    fix i
    assume i-porp:
      i ∈ {i. length i = max-vars P + 1 ∧ sum-list i ≤ total-degree P}
    hence i ≠ [] length i − 1 = max-vars P by auto
    thus monom (Abs-poly-mapping ((!0) i)) (coeff P (Abs-poly-mapping ((!0) i)))
=
      Const (coeff P (Abs-poly-mapping ((!0) i))) *
      (∏ s = 0..max-vars P. Var s ^ i ! s)
      by (simp add: term-expansion)
  qed
  finally show ?thesis .
qed


lemma mpoly-univariate-expansion-sum:
  fixes P :: ('a::comm-ring-1) mpoly
  assumes vars P ⊆ {v}
  defines q ≡ MPoly-Type.degree P v
  defines coeff-P ≡ (λd. coeff P (Poly-Mapping.single v d))
  shows P = (∑ d = 0..q. Const (coeff-P d) * (Var v) ^ d)
  unfolding coeff-P-def
  apply (subst mpoly-univariate-expansion[of P v, OF assms(1)])
  unfolding monom-single
  unfolding Sum-any.expand-set
  apply (rule sum.mono-neutral-cong-left)
  subgoal by auto
  subgoal unfolding q-def degree.rep-eq
  proof −
    have Const (coeff P (Poly-Mapping.single v p)) * Var v ^ p ≠ 0 ⟹
      p ∈ {0..Max (insert 0 ((λm. lookup m v) ‘ keys (mapping-of P)))} for p
    proof −
      assume Const (coeff P (Poly-Mapping.single v p)) * Var v ^ p ≠ 0
      hence coeff P (Poly-Mapping.single v p) ≠ 0
        by (metis Const-zero lambda-zero)
      hence p ∈ (λm. lookup m v) ‘ keys (mapping-of P)
        by (metis coeff-def imageI in-keys-iff lookup-single-eq)
      thus p ∈ {0..Max (insert 0 ((λm. lookup m v) ‘ keys (mapping-of P)))}
        by simp
    qed
```

thus $\{a.\ Const\ (coeff\ P\ (Poly\text{-}Mapping.single\ v\ a)) * Var\ v\ \hat{}\ a \neq 0\}$
$\subseteq \{0..Max\ (insert\ 0\ ((\lambda m.\ lookup\ m\ v)\ `\ keys\ (mapping\text{-}of\ P)))\}$
by *blast*
qed
subgoal by *auto*
subgoal by *auto*
done

end
theory *Substitutions*
imports *Poly-Expansions*
begin

1.6 Substitution

The following definitions allow substituting polynomials into the variables of the given polynomial p. They correspond to {@*const subst-pp*} and {@*const poly-subst* in the AFP entry {@*afp Polynomials.Poly-PM*}

definition *poly-subst-monom* :: $(nat \Rightarrow 'a::comm\text{-}semiring\text{-}1\ mpoly) \Rightarrow (nat \Rightarrow_0 nat) \Rightarrow 'a\ mpoly$
where $poly\text{-}subst\text{-}monom\ f\ t = (\prod x.\ (f\ x)\ \hat{}\ (lookup\ t\ x))$

definition *poly-subst* :: $(nat \Rightarrow 'a::comm\text{-}semiring\text{-}1\ mpoly) \Rightarrow 'a\ mpoly \Rightarrow 'a\ mpoly$
where $poly\text{-}subst\ f\ p = Sum\text{-}any\ (\lambda t.\ (Const\ (coeff\ p\ t)) * (poly\text{-}subst\text{-}monom\ f\ t))$

definition *poly-subst-list* **where** $poly\text{-}subst\text{-}list \equiv poly\text{-}subst \circ (!_0)$
abbreviation *insertion-list* **where** $insertion\text{-}list \equiv insertion \circ (!_0)$

lemma *poly-subst-monom-alt*: $poly\text{-}subst\text{-}monom\ f\ t = (\prod x \in keys\ t.\ (f\ x)\ \hat{}\ (lookup\ t\ x))$
unfolding *poly-subst-monom-def*
apply (*rule Prod-any.expand-superset*)
subgoal by *simp*
subgoal using *in-keys-iff* **by** *fastforce*
done

lemma *poly-subst-alt*: $poly\text{-}subst\ f\ p = (\sum t \in keys\ (mapping\text{-}of\ p).\ (Const\ (coeff\ p\ t)) * (poly\text{-}subst\text{-}monom\ f\ t))$
unfolding *poly-subst-def*
apply (*rule Sum-any.expand-superset*)
subgoal by *simp*
subgoal using *Const-zero coeff-keys mult-not-zero* **by** *fastforce*
done

lemma *poly-subst-list-id*:
fixes $p :: 'a::\{comm\text{-}semiring\text{-}1, ring\text{-}no\text{-}zero\text{-}divisors\}\ mpoly$
assumes $k \geq max\text{-}vars\ p$

```
  shows poly-subst-list (map Var [0..<Suc k]) p = p
proof −
  have h0: ⋀x t. x∈keys t ∧ t ∈ keys (mapping-of p) ⟹ x < Suc (max-vars p)
    unfolding max-vars-def vars-def
    apply (subst less-Suc-eq-le)
    by (subst Max.coboundedI, auto)
  hence ⋀x t. x∈keys t ∧ t ∈ keys (mapping-of p) ⟹ x < Suc k
    using assms by fastforce
  hence h1: ⋀x t. x∈keys t ∧ t ∈ keys (mapping-of p) ⟹
              map Var [0..<Suc k] !0 x = (Var x)
    unfolding nth0-def
    apply (subst nth-map, simp)
    by (subst nth-upt, simp-all)

  show ?thesis
    unfolding poly-subst-list-def comp-def
    unfolding poly-subst-alt poly-subst-monom-alt
    apply (subst (9) mpoly-multivariate-expansion')
    apply (rule sum.cong, simp)
    apply (subst mult-left-cancel, metis Const-zero coeff-Const-zero coeff-keys)
    apply (rule prod.mono-neutral-cong-left, simp)
    using h0 less-Suc-eq-le apply force
    apply (simp add: in-keys-iff)
    by (subst h1, simp-all add: assms)
qed


lemma insertion-poly-subst-monom:
  insertion g (poly-subst-monom f t) = (∏ x. (insertion g (f x)) ^ (lookup t x))
  unfolding poly-subst-monom-alt
  apply (subst insertion-prod)
  subgoal by simp
  subgoal
    unfolding insertion-pow
    apply (rule Prod-any.expand-superset[symmetric])
    subgoal by simp
    subgoal using in-keys-iff by fastforce
    done
  done

lemma insertion-poly-subst:
  insertion g (poly-subst f p) = insertion ((insertion g) ∘ f) p
  unfolding poly-subst-alt
  apply (subst (7) poly-eq-sum-monom-alt)
  apply (subst (1 2) insertion-sum)
  subgoal by simp
  subgoal
    unfolding insertion-mult insertion-Const
    unfolding insertion-poly-subst-monom
```

unfolding *insertion-monom*
by *simp*
done

lemma *insertion-nth0*: *insertion f* (*l* !$_0$ *x*) = (*map* (*insertion f*) *l*) !$_0$ *x*
unfolding *nth0-def*
apply (*induction l*)
by *simp* (*metis insertion-zero length-map nth-map when-def*)

lemma *poly-subst-monom-zero* [*simp*]:
poly-subst-monom f 0 = 1
unfolding *poly-subst-monom-alt keys-zero*
by *simp*

lemma *poly-subst-monom-single* [*simp*]:
poly-subst-monom f (*Poly-Mapping.single v 1*) = *f v*
unfolding *poly-subst-monom-alt*
by *simp*

lemma *poly-subst-monom-add*: *poly-subst-monom f* (m_1 + m_2) = *poly-subst-monom f* m_1 ∗ *poly-subst-monom f* m_2
unfolding *poly-subst-monom-def lookup-add power-add*
apply (*rule Prod-any.distrib*)
apply (*metis* (*mono-tags*, *lifting*) *finite-keys finite-nat-set-iff-bounded in-keys-iff mem-Collect-eq power-0*)+
done

lemma *poly-subst-zero* [*simp*]:
poly-subst f 0 = 0
unfolding *poly-subst-def poly-subst-monom-def coeff-def zero-mpoly.rep-eq zero-poly-mapping.rep-eq Const-when when-mult Sum-any-when-equal Const-zero*
by *simp*

lemma *poly-subst-one* [*simp*]:
poly-subst f 1 = 1
unfolding *poly-subst-def poly-subst-monom-def coeff-def one-mpoly.rep-eq one-poly-mapping.rep-eq Const-when when-mult Sum-any-when-equal Const-one*
by *simp*

lemma *poly-subst-Var* [*simp*]:
poly-subst f (*Var v*) = *f v*
unfolding *poly-subst-alt Var.rep-eq keys.rep-eq Var*$_0$*-def lookup-single when-neq-zero*
apply *simp*
apply (*subst* (*2*) *power-one-right*[*symmetric*])
unfolding *One-nat-def*[*symmetric*]
apply (*subst coeff-var-power-eq*)
unfolding *poly-subst-monom-single Const-one*
by *simp*

lemma *poly-subst-Const* [*simp*]:
poly-subst f (*Const c*) = (*Const c*)
unfolding *poly-subst-alt Const.rep-eq keys.rep-eq* $Const_0$*-def lookup-single when-neq-zero*
apply (*cases c* = *0*)
by (*auto simp add*: *Const-zero coeff-Const-zero*)

lemma *poly-subst-numeral*[*simp*]:
poly-subst f (*numeral n*) = (*numeral n*)
unfolding *Const-numeral*[*symmetric*]
by *simp*

lemma *poly-subst-add* [*simp*]:
poly-subst f (*P* + *Q*) = *poly-subst f P* + *poly-subst f Q*
proof −
have *0*: ⋀*P*. {*a*. *Const* (*lookup* (*mapping-of P*) *a*) ∗ *poly-subst-monom f a* ≠ *0*}
⊆ {*a*. *lookup* (*mapping-of P*) *a* ≠ *0*}
by (*auto simp*: *Const-zero*)

have *H*: *finite* {*a*. *Const* (*lookup* (*mapping-of P*) *a*) ∗ *poly-subst-monom f a* ≠ *0*} **for** *P*
using *0*[*of P*] *finite-lookup*
by (*rule finite-subset*)

thus *?thesis*
unfolding *poly-subst-def coeff-add*[*symmetric*] *Const-add*[*symmetric*]
unfolding *coeff-def*
unfolding *Sum-any.distrib*[*OF H H*, *symmetric*]
unfolding *coeff-def*[*symmetric*]
by (*auto simp*: *algebra-simps*)
qed

lemma *poly-subst-uminus* [*simp*]:
poly-subst f (− *P*) = − *poly-subst f P*
by (*metis add-cancel-right-right add-eq-0-iff poly-subst-add*)

lemma *poly-subst-diff* [*simp*]:
poly-subst f ((*P*::('*a*::{*ab-group-add*,*comm-semiring-1*}) *mpoly*) − *Q*) = *poly-subst f P* − *poly-subst f Q*
by (*metis diff-eq-eq poly-subst-add*)

lemma *poly-subst-sum*:
poly-subst f (*sum P A*) = *sum* (*poly-subst f* ∘ *P*) *A*
by (*induct A rule*: *infinite-finite-induct*; *auto*)

lemma *poly-subst-mult* [*simp*]:
poly-subst f (*P* ∗ *Q*) = *poly-subst f P* ∗ *poly-subst f Q*
unfolding *poly-subst-def*
unfolding *coeff-def times-mpoly.rep-eq times-poly-mapping.rep-eq prod-fun-def*
unfolding *coeff-def*[*symmetric*]

proof −
have *0*: *finite* {*m*. *coeff P m* ≠ *0*} **by** (*simp add*: *coeff-def*)
have *1*: *finite* {*m*. *coeff Q m* ≠ *0*} **by** (*simp add*: *coeff-def*)

have *finite* ({m_1. *coeff P* m_1 ≠ *0*} × {m_2. *coeff Q* m_2 ≠ *0*})
using *0 1* **by** *simp*
hence *finite* {(m_1, m_2). *coeff P* m_1 ≠ *0* ∧ *coeff Q* m_2 ≠ *0*}
apply (*rule back-subst*)
apply (*rule arg-cong*[*of* - - *finite*])
apply *auto*
done
hence *2*: *finite* {(m_1, m_2). *Const* (*coeff P* m_1) ∗ *Const* (*coeff Q* m_2) ∗ *poly-subst-monom f* (m_1 + m_2) ≠ *0*}
apply (*rule rev-finite-subset*)
apply (*auto simp add*: *Const-zero*)
done

have *Sum-any* (λ*m*. *Const* (*Sum-any* (λm_1. *coeff P* m_1 ∗ *Sum-any* (λm_2. *coeff Q* m_2 *when m* = m_1 + m_2))) ∗ *poly-subst-monom f m*) =
Sum-any (λ*m*. *Sum-any* (λm_1. (*Const* (*coeff P* m_1)) ∗ *Sum-any* (λm_2. *Const* ((*coeff Q* m_2) *when m* = m_1 + m_2))) ∗ *poly-subst-monom f m*)
unfolding *Const-sum-Any comp-def Const-mult*[*symmetric*] **..**
also have ... = *Sum-any* (λ*m*. *Sum-any* (λm_1. *Sum-any* (λm_2. (*Const* (*coeff P* m_1)) ∗ *Const* ((*coeff Q* m_2) *when m* = m_1 + m_2))) ∗ *poly-subst-monom f m*)
apply (*subst Sum-any-right-distrib*)
subgoal by (*rule rev-finite-subset*[*OF 1*]; *auto simp add*: *Const-zero*)
subgoal ..
done
also have ... = *Sum-any* (λ*m*. *Sum-any* (λm_1. *Sum-any* (λm_2. (*Const* (*coeff P* m_1)) ∗ *Const* (*coeff Q* m_2) *when m* = m_1 + m_2)) ∗ *poly-subst-monom f m*)
unfolding *Const-when mult-when* **..**
also have ... = *Sum-any* (λ*m*. *Sum-any* (λm_1. *Sum-any* (λm_2. (*Const* (*coeff P* m_1)) ∗ *Const* (*coeff Q* m_2) ∗ *poly-subst-monom f* (m_1 + m_2) *when m* = m_1 + m_2)))
apply (*subst Sum-any-left-distrib*)
subgoal by (*rule finite-subset*[*OF - 0*]; *auto simp add*: *Const-zero*)
subgoal
apply (*subst Sum-any-left-distrib*)
subgoal by (*rule finite-subset*[*OF - 1*]; *auto simp add*: *Const-zero*)
subgoal
apply (*rule Sum-any.cong*)
apply (*rule Sum-any.cong*)
apply (*rule Sum-any.cong*)
unfolding *when-mult*
apply (*rule when-cong*[*OF refl*])
apply *simp*
done
done
done

also have ... = *Sum-any* (λm_1. *Sum-any* (λm_2. (*Const* (*coeff P* m_1)) $*$ (*Const* (*coeff Q* m_2)) $*$ *poly-subst-monom f* (m_1 + m_2)))
by (*rule Sum-any-rev-image-add*[*OF 2*])
also have ... = *Sum-any* (λm_1. *Sum-any* (λm_2. (*Const* (*coeff P* m_1) $*$ *poly-subst-monom f* m_1) $*$ (*Const* (*coeff Q* m_2) $*$ *poly-subst-monom f* m_2)))
unfolding *poly-subst-monom-add* **by** (*simp add*: *algebra-simps*)
also have ... = *Sum-any* (λm_1. *Const* (*coeff P* m_1) $*$ *poly-subst-monom f* m_1) $*$ *Sum-any* (λm_2. *Const* (*coeff Q* m_2) $*$ *poly-subst-monom f* m_2)
apply (*rule Sum-any-product*[*symmetric*])
subgoal by (*rule finite-subset*[*OF - 0*]; *auto simp add*: *Const-zero*)
subgoal by (*rule finite-subset*[*OF - 1*]; *auto simp add*: *Const-zero*)
done
finally show *Sum-any* (λm. *Const* (*Sum-any* (λm_1. *coeff P* m_1 $*$ *Sum-any* (λm_2. *coeff Q* m_2 *when* $m = m_1 + m_2$))) $*$ *poly-subst-monom f m*) =
Sum-any (λm_1. *Const* (*coeff P* m_1) $*$ *poly-subst-monom f* m_1) $*$ *Sum-any* (λm_2. *Const* (*coeff Q* m_2) $*$ *poly-subst-monom f* m_2)
.
qed

lemma *poly-subst-prod*:
poly-subst f (*prod P A*) = *prod* (*poly-subst f* $\circ$ *P*) *A*
by (*induct A rule*: *infinite-finite-induct*; *auto*)

lemma *poly-subst-monom-id*: *poly-subst-monom* (*Var*) *t* = *monom t 1*
proof −
have *mapping-of* (*poly-subst-monom* (*Var*) *t*) = *Poly-Mapping.single t 1*
unfolding *poly-subst-monom-alt*
apply (*subst monom.rep-eq*[*symmetric*])
apply (*subst monom-expansion*)
by (*subst mapping-of-inject*, *simp add*: *Const-one*)

thus *?thesis*
unfolding *monom.abs-eq* **by** (*smt* (*verit*) *mapping-of-inverse*)
qed

lemma *poly-subst-id*: *poly-subst* (*Var*) *p* = *p*
apply (*induct p rule*: *mpoly-induct*, *simp-all*)
apply (*simp add*: *poly-subst-def poly-subst-monom-id*)
by (*smt* (*verit*, *ccfv-SIG*) *Const.abs-eq Const*$_0$*-def Sum-any.cong monom.abs-eq mult.right-neutral*
poly-eq-sum-monom smult-conv-mult smult-monom)

Vars of substitutions

lemma *vars-poly-subst-monom*: *vars* (*poly-subst-monom f t*) $\subseteq \bigcup$ (*vars* ‘ (*f* ‘ *keys t*))
unfolding *poly-subst-monom-alt*
apply (*rule subset-trans*[*OF vars-prod*])
using *vars-pow* **by** *auto*

lemma *vars-poly-subst-monom'*: *vars* (*poly-subst-monom* ((!$_0$) *ls*) *t*) $\subseteq \bigcup$ (*vars* ‘ *set ls*)
apply (*rule subset-trans*[*OF vars-poly-subst-monom*])
unfolding *nth0-def when-def keys.rep-eq*
using *nth-mem Const-zero MPoly-Type.degree-zero vars-non-zero-degree* **by** *force*

lemma *vars-poly-subst-list*: *vars* (*poly-subst-list ls p*) $\subseteq \bigcup$ (*vars* ‘ *set ls*)
unfolding *poly-subst-alt comp-def poly-subst-list-def*
apply (*rule subset-trans*[*OF vars-setsum*], *simp-all*)
using *vars-poly-subst-monom' vars-mult* **by** *fastforce*

lemma *vars-poly-subst-monom-bounded*:
$\forall\, v{\in}$(*keys t*). *v* $\leq$ *bound* $\Longrightarrow$ *vars* (*poly-subst-monom* ((!$_0$) *ls*) *t*) $\subseteq \bigcup$ (*vars* ‘ *set* (*take* (*Suc bound*) *ls*))
unfolding *poly-subst-monom-alt*
apply (*rule subset-trans*[*OF vars-prod*], *auto*)
apply (*drule set-mp*[*OF vars-pow*])
subgoal for *m v*
proof (*rule bexI*[*of - ls* !$_0$ *v*], *assumption*)
assume $\forall\, v{\in}$(*keys t*). *v* $\leq$ *bound*
moreover assume *v* $\in$ *keys t*
ultimately have *v* $\leq$ *bound* **by** *auto*
moreover assume *m* $\in$ *vars* (*ls* !$_0$ *v*)
ultimately show *ls* !$_0$ *v* $\in$ *set* (*take* (*Suc bound*) *ls*)
apply (*cases v* < *length ls*, *simp-all add*: *nth0-def*)
unfolding *in-set-conv-nth* **by** (*intro exI*[*of - v*], *auto*)
qed done

lemma *aux0*: *max-vars p* $\leq$ *bound* $\Longrightarrow$ *m* $\in$ *keys* (*mapping-of p*) $\Longrightarrow$ $\forall\, v{\in}$(*keys m*). *v* $\leq$ *bound*
unfolding *max-vars-def vars-def* **by** *auto*

lemma *vars-poly-subst-list-bounded*:
assumes *max-vars p* $\leq$ *bound*
shows *vars* (*poly-subst-list ls p*) $\subseteq \bigcup$ (*vars* ‘ *set* (*take* (*Suc bound*) *ls*))
proof −
from *assms* **have** *a*: *t*$\in$*keys* (*mapping-of p*) $\Longrightarrow$ $\forall\, v{\in}$(*keys t*). *v* $\leq$ *bound* **for** *t*
by (*simp add*: *aux0*)
hence *b*: *t*$\in$*keys* (*mapping-of p*) $\Longrightarrow$ *vars* (*poly-subst-monom* ((!$_0$) *ls*) *t*)
$\subseteq \bigcup$ (*vars* ‘ *set* (*take* (*Suc bound*) *ls*)) **for** *t*
by (*rule vars-poly-subst-monom-bounded*, *simp*)
show *?thesis*
apply (*simp add*: *poly-subst-list-def poly-subst-alt*)
apply (*rule subset-trans*[*OF vars-setsum*], *auto simp*: *vars-mult*)
subgoal for *x t*
using *vars-mult b*[*of t*] **apply** *simp*
by (*smt* (*verit*, *del-insts*) *UN-iff subset-iff sup-bot-left vars-Const vars-mult*)
done
qed

```
lemma vars-poly-subst:
  vars (poly-subst f p) ⊆ (⋃ t ∈ vars p. vars (f t))
proof −
  {
    fix x y
    assume x ∈ keys (mapping-of p)
      and y ∈ (λxa. f xa ^ lookup x xa) ‘ keys x
    from this obtain z where z ∈ keys x and y-def: y = f z ^ lookup x z
      by auto

    from this have z-in: z ∈ vars p
      unfolding vars-def image-def
      apply (intro UnionI[of keys x {y. ∃ x∈keys (mapping-of p). y = keys x} z])
      unfolding mem-Collect-eq
       apply (rule bexI[of - x])
      using ‹x ∈ keys (mapping-of p)›
      by auto

    have 0: vars (f z) ∈ {y. ∃ x∈⋃ {y. ∃ x∈keys (mapping-of p). y = keys x}.
                        y = ⋃ {y. ∃ x∈keys (mapping-of (f x)). y = keys x}}
      unfolding mem-Collect-eq
      unfolding image-def[symmetric] vars-def[symmetric] y-def
      by (rule bexI[of - z, OF - z-in]; simp)

    {
      fix w
      assume w ∈ {ya. ∃ x∈keys (mapping-of y). ya = keys x}
      from this obtain v where v ∈ keys (mapping-of y) and w-def: w = keys v
        by auto
      hence keys v ⊆ vars y
        unfolding vars-def
        by auto

      have w ⊆ vars (f z)
        unfolding w-def
        apply (rule subset-trans[OF ‹keys v ⊆ vars y›])
        unfolding y-def
        by (simp add: vars-pow)

      from this and 0
      have ∃ y. y ∈ {y. ∃ x∈⋃ {y. ∃ x∈keys (mapping-of p). y = keys x}.
                        y = ⋃ {y. ∃ x∈keys (mapping-of (f x)). y = keys x}} ∧
              w ⊆ y
        by auto
    }
    note 0 = this
```

```
    hence vars y ⊆ (⋃t∈vars p. vars (f t))
      unfolding vars-def UN-simps image-def
      apply (intro Union-subsetI)
      by assumption
  }
  note 0 = this
  {
    fix x
    assume Hx: x ∈ keys (mapping-of p)
   have vars (Const (coeff p x) * poly-subst-monom f x) ⊆ vars (poly-subst-monom
f x)
      apply (rule subset-trans[OF vars-mult Un-least])
      by simp-all

    also have ... ⊆ (⋃t∈vars p. vars (f t))
      unfolding poly-subst-monom-alt
      apply (rule subset-trans[OF vars-prod])
      apply (rule UN-least)
      using 0[OF Hx] by assumption

    finally have vars (Const (coeff p x) * poly-subst-monom f x) ⊆ (⋃t∈vars p.
vars (f t)) .
  }
  note 0 = this
  show ?thesis
    unfolding poly-subst-alt
    apply (rule subset-trans[OF vars-setsum[OF finite-keys]])
    apply (rule UN-least)
    using 0 by assumption
qed

lemma max-vars-poly-subst-list-general:
  shows max-vars (poly-subst-list ls p) ≤ Max (max-vars ‘ set ls)
  unfolding max-vars-def
  apply (rule Max.boundedI, auto simp add: vars-finite)
  subgoal for a
  proof −
    assume a ∈ vars (poly-subst-list ls p)
    hence a ∈ ⋃ (vars ‘ set ls) using vars-poly-subst-list by force
    thus a ≤ (MAX P∈set ls. Max (insert 0 (vars P)))
      apply simp
        by (smt (verit, best) List.finite-set Max-ge dual-order.trans finite-imageI
finite-insert
            imageI insert-absorb insert-subset subset-insertI vars-finite)
  qed done

lemma max-vars-poly-subst-list-bounded:
  max-vars p ≤ bound ⟹ max-vars (poly-subst-list ls p) ≤ Max (max-vars ‘ set
(take (Suc bound) ls))
```

proof −
assume *a*: *max-vars p* ≤ *bound*

{
fix *a*
assume *a* ∈ *vars* (*poly-subst-list ls p*)
hence *a* ∈ ⋃ (*vars* ‘ *set* (*take* (*Suc bound*) *ls*))
using *vars-poly-subst-list-bounded*[*OF a*] **by** *auto*
hence *a* ≤ (*MAX P*∈*set* (*take* (*Suc bound*) *ls*). *Max* (*insert 0* (*vars P*)))
apply *simp*
by (*smt* (*verit*, *best*) *List.finite-set Max-ge dual-order.trans finite-imageI finite-insert*
imageI insert-absorb insert-subset subset-insertI vars-finite)
} note *b* = *this*

show *?thesis*
unfolding *max-vars-def*
by (*rule Max.boundedI*, *auto simp add*: *vars-finite b*)
qed

lemma *max-vars-id*:
fixes *p* :: *'a*::{*comm-semiring-1*,*ring-no-zero-divisors*} *mpoly*
shows *max-vars* (*poly-subst-list* (*map Var* [*0*..<*Suc k*]) *p*) ≤ *k*
by (*rule le-trans*[*OF max-vars-poly-subst-list-general*], *auto simp*: *max-vars-def*)

Degrees of substitutions

lemma *degree-poly-subst-monom*:
fixes *f*
assumes *finite* {*k*. *f k* ≠ *0*}
defines *degree-monom* ≡ (λ*m t*. (*lookup m*) *t*)
shows *degree* (*poly-subst-monom f m*) *v*
≤ (∑ *t* | *v*∈*vars* (*f t*). *degree-monom m t* ∗ *degree* (*f t*) *v*)
proof −
have *a0*: *v* ∉ *vars* (*f k*) ⟹ *degree* (*f k* ^ *lookup m k*) *v* = *0* **for** *k*
unfolding *le-0-eq*[*of degree - v*, *symmetric*]
apply (*rule le-trans*[*OF degree-pow*])
by (*simp add*: *in-vars-non-zero-degree*)

have *a1*: (∑ *t* | *v* ∈ *vars* (*f t*). *lookup m t* ∗ *degree* (*f t*) *v*)
= (∑ *t* | *t* ∈ *keys m* ∧ *v* ∈ *vars* (*f t*). *lookup m t* ∗ *degree* (*f t*) *v*)
unfolding *keys.rep-eq* **apply** (*rule sum.mono-neutral-right*, *auto*)
using *assms*
by (*smt* (*verit*, *del-insts*) *MPoly-Type.degree-zero finite-nat-set-iff-bounded*
in-vars-non-zero-degree mem-Collect-eq)

show *?thesis*
unfolding *poly-subst-monom-alt degree-monom-def*
apply (*subst a1*)
apply (*rule le-trans*[*OF degree-prod*[*OF finite-keys*]])

```
    apply (subst sum.setdiff-irrelevant[OF finite-keys,
                                        of λk. degree (f k ^ lookup m k) v, symmetric])
    apply (rule le-trans[OF sum-mono[OF degree-pow]])
    apply (rule sum-mono2, auto)
    by (rule ccontr, auto simp: a0)
qed

lemma degree-poly-subst:
  fixes p :: 'a::comm-ring-1 mpoly
  fixes f :: nat ⇒ 'a mpoly
  assumes finite {k. f k ≠ 0}
  shows degree (poly-subst f p) v ≤ (∑ t | v ∈ vars (f t). degree p t * degree (f t)
v)
proof −
  have a ∈ keys (mapping-of p) ⟹ degree (poly-subst-monom f a) v
          ≤ (∑ t | v ∈ vars (f t). degree p t * degree (f t) v) for a
    apply (rule le-trans[OF degree-poly-subst-monom[of f, OF assms]])
    apply (rule sum-mono, simp)
    unfolding degree.rep-eq by auto

  hence a0: Max (insert 0 ((λi. degree (poly-subst-monom f i) v) ' keys (mapping-of
p)))
       ≤ (∑ t | v ∈ vars (f t). degree p t * degree (f t) v)
    by auto

  show ?thesis
    unfolding poly-subst-alt
    apply (rule le-trans[OF - a0])
    apply (rule le-trans[OF degree-sum[OF finite-keys, of λt. Const (coeff p t) *
poly-subst-monom f t
                                       mapping-of p v]])
    using degree-mult degree-Const
    by simp (smt (verit, best) Max.coboundedI add-cancel-right-right degree-Const
degree-mult
                  finite-imageI finite-insert finite-keys imageI insert-iff le-trans
mult.commute)
qed

lemma degree-poly-subst':
  fixes p :: 'a::comm-ring-1 mpoly
  fixes f :: nat ⇒ 'a mpoly
  assumes finite {k. f k ≠ 0}
  shows degree (poly-subst f p) v ≤ (∑ t∈vars p. degree p t * degree (f t) v)
  apply (rule le-trans[OF degree-poly-subst], simp add: assms)
  apply (subst sum.mono-neutral-right[of {t. v ∈ vars (f t)} vars p ∩ {t. v ∈ vars
(f t)}])
  using in-vars-non-zero-degree assms
 subgoal by (smt (verit) Collect-mono emptyE rev-finite-subset vars-zero vars-finite)
  subgoal by auto
```

subgoal using *in-vars-non-zero-degree* **by** *auto*
subgoal by (*simp add*: *sum-mono2 vars-finite*)
done

lemma *degree-poly-subst-list*:
fixes p :: '*a*::*comm-ring-1 mpoly*
shows *degree* (*poly-subst-list ls p*) $v \leq (\sum t \mid v \in vars\ (ls\ !_0\ t).\ degree\ p\ t * degree\ (ls\ !_0\ t)\ v)$
unfolding *poly-subst-list-def*
using *degree-poly-subst*[*OF nth0-finite*] **by** *auto*

lemma *degree-poly-subst-list'*:
fixes p :: '*a*::*comm-ring-1 mpoly*
shows *degree* (*poly-subst-list ls p*) $v \leq (\sum t \leq length\ ls.\ degree\ p\ t * degree\ (ls\ !_0\ t)\ v)$
apply (*rule le-trans*[*OF degree-poly-subst-list*])
by (*rule sum-mono2*, *auto simp*: *nth0-def when-def in-vars-non-zero-degree*)

Total degree of substitions

lemma *deg-imp-tot-deg-zero*: $(\forall v \in vars\ P.\ degree\ P\ v = 0) \Longrightarrow total\text{-}degree\ P = 0$
by (*metis bot-nat-0.extremum-uniqueI sum.neutral total-degree-bound*)

lemma *total-degree-poly-subst-monom*:
fixes f
defines *degree-monom* $\equiv (\lambda m\ t.\ (lookup\ m)\ t)$
shows *total-degree* (*poly-subst-monom f m*)
$\leq (\sum t{\in}keys\ m.\ degree\text{-}monom\ m\ t * total\text{-}degree\ (f\ t))$
unfolding *poly-subst-monom-alt degree-monom-def*
apply (*rule le-trans*[*OF total-degree-prod*[*OF finite-keys*]])
apply (*rule le-trans*[*OF sum-mono*[*OF total-degree-pow*]])
by (*rule sum-mono2*) *auto*

lemma *total-degree-poly-subst*:
shows *total-degree* (*poly-subst f p*) $\leq (\sum t{\in}vars\ p.\ degree\ p\ t * total\text{-}degree\ (f\ t))$
proof −
have $a \in keys\ (mapping\text{-}of\ p) \Longrightarrow total\text{-}degree\ (poly\text{-}subst\text{-}monom\ f\ a)$
$\leq (\sum t{\in}vars\ p.\ degree\ p\ t * total\text{-}degree\ (f\ t))$ **for** a
apply (*rule le-trans*[*OF total-degree-poly-subst-monom*[*of f*]])
apply (*rule sum-le-included*[*of - - - λx. x*], *auto simp add*: *vars-finite vars-def*)
unfolding *degree.rep-eq* **by** *auto*

hence *a0*: *Max* (*insert 0* ((λ*i*. *total-degree* (*poly-subst-monom f i*)) ‘ *keys* (*mapping-of p*)))
$\leq (\sum t{\in}vars\ p.\ degree\ p\ t * total\text{-}degree\ (f\ t))$
by *auto*

show *?thesis*
unfolding *poly-subst-alt*

apply (*rule le-trans*[*OF* - *a0*])
apply (*rule le-trans*[*OF total-degree-sum*[*OF finite-keys*]])
using *total-degree-mult total-degree-Const*
apply *simp*
by (*smt* (*verit*, *del-insts*) *Max-ge add-0 dual-order.trans finite-imageI finite-insert finite-keys image-eqI insert-iff total-degree-Const total-degree-mult*)
qed

lemma *total-degree-poly-subst-list*:
fixes p :: *'a*::*comm-ring-1 mpoly*
shows *total-degree* (*poly-subst-list ls p*) $\leq$ ($\sum$ *t*$\in$*vars p*. *degree p t* $*$ *total-degree* (*ls* !$_0$ *t*))
unfolding *poly-subst-list-def*
using *total-degree-poly-subst* **by** *auto*

lemma *total-degree-poly-subst-list'*:
fixes p :: *'a*::*comm-ring-1 mpoly*
assumes *max-vars p* $\leq$ *length ls*
shows *total-degree* (*poly-subst-list ls p*) $\leq$ ($\sum$ *t*$\leq$*length ls*. *degree p t* $*$ *total-degree* (*ls* !$_0$ *t*))
apply (*rule le-trans*[*OF total-degree-poly-subst-list*])
apply (*cases vars p* = {}, *simp-all*)
apply (*rule sum-mono2*, *simp-all*)
using *assms*[*unfolded max-vars-def*]
by (*simp add*: *subset-iff vars-finite*)

lemma *total-degree-poly-subst-list''*:
fixes p :: *'a*::*comm-ring-1 mpoly*
assumes $\forall$ *i* $\leq$ *length ls*. *card* (*vars* (*ls* ! *i*)) $\leq$ *1*
shows *total-degree* (*poly-subst-list ls p*) $\leq$
length ls $*$ ($\sum$ *t* $\leq$ *length ls*. *degree p t* $*$ *total-degree* (*ls* !$_0$ *t*))
proof −
from *assms* **have** *asm*: $\bigwedge$*i*. *i* $\in$ *set ls* $\Longrightarrow$ *card* (*vars i*) $\leq$ *1*
by (*metis in-set-conv-nth less-or-eq-imp-le*)

have *card0*: *card* (*vars* (*poly-subst-list ls p*)) $\leq$ *length ls*
apply (*rule le-trans*[*OF card-mono*[*OF* - *vars-poly-subst-list*]], *simp add*: *vars-finite*)
apply (*rule le-trans*[*OF card-UN-le*], *simp*)
apply (*rule le-trans*[*OF sum-bounded-above*[*OF asm*]])
by (*auto simp add*: *card-length*)

have *sum* (*degree* (*poly-subst-list ls p*)) (*vars* (*poly-subst-list ls p*))
$\leq$ *sum* (λ*v*. $\sum$ *t* $\leq$ *length ls*. *degree p t* $*$ *degree* (*ls* !$_0$ *t*) *v*) (*vars* (*poly-subst-list ls p*))
apply (*rule sum-mono*)
using *degree-poly-subst-list'* **by** *auto*
also have ... = ($\sum$ *t* $\leq$ *length ls*. *degree p t* $*$ *sum* (*degree* (*ls* !$_0$ *t*)) (*vars* (*poly-subst-list ls p*)))

apply (*subst sum.swap*)
by (*subst sum-distrib-left*, *simp*)
also have ... ≤ (∑ *t* ≤ *length ls*. *degree p t* ∗ *length ls* ∗ *total-degree* (*ls* !$_0$ *t*))
apply (*rule sum-mono*, *auto*)
apply (*rule le-trans*[*OF sum-bounded-above*, *of* - - *total-degree* (*ls* !$_0$ -)])
using *degree-total-degree-bound vars-poly-subst-list card0* **by** *auto*

finally show *?thesis*
apply *simp*
apply (*rule le-trans*[*OF total-degree-bound*])
apply (*subst sum-distrib-left*)
by (*smt* (*verit*) *ab-semigroup-mult-class.mult-ac*(*1*) *mult.commute sum.cong*)
qed

end
theory *Type-Casting*
imports *Substitutions*
begin

1.7 Type casting for polynomials

named-theorems *of-int-mpoly-simps Lemmas about of-int-mpoly*

definition *of-int-mpoly* :: *int mpoly* ⇒ *'a*::*ring-1 mpoly* **where**
of-int-mpoly P = *MPoly* (*Abs-poly-mapping* (*of-int* ∘ *lookup* (*mapping-of P*)))

lemma *of-int-mpoly-coeff* [*simp*, *of-int-mpoly-simps*]:
coeff (*of-int-mpoly P*) *a* = *of-int* (*coeff P a*)
unfolding *of-int-mpoly-def comp-def coeff-def*
apply (*subst MPoly-inverse*)
subgoal by *blast*
subgoal
apply (*subst Abs-poly-mapping-inverse*)
subgoal by (*simp*; *metis* (*mono-tags*, *lifting*) *finite-lookup finite-subset mem-Collect-eq of-int-0 subsetI*)
subgoal ..
done
done

lemma *of-int-mpoly-zero* [*of-int-mpoly-simps*]:
of-int-mpoly 0 = *0*
unfolding *of-int-mpoly-def zero-mpoly.rep-eq lookup-zero comp-apply of-int-0*
unfolding *zero-mpoly-def*
by *auto*

lemma *eq-onp-intF*:
fixes *F* :: *int mpoly*
shows *eq-onp* (*λf*. *finite* {*x*. *f x* ≠ *0*}) (*λx*. *of-int* (*lookup* (*mapping-of F*) *x*))
(*λx*. *of-int* (*lookup* (*mapping-of F*) *x*))

proof −
have *incl*:{*x*. *of-int* (*lookup* (*mapping-of* $\mathcal{F}$) *x*) ≠ *0*} ⊆ {*x*. (*lookup* (*mapping-of* $\mathcal{F}$) *x*) ≠ *0*} **by** *auto*
have *finite* {*x*. (*lookup* (*mapping-of* $\mathcal{F}$) *x*) ≠ *0*} **by** *auto*
then have *fin*:*finite* {*x*. *of-int* (*lookup* (*mapping-of* $\mathcal{F}$) *x*) ≠ *0*}
using *incl Finite-Set.rev-finite-subset*[*of* {*x*. (*lookup* (*mapping-of* $\mathcal{F}$) *x*) ≠ *0*} {*x*. *of-int* (*lookup* (*mapping-of* $\mathcal{F}$) *x*) ≠ *0*}] **by** *auto*
thus *?thesis* **unfolding** *eq-onp-def* **by** *auto*
qed

lemma *of-int-mpoly-one* [*of-int-mpoly-simps*]:
of-int-mpoly 1 = 1
unfolding *of-int-mpoly-def one-mpoly.rep-eq lookup-one comp-apply*
unfolding *one-mpoly-def one-poly-mapping-def*
apply (*simp add*: *MPoly-inject*)
apply (*subst Abs-poly-mapping-inject*)
by (*auto simp*: *when-def*)

lemma *of-int-mpoly-Var* [*of-int-mpoly-simps*]:
of-int-mpoly (*Var n*) = *Var n*
unfolding *of-int-mpoly-def Var.rep-eq* Var_0*-def comp-def*
unfolding *Var.abs-eq* Var_0*-def*
apply (*simp add*: *MPoly-inject*)
apply (*subst* (*3*) *Poly-Mapping.single.abs-eq*)
apply (*subst Abs-poly-mapping-inject*, *simp-all*)
by (*auto simp add*: *when-def*)

lemma *of-int-mpoly-Const* [*of-int-mpoly-simps*]:
of-int-mpoly (*Const c*) = *Const* (*of-int c*)
unfolding *of-int-mpoly-def Const.rep-eq* $Const_0$*-def Poly-Mapping.single.abs-eq comp-def*
apply (*simp add*: *Const-def MPoly-inject*)
unfolding $Const_0$*-def Poly-Mapping.single.abs-eq*
apply (*subst Abs-poly-mapping-inject*)
apply *simp-all*
subgoal by (*smt* (*verit*, *ccfv-SIG*) *when*(*2*) *finite.intros*(*1*) *finite-insert finite-subset insert-iff mem-Collect-eq of-int-0 subsetI*)
subgoal by (*metis when*(*1*) *when*(*2*) *of-int-0*)
done

lemma *of-int-mpoly-numeral* [*of-int-mpoly-simps*]:
of-int-mpoly (*numeral n*) = *numeral n*
unfolding *Const-numeral*[*symmetric*]
by (*simp add*: *of-int-mpoly-simps*)

lemma *of-int-mpoly-add* [*of-int-mpoly-simps*]:
fixes $\mathcal{F}$ $\mathcal{G}$ :: *int mpoly*
shows *of-int-mpoly* ($\mathcal{F}$ + $\mathcal{G}$) = *of-int-mpoly* $\mathcal{F}$ + *of-int-mpoly* $\mathcal{G}$
proof −

have *Abs-distrib*:(*Abs-poly-mapping* (λ*x*. *of-int* (*lookup* (*mapping-of* $\mathcal{F}$) *x*)) +
Abs-poly-mapping (λ*x*. *of-int* (*lookup* (*mapping-of* $\mathcal{G}$) *x*)))
= (*Abs-poly-mapping*
(λ*x*. *of-int* (*lookup* (*mapping-of* $\mathcal{F}$) *x*) + *of-int* (*lookup* (*mapping-of* $\mathcal{G}$) *x*)))
using *eq-onp-intF*[*of* $\mathcal{F}$] *eq-onp-intF*[*of* $\mathcal{G}$]
Poly-Mapping.plus-poly-mapping.abs-eq[*of* λ*x*. *of-int* (*lookup* (*mapping-of* $\mathcal{F}$) *x*) λ*x*. *of-int* (*lookup* (*mapping-of* $\mathcal{G}$) *x*)]
by *auto*

have *LHS*:*of-int-mpoly* ($\mathcal{F}$ + $\mathcal{G}$) = *MPoly*
(*Abs-poly-mapping*
(λ*x*. *of-int* (*lookup* (*mapping-of* $\mathcal{F}$) *x*) + *of-int* (*lookup* (*mapping-of* $\mathcal{G}$) *x*)))
unfolding *of-int-mpoly-def*
unfolding *MPoly-Type.plus-mpoly.rep-eq MPoly-Type.plus-mpoly.abs-eq*
unfolding *Poly-Mapping.plus-poly-mapping.rep-eq*
unfolding *comp-apply*
unfolding *Int.ring-1-class.of-int-add*
by *auto*

have *of-int-mpoly* $\mathcal{F}$ + *of-int-mpoly* $\mathcal{G}$ = *MPoly*
(*Abs-poly-mapping* (λ*x*. *of-int* (*lookup* (*mapping-of* $\mathcal{F}$) *x*)) +
Abs-poly-mapping (λ*x*. *of-int* (*lookup* (*mapping-of* $\mathcal{G}$) *x*)))
unfolding *of-int-mpoly-def*
unfolding *MPoly-Type.plus-mpoly.rep-eq MPoly-Type.plus-mpoly.abs-eq*
unfolding *comp-apply*
by *auto*

also have ... = *MPoly* (*Abs-poly-mapping*
(λ*x*. *of-int* (*lookup* (*mapping-of* $\mathcal{F}$) *x*) + *of-int* (*lookup* (*mapping-of* $\mathcal{G}$) *x*)))
using *Abs-distrib MPoly-inject* **by** *auto*
finally have *RHS*:*of-int-mpoly* $\mathcal{F}$ + *of-int-mpoly* $\mathcal{G}$ = *MPoly* (*Abs-poly-mapping*
(λ*x*. *of-int* (*lookup* (*mapping-of* $\mathcal{F}$) *x*) + *of-int* (*lookup* (*mapping-of* $\mathcal{G}$) *x*)))
by *auto*
show *?thesis* **unfolding** *LHS RHS* **by** *auto*
qed

lemma *of-int-mpoly-neg* [*of-int-mpoly-simps*]:
fixes $\mathcal{G}$ :: *int mpoly*
shows *of-int-mpoly* (− $\mathcal{G}$) = − *of-int-mpoly* $\mathcal{G}$
proof −
have *Abs-minus*:− *Abs-poly-mapping* (λ*x*. *of-int* ((*lookup* (*mapping-of* $\mathcal{G}$)) *x*))
= *Abs-poly-mapping* (− (λ*x*. *of-int* ((*lookup* (*mapping-of* $\mathcal{G}$)) *x*)))
using *eq-onp-intF Poly-Mapping.uminus-poly-mapping.abs-eq*[*of* λ*x*. *of-int* ((*lookup* (*mapping-of* $\mathcal{G}$)) *x*)]
by *auto*
have *of-int-mpoly* (− $\mathcal{G}$) = *MPoly* (*Abs-poly-mapping* (λ*x*. *of-int* ((− *lookup* (*mapping-of* $\mathcal{G}$)) *x*)))
unfolding *of-int-mpoly-def MPoly-Type.uminus-mpoly.rep-eq*
unfolding *Poly-Mapping.uminus-poly-mapping.rep-eq*

unfolding *comp-apply* **by** *auto*
also have ... = *MPoly* (*Abs-poly-mapping* (λ*x*. − *of-int* ((*lookup* (*mapping-of* G)) *x*)))
unfolding *Int.ring-1-class.of-int-minus*[*symmetric*] **by** *auto*
also have ... = *MPoly* (*Abs-poly-mapping* (− (λ*x*. *of-int* ((*lookup* (*mapping-of* G)) *x*))))
unfolding *uminus-apply* **by** *auto*
also have ... = *MPoly* (− *Abs-poly-mapping* (λ*x*. *of-int* ((*lookup* (*mapping-of* G)) *x*)))
unfolding *Abs-minus MPoly-inject* **by** *auto*
also have ... = − *MPoly* (*Abs-poly-mapping* (λ*x*. *of-int* ((*lookup* (*mapping-of* G)) *x*)))
unfolding *uminus-mpoly.abs-eq* **by** *auto*
also have ... = − *of-int-mpoly* G **unfolding** *of-int-mpoly-def comp-apply* **by** *auto*
finally show *?thesis* **by** *auto*
qed

lemma *of-int-mpoly-diff* [*of-int-mpoly-simps*]:
fixes F G :: *int mpoly*
shows *of-int-mpoly* (F − G) = *of-int-mpoly* F − *of-int-mpoly* G
unfolding *group-add-class.diff-conv-add-uminus*
unfolding *of-int-mpoly-add of-int-mpoly-neg*
by *auto*

lemma *of-int-mpoly-mult* [*of-int-mpoly-simps*]:
fixes F G :: *int mpoly*
shows (*of-int-mpoly* (F ∗ G) :: ('*a*::*ring-1*) *mpoly*) = *of-int-mpoly* F ∗ *of-int-mpoly* G
proof −
have *Abs-distrib*:(*Abs-poly-mapping* (λ*x*. *of-int* (*lookup* (*mapping-of* F) *x*)) ∗
Abs-poly-mapping (λ*x*. *of-int* (*lookup* (*mapping-of* G) *x*)))
= (*Abs-poly-mapping*
(*prod-fun* (λ*x*. *of-int* (*lookup* (*mapping-of* F) *x*)) (λ*x*. *of-int* (*lookup* (*mapping-of* G) *x*))))
using *eq-onp-intF*[*of* F] *eq-onp-intF*[*of* G]
Poly-Mapping.times-poly-mapping.abs-eq[*of* λ*x*. *of-int* (*lookup* (*mapping-of* F) *x*) λ*x*. *of-int* (*lookup* (*mapping-of* G) *x*)]
by *auto*

have *of-int-prod-fun*: *of-int* (*prod-fun* (*lookup* (*mapping-of* F)) (*lookup* (*mapping-of* G)) *x*)
= *prod-fun* (λ*x*. *of-int* (*lookup* (*mapping-of* F) *x*)) (λ*x*. *of-int* (*lookup* (*mapping-of* G) *x*)) *x* **for** *x* :: *nat* ⇒$_0$ *nat*
unfolding *prod-fun-def*
unfolding *of-int-Sum-any*
apply (*subst of-int-Sum-any*, *simp*)
unfolding *comp-def of-int-sum of-int-mult*
apply (*subst of-int-Sum-any*, *simp*)
apply (*rule Sum-any.cong*)

apply (*simp add*: *when-def*)
by (*metis* (*mono-tags*, *opaque-lifting*) *of-int-0*)

have *of-int-mpoly* ($\mathcal{F} * \mathcal{G}$) = *MPoly* (*Abs-poly-mapping*
(λx. *of-int* (*prod-fun* (*lookup* (*mapping-of* $\mathcal{F}$)) (*lookup* (*mapping-of* $\mathcal{G}$)) x)))
unfolding *of-int-mpoly-def*
unfolding *MPoly-Type.times-mpoly.rep-eq MPoly-Type.times-mpoly.abs-eq*
unfolding *Poly-Mapping.times-poly-mapping.rep-eq*
unfolding *comp-apply*
by *auto*
also have ... = *MPoly* (*Abs-poly-mapping*
(*prod-fun* (λx. *of-int* (*lookup* (*mapping-of* $\mathcal{F}$) x)) (λx. *of-int* (*lookup* (*mapping-of* $\mathcal{G}$) x))))
using *of-int-prod-fun*
apply (*simp add*: *MPoly-inject*)
apply (*subst Abs-poly-mapping-inject*)
subgoal
by (*metis eq-onp-def eq-onp-intF mem-Collect-eq times-mpoly.rep-eq*
times-poly-mapping.rep-eq)
subgoal
by (*metis eq-onp-def eq-onp-intF finite-prod-fun mem-Collect-eq*)
subgoal
using *of-int-prod-fun* **by** *blast*
done
finally have *LHS*: *of-int-mpoly* ($\mathcal{F} * \mathcal{G}$) = *MPoly* (*Abs-poly-mapping*
(*prod-fun* (λx. *of-int* (*lookup* (*mapping-of* $\mathcal{F}$) x)) (λx. *of-int* (*lookup* (*mapping-of* $\mathcal{G}$) x))))
by *auto*

have *of-int-mpoly* $\mathcal{F}$ * *of-int-mpoly* $\mathcal{G}$ = *MPoly*
(*Abs-poly-mapping* (λx. *of-int* (*lookup* (*mapping-of* $\mathcal{F}$) x)) *
Abs-poly-mapping (λx. *of-int* (*lookup* (*mapping-of* $\mathcal{G}$) x)))
unfolding *of-int-mpoly-def*
unfolding *MPoly-Type.times-mpoly.rep-eq MPoly-Type.times-mpoly.abs-eq*
unfolding *comp-apply*
by *auto*

also have ... = *MPoly* (*Abs-poly-mapping*
(*prod-fun* (λx. *of-int* (*lookup* (*mapping-of* $\mathcal{F}$) x)) (λx. *of-int* (*lookup* (*mapping-of* $\mathcal{G}$) x))))
using *Abs-distrib MPoly-inject* **by** *auto*
finally have *RHS*:*of-int-mpoly* $\mathcal{F}$ * *of-int-mpoly* $\mathcal{G}$ =
MPoly (*Abs-poly-mapping*
(*prod-fun* (λx. *of-int* (*lookup* (*mapping-of* $\mathcal{F}$) x)) (λx. *of-int* (*lookup* (*mapping-of* $\mathcal{G}$) x))))
by *auto*
show *?thesis* **unfolding** *LHS RHS* **by** *auto*
qed

lemma *of-int-mpoly-power* [*of-int-mpoly-simps*]:
fixes $\mathcal{F}$ :: *int mpoly*
shows *of-int-mpoly* ($\mathcal{F}$ ^ *n*) = (*of-int-mpoly* $\mathcal{F}$) ^ *n*
by (*induction n*; *simp add*: *of-int-mpoly-simps*)

lemma *of-int-mpoly-sum* [*of-int-mpoly-simps*]:
fixes *f* :: *'a* ⇒ *int mpoly* **and** *S*
shows *of-int-mpoly* (*sum f S*) = *sum* (*of-int-mpoly* ∘ *f*) *S*
by (*rule infinite-finite-induct*; *auto simp add*: *of-int-mpoly-simps*)

lemma *of-int-mpoly-prod* [*of-int-mpoly-simps*]:
fixes *f* :: *'a* ⇒ *int mpoly* **and** *S*
shows *of-int-mpoly* (*prod f S*) = *prod* (*of-int-mpoly* ∘ *f*) *S*
by (*rule infinite-finite-induct*; *auto simp add*: *of-int-mpoly-simps*)

lemma *of-int-mpoly-Sum-any*:
fixes *f* :: *'a* ⇒ *int mpoly*
assumes *finite* {*a*. *f a* ≠ *0*}
shows (*of-int-mpoly* (*Sum-any f*) :: *'b*::*ring-1 mpoly*) = *Sum-any* (*of-int-mpoly* ∘ *f*)
proof −
have *of-int-mpoly* (*sum f* {*a*. *f a* ≠ *0*}) = *sum* (*of-int-mpoly* ∘ *f*) {*a*. *f a* ≠ *0*}
by (*rule of-int-mpoly-sum*)
also have ... = *sum* (*of-int-mpoly* ∘ *f* :: *'a* ⇒ *'b mpoly*) {*a*. (*of-int-mpoly* ∘ *f* :: *'a* ⇒ *'b mpoly*) *a* ≠ *0*}
apply (*rule sum.mono-neutral-cong-right*)
using *assms* **apply** (*auto simp add*: *of-int-mpoly-simps*)
done
finally show *?thesis* **unfolding** *Sum-any.expand-set* **.**
qed

lemma *of-int-mpoly-Prod-any*:
fixes *f* :: *'a* ⇒ *int mpoly*
assumes *finite* {*a*. *f a* ≠ *1*}
shows (*of-int-mpoly* (*Prod-any f*) :: *'b*::*comm-ring-1 mpoly*) = *Prod-any* (*of-int-mpoly* ∘ *f*)
proof −
have *of-int-mpoly* (*prod f* {*a*. *f a* ≠ *1*}) = *prod* (*of-int-mpoly* ∘ *f*) {*a*. *f a* ≠ *1*}
by (*rule of-int-mpoly-prod*)
also have ... = *prod* (*of-int-mpoly* ∘ *f* :: *'a* ⇒ *'b mpoly*) {*a*. (*of-int-mpoly* ∘ *f* :: *'a* ⇒ *'b mpoly*) *a* ≠ *1*}
apply (*rule prod.mono-neutral-cong-right*)
using *assms* **apply** (*auto simp add*: *of-int-mpoly-simps*)
done
finally show *?thesis* **unfolding** *Prod-any.expand-set* **.**
qed

lemma *insertion-of-int-mpoly*:
insertion (*of-int* ∘ *α*) ((*of-int-mpoly P*) :: *'a*::*comm-ring-1 mpoly*) = *of-int* (*insertion*

```
α P)
proof −
  show ?thesis
    unfolding of-int-mpoly-def
    unfolding insertion-def insertion-aux-def insertion-fun-def
    apply (simp add: MPoly-inverse)
    apply (subst of-int-Sum-any)
    subgoal by simp
    subgoal
      unfolding comp-def
      apply (rule Sum-any.cong)
      unfolding of-int-mult
      apply (rule arg-cong2[of - - - - times])
      subgoal
        apply (subst Abs-poly-mapping-inverse)
        subgoal
          using finite-imageI[of - of-int] finite-lookup
          by (smt (verit, best) finite-subset mem-Collect-eq of-int-0 subsetI)
        subgoal ..
        done
      subgoal for a
        apply (subst of-int-Prod-any)
        subgoal
          apply (rule rev-finite-subset[of {x. lookup a x ≠ 0}])
          subgoal by simp
          subgoal using power-0 Collect-mono-iff by fastforce
          done
        subgoal by simp
        done
      done
    done
qed

lemma of-int-mpoly-poly-subst-monom [of-int-mpoly-simps]:
  of-int-mpoly (poly-subst-monom f a) = poly-subst-monom (of-int-mpoly ∘ f :: nat
⇒ 'a::comm-ring-1 mpoly) a
  unfolding poly-subst-monom-def comp-def
  apply (subst of-int-mpoly-Prod-any)
 subgoal by (smt (verit, best) Collect-mono-iff finite-lookup power-0 rev-finite-subset)
  subgoal by (simp add: of-int-mpoly-simps)
  done

lemma of-int-mpoly-poly-subst [of-int-mpoly-simps]:
  of-int-mpoly (poly-subst f P) = poly-subst (of-int-mpoly ∘ f :: nat ⇒ 'a::comm-ring-1
mpoly) (of-int-mpoly P)
  unfolding poly-subst-def
  unfolding coeff-def times-mpoly.rep-eq times-poly-mapping.rep-eq prod-fun-def
  unfolding coeff-def[symmetric]
  apply (subst of-int-mpoly-Sum-any)
```

subgoal by (*metis* (*mono-tags*, *lifting*) *Collect-mono coeff-def finite-lookup mult-zero-left of-int-0-eq-iff of-int-Const rev-finite-subset*)
subgoal
apply (*rule Sum-any.cong*)
apply (*simp add*: *of-int-mpoly-simps*)
done
done

lemma *of-int-general-Const*: (*of-int x* :: (*'a*::*ring-1*) *mpoly*) = *of-int-mpoly* (*Const x*)
by (*metis Const.abs-eq Const*$_0$*-def monom.abs-eq monom-of-int of-int-mpoly-Const*)

lemma *of-int-mpoly-degree*[*simp*]:
MPoly-Type.degree (*of-int-mpoly P* :: (*'a*::*ring-1*) *mpoly*) ≤ *MPoly-Type.degree P*
proof −

have *a*: {*k*. (*of-int* ∘ *lookup* (*mapping-of P*)) *k* ≠ *0*} ⊆ {*k*. (*lookup* (*mapping-of P*)) *k* ≠ *0*}
by *auto*

hence *b*: *insert 0* ((λ*m*. *lookup m x*) ‘ {*k*. (*of-int* ∘ *lookup* (*mapping-of P*)) *k* ≠ *0*})
⊆ *insert 0* ((λ*m*. *lookup m x*) ‘ {*k*. (*lookup* (*mapping-of P*)) *k* ≠ *0*}) **for** *x*
by *auto*

show *?thesis*
unfolding *MPoly-Type.degree.rep-eq of-int-mpoly-def*
apply (*subst MPoly-inverse*)
subgoal by *simp*
apply (*rule le-funI*)
unfolding *keys.rep-eq*
apply (*subst Abs-poly-mapping-inverse*)
subgoal by *auto* (*metis* (*mono-tags*, *lifting*) *finite-lookup finite-subset mem-Collect-eq of-int-0 subsetI*)
unfolding *comp-def*
using *Max-mono*[*OF b*] **by** *auto*
qed

lemma *vars-of-int-mpoly*:
vars (*of-int-mpoly P* :: (*'a* :: {*comm-semiring-1*, *ring-1*}) *mpoly*) ⊆ *vars P*
unfolding *of-int-mpoly-def vars-def*
by (*metis SUP-mono coeff-keys dual-order.refl of-int-0 of-int-mpoly-coeff of-int-mpoly-def*)

end
theory *More-More-MPoly-Type*
imports *Type-Casting Substitutions Poly-Expansions Total-Degree*
Variables Degree Notation

begin

end
theory *Poly-Extract*
imports *More-More-MPoly-Type*
keywords *poly-extract* :: *thy-defn*
begin

1.8 Automatic generation of polynomials from Isabelle terms

ML‹
```
fun gen-def spec lthy =
  let
    val atts = [];
    val (((x, T), rhs), prove) = Local-Defs.derived-def lthy (fn - => []) {conditional
= true} spec;
    val - = Name.reject-internal (x, []);
    val (b, mx) = (Binding.make (x, Position.none), NoSyn)
    val name = Thm.def-binding b;
    val ((lhs, (-, raw-th)), lthy2) = lthy
      |> Local-Theory.define-internal ((b, mx), ((Binding.suffix-name -raw name,
[]), rhs));
    val th = prove lthy2 raw-th;
    val lthy3 = lthy2 |> Spec-Rules.add name Spec-Rules.equational [lhs] [th];
    val ([(def-name, [th'])], lthy4) = lthy3
      |> Local-Theory.notes [((name, atts), [([th], [])])];
    val lthy5 = lthy4
      |> Code.declare-default-eqns [(th', true)];
    val lhs' = Morphism.term (Local-Theory.target-morphism lthy5) lhs;
    val - =
      Proof-Display.print-consts true (Position.thread-data ()) lthy5
        (Frees.defined (Frees.build (Frees.add-frees lhs'))) [(x, T)];
  in ((lhs, (def-name, th')), lthy5) end;

fun gen-theorems (name, thms) lthy =
  let
    val kind = lemma;
    val facts = [((Binding.make (name, Position.none), []), [(thms, [])])];
    val facts' = facts |> Attrib.partial-evaluation lthy;
    val (res, lthy') = lthy |> Local-Theory.notes-kind kind facts';
    val - =
      Proof-Display.print-results
        {interactive = true, pos = Position.thread-data (), proof-state = false}
        lthy' ((kind, ), res);
  in (res, lthy') end;

fun find-def (ctxt : Proof.context) s =
  let
    val thy = Proof-Context.theory-of ctxt;
```

```
  val axioms = (Theory.all-axioms-of thy |> map #2) @
  (Locale.get-locales thy |> maps (Locale.axioms-of thy) |> map Thm.full-prop-of)
in
  case
    axioms
    |> List.mapPartial (fn axiom-statement =>
      case axiom-statement of
        Const (Pure.eq, -) $ lhs $ rhs =>
          let
            val (lhs-fn, lhs-args) = strip-comb lhs;
          in
            if lhs-fn aconv s then SOME (lhs-args, rhs) else NONE
          end
      | - => NONE
      )
  of
      def :: - => def
    | [] => error (Definition not found: ^ @{make-string} s)
end;

fun find-thm (ctxt : Proof.context) (name: string) : thm =
  case
    Proof-Context.get-fact ctxt (Facts.named name)
  of
      thm :: - => thm
    | [] => error (Theorem not found: ^ name);
```

ML‹

```
fun convert-num n =
  if n < 1 then
    raise Match
  else if n = 1 then
    Const (Num.num.One, @{typ num})
  else if n mod 2 = 0 then
    Const (Num.num.Bit0, @{typ num ⇒ num}) $ convert-num (n div 2)
  else
    Const (Num.num.Bit1, @{typ num ⇒ num}) $ convert-num (n div 2);

fun convert-nat n =
  if n < 0 then
    raise Match
  else if n = 0 then
    Const (Groups.zero-class.zero, @{typ nat})
  else if n = 1 then
    Const (Groups.one-class.one, @{typ nat})
  else
    Const (Num.numeral-class.numeral, @{typ num ⇒ nat}) $ (convert-num n);
```

```
fun convert-nat-normal n =
  if n < 0 then
    raise Match
  else if n = 0 then
    Const (Groups.zero-class.zero, @{typ nat})
  else
    Const (Nat.Suc, @{typ nat ⇒ nat}) $ (convert-nat-normal (n − 1));

fun convert-list typ l =
  case l of
    [] => Const (List.list.Nil, Type(List.list, [typ]))
 | s :: l' => Const (List.list.Cons, typ --> Type(List.list, [typ]) --> Type(List.list,
[typ])) $ s $ convert-list typ l';

fun dest-list l =
  case l of
    Const (List.list.Nil, -) => []
  | Const (List.list.Cons, -) $ s $ l' => s :: dest-list l'
  | - => raise Match

fun get-deps input =
  let
    val (input-fn, input-args) = strip-comb input;
  in
    case (input-fn, input-args) of
      (Bound -, -) => []
    | (Const (Num.num.One, -), -) => []
    | (Const (Num.num.Bit0, -), -) => []
    | (Const (Num.num.Bit1, -), -) => []
    | (Const (Groups.zero-class.zero, -), -) => []
    | (Const (Groups.one-class.one, -), -) => []
    | (Const (Num.numeral-class.numeral, -), -) => []
    | (Const (Nat.semiring-1-class.of-nat, -), -) => []
    | (Const (Groups.plus-class.plus, -), [s, t]) => get-deps s @ get-deps t
    | (Const (Groups.minus-class.minus, -), [s, t]) => get-deps s @ get-deps t
    | (Const (Groups.uminus-class.uminus, -), [s]) => get-deps s
    | (Const (Groups.times-class.times, -), [s, t]) => get-deps s @ get-deps t
    | (Const (Power.power-class.power, -), [s, -]) => get-deps s
   | (Const (Fun.comp, -), [Const(MPoly-Type.insertion, -), Const(Notation.nth0,
-), -, -]) => []
    | (Const (dep, -), -) => [dep] @ (input-args |> filter (not o is-Free) |> maps
get-deps)
    | - => error (Unknown term: ^ (@{make-string} input))
  end;

fun make-mpoly var-count (input : term) : term =
  let
    val (input-fn, input-args) = strip-comb input;
  in
```

```
    case (input-fn, input-args) of
      (Bound i, []) =>
       Const (MPoly-Type.Var, @{typ nat ⇒ int mpoly}) $ convert-nat (var-count
- i - 1)
    | (Const (Groups.zero-class.zero, -), []) =>
      Const (Groups.zero-class.zero, @{typ int mpoly})
    | (Const (Groups.one-class.one, -), []) =>
      Const (Groups.one-class.one, @{typ int mpoly})
    | (Const (Num.numeral-class.numeral, -), [s]) =>
      Const (Num.numeral-class.numeral, @{typ num => int mpoly}) $ s
    | (Const (Nat.semiring-1-class.of-nat, -), -) =>
      Const (MPoly-Type.Const, @{typ int ⇒ int mpoly}) $ input
    | (Const (Groups.plus-class.plus, -), [s, t]) =>
      Const (Groups.plus-class.plus, @{typ int mpoly ⇒ int mpoly ⇒ int mpoly}) $
make-mpoly var-count s $ make-mpoly var-count t
    | (Const (Groups.minus-class.minus, -), [s, t]) =>
     Const (Groups.minus-class.minus, @{typ int mpoly ⇒ int mpoly ⇒ int mpoly})
$ make-mpoly var-count s $ make-mpoly var-count t
    | (Const (Groups.uminus-class.uminus, -), [s]) =>
        Const (Groups.uminus-class.uminus, @{typ int mpoly ⇒ int mpoly}) $
make-mpoly var-count s
    | (Const (Groups.times-class.times, -), [s, t]) =>
     Const (Groups.times-class.times, @{typ int mpoly ⇒ int mpoly ⇒ int mpoly})
$ make-mpoly var-count s $ make-mpoly var-count t
    | (Const (Power.power-class.power, -), [s, t]) =>
       Const (Power.power-class.power, @{typ int mpoly ⇒ nat ⇒ int mpoly}) $
make-mpoly var-count s $ t
    | (Const (Fun.comp, -), [Const(MPoly-Type.insertion, -), Const(Notation.nth0,
-), args, poly]) =>
      @{term poly-subst-list :: - ⇒ - ⇒ int mpoly} $
       (args |> dest-list |> map (make-mpoly var-count) |> convert-list @{typ int
mpoly}) $
      poly
    | (Const(const-name, -), -) =>
      let
        val params = input-args |> filter is-Free;
        val param-types = params |> map fastype-of;
        val args = input-args |> filter (not o is-Free);
      in
        @{term poly-subst-list :: - ⇒ - ⇒ int mpoly} $
        (args |> map (make-mpoly var-count) |> convert-list @{typ int mpoly}) $
        (list-comb (
          Const (const-name ^ -poly, param-types ---> @{typ int mpoly}),
          params
        ))
      end
    | - => error (Unknown term: ^ (@{make-string} input))
  end
```

```
fun prove-nat-normal (ctxt : Proof.context) =
  List.tabulate (30, fn i =>
    let
      val statement =
        HOLogic.mk-Trueprop (HOLogic.mk-eq (
          convert-nat i,
          convert-nat-normal i
        ))
    in
      Goal.prove ctxt [] [] statement (fn {prems = -, context} =>
        Clasimp.auto-tac context
      )
    end
  ) |> tl

fun string-of-string-list (xs : string list) : string =
  case xs of
    [] => []
  | [x] => x
  | x::ys => x ^ ,  ^ string-of-string-list ys

fun string-of-terms (ctxt : Proof.context) (terms: term list): string =
  string-of-string-list (map (Syntax.string-of-term ctxt) terms)

fun string-of-typs (ctxt : Proof.context) (typs: typ list): string =
  string-of-string-list (map (Syntax.string-of-typ ctxt) typs)

fun print-strings (strings : string list) =
  writeln (string-of-string-list strings)

fun print-terms (ctxt : Proof.context) (terms : term list) =
  print-strings (map (Syntax.string-of-term ctxt) terms)

fun print-typs (ctxt : Proof.context) (typs : typ list) =
  print-strings (map (Syntax.string-of-typ ctxt) typs)

›
```

ML‹

```
fun make-thm-statement (converted-const: term) (input-const: term)
  (input-def-params: term list) (var-count: int) : term =
  Const (Pure.all, (@{typ nat ⇒ int} --> propT) --> propT) $ Abs (
    fn,
    @{typ nat ⇒ int},
    HOLogic.mk-Trueprop (
      HOLogic.mk-eq (
        @{term insertion :: - ⇒ - ⇒ int} $ Bound 0 $ converted-const,
```

```
      list-comb (
        list-comb(Term.close-schematic-term input-const, input-def-params),
        List.tabulate (var-count, fn i => Bound 0 $ convert-nat i)
      )
    )
  )
)

fun prove-correctness-theorem lthy' correct-thm-statement input-def converted-def
dep-correct-thms =
  Goal.prove lthy' [] [] correct-thm-statement (fn {prems = -, context} =>
    Local-Defs.unfold-tac context (prove-nat-normal lthy' @ [
      @{thm insertion-Const},
      @{thm insertion-Var},
      @{thm insertion-add},
      @{thm insertion-diff},
      @{thm insertion-neg},
      @{thm insertion-mult},
      @{thm insertion-pow},
      @{thm insertion-nth0},
      @{thm insertion-poly-subst},
      @{thm poly-subst-list-def},
      @{thm comp-def},
      (nth @{thms List.list.map} 0),
      (nth @{thms List.list.map} 1),
      @{thm nth0-Cons-0},
      @{thm nth0-Cons-Suc},
      input-def,
      converted-def
    ] @ dep-correct-thms) THEN Clasimp.auto-tac context
  );

val - =
  Outer-Syntax.local-theory command-keyword‹poly-extract›
    extract terms as polynomials
    (Parse.term >> (fn raw-input => fn lthy =>
      let
        val input = Syntax.read-term lthy raw-input;
        val (input-const : term, input-params : term list) = strip-comb input;

        val input-const-name = input-const |> dest-Const |> #1;
          val input-const-short-name = Binding.name-of (Binding.qualified-name
input-const-name);

        val input-def : thm = find-thm lthy (input-const-name ^ -def);
          val (input-def-params : term list, input-def-value : term) = find-def lthy
input-const;

        val intparams = List.filter (fn term => let
```

```
    val termtype = fastype-of term
    in termtype = @{typ int}
  end) input-def-params

  val nonintparams = List.filter (fn term => let
    val termtype = fastype-of term
    in termtype <> @{typ int}
  end) input-def-params


   val input-def-value-with-closed-ints : term = fold lambda (rev intparams)
input-def-value

  val full-substitution : (indexname * term) list =
    ListPair.zip (map #1 (map dest-Var input-def-params), input-params)

  val substitution : (indexname * term) list =
    List.filter (fn (-, term) =>
      let
        val termtype : typ = fastype-of term
      in termtype <> @{typ int} end)
    full-substitution

 val applied-input-def-value = subst-Vars substitution input-def-value-with-closed-ints;

  val (vars, input-def-value-content) = strip-abs applied-input-def-value;

  val deps = get-deps input-def-value-content;
    val dep-correct-thms = deps |> map (fn dep => find-thm lthy (dep ^
-correct));
  val var-count = vars |> length;

  val mpoly : term = make-mpoly var-count input-def-value-content;
  val ((mpoly-const, (mpoly-def-name, -)), lthy'') =
    lthy
    |> gen-def (Logic.mk-equals
          (Free (input-const-short-name ^ -poly, @{typ int mpoly}), mpoly))

  val mpoly-fact : thm = find-thm lthy'' mpoly-def-name;

  val - = writeln (Generated definition:  ^ Syntax.string-of-term lthy mpoly)

  val correct-thm-statement : term =
    make-thm-statement mpoly-const input-const nonintparams var-count

   val - = writeln (Proved correctness theorem:  ^ Syntax.string-of-term lthy
correct-thm-statement)

    val correct-thm = prove-correctness-theorem lthy'' correct-thm-statement
```

```
input-def mpoly-fact dep-correct-thms
      val (-, lthy''') =
        lthy'' |> gen-theorems (input-const-short-name ^ -correct, [correct-thm]);
      in lthy'''
end));
›
```

end
theory *Bit-Counting*
imports *Digit-Expansions.Binary-Operations HOL−Computational-Algebra.Primes*
begin

2 The Coding Technique

2.1 Counting bits and number of carries

Count the number of bits in the binary expansion of n

definition *bit-set* :: *nat* ⇒ *nat set* **where**
bit-set n = {*i*. *n* ¡ *i* = *1*}

definition *count-bits* :: *nat* ⇒ *nat* **where**
count-bits n = *card* (*bit-set n*)

Count the number of carries in the binary addition of a and b

definition *carry-set* :: *nat* ⇒ *nat* ⇒ *nat set* **where**
carry-set a b = {*i*. *bin-carry a b i* = *1*}

definition *count-carries* :: *nat* ⇒ *nat* ⇒ *nat* **where**
count-carries a b = *card* (*carry-set a b*)

This shows that {@*const count-bits*} is well defined

lemma *bit-set-subset*: *bit-set n* ⊆ {..<*n*}
proof −
{ fix *i* **assume** *i* ≥ *n*
hence *n* ¡ *i* = *0* **unfolding** *nth-bit-def* **by** (*simp add*: *order-le-less-trans*) **}**
thus *?thesis*
unfolding *bit-set-def*
by (*metis* (*mono-tags*, *lifting*) *lessThan-iff dual-order.strict-iff-not mem-Collect-eq*

nle-le subsetI zero-neq-one)
qed

corollary *bit-set-finite*: *finite* (*bit-set n*)
using *bit-set-subset finite-subset* **by** *fastforce*

We can be more precise when we know how many bits n requires

lemma *bit-set-subset-variant*: *n* < *2*^*k* ⟹ *bit-set n* ⊆ {..<*k*}

proof −
assume $n < 2\hat{}k$
hence $i \geq k \Longrightarrow n \,¦\, i = 0$ **for** i
by (*metis add.commute add-cancel-left-right aux1-digit-lt-linear aux1-digit-wise-equiv mult-0*)
thus *?thesis*
unfolding *bit-set-def*
by (*metis* (*mono-tags*, *lifting*) *One-nat-def lessThan-iff less-le-not-le linorder-neqE-nat mem-Collect-eq nat.simps*(*3*) *order-refl subsetI*)
qed

corollary *count-bits-def-sum*: $n < 2\hat{}k \Longrightarrow$ *count-bits* $n = (\sum i{<}k.\ n \,¦\, i)$
proof −
assume *assm*: $n < 2\hat{}k$

have $(\sum i{<}k.\ n \,¦\, i) = (\sum i{\in}\{..{<}k\}.\ \textit{of-bool}\ (n \,¦\, i = 1))$
by (*smt* (*verit*, *del-insts*) *sum.cong not-mod-2-eq-1-eq-0 nth-bit-def of-bool-eq*(*1*) *of-bool-eq*(*2*))
also have ... = *card* $(\{..{<}k\} \cap$ *bit-set* $n)$
unfolding *bit-set-def* **using** *Groups-Big.sum-of-bool-eq finite-lessThan* **by** *simp*
also have ... = *count-bits* n
unfolding *count-bits-def* **using** *bit-set-subset-variant*[*OF assm*] **by** (*simp add*: *inf.absorb2*)
finally show *?thesis* **by** *simp*
qed

corollary *count-bits-bounded*: $n < 2\hat{}k \Longrightarrow$ *count-bits* $n \leq k$
unfolding *count-bits-def* **using** *bit-set-subset-variant*
by (*metis card-lessThan card-mono finite-lessThan*)

The following lemma shows that {@*const count-carries*} is well defined

lemma *carry-set-subset*: *carry-set* $a\ b \subseteq \{..\textit{max}\ a\ b\}$
proof −
{ fix i **assume** $i > \textit{max}\ a\ b$
hence $i > a$ **and** $i > b$
by *simp-all*
have $a < 2\hat{}(i{-}1)$ **using** ‹$i > a$›
by (*metis Suc-diff-1 Suc-leI gr-implies-not0 less-exp not-le-imp-less order-less-le-trans*)
moreover have $b < 2\hat{}(i{-}1)$ **using** ‹$i > b$›
by (*metis Suc-diff-1 Suc-leI gr-implies-not0 less-exp not-le-imp-less order-less-le-trans*)
ultimately have $a + b < 2\hat{}(\textit{Suc}\ (i{-}1))$
by *simp*
hence *ab-bound*: $a + b < 2\hat{}i$
using ‹$i > a$› **by** *simp*

have $(a\ \textit{mod}\ 2\hat{}i) + (b\ \textit{mod}\ 2\hat{}i) = a + b$
using ‹$i > a$› ‹$i > b$› **by** (*metis less-exp less-imp-add-positive mod-less trans-less-add1*)
hence *bin-carry* $a\ b\ i = 0$

unfolding *bin-carry-def* **using** *ab-bound* **by** *simp* **}**
thus *?thesis*
unfolding *carry-set-def*
by (*metis* (*mono-tags*, *lifting*) *atMost-iff mem-Collect-eq subsetI verit-comp-simplify1*(*3*) *zero-neq-one*)
qed

corollary *carry-set-finite*: *finite* (*carry-set a b*)
using *carry-set-subset finite-subset* **by** *blast*

We can be more precise when we know how many bits $a + b$ requires

lemma *carry-set-subset-variant*: *a + b < 2^k ⟹ carry-set a b ⊆ {..<k}*
proof −
assume *assm*: *a + b < 2^k*
hence *i ≥ k ⟹ bin-carry a b i = 0* **for** *i*
unfolding *bin-carry-def*
using *Euclidean-Rings.div-eq-0-iff add-increasing add-increasing2 div-exp-eq le-antisym le-eq-less-or-eq le-iff-add mod-eq-self-iff-div-eq-0 zero-le*
by (*smt* (*verit*, *ccfv-threshold*) *div-greater-zero-iff*)
thus *?thesis*
unfolding *carry-set-def*
by (*metis* (*full-types*, *lifting*) *lessThan-iff linorder-not-less mem-Collect-eq subsetI zero-neq-one*)
qed

corollary *count-carries-def-sum*: *a + b < 2^k ⟹ count-carries a b* = (∑ *i*<*k*. *bin-carry a b i*)
proof −
assume *assm*: *a + b < 2^k*

have (∑ *i*<*k*. *bin-carry a b i*) = (∑ *i*∈{..<*k*}. *of-bool* (*bin-carry a b i = 1*))
using *sum.cong bin-carry-def*
by (*smt* (*verit*, *best*) *bin-carry-bounded not-mod-2-eq-0-eq-1 of-bool-eq*(*1*) *of-bool-eq*(*2*))

also have ... = *card* ({..<*k*} ∩ *carry-set a b*)
unfolding *carry-set-def* **using** *Groups-Big.sum-of-bool-eq finite-lessThan* **by** *simp*
also have ... = *count-carries a b*
unfolding *count-carries-def* **using** *carry-set-subset-variant*[*OF assm*] **by** (*simp add*: *inf.absorb2*)
finally show *?thesis* **by** *auto*
qed

Some elementary properties of {@*const count-bits*} and {@*const count-carries*}

lemma *bit-set-0*[*simp*]: *bit-set 0* = {}
unfolding *bit-set-def nth-bit-def* **by** *simp*

corollary *count-bits-0*[*simp*]: *count-bits 0 = 0*
unfolding *count-bits-def* **by** *simp*

lemma *bit-set-1*[*simp*]: *bit-set 1* = {*0*}
proof −
  have *bit-one*: *1* ¡ *i* = (*if i* = *0 then 1 else 0*) **for** *i*
    unfolding *nth-bit-def*
    using *div-eq-0-iff bits-one-mod-two-eq-one gr-zeroI mod-0*
        one-div-two-eq-zero one-less-power power-0 zero-neq-numeral
    by (*metis div-by-1*)
  thus *?thesis* **unfolding** *bit-set-def bit-one* **by** *simp*
qed

corollary *count-bits-1*[*simp*]: *count-bits 1* = *1*
  unfolding *count-bits-def bit-set-1* **by** *simp*

lemma *carry-set-n0*[*simp*]: *carry-set n 0* = {}
  unfolding *carry-set-def bin-carry-def* **by** *simp*
lemma *carry-set-0n*[*simp*]: *carry-set 0 n* = {}
  unfolding *carry-set-def bin-carry-def* **by** *simp*

corollary *count-carries-n0*[*simp*]: *count-carries n 0* = *0*
  unfolding *count-carries-def* **by** *simp*
corollary *count-carries-0n*[*simp*]: *count-carries 0 n* = *0*
  unfolding *count-carries-def* **by** *simp*

lemma *carry-set-sym*: *carry-set a b* = *carry-set b a*
  unfolding *carry-set-def bin-carry-def* **by** (*simp add*: *add.commute*)

corollary *count-carries-sym*: *count-carries a b* = *count-carries b a*
  unfolding *count-carries-def* **by** (*simp add*: *carry-set-sym*)

lemma *aux-geometric-sum*:
  (*x*::*nat*) > *1* ⟹ (*x*−*1*) ∗ (∑ *i*<*n*. *x*^*i*) = *x*^*n* − *1*
proof (*induct n*)
  case *0* **show** *?case* **using** *lessThan-0* **by** (*simp add*: *algebra-simps*)
next
  case (*Suc n*)
  have (*x*−*1*) ∗ (∑ *i*<*Suc n*. *x*^*i*) = (*x*−*1*) ∗ (∑ *i*<*n*. *x*^*i*) + (*x*−*1*) ∗ *x*^*n*
    using *add-mult-distrib2* **by** *auto*
  also have ... = *x*^*n* − *1* + *x*^(*Suc n*) − *x*^*n*
    using *Suc less-imp-Suc-add* **by** *fastforce*
  also have ... = *x*^(*Suc n*) − *1*
    using *Suc.prems* **by** *auto*
  finally show *?case* **.**
qed

lemma *aux-digit-sum-bound*:
  assumes *1* < (*b*::*nat*) **and** ∀ *i*<*q*. *f i* < *b*
  shows (∑ *i*<*q*. *f i* ∗ *b*^*i*) < *b*^*q*
proof −

```
  have i < q ⟹ f i * b^i ≤ (b−1) * b^i for i
    by (metis assms le-add-diff-inverse less-Suc-eq-le less-imp-le-nat mult-le-mono1
plus-1-eq-Suc)
  hence (∑ i<q. f i * b^i) ≤ (∑ i<q. (b−1) * b^i)
    by (meson lessThan-iff sum-mono)
  also have ... = b^q − 1
    using sum-distrib-left aux-geometric-sum assms(1) by metis
  finally show ?thesis
   by (metis assms(1) le-add-diff-inverse le-imp-less-Suc less-imp-le-nat one-le-power

      plus-1-eq-Suc)
qed

lemma carry-set-same[simp]: carry-set a a = Suc ` bit-set a
  (is ?A = ?B)
proof −
  have 0: bin-carry a a (Suc i) = a ¡ i for i
    unfolding bin-carry-def nth-bit-def
   by (metis One-nat-def add.commute add-self-div-2 div-exp-eq div-exp-mod-exp-eq
plus-1-eq-Suc
      power-Suc0-right)

  have 1: ?A ⊆ ?B
  proof
    fix i assume i-def: i ∈ carry-set a a
    hence i > 0
      unfolding carry-set-def bin-carry-def using gr0I by force
    hence a ¡ (i−1) = 1
      using i-def 0 unfolding carry-set-def
      by (metis (mono-tags, lifting) One-nat-def Suc-pred mem-Collect-eq)
    thus i ∈ ?B
      using bit-set-def Suc-pred' ‹0 < i› by blast
  qed

  have 2: ?B ⊆ ?A
  proof
    fix i assume i-def: i ∈ ?B
    hence a ¡ (i−1) = 1
      using bit-set-def by auto
    hence bin-carry a a i = 1
      using 0 i-def by fastforce
    thus i ∈ ?A
      using carry-set-def by simp
  qed

  from 1 2 show ?thesis by simp
qed

corollary count-carries-same[simp]: count-carries a a = count-bits a
```

using *count-carries-def count-bits-def carry-set-same* **by** (*simp add*: *card-image*)

lemma *bit-set-pow2*[*simp*]: *bit-set* (*2^k*) = {*k*}
proof −
have (*2^k*) ¡ *i* = (*if i* = *k then 1 else 0*) **for** *i*
unfolding *nth-bit-def*
by (*metis Suc-mask-eq-exp bit-exp-iff bit-iff-odd nat.simps*(*3*) *not-mod-2-eq-1-eq-0*

odd-iff-mod-2-eq-one possible-bit-def)
thus *?thesis* **unfolding** *bit-set-def* **by** *simp*
qed

corollary *count-bits-pow2*[*simp*]: *count-bits* (*2^k*) = *1*
unfolding *count-bits-def* **by** *simp*

lemma *bit-set-block-ones*[*simp*]: *bit-set* (*2^k* − *1*) = {..<*k*}
proof −
have *bit*: (*2^k* − *1*) ¡ *i* = (*if i* < *k then 1 else 0*) **for** *i*
unfolding *nth-bit-def*
by (*meson even-decr-exp-div-exp-iff' mod2-eq-if not-less*)

hence *bit-set* (*2^k* − *1*) = *bit-set* (*2^k*−*1*) ∩ {..<*k*}
unfolding *count-bits-def* **using** *bit-set-subset-variant* **by** (*simp add*: *inf-absorb1*)
also have ... = {*i*. (*2^k*−*1*) ¡ *i* = *1* ∧ *i* < *k*}
unfolding *bit-set-def* **by** (*simp add*: *Collect-conj-eq lessThan-def*)
also have ... = {*i*. *i* < *k*}
unfolding *bit* **by** *simp*
finally show *?thesis*
by *auto*
qed

corollary *count-bits-block-ones*[*simp*]: *count-bits* (*2^k*−*1*) = *k*
unfolding *count-bits-def bit-set-block-ones* **by** *simp*

The binary complement of a number with k bits

lemma *nth-bit-complement*:
a < *2^k* ⟹ (*2^k*−*1*−*a*) ¡ *i* = (*if i* < *k then 1* − (*a* ¡ *i*) *else 0*)
proof (*cases k* = *0*)
case *True* **assume** *a* < *2^k*
hence *a* = *0* **using** *True* **by** *simp*
thus *?thesis* **using** *True nth-bit-def* **by** *simp*
next
case *False* **assume** *assm*: *a* < *2^k*
have *0*: ∀ *i*. (*a* ¡ *i*) ∗ *2^i* ≤ *2^i* **using** *nth-bit-bounded* **by** *simp*
have *1*: ∀ *i*. *1* − (*a* ¡ *i*) < *2^*(*Suc 0*) **by** *auto*
have *2*: *k* = *Suc* (*k*−*1*) **using** *False* **by** *simp*

have *2^k*−*1*−*a* = (*2^k* − *1*) − (∑ *i*<*k*. (*a* ¡ *i*) ∗ *2^i*)
using *digit-sum-repr*[*OF assm*] **by** *simp*

```
  also have ... = (∑ i<k. 2^i) − (∑ i<k. (a ¡ i) * 2^i)
    using aux-geometric-sum[of 2::nat] by simp
  also have ... = (∑ i<k. 2^i − (a ¡ i) * 2^i)
    using sum-diff-distrib[OF 0] .
  also have ... = (∑ i<k. (1 − a ¡ i) * 2^i)
    using nth-bit-bounded
    by (metis (no-types, opaque-lifting) cancel-comm-monoid-add-class.diff-cancel
diff-zero
      mult.commute mult.right-neutral mult-zero-right not-mod-2-eq-1-eq-0 nth-bit-def)
  finally have complement: 2^k−1−a = (∑ i=0..k−1. (1 − a ¡ i) * 2^i)
    using 2 by (metis atLeast0LessThan atLeastLessThanSuc-atLeastAtMost)

  have (2^k−1−a) ¡ i = (if i ≤ k−1 then 1 − a ¡ i else 0)
    using nth-digit-gen-power-series[OF 1]
    unfolding complement nth-digit-base2-equiv by simp
  thus ?thesis using False by simp
qed

lemma bit-set-complement:
  a < 2^k ⟹ bit-set (2^k−1−a) = {..<k} − bit-set a
proof
  assume assm: a < 2^k
  show bit-set (2^k−1−a) ⊆ {..<k} − bit-set a
  proof
    fix i assume i-def: i ∈ bit-set (2^k−1−a)
    hence i < k
      by (metis (mono-tags, lifting) One-nat-def assm bit-set-def mem-Collect-eq
        nat.simps(3) nth-bit-complement)
    thus i ∈ {..<k} − bit-set a
      using nth-bit-complement i-def unfolding bit-set-def by (simp add: assm)
  qed
next
  assume assm: a < 2^k
  show {..<k} − bit-set a ⊆ bit-set (2^k−1−a)
  proof
    fix i assume i ∈ {..<k} − bit-set a
    thus i ∈ bit-set (2^k−1−a)
      using bit-set-def nth-bit-complement[OF assm] by (simp add: nth-bit-def)
  qed
qed

corollary count-bits-complement:
  a < 2^k ⟹ count-bits (2^k−1−a) = k − count-bits a
  unfolding count-bits-def using bit-set-complement bit-set-subset-variant
  by (simp add: bit-set-finite card-Diff-subset)

lemma carry-set-pow2-block-ones[simp]: carry-set (2^k) (2^k−1) = {}
proof −
  have bin-carry (2^k) (2^k−1) i = 0 for i
```

```
  proof (induct i)
    case 0 show ?case unfolding bin-carry-def by simp
  next
    case (Suc i)
    have i < k ⟹ (2^k) ¡ i + (2^k−1) ¡ i = 1
      using bit-set-pow2 bit-set-block-ones nth-bit-bounded unfolding bit-set-def
     by (metis (mono-tags, lifting) One-nat-def Suc-eq-plus1 lessThan-iff mem-Collect-eq

         nat-less-le not-mod-2-eq-1-eq-0 nth-bit-def singletonD)
    moreover have i = k ⟹ (2^k) ¡ i + (2^k−1) ¡ i = 1
      using bit-set-pow2 bit-set-block-ones nth-bit-bounded unfolding bit-set-def
      by (simp add: nth-bit-def)
    moreover have i > k ⟹ (2^k) ¡ i + (2^k−1) ¡ i = 0
      using nth-bit-bounded unfolding bit-set-def
      using add.right-neutral diff-less div-less less-exp linorder-not-less
       nat-less-le not-mod-2-eq-1-eq-0 nth-bit-def odd-iff-mod-2-eq-one power-one-right

         power-strict-increasing-iff
      by (metis even-decr-exp-div-exp-iff')
    ultimately show ?case
      using sum-carry-formula Suc by force
  qed
  thus ?thesis unfolding carry-set-def by simp
qed

corollary count-carries-pow2-block-ones[simp]: count-carries (2^k) (2^k−1) = 0
  unfolding count-carries-def carry-set-pow2-block-ones by simp

lemma bit-set-add-shift:
  a < 2^k ⟹ bit-set (a + b * 2^k) = bit-set a ∪ ((+) k) ‘ bit-set b
  (is - ⟹ ?A = ?B)
proof
  assume assm: a < 2^k show ?A ⊆ ?B
  proof
    fix i assume i ∈ ?A
    hence i-def: (a + b * 2^k) ¡ i = 1
      using bit-set-def by simp
    { assume i < k
      hence a ¡ i = 1
        using aux2-digit-sum-repr[OF assm] i-def by (simp add: add.commute)
      hence i ∈ bit-set a
        using bit-set-def by simp }
    note 0 = this
    { assume i ≥ k
      hence (b * 2^k) ¡ i = 1
        using aux1-digit-lt-linear[OF assm] i-def by (simp add: add.commute)
      hence b ¡ (i−k) = 1
        using aux-digit-shift by (metis ‹k ≤ i› le-add-diff-inverse2)
      hence i ∈ ((+) k) ‘ bit-set b
```

```
      using ‹k ≤ i› bit-set-def image-iff by fastforce }
    note 1 = this
    from 0 1 show i ∈ ?B by force
  qed
next
  assume assm: a < 2^k show ?B ⊆ ?A
  proof
    fix i assume i ∈ ?B
    hence i-def: (a ¡ i = 1 ∧ i < k) ∨ (b ¡ (i−k) = 1 ∧ i ≥ k)
      using bit-set-subset-variant[OF assm] unfolding bit-set-def by auto
    thus i ∈ ?A
    proof
      assume a ¡ i = 1 ∧ i < k
      hence (a + b * 2^k) ¡ i = 1
        using aux2-digit-sum-repr[OF assm] by (simp add: add.commute)
      thus ?thesis
        using bit-set-def by simp
    next
      assume b ¡ (i−k) = 1 ∧ i ≥ k
      hence (a + b * 2^k) ¡ i = 1
        using aux1-digit-lt-linear[OF assm] aux-digit-shift
        by (metis add.commute le-add-diff-inverse2)
      thus ?thesis
        using bit-set-def by simp
    qed
  qed
qed

corollary count-bits-add-shift:
  a < 2^k ⟹ count-bits (a + b * 2^k) = count-bits a + count-bits b
proof −
  assume assm: a < 2^k

  have disjoint: bit-set a ∩ ((+) k) ‘ bit-set b = {}
  proof −
    { fix i assume i ∈ bit-set a ∩ ((+) k) ‘ bit-set b
      hence i < k ∧ i ≥ k
        using bit-set-subset-variant[OF assm] by blast
      hence False by auto }
    thus ?thesis by blast
  qed

  have count-bits (a + b * 2^k) = count-bits a + card (((+) k) ‘ bit-set b)
    unfolding count-bits-def using bit-set-add-shift[OF assm] disjoint
    by (simp add: bit-set-finite card-Un-Int)
  also have ... = count-bits a + count-bits b
    unfolding count-bits-def by (metis card-image inj-on-add)
  finally show ?thesis .
qed
```

corollary *count-bits-even*[*simp*]: *count-bits* (*2*∗*n*) = *count-bits n*
using *count-bits-add-shift count-bits-0*
by (*metis add-lessD1 comm-monoid-add-class.add-0 less-exp mult.commute power-one-right*)

corollary *count-bits-odd*[*simp*]: *count-bits* (*2*∗*n*+*1*) = *1* + *count-bits n*
using *count-bits-add-shift count-bits-1*
by (*metis add.commute less-exp mult.commute power-one-right*)

lemma *count-bits-digitwise*:
assumes *1* ≤ *k* **and** ∀ *i*<*q*. *f i* < *2*^*k*
shows *count-bits* (∑ *i*<*q*. *f i* ∗ (*2*^*k*) ^*i*) = (∑ *i*<*q*. *count-bits* (*f i*))
using *assms*
proof (*induct q*)
case *0* **show** *?case* **by** *simp*
next
case (*Suc q*)
have *0*: *1* < (*2*::*nat*) ^*k*
using *assms*(*1*) **by** (*meson dual-order.strict-trans2 less-exp*)
hence *bound*: (∑ *i*<*q*. *f i* ∗ (*2*^*k*) ^*i*) < *2*^(*k*∗*q*)
using *aux-digit-sum-bound*[*OF 0*] *Suc.prems*(*2*) *power-mult*
by (*metis* (*full-types*) *less-Suc-eq-le less-imp-le-nat*)

have *count-bits* (∑ *i*<*Suc q*. *f i* ∗ (*2*^*k*) ^*i*) =
count-bits ((∑ *i*<*q*. *f i* ∗ (*2*^*k*) ^*i*) + *f q* ∗ *2*^(*k*∗*q*))
by (*simp add*: *power-mult*)
also have ... = *count-bits* (∑ *i*<*q*. *f i* ∗ (*2*^*k*) ^*i*) + *count-bits* (*f q*)
using *count-bits-add-shift*[*OF bound*] **by** *simp*
also have ... = (∑ *i*<*Suc q*. *count-bits* (*f i*))
using *Suc* **by** *simp*
finally show *?case* **.**
qed

lemma *count-carries-count-bits*:
count-bits (*a*+*b*) + *count-carries a b* = *count-bits a* + *count-bits b*
proof −
obtain *k* **where** *k-def*: *a* + *b* < *2*^*k* **using** *less-exp* **by** *blast*

have *0*: (∑ *i*<*k*. *bin-carry a b* (*i*+*1*)) = (∑ *i*<*k*. *bin-carry a b i*)
proof −
have *carry-0*: *bin-carry a b 0* = *0*
using *bin-carry-def* **by** *auto*

have *carry-k*: *bin-carry a b k* = *0*
by (*metis* (*no-types*, *lifting*) *div-eq-0-iff add.commute add-lessD1*
bin-carry-def k-def mod-less)

have (∑ *i*<*k*. *bin-carry a b* (*i*+*1*)) = (∑ *i*=*1*..<*k*+*1*. *bin-carry a b i*)
by (*metis add-0 atLeast0LessThan sum.shift-bounds-nat-ivl*)

also have ... = *bin-carry a b 0* + (∑ *i*=*1*..<*k*. *bin-carry a b i*)+ *bin-carry a b k*
using *carry-0* **by** *auto*
also have ... = (∑ *i*<*k*. *bin-carry a b i*)
using *carry-k* **by** (*simp add*: *atLeast0LessThan carry-0 sum-shift-lb-Suc0-0-upt*)
finally show *?thesis* **.**
qed

have (*a* + *b*) ¡ *i* + *2* ∗ *bin-carry a b* (*i*+*1*) = *a* ¡ *i* + *b* ¡ *i* + *bin-carry a b i* **for** *i*
using *sum-carry-formula sum-digit-formula* **by** *simp*
hence (∑ *i*<*k*. (*a* + *b*) ¡ *i* + *2* ∗ *bin-carry a b* (*i*+*1*)) = (∑ *i*<*k*. *a* ¡ *i* + *b* ¡ *i* + *bin-carry a b i*)
using *sum.cong* **by** *presburger*
hence (∑ *i*<*k*. (*a* + *b*) ¡ *i*) + *2* ∗ (∑ *i*<*k*. *bin-carry a b* (*i*+*1*)) =
(∑ *i*<*k*. *a* ¡ *i*) + (∑ *i*<*k*. *b* ¡ *i*) + (∑ *i*<*k*. *bin-carry a b i*)
using *sum.distrib sum-distrib-left* **by** (*smt* (*z3*) *sum.cong*)
hence *count-bits* (*a*+*b*) + *2* ∗ *count-carries a b* = *count-bits a* + *count-bits b* + *count-carries a b*
using *0 count-bits-def-sum count-carries-def-sum k-def* **by** (*metis add.commute add-lessD1*)
thus *?thesis* **by** *simp*
qed

corollary *count-bits-add-le*: *count-bits* (*a*+*b*) ≤ *count-bits a* + *count-bits b*
using *count-carries-count-bits* **by** (*metis le-add1*)

{@*const count-carries*} can be defined in term of {@*const count-bits*}

corollary *count-carries-def-alt*:
count-carries a b = *count-bits a* + *count-bits b* − *count-bits* (*a*+*b*)
using *count-carries-count-bits* **by** (*metis diff-add-inverse*)

lemma *count-bits-sum-le*:
assumes *S-fin*: *finite S*
shows *count-bits* (*sum f S*) ≤ (∑ *i*∈*S*. *count-bits* (*f i*))
proof (*induct rule*: *finite-induct*[*OF S-fin*])
case *1* **show** *?case* **by** *simp*
next
case (*2 x S*)
have *count-bits* (*sum f* (*insert x S*)) = *count-bits* (*f x* + *sum f S*)
by (*simp add*: *2.hyps*(*1*) *2.hyps*(*2*))
also have ... ≤ *count-bits* (*f x*) + *count-bits* (*sum f S*)
using *count-bits-add-le* **by** *simp*
also have ... ≤ *count-bits* (*f x*) + (∑ *i*∈*S*. *count-bits* (*f i*))
by (*simp add*: *2.hyps*(*3*))
also have ... = (∑ *i*∈*insert x S*. *count-bits* (*f i*))
by (*simp add*: *2.hyps*(*1*) *2.hyps*(*2*))
finally show *?case* **.**
qed

lemma *aux1-carry-set-add-shift*:
a < 2^k ⟹ c < 2^k ⟹ i ≤ k ⟹ bin-carry (a+b∗2^k) (c+d∗2^k) i = bin-carry a c i
proof (*induct i*)
case *0* **thus** *?case* **unfolding** *bin-carry-def* **by** *simp*
next
case (*Suc i*)
have *bin-carry (a+b∗2^k) (c+d∗2^k) (Suc i) =*
((a+b∗2^k) ¡ i + (c+d∗2^k) ¡ i + bin-carry (a+b∗2^k) (c+d∗2^k) i) div 2
using *sum-carry-formula* **by** *simp*
also have *... = (a ¡ i + c ¡ i + bin-carry a c i) div 2*
using *Suc aux2-digit-sum-repr* **by** (*simp add*: *add.commute*)
also have *... = bin-carry a c (Suc i)*
using *sum-carry-formula* **by** *simp*
finally show *?case* **.**
qed

lemma *aux2-carry-set-add-shift*:
a < 2^k ⟹ c < 2^k ⟹ k ≤ i ⟹ bin-carry (a+b∗2^k) (c+d∗2^k) i ≥ bin-carry b d (i−k)
proof −
assume *assm1*: *a < 2^k* **and** *assm2*: *c < 2^k* **and** *i-def*: *k ≤ i*
show *?thesis*
proof (*induct rule*: *nat-induct-at-least*[*OF i-def*])
case *1* **show** *?case* **using** *bin-carry-def assm1 assm2* **by** *simp*
next
case (*2 i*)
have *bin-carry (a + b ∗ 2 ^ k) (c + d ∗ 2 ^ k) (Suc i) =*
((a+b∗2^k) ¡ i + (c+d∗2^k) ¡ i + bin-carry (a+b∗2^k) (c+d∗2^k) i) div 2
using *sum-carry-formula* **by** *simp*
also have *... ≥ (b ¡ (i−k) + d ¡ (i−k) + bin-carry b d (i−k)) div 2*
using *2 aux1-digit-lt-linear aux-digit-shift assm1 assm2*
by (*smt* (*verit*, *ccfv-SIG*) *add.commute add-le-mono div-le-mono le-add-diff-inverse2 order.refl*)
finally show *?case*
using *sum-carry-formula 2* **by** (*metis Suc-diff-le Suc-eq-plus1*)
qed
qed

lemma *aux3-carry-set-add-shift*:
a + c < 2^k ⟹ k ≤ i ⟹ bin-carry (a+b∗2^k) (c+d∗2^k) i = bin-carry b d (i−k)
proof −
assume *assm*: *a + c < 2^k*
assume *i-def*: *k ≤ i*
show *?thesis*
proof (*induct rule*: *nat-induct-at-least*[*OF i-def*])
case *1* **show** *?case* **using** *bin-carry-def assm* **by** *simp*

```
  next
    case (2 i)
    have bin-carry (a + b * 2 ^ k) (c + d * 2 ^ k) (Suc i) =
      ((a+b*2^k) ¡ i + (c+d*2^k) ¡ i + bin-carry (a+b*2^k) (c+d*2^k) i) div 2
      using sum-carry-formula by simp
    also have ... = (b ¡ (i−k) + d ¡ (i−k) + bin-carry b d (i−k)) div 2
      using 2 aux1-digit-lt-linear aux-digit-shift assm
      by (smt (verit, best) add.commute add-lessD1 le-add-diff-inverse2)
    also have ... = bin-carry b d (Suc i − k)
      using sum-carry-formula 2 by (metis Suc-diff-le Suc-eq-plus1)
    finally show ?case .
  qed
qed

lemma carry-set-add-shift:
  a < 2^k ⟹ c < 2^k ⟹ carry-set a c ∪ ((+) k) ‘ carry-set b d ⊆ carry-set
(a+b*2^k) (c+d*2^k)
  (is - ⟹ - ⟹ ?A1 ∪ ?A2 ⊆ ?B)
proof −
  assume assm1: a < 2^k and assm2: c < 2^k

  { fix i assume i-def: i ∈ ?A1
    have a + c < 2^(Suc k)
      using assm1 assm2 by auto
    hence i ≤ k
      using i-def carry-set-subset-variant by fastforce
    hence i ∈ ?B
      using carry-set-def i-def aux1-carry-set-add-shift[OF assm1 assm2] by auto
}
  note 0 = this

  { fix i assume i-def: i ∈ ?A2
    hence k ≤ i
      by auto
    hence i ∈ ?B
     using i-def aux2-carry-set-add-shift[OF assm1 assm2] unfolding carry-set-def
      by (smt (verit) add-diff-cancel-left' carry-bounded image-iff linorder-not-less
          mem-Collect-eq nat-less-le) }
  note 1 = this

  from 0 1 show ?thesis by auto
qed

corollary count-carries-add-shift:
  a < 2^k ⟹ c < 2^k ⟹ count-carries (a + b*2^k) (c + d*2^k)
                    ≥ count-carries a c + count-carries b d
proof −
  assume assm1: a < 2^k and assm2: c < 2^k
  have carry-set a c ∩ ((+) k) ‘ carry-set b d = {}
```

```
  proof −
    { fix i assume i-def: i ∈ carry-set a c ∩ ((+) k) ‘ carry-set b d
      have a + c < 2^(Suc k)
        using assm1 assm2 by auto
      hence i < Suc k ∧ i ≥ k
        using carry-set-subset-variant i-def by fastforce
      hence i = k
        by auto
      moreover have 0 ∉ carry-set b d
        unfolding carry-set-def bin-carry-def by simp
      ultimately have False
        using i-def by auto }
    thus ?thesis by auto
  qed
  hence card (carry-set a c ∪ ((+) k) ‘ carry-set b d) =
    card (carry-set a c) + card (carry-set b d)
    by (metis card-Un-disjoint card-image carry-set-finite finite-imageI inj-on-add)
  thus ?thesis
    unfolding count-carries-def using carry-set-add-shift[OF assm1 assm2]
    by (metis card-mono carry-set-finite)
qed

lemma carry-set-add-shift-no-overflow:
  a + c < 2^k ⟹ carry-set (a+b*2^k) (c+d*2^k) = carry-set a c ∪ ((+) k) ‘
carry-set b d
  (is - ⟹ ?A = ?B)
proof
  assume assm: a + c < 2^k show ?A ⊆ ?B
  proof
    fix i assume i-def: i ∈ ?A
    { assume i < k
      hence bin-carry a c i = 1
        using aux1-carry-set-add-shift assm i-def carry-set-def by simp
      hence i ∈ carry-set a c
        using carry-set-def by simp }
    note 0 = this
    { assume i ≥ k
      hence bin-carry b d (i−k) = 1
        using aux3-carry-set-add-shift[OF assm] i-def carry-set-def by simp
      hence i ∈ ((+) k) ‘ carry-set b d
        using carry-set-def ‹k ≤ i› image-iff by fastforce }
    note 1 = this
    from 0 1 show i ∈ ?B by force
  qed
next
  assume a + c < 2^k thus ?B ⊆ ?A
    using carry-set-add-shift
    by (meson add-lessD1 linorder-not-less trans-le-add2)
qed
```

corollary *count-carries-add-shift-no-overflow*:
a + c < 2^k ⟹ count-carries (a + b∗2^k) (c + d∗2^k) = count-carries a c + count-carries b d
proof −
assume *assm*: *a + c < 2^k*
have *carry-set a c ∩ ((+) k) ‘ carry-set b d = {}*
proof −
{ fix *i* **assume** *i ∈ carry-set a c ∩ ((+) k) ‘ carry-set b d*
hence *i < k ∧ i ≥ k*
using *carry-set-subset-variant*[*OF assm*] **by** *auto* **}**
thus *?thesis* **by** *auto*
qed
thus *?thesis*
unfolding *count-carries-def carry-set-add-shift-no-overflow*[*OF assm*]
by (*simp add*: *card-Un-disjoint card-image carry-set-finite*)
qed

corollary *count-carries-even-even*: *count-carries (2∗a) (2∗b) = count-carries a b*
using *count-carries-add-shift-no-overflow*[*of 0 0 1 a b*] **by** (*simp add*: *mult.commute*)

lemma *count-carries-digitwise*:
assumes *1 ≤ k* **and** *∀ i<q. f i < 2^k ∧ g i < 2^k*
shows *count-carries* (∑ *i<q. f i ∗ (2^k)^i*) (∑ *i<q. g i ∗ (2^k)^i*) ≥
(∑ *i<q. count-carries* (*f i*) (*g i*))
using *assms*
proof (*induct q*)
case *0* **thus** *?case* **by** *auto*
next
case (*Suc q*)
have *0:1 < (2::nat)^k*
using *Suc.prems*(*1*) *dual-order.strict-trans2 less-exp* **by** *blast*
have *bound*: (∑ *i<q. f i ∗ (2^k)^i*) *< 2^(k∗q) ∧* (∑ *i<q. g i ∗ (2^k)^i*) *< 2^(k∗q)*
using *aux-digit-sum-bound*[*OF 0*] *Suc.prems*(*2*) *power-mult* **by** (*metis* (*full-types*) *less-Suc-eq*)

have *count-carries* (∑ *i<q. f i ∗ (2^k)^i*) (∑ *i<q. g i ∗ (2^k)^i*) ≥
(∑ *i<q. count-carries* (*f i*) (*g i*))
using *Suc less-Suc-eq* **by** *presburger*
hence (∑ *i<Suc q. count-carries* (*f i*) (*g i*)) ≤
count-carries (∑ *i<q. f i ∗ (2^k)^i*) (∑ *i<q. g i ∗ (2^k)^i*) + *count-carries* (*f q*) (*g q*)
by *simp*
also have *... ≤ count-carries* ((∑ *i<q. f i ∗ (2^k)^i*) + *f q ∗ 2^(k∗q)*) ((∑ *i<q. g i ∗ (2^k)^i*) + *g q ∗ 2^(k∗q)*)
using *bound count-carries-add-shift* **by** *simp*
also have *... = count-carries* (∑ *i<Suc q. f i ∗ (2^k)^i*) (∑ *i<Suc q. g i ∗ (2^k)^i*)

```
    using power-mult by (smt (verit, del-insts) mult.commute sum.lessThan-Suc)
  finally show ?case .
qed

corollary count-carries-digitwise-specific:
  assumes 1 ≤ k and ∀ i<q. f i < 2^k ∧ g i < 2^k
  shows i < q ⟹ count-carries (∑ i<q. f i * (2^k)^i) (∑ i<q. g i * (2^k)^i)
≥
    count-carries (f i) (g i)
proof −
  assume i < q
  hence count-carries (f i) (g i) ≤ (∑ i<q. count-carries (f i) (g i))
    using member-le-sum by (metis (no-types, lifting) finite-lessThan lessThan-iff
zero-le)
  thus ?thesis
    using count-carries-digitwise[OF assms(1) assms(2)] by auto
qed

lemma count-carries-digitwise-no-overflow:
  assumes k ≥ 1 and ∀ i<q. f i + g i < 2^k
  shows count-carries (∑ i<q. f i * (2^k)^i) (∑ i<q. g i * (2^k)^i) =
    (∑ i<q. count-carries (f i) (g i))
using assms
proof (induct q)
  case 0 thus ?case by auto
next
  case (Suc q)
  have bound: (∑ i<q. f i * (2^k)^i) + (∑ i<q. g i * (2^k)^i) < 2^(k*q)
  proof −
    have 0: 1 < (2::nat)^k
      using Suc.prems(1) dual-order.strict-trans2 less-exp by blast

    have (∑ i<q. f i * (2^k)^i) + (∑ i<q. g i * (2^k)^i) = (∑ i<q. (f i + g i)
* (2^k)^i)
      by (simp add: sum.distrib algebra-simps)
    also have ... < (2^k)^q
      using aux-digit-sum-bound[OF 0] Suc.prems(2) less-Suc-eq by presburger
    finally show ?thesis
      unfolding power-mult .
  qed

  have (∑ i<Suc q. count-carries (f i) (g i)) =
    count-carries (∑ i<q. f i * (2^k)^i) (∑ i<q. g i * (2^k)^i) + count-carries
(f q) (g q)
    using Suc by simp
  also have ... = count-carries ((∑ i<q. f i * (2^k)^i) + f q * 2^(k*q))
    ((∑ i<q. g i * (2^k)^i) + g q * 2^(k*q))
    using bound count-carries-add-shift-no-overflow assms by auto
  also have ... = count-carries (∑ i<Suc q. f i * (2^k)^i) (∑ i<Suc q. g i *
```

$(2\hat{}k)\hat{}i)$
using *power-mult* **by** (*smt* (*verit*, *del-insts*) *mult.commute sum.lessThan-Suc*)
finally show *?case*
by *simp*
qed

lemma *carry-set-empty-iff*:
carry-set a b = {} ⟷ (∀ *i*. *a* ¡ *i* + *b* ¡ *i* ≤ *1*)
proof
assume *carry-set a b* = {}
hence *bin-carry a b i* = *0* **for** *i*
unfolding *carry-set-def* **using** *carry-bounded*
by (*smt* (*verit*) *Collect-empty-eq-bot div-le-dividend div-self empty-def nle-le*)
hence (*a* ¡ *i* + *b* ¡ *i*) *div 2* = *0* **for** *i*
using *sum-carry-formula* **by** (*metis add.right-neutral*)
thus ∀ *i*. *a* ¡ *i* + *b* ¡ *i* ≤ *1*
by (*metis Suc-1 add-self-div-2 div-le-mono nat-1-add-1 not-less-eq-eq not-one-le-zero*)
next
assume *assm*: ∀ *i*. *a* ¡ *i* + *b* ¡ *i* ≤ *1*
hence *bin-carry a b k* = *0* **for** *k*
using *carry-digit-impl* **by** (*metis Suc-1 Suc-n-not-le-n*)
thus *carry-set a b* = {}
unfolding *carry-set-def* **by** *simp*
qed

corollary *count-carries-zero-iff*:
count-carries a b = *0* ⟷ (∀ *i*. *a* ¡ *i* + *b* ¡ *i* ≤ *1*)
using *count-carries-def carry-set-empty-iff* **by** (*simp add*: *carry-set-finite*)

lemma *no-carry-no-overflow*:
assumes *a* < *2*^*k* **and** *b* < *2*^*k* **and** *count-carries a b* = *0*
shows *a* + *b* < *2*^*k*
proof −
have *carry-set a b* = {}
using *assms*(*3*) *count-carries-def carry-set-finite* **by** *simp*
hence *bin-carry a b i* = *0* **for** *i*
unfolding *carry-set-def* **using** *carry-bounded*
by (*smt* (*verit*) *Collect-empty-eq-bot div-le-dividend div-self empty-def nle-le*)
hence (*a mod 2*^*k* + *b mod 2*^*k*) *div 2*^*k* = *0*
using *bin-carry-def* **by** *simp*
hence (*a* + *b*) *div 2*^*k* = *0*
using *assms*(*1*−*2*) **by** *simp*
thus *a* + *b* < *2*^*k*
by (*simp add*: *div-eq-0-iff*)
qed

lemma *count-carries-divisibility-pow2*: *count-carries* (*2*^*k*−*1*) *x* = *0* ⟷ *2*^*k dvd*
x
proof

```
  assume count-carries (2^k−1) x = 0
  hence ((2^k−1) ¡ i) + (x ¡ i) ≤ 1 for i
    using count-carries-zero-iff by simp
  hence i < k ⟹ x ¡ i = 0 for i
    by (metis add-le-same-cancel2 bot-nat-0.extremum even-decr-exp-div-exp-iff'
        le-antisym mod2-eq-if not-less nth-bit-def)
  thus 2^k dvd x using aux2-digit-wise-equiv by presburger
next
  assume assm: 2^k dvd x
  { fix i assume i < k
    hence 2 * 2^i dvd x
      by (metis Suc-leI assm dvd-trans le-imp-power-dvd power-Suc)
    hence x ¡ i = 0
      unfolding nth-bit-def by auto }
  hence ((2^k−1) ¡ i) + (x ¡ i) ≤ 1 for i
    using nth-bit-bounded
   by (metis Nat.le-diff-conv2 add.right-neutral bot-nat-0.extremum even-decr-exp-div-exp-iff'

        mod-eq-0-iff-dvd not-less nth-bit-def)
  thus count-carries (2^k−1) x = 0
    using count-carries-zero-iff by simp
qed

lemma nth-digit-gen-power-series-general:
  assumes 1 < b and ∀k≤q. f k < b
  shows nth-digit (∑k=0..q. f k * b^k) i b = (if i ≤ q then f i else 0)
  (is nth-digit ?X - - = -)
proof (cases i ≤ q)
  case False
  have ?X < b^(Suc q)
    using aux-digit-sum-bound[OF assms(1)] assms(2)
   by (metis (full-types) Suc-leI atLeast0AtMost lessThan-Suc-atMost verit-comp-simplify1(3))
  thus ?thesis
    using False unfolding nth-digit-def
    by (metis (no-types, lifting) assms(1) div-less dual-order.strict-trans2
      linordered-nonzero-semiring-class.zero-le-one mod-less not-less-eq-eq power-increasing-iff

        verit-comp-simplify1(3))
next
  case True

  have split0: ?X = (∑k<i+1. f k * b^k) + (∑k<q−i. f (k+i+1) * b^k) *
b^(i+1)
  proof −
    have (∑k=i+1..q. f k * b^k) = (∑k=Suc i..<Suc q. f k * b^k)
      using Suc-eq-plus1 atLeastLessThanSuc-atLeastAtMost by presburger
    also have ... = (∑k=0 + Suc i..<(Suc q − Suc i) + Suc i. f k * b^k)
      by (simp add: True)
    also have ... = (∑k=0..<(Suc q − Suc i). f (k + Suc i) * b^(k + Suc i))
```

```
    using sum.shift-bounds-nat-ivl by blast
  also have ... = (∑k<q−i. f (k+i+1) * b^(k+i+1))
    using atLeast0LessThan by force
  also have ... = (∑k<q−i. f (k+i+1) * b^k * b^(i+1))
    by (simp add: power-add algebra-simps)
  finally have rewrite: (∑k=i+1..q. f k * b^k) = (∑k<q−i. f (k+i+1) *
b^k) * b^(i+1)
    using sum-distrib-right by (smt (verit) sum.cong)

  have ?X = (∑k=0..i. f k * b^k) + (∑k=i+1..q. f k * b^k)
    using sum.ub-add-nat
    by (smt (verit, ccfv-threshold) True bot-nat-0.extremum nat-le-iff-add)
  thus ?thesis
    using rewrite by (metis Suc-eq-plus1 atLeast0AtMost lessThan-Suc-atMost)
 qed
 have bound0: (∑k<i+1. f k * b^k) < b^(i+1)
  using aux-digit-sum-bound[OF assms(1)] assms(2) True
  using Suc-eq-plus1 le-trans less-Suc-eq-le by presburger
 have 0: nth-digit ?X i b = nth-digit (∑k<i+1. f k * b^k) i b
  (is - = nth-digit ?Y - -)
  using aux2-digit-gen-sum-repr[OF bound0] unfolding split0
  by (metis (no-types, lifting) add.commute less-add-same-cancel1 zero-less-one)


 have split1: ?Y = (∑k<i. f k * b^k) + f i * b^i
  by simp
 have bound1: (∑k<i. f k * b^k) < b^i
  using aux-digit-sum-bound[OF assms(1)] assms(2) True by auto
 have 1: nth-digit ?Y i b = nth-digit (f i * b^i) i b
  using aux3-digit-gen-sum-repr[OF bound1 assms(1)] unfolding split1
  by (simp add: add.commute)

 have 2: nth-digit (f i * b^i) i b = f i
  unfolding nth-digit-def using assms(2) True by auto

 show ?thesis
  using 0 1 2 True by simp
qed


lemma aux-count-bits-multiplicity:
 count-bits (Suc x) + multiplicity 2 (Suc x) = count-bits x + 1
proof (induct x rule: Bit-Operations.nat-bit-induct)
 case zero thus ?case using count-bits-1 by auto
next
 case (even x)
 have ¬ 2 dvd Suc (2*x)
  by simp
 thus ?case
```

```
  using count-bits-even count-bits-odd by (simp add: multiplicity-eq-zero-iff)
next
 case (odd x)
 have multiplicity 2 (2 * Suc x) = multiplicity 2 (Suc x) + 1
  by (metis Suc-1 Suc-eq-plus1 multiplicity-times-same nat.simps(3) odd-one)
 thus ?case
  using count-bits-even count-bits-odd
  by (metis add.commute add-2-eq-Suc add-Suc-right mult-Suc-right odd plus-1-eq-Suc)
qed

lemma count-bits-multiplicity:
 count-bits x = multiplicity 2 (2*x choose x)
proof (induct x)
 case 0
 then show ?case
  using One-nat-def binomial-n-0 multiplicity-one-nat count-bits-0 by presburger
next
 case (Suc x)

 have p: prime-elem (2::nat)
  using prime-elem-nat-iff two-is-prime-nat by presburger
 have e: (2 * Suc x choose Suc x) * Suc x * Suc x = (2*x choose x) * 2 * Suc x
* (2*x+1)
  by (smt (z3) Suc-times-binomial-add add.right-neutral add-2-eq-Suc add-Suc-right

      add-diff-cancel-left' binomial-absorption mult.commute mult.left-commute
mult-2
      mult-Suc-right plus-1-eq-Suc)

 let ?m = multiplicity (2::nat)

 have ¬ (2 dvd (2*x+1)) by simp
 hence ?m ((2 * Suc x choose Suc x) * Suc x * Suc x) = ?m ((2*x choose x) *
2 * (Suc x))
  by (metis (no-types, lifting) e mult.commute multiplicity-prime-elem-times-other
p)
 hence ?m ((2 * Suc x choose Suc x) * Suc x) = ?m ((2*x choose x) * 2)
  using prime-elem-multiplicity-mult-distrib[OF p]
  by (metis add-is-0 add-right-cancel e mult-is-0 nat.simps(3) zero-neq-one)
 hence ?m (2 * Suc x choose Suc x) + ?m (Suc x) = ?m (2*x choose x) + ?m 2
  using prime-elem-multiplicity-mult-distrib[OF p] binomial-eq-0-iff
   by (metis less-numeral-extra(3) mult-2 nat.simps(3) nat-zero-less-power-iff
not-add-less1
      zero-power2)
 hence ?m (2 * Suc x choose Suc x) + ?m (Suc x) = count-bits x + 1
  using multiplicity-prime[OF p] Suc by metis
 hence ?m (2 * Suc x choose Suc x) = count-bits (Suc x)
  using aux-count-bits-multiplicity by (metis add-diff-cancel-right')
 thus ?case by simp
```

qed

corollary *count-bits-divibility-binomial*:
2^k dvd (2∗x choose x) ⟷ k ≤ count-bits x
using *count-bits-multiplicity* **by** (*simp add*: *power-dvd-iff-le-multiplicity*)

end
theory *Utils*
imports *Main*
begin

definition *is-square*::*int*⇒*bool* **where**
is-square n = (∃ *k*. *n* = *k*^*2*)

definition *is-power2*::*int*⇒*bool* **where**
is-power2 x ≡ (∃ *n*::*nat*. *x* = *2*^*n*)

lemma *is-power2-ge1*: *is-power2 x* ⟹ *1* ≤ *x*
unfolding *is-power2-def* **by** *force*

lemma *is-power2-mult*[*simp*]: *is-power2 x* ⟹ *is-power2 y* ⟹ *is-power2* (*x* ∗ *y*)
unfolding *is-power2-def* **by** (*metis power-add*)

lemma *is-power2-pow*[*simp*]: *is-power2 x* ⟹ *is-power2* (*x*^*n*)
unfolding *is-power2-def* **by** (*metis power-mult*)

lemma *is-power2-1*[*simp*]: *is-power2 1*
unfolding *is-power2-def* **by** (*metis power-0*)
lemma *is-power2-2*[*simp*]: *is-power2 2*
unfolding *is-power2-def* **by** (*metis power-one-right*)
lemma *is-power2-4*[*simp*]: *is-power2 4*
unfolding *is-power2-def* **by** (*metis mult-2-right numeral-Bit0 power2-eq-square*)

lemma *is-power2-div2*: *is-power2 x* ⟹ *2* ≤ *x* ⟹ *is-power2* (*x div 2*)
unfolding *is-power2-def*
by (*smt* (*verit*, *ccfv-threshold*) *bot-nat-0.not-eq-extremum int-power-div-base power-0*)

lemma *digit-repr-lt*:
fixes *q* :: *nat*
fixes *b* :: *int*
assumes *b* > *1*
assumes ∀ *k*. *f k* < *b*
shows (∑ *k* = *0*..*q*. *f k* ∗ *b* ^ *k*) < *b*^(*Suc q*)
proof (*induct q*)
case *0*
then show *?case* **using** *assms*(*2*) **by** *auto*
next

case (*Suc q*)
have *le-bound*: $f\ k \leq b - 1$ **for** k
using *assms*(*2*) **by** *auto*
have $(\sum k = 0..q.\ f\ k * b\ \hat{}\ k) + f\ (Suc\ q) * (b * b\ \hat{}\ q)$
$\leq (\sum k = 0..q.\ f\ k * b\ \hat{}\ k) + (b-1) * (b * b\ \hat{}\ q)$ **using** *le-bound assms*(*1*)
by *force*
also have ... $< b\hat{}(Suc\ q) + (b-1) * (b * b\ \hat{}\ q)$ **using** *Suc* **by** *auto*
also have ... $\leq b * b * b\hat{}q$
by *simp* (*metis* (*mono-tags*, *opaque-lifting*) *add.commute diff-add-cancel distrib-left linorder-not-less mult.left-commute mult.right-neutral order-less-imp-le*)
finally show *?case* **using** *Suc* **by** *auto*
qed

end
theory *Tau-Reduction*
imports *Bit-Counting Utils*
begin

2.2 Expressing the bit counting function with a binomial coefficient

locale *Tau-Reduction* =
fixes N::*nat* **and** S::*nat* **and** T::*nat*
assumes *HN*: *is-power2* (*int N*)
and *HS*: $S < N$
and *HT*: $T < N$
begin

abbreviation σ **where** $\sigma \equiv$ *count-bits*
abbreviation τ **where** $\tau \equiv$ *count-carries*

definition R **where** $R \equiv (S+T+1)*N + T + 1$

We prove an identity on natural numbers. To make it more tractable we transpose it to integers.

lemma *rewrite-R*:
$N\hat{}2 * (S+T) + N * (N-1-S) + (N-1-T) = (N-1) * R$
proof −
let *?iS* = *int S* **and** *?iT* = *int T* **and** *?iN* = *int N*

have *int* $(N*N * (S+T) + N * (N-1-S) + (N-1-T)) =$
*?iN***?iN* $*$ (*?iS*+*?iT*) + *?iN* $*$ (*?iN*−1−*?iS*) + (*?iN*−1−*?iT*)
using *HS HT int-ops* **by** *auto*
also have ... = (*?iN*−1)*((*?iS*+*?iT*+1) $*$ *?iN* + *?iT* + 1)
by *algebra*
also have ... = (*?iN* − 1) $*$ *int* ((*S*+*T*+1) $*$ *N* + *T* + 1)
using *int-ops* **by** *presburger*
also have ... = *int* ((*N*−1) $*$ ((*S*+*T*+1) $*$ *N* + *T* + 1))
using *HS HT int-ops*

by (*metis* (*mono-tags*, *opaque-lifting*) *One-nat-def R-def Suc-leI diff-is-0-eq not-gr-zero*
of-nat-diff verit-comp-simplify1(*3*) *zero-diff*)
finally show *?thesis* **using** *R-def* **by** (*metis nat-int-comparison*(*1*) *power2-eq-square*)
qed

This is a direct consequence of the properties of sigma and tau.

Lemma 1.4 in the article

lemma *tau-as-binomial-coefficient*:
$\tau\ S\ T = 0 \longleftrightarrow N\hat{}2\ dvd\ 2 * (N-1) * R\ choose\ (N-1) * R$
proof −
from *HN* **obtain** *n* **where** *n-def*: $N = 2\hat{}n$ **unfolding** *is-power2-def* **by** *auto*

have *1*: $N-1-S < N$ **using** *HS* **by** *auto*
have *2*: $N-1-T < N$ **using** *HT* **by** *auto*

have $N * (N-1-S) + (N-1-T) < N * (N-1) + N$
using *1 2*
by (*meson add-mono-thms-linordered-field*(*4*) *diff-le-self mult-le-mono2*)
also have $... = N\hat{}2$
by (*simp add*: *algebra-simps numeral-eq-Suc*)
finally have *3*: $N * (N-1-S) + (N-1-T) < N\hat{}2$ **.**

have $\tau\ S\ T = 0 \longleftrightarrow \sigma\ S + \sigma\ T - \sigma\ (S+T) = 0$
using *count-carries-def-alt* **by** *simp*
also have $... \longleftrightarrow \sigma\ S + \sigma\ T \leq \sigma\ (S+T)$
by *auto*
also have $... \longleftrightarrow (\sigma\ S + \sigma\ (N-1-S)) + (\sigma\ T + \sigma\ (N-1-T)) \leq \sigma\ (S+T) + \sigma\ (N-1-S) + \sigma\ (N-1-T)$
by *auto*
also have $... \longleftrightarrow 2 * \sigma\ (N-1) \leq \sigma\ (S+T) + \sigma\ (N-1-T) + \sigma\ (N-1-S)$
using *HS HT count-bits-complement count-bits-bounded n-def* **by** *auto*
also have $... \longleftrightarrow 2 * n \leq \sigma\ (S+T) + \sigma\ (N-1-T) + \sigma\ (N-1-S)$
using *count-bits-block-ones n-def* **by** *simp*
also have $... \longleftrightarrow 2 * n \leq \sigma\ (S+T) + \sigma\ (N * (N-1-S) + (N-1-T))$
using *count-bits-add-shift 2 n-def*
by (*smt* (*verit*, *best*) *add.commute add.left-commute mult.commute*)
also have $... \longleftrightarrow 2 * n \leq \sigma\ (N\hat{}2 * (S+T) + N * (N-1-S) + (N-1-T))$
using *count-bits-add-shift 3 n-def*
by (*smt* (*verit*) *add.assoc add.commute mult.commute power-mult*)
also have $... \longleftrightarrow 2 * n \leq \sigma\ ((N-1) * R)$
using *rewrite-R HS HT* **by** *simp*
also have $... \longleftrightarrow N\hat{}2\ dvd\ 2 * (N-1) * R\ choose\ (N-1) * R$
using *count-bits-divibility-binomial n-def*
by (*metis distrib-right mult-2 mult-2-right power-mult*)
finally show *?thesis* **.**
qed

end

end
theory *Masking*
imports *Complex-Main Bit-Counting Utils*
begin

2.3 Masking

abbreviation σ **where** $\sigma \equiv$ *count-bits*
abbreviation τ **where** $\tau \equiv$ *count-carries*

locale *masking-lemma* =
fixes δ ν $\mathcal{B}$ b C :: *nat*
assumes δ*-pos*: $\delta > 0$
and $\mathcal{B}$*-power2*: *is-power2* (*int* $\mathcal{B}$)
and *b-power2*: *is-power2* (*int* b)
and $\mathcal{B}$*-ge-2*: $2 \leq \mathcal{B}$
and *b-le-*$\mathcal{B}$: $b \leq \mathcal{B}$
and *C-lower-bound*: $C \geq b * \mathcal{B}\hat{}(\delta+1)\hat{}\nu$
and *C-upper-bound*: $C \leq \mathcal{B} * \mathcal{B}\hat{}(\delta+1)\hat{}\nu$
begin

definition n :: *nat* $\Rightarrow$ *nat* **where** $n\ j = (\delta+1)\hat{}j$

definition m::*nat* $\Rightarrow$ *nat* **where**
$m\ j =$ (*if* $j \in n$ ‘ $\{1..\nu\}$ *then* $\mathcal{B} - b$ *else* $\mathcal{B} - 1$)

mask(b, $\mathcal{B}$, δ, ν)

definition M::*nat* **where** $M = (\sum j{=}0..n\ \nu.\ m\ j * \mathcal{B}\hat{}j)$

lemma *b-ge-1*: $1 \leq b$
using *b-power2 is-power2-ge1* **by** *force*

lemma *b-dvd-*$\mathcal{B}$: b *dvd* $\mathcal{B}$
using *b-le-*$\mathcal{B}$ *b-power2* $\mathcal{B}$*-power2 is-power2-def le-imp-power-dvd* **by** *fastforce*

lemma *n-inj-on*: *inj-on n A*
unfolding *n-def inj-on-def* **by** (*simp add*: δ*-pos*)

lemma *direct-g-bound*:
assumes *z-bound*: $\forall\, i.\ z\ i < b$
and *g-code*: $g = (\sum i{=}1..\nu.\ z\ i * \mathcal{B}\hat{}(n\ i))$
shows $g < C$
proof (*cases* $b > 1$)
case *False*
hence $b = 1$ **using** *b-ge-1* **by** *simp*
hence $g = 0$ **using** *z-bound g-code* **by** *simp*
thus *?thesis* **using** *C-lower-bound b-ge-1* $\mathcal{B}$*-ge-2* **using** *zero-less-iff-neq-zero* **by** *fastforce*

next
case *True*

define z' **where** $z' = (\lambda j.\ \textit{if } j \in \textit{range } n \textit{ then } z\ (\textit{inv } n\ j) \textit{ else } 0)$

have $z'\textit{-bound}$: $z'\ j \leq b - 1$ **for** j
unfolding $z'\textit{-def}$ **using** *z-bound b-ge-1 less-Suc-eq-le* **by** *fastforce*

have $g = (\sum i{\in}\{1..\nu\}.\ z'\ (n\ i) * \mathcal{B}\hat{}(n\ i))$
using *sum.cong* $z'\textit{-def}$ *n-inj-on* **by** (*simp add*: *g-code*)
also have $... = (\sum j{\in}n`\{1..\nu\}.\ z'\ j * \mathcal{B}\hat{}j)$
using *n-inj-on*[*of* $\{1..\nu\}$] **by** (*simp add*: *sum.reindex-cong*)
also have $... \leq (\sum j{\in}\{0..n\ \nu\}.\ z'\ j * \mathcal{B}\hat{}j)$
proof −
have $n\ `\ \{1..\nu\} \subseteq \{0..n\ \nu\}$
unfolding *n-def*
by (*metis* (*no-types*, *opaque-lifting*) *One-nat-def Suc-eq-plus1 Suc-le-mono atLeast0AtMost*
atLeastAtMost-iff image-subset-iff power-increasing zero-le)
thus *?thesis* **using** *sum-mono2* **by** *blast*
qed
also have $... \leq (\sum j{=}0..n\ \nu.\ (b - 1) * \mathcal{B}\hat{}j)$
using $z'\textit{-bound}$ *sum-mono* **by** (*metis* (*no-types*, *lifting*) *mult-le-cancel2*)
finally have $g \leq (b{-}1) * (\sum j{<}(n\ \nu){+}1.\ \mathcal{B}\hat{}j)$
using *sum-distrib-left* **by** (*metis Suc-eq-plus1 atMost-atLeast0 lessThan-Suc-atMost*)

hence $(\mathcal{B}{-}1) * g \leq (\mathcal{B}{-}1) * (\sum j{<}(n\ \nu){+}1.\ \mathcal{B}\hat{}j) * (b{-}1)$
by (*simp add*: *mult.commute*)
also have $... = (\mathcal{B}\hat{}(n\ \nu + 1) - 1) * (b{-}1)$
using *aux-geometric-sum* $\mathcal{B}\textit{-ge-2}$
by (*metis le-imp-less-Suc nat-add-left-cancel-less one-add-one plus-1-eq-Suc*)
also have $... < (b{-}1) * \mathcal{B} * \mathcal{B}\hat{}(n\ \nu)$
using *True* $\mathcal{B}\textit{-ge-2}$ **by** *auto*
also have $... \leq (\mathcal{B}{-}1) * b * \mathcal{B}\hat{}(n\ \nu)$
using $b\textit{-le-}\mathcal{B}$ $\mathcal{B}\textit{-ge-2}$
by (*smt* (*verit*) *diff-le-mono2 diff-mult-distrib mult.commute mult-le-mono1*)
finally have $g < b * \mathcal{B}\hat{}(n\ \nu)$
by *auto*
thus *?thesis*
using *C-lower-bound n-def* **by** *simp*
qed

lemma *direct-tau-zero*:
assumes *z-bound*: $\forall\, i.\ z\ i < b$
and *g-code*: $g = (\sum i{=}1..\nu.\ z\ i * \mathcal{B}\hat{}(n\ i))$
shows $\tau\ g\ M = 0$
proof −
define z' **where** $z' = (\lambda j.\ \textit{if } j \in n\ `\ \{1..\nu\} \textit{ then } z\ (\textit{inv } n\ j) \textit{ else } 0)$

```
  have g = (∑ i∈{1..ν}. z' (n i) * ℬ^(n i))
    using sum.cong z'-def n-inj-on by (simp add: g-code)
  also have ... = (∑ j∈n`{1..ν}. z' j * ℬ^j)
    using n-inj-on[of {1..ν}] by (simp add: sum.reindex-cong)
  also have ... = (∑ j∈{0..n ν}. z' j * ℬ^j)
  proof −
    have n ` {1..ν} ⊆ {0..n ν}
      unfolding n-def
       by (metis (no-types, opaque-lifting) One-nat-def Suc-eq-plus1 Suc-le-mono
atLeast0AtMost
          atLeastAtMost-iff image-subset-iff power-increasing zero-le)
    moreover have j ∈ {0..n ν} − n ` {1..ν} ⟹ z' j * ℬ^j = 0 for j
      unfolding z'-def by auto
    ultimately show ?thesis
      using sum.mono-neutral-left by (simp add: sum.subset-diff)
  qed
  finally have g-sum: g = (∑ j<n ν + 1. z' j * ℬ^j)
    using Suc-eq-plus1 atLeast0AtMost lessThan-Suc-atMost by presburger

  have zm-bound: z' j + m j < ℬ for j
    unfolding z'-def m-def using b-le-ℬ z-bound ℬ-ge-2
   by (smt (verit, best) add-diff-inverse-nat add-le-cancel-right b-ge-1 dual-order.strict-trans2

       linorder-not-less zero-less-one)

  have tau-digits: τ (z' j) (m j) = 0 for j
  proof (cases j ∈ n ` {1.. ν})
    case False thus ?thesis unfolding z'-def by simp
  next
    obtain k where k-def: b = 2^k using b-power2 is-power2-def by auto

    case True
    hence τ (z' j) (m j) = τ (z (inv n j)) (ℬ − b)
      using z'-def m-def by simp
    also have ... = τ (z (inv n j) + 0 * b) (0 + (ℬ div b − 1) * b)
      using b-dvd-ℬ by (simp add: mult.commute right-diff-distrib')
    also have ... = τ (z (inv n j)) 0 + τ 0 (ℬ div b − 1)
      using count-carries-add-shift-no-overflow k-def z-bound
      by (metis (no-types, lifting) add.right-neutral)
    also have ... = 0
      by simp
    finally show ?thesis .
  qed

  obtain k where k-def: ℬ = 2^k
    using ℬ-power2 is-power2-def by auto
  have k-ge-1: 1 ≤ k
    using k-def ℬ-ge-2
     by (metis Suc-eq-plus1 Suc-leD antisym div-le-dividend div-self nat-1-add-1
```

```
numeral-eq-one-iff
        order-refl power-0 semiring-norm(85))

  have τ g M = τ (∑j<n ν + 1. z' j * ℬ^j) (∑j<n ν + 1. m j * ℬ^j)
   using g-sum M-def Suc-eq-plus1 atLeast0AtMost lessThan-Suc-atMost by presburger
  also have ... = (∑j<n ν + 1. τ (z' j) (m j))
    using count-carries-digitwise-no-overflow[OF k-ge-1] k-def zm-bound by blast
  also have ... = 0
    using tau-digits by simp
  finally show ?thesis .
qed

lemma reverse-impl:
  assumes tau-zero: τ g M = 0
      and g-bound-C: g < C
  shows ∃z. (∀i. z i < b) ∧ g = (∑i=1..ν. z i * ℬ^(n i))
proof −
  have g-bound-ℬ: g < ℬ^(n ν + 1)
    using C-upper-bound g-bound-C n-def by simp

  have g-digit: j > n ν ⟹ nth-digit g j ℬ = 0 for j
    using nth-digit-def ℬ-ge-2 g-bound-ℬ
   by (metis Suc-eq-plus1 div-less linorder-not-less mod-less nat-1-add-1 not-less-eq-eq

        order-less-le-trans power-increasing-iff zero-less-two)
  have g-sum: g = (∑j<n ν + 1. nth-digit g j ℬ * ℬ^j)
    using digit-gen-sum-repr[OF g-bound-ℬ] ℬ-ge-2 by auto

  have digit-bound: j ∈ {0..n ν} ⟹ nth-digit g j ℬ + m j < ℬ for j
  proof −
    obtain k where k-def: ℬ = 2^k
      using ℬ-power2 is-power2-def by auto
    have k-ge-1: 1 ≤ k
      using k-def ℬ-ge-2
       by (metis Suc-eq-plus1 Suc-leD antisym div-le-dividend div-self nat-1-add-1
numeral-eq-one-iff
          order-refl power-0 semiring-norm(85))
    assume j ∈ {0..n ν}
    hence j < n ν + 1
      by auto
    moreover have 0: m j < ℬ ∧ nth-digit g j ℬ < ℬ for j
      unfolding m-def nth-digit-def using ℬ-ge-2 b-ge-1 by fastforce
    ultimately have τ (∑j<n ν + 1. nth-digit g j ℬ * ℬ^j) (∑j<n ν + 1. m j
* ℬ^j) ≥
      τ (nth-digit g j ℬ) (m j)
       using count-carries-digitwise-specific[OF k-ge-1 - ‹j < n ν + 1›] k-def by
force
    hence τ (nth-digit g j ℬ) (m j) = 0
      using tau-zero g-sum M-def
```

```
    by (metis Suc-eq-plus1 atLeast0AtMost le-zero-eq lessThan-Suc-atMost)
  thus ?thesis
    using no-carry-no-overflow k-def 0 by auto
qed

define z where z = (λj. nth-digit g (n j) B)

have digit-zero: j ∉ n ‘ {1..ν} ⟹ nth-digit g j B = 0 for j
proof (cases j ∈ {0..n ν})
  case False thus ?thesis using g-digit by auto
next
  case True
  assume assm: j ∉ n ‘ {1..ν}
  have nth-digit g j B + m j < B
    using True digit-bound by simp
  hence nth-digit g j B + (B − 1) < B
    using m-def assm by auto
  thus ?thesis
    using nth-digit-def by simp
qed

have z-bound: z i < b for i
proof (cases i ∈ {1..ν})
  case False
  hence (n i) ∉ n ‘ {1..ν}
    using n-inj-on by blast
  hence nth-digit g (n i) B = 0
    unfolding z-def using digit-zero by auto
  thus ?thesis
    using b-ge-1 z-def by auto
next
  case True
  hence n i ∈ {0..n ν}
    unfolding n-def by (simp add: power-increasing)
  hence z i + m (n i) < B
    using digit-bound z-def by auto
  hence z i + (B − b) < B
    using m-def by (metis True imageI)
  thus z i < b
    using b-le-B by auto
qed

have g-sum-z: g = (∑ i=1..ν. z i * B ^ (n i))
proof −
  have g = (∑ j∈{0..n ν}. nth-digit g j B * B ^ j)
    using g-sum by (metis Suc-eq-plus1 atLeast0AtMost lessThan-Suc-atMost)
  also have ... = (∑ j∈n‘{1..ν}. nth-digit g j B * B ^ j)
  proof −
    have n ‘ {1..ν} ⊆ {0..n ν}
```

```
    unfolding n-def
    by (metis (no-types, opaque-lifting) One-nat-def Suc-eq-plus1 Suc-le-mono
atLeast0AtMost
        atLeastAtMost-iff image-subset-iff power-increasing zero-le)
  moreover have j ∈ {0..n ν} − n ‘ {1..ν} ⟹ nth-digit g j ℬ = ℬ^j = 0
for j
    using digit-zero by auto
  ultimately show ?thesis
  using sum.mono-neutral-right by (metis (no-types, lifting) finite-atLeastAtMost)
 qed
 also have ... = (∑ i∈{1..ν}. nth-digit g (n i) ℬ * ℬ^(n i))
  using n-inj-on sum.reindex by auto
 finally show ?thesis
  using z-def by simp
qed

show ?thesis
 using z-bound g-sum-z by auto
qed

lemma masking-lemma:
(∃ z. (∀ i. z i < b) ∧ g = (∑ i=1..ν. z i * ℬ^(n i))) ⟷ (g < C ∧ τ g M = 0)
using reverse-impl direct-tau-zero direct-g-bound by auto

end

end
theory Multinomial
imports Main HOL−Library.Disjoint-Sets
begin
```

The factorial of a list of natural numbers is the product of all factorials

```
fun mfact′ :: nat list ⇒ nat where
mfact′ [] = 1 |
mfact′ (i # is) = (fact i :: nat) * mfact′ is

definition mfact :: nat list ⇒ nat where
mfact i = (∏ s<length i. fact (i ! s) :: nat)

lemma mfact-Nil[simp]: mfact [] = 1
by (simp add: mfact-def)

lemma mfact-Cons[simp]: mfact (i # is) = fact i * mfact is
proof −
have (∏ s<length is + 1. fact ((i # is) ! s)) =
 fact i * (∏ s<length is. fact (is ! s)) proof −
 have (∏ s<length is + 1. fact ((i # is) ! s)) =
  (∏ s<Suc (length is). fact ((i # is) ! s)) by auto
 also have ... = fact ((i # is) ! 0) *
```

```
    (∏ s<length is. fact ((i # is) ! (Suc s)))
    by (rule prod.lessThan-Suc-shift)
  also have ... = fact i * (∏ s<length is. fact (is ! s)) by auto
  finally show ?thesis .
 qed
 thus ?thesis unfolding mfact-def by auto
qed

lemma mfact'-equiv: mfact' = mfact proof −
 have mfact' xs = mfact xs for xs by (induction xs; auto)
 thus ?thesis by auto
qed
```

The "multi-power" of a list of natural numbers.

```
fun mpow' :: 'a::comm-semiring-1 list ⇒ nat list ⇒ 'a where
 mpow' [] ns = 1 |
 mpow' ns [] = 1 |
 mpow' (x # xs) (n # ns) = x ^ n * mpow' xs ns

definition mpow :: 'a::comm-semiring-1 list ⇒ nat list ⇒ 'a where
 mpow xs ns = (∏ i<min (length xs) (length ns). (xs ! i) ^ (ns ! i))

lemma mpow-Nil-any[simp]: mpow [] ns = 1
 by (simp add: mpow-def)

lemma mpow-any-Nil[simp]: mpow xs [] = 1
 by (simp add: mpow-def)

lemma mpow-Cons[simp]: mpow (x # xs) (n # ns) = (x ^ n) * (mpow xs ns)
proof −
 let ?l = min (length xs) (length ns)
 have (∏ i<?l. (x # xs) ! i ^ (n # ns) ! i) * (x # xs) ! ?l ^ (n # ns) ! ?l =
   x ^ n * (∏ i<?l. xs ! i ^ ns ! i) proof cases
   assume ?l = 0
   thus ?thesis by auto
 next
   assume ?l ≠ 0
   hence 0: Suc (?l − 1) = ?l by auto
   have (∏ i<?l. (x # xs) ! i ^ (n # ns) ! i) * (x # xs) ! ?l ^ (n # ns) ! ?l =
     (∏ i<(Suc (?l − 1)). (x # xs) ! i ^ (n # ns) ! i) *
     (x # xs) ! ?l ^ (n # ns) ! ?l
     unfolding 0 by auto
   also have ... = (x # xs) ! 0 ^ (n # ns) ! 0 *
     (∏ i<(?l − 1). (x # xs) ! (Suc i) ^ (n # ns) ! (Suc i)) *
     (x # xs) ! ?l ^ (n # ns) ! ?l unfolding prod.lessThan-Suc-shift by auto
   also have ... = x ^ n *
     (∏ i<(?l − 1). xs ! i ^ ns ! i) * (x # xs) ! ?l ^ (n # ns) ! ?l by auto
   also have ... = x ^ n *
     (∏ i<(?l − 1). xs ! i ^ ns ! i) *
```

```
    (x # xs) ! (Suc (?l − 1)) ^ (n # ns) ! (Suc (?l − 1)) unfolding 0 by auto
  also have ... = x ^ n *
    ((∏ i<(?l − 1). xs ! i ^ ns ! i) *
    (x # xs) ! (Suc (?l − 1)) ^ (n # ns) ! (Suc (?l − 1)))
    by (simp add: algebra-simps)
  also have ... = x ^ n * (∏ i<(Suc (?l − 1)). xs ! i ^ ns ! i)
    unfolding prod.atMost-Suc[symmetric] by auto
  also have ... = x ^ n * (∏ i<?l. xs ! i ^ ns ! i) unfolding 0 by auto
  finally show ?thesis .
 qed
 thus ?thesis unfolding mpow-def by auto
qed

lemma mpow'-equiv: mpow' = mpow proof −
 have mpow' (xs :: ('a::comm-semiring-1 list)) ns = mpow xs ns for xs ns
  by (induction xs ns rule: mpow'.induct; auto)
 thus ?thesis by auto
qed

lemma multinomial'-dvd: mfact ks dvd fact (sum-list ks)
proof (induction ks)
 case Nil
 show ?case by auto
next
 case (Cons k ks)
 hence mfact ks dvd fact (sum-list ks) .
 hence fact k * mfact ks dvd (fact k :: nat) * fact (sum-list ks) by auto
 also have ... dvd fact (k + sum-list ks) by (rule Binomial.fact-fact-dvd-fact)
 finally have fact k * mfact ks dvd fact (k + sum-list ks) .
 thus ?case by auto
qed

lemma mchoose-dvd: sum-list ks ≤ n ⟹
 mfact ks * fact (n − sum-list ks) dvd fact n
 using multinomial'-dvd[of (n − sum-list ks) # ks] by auto

lemma mchoose-le:
 sum-list ks ≤ n ⟹ mfact ks * fact (n − sum-list ks) ≤ fact n
using mchoose-dvd by (simp add: dvd-imp-le)
```

The multinomial coefficient.

```
definition multinomial' :: nat list ⇒ nat where
 multinomial' ks = fact (sum-list ks) div mfact ks

lemma multinomial'-Nil[simp]: multinomial' [] = 1
 unfolding multinomial'-def by auto

lemma multinomial'-Cons[simp]: multinomial' (k # ks) =
 ((k + sum-list ks) choose k) * multinomial' ks
```

```
  unfolding multinomial'-def
  using Rings.algebraic-semidom-class.div-mult-div-if-dvd[
    of mfact ks fact (sum-list ks)
    fact k * fact (k + sum-list ks - k) fact (k + sum-list ks),
    OF multinomial'-dvd Binomial.choose-dvd[OF Nat.le-add1]
  ]
  by (simp add: algebra-simps binomial-fact')

definition multinomial :: nat ⇒ nat list ⇒ nat (infixl mchoose 65) where
  n mchoose ks = multinomial' ((n - sum-list ks) # ks)

lemma sum-exists:
  fixes n :: nat
  assumes 0: inj f
  shows (∑ s | ∃ m≤n. s = f m. v s) = (∑ m≤n. v (f m))
proof (induction n)
  case 0
  thus ?case by auto
next
  case (Suc n)
  have (∑ s | ∃ m≤Suc n. s = f m. v s) =
    (∑ s | (∃ m≤n. s = f m) ∨ s = f (Suc n). v s) by (metis le-Suc-eq)
  also have ... = sum v ({s. (∃ m≤n. s = f m)} ∪ {f (Suc n)})
    by (simp add: Collect-disj-eq)
  also have ... = sum v {s. (∃ m≤n. s = f m)} + sum v {f (Suc n)}
  proof (rule sum.union-disjoint)
    show finite {s. ∃ m≤n. s = f m} by auto
    show finite {f (Suc n)} by auto
    show {s. ∃ m≤n. s = f m} ∩ {f (Suc n)} = {} using 0 injD by fastforce
  qed
  also have ... = (∑ s | ∃ m≤n. s = f m. v s) + v (f (Suc n)) by auto
  also have ... = (∑ m≤n. v (f m)) + v (f (Suc n)) unfolding Suc by blast
  also have ... = (∑ m≤(Suc n). v (f m)) by auto
  finally show ?case by auto
qed
```

The proof is by induction on *xs*, and using the standard binomial theorem ($(?a + ?b)^{?n} = (\sum k \leq ?n.\ of\text{-}nat\ (?n\ choose\ k) * ?a^k * ?b^{?n - k})$) See the Wikipedia article (https://en.wikipedia.org/wiki/Multinomial_theorem) for reference.

```
theorem multinomial-ring:
  fixes xs :: 'a::comm-semiring-1 list
  shows (sum-list xs) ^n = (∑ ks | length ks = length xs ∧ sum-list ks = n.
    of-nat (multinomial' ks) * mpow xs ks)
    (is - = (∑ ks∈?indices xs n. ?v xs ks))
proof (induction xs arbitrary: n)
  case Nil
  show ?case proof (cases n)
    case 0
```

```
    hence ?indices [] n = {[]} by auto
    thus ?thesis using 0 by auto
  next
    case (Suc n')
    hence 1: ?indices [] n = {} by auto
    thus ?thesis unfolding 1 using Suc by auto
  qed
next
  case (Cons x xs')
  have 2: ?indices (x # xs') n = ⋃ {s. ∃ m≤n. s = (#) (n − m) ‘ ?indices xs' m}
    (is - = ⋃ ?S is - = ⋃ {s. ?P s})
  proof
    have length ks = Suc (length xs') ∧ sum-list ks = n ⟹
      ∃ s. (∃ m≤n. s = (#) (n − m) ‘ ?indices xs' m) ∧ ks ∈ s for ks proof −
      assume 3: length ks = Suc (length xs') ∧ sum-list ks = n
      have ∃ m≤n. ks ∈ (#) (n − m) ‘ ?indices xs' m proof (cases ks)
        case Nil
        thus ?thesis using 3 by auto
      next
        case (Cons k ks')
        let ?m = n − k
        have ks ∈ (#) (n − ?m) ‘ ?indices xs' ?m using 3 Cons by auto
        moreover have ?m ≤ n by auto
        ultimately show ?thesis by blast
      qed
      thus ∃ s. (∃ m≤n. s = (#) (n − m) ‘ ?indices xs' m) ∧ ks ∈ s by blast
    qed
    thus ?indices (x # xs') n ⊆ ⋃ ?S by auto
    show ⋃ ?S ⊆ ?indices (x # xs') n by auto
  qed
  have 4: disjoint ?S proof
    fix s t
    assume s ∈ ?S
    hence ?P s by blast
    from this obtain m where
      m-bound: m ≤ n and
      s-def: s = (#) (n − m) ‘ ?indices xs' m
      by blast
    assume t ∈ ?S
    hence ?P t by blast
    from this obtain l where
      l-bound: l ≤ n and
      t-def: t = (#) (n − l) ‘ ?indices xs' l
      by blast
    have 5: ks ∈ s ⟹ ks ∈ t ⟹ s = t for ks proof −
      assume ks ∈ s
      from this obtain ks'1 where ks-def-1: ks = (n − m) # ks'1 and
        ks'1 ∈ ?indices xs' m unfolding s-def by blast
```

assume $ks \in t$
from *this* **obtain** *ks'2* **where** *ks-def-2*: $ks = (n - l) \# ks'2$ **and**
$ks'2 \in$ *?indices xs' l* **unfolding** *t-def* **by** *blast*
from *m-bound l-bound ks-def-1 ks-def-2* **have** $m = l$ **by** *auto*
thus *?thesis* **unfolding** *s-def t-def* **by** *auto*
qed
assume $s \neq t$
thus *disjnt s t* **unfolding** *disjnt-iff* **using** *5* **by** *blast*
qed
have *5*: $\forall s \in ?S$. *finite s* **proof**
fix *s*
assume $s \in ?S$
hence *?P s* **by** *blast*
from *this* **obtain** *m* **where**
s-def: $s = (\#)\ (n - m)$ ‘ *?indices xs' m* **by** *blast*
have *?indices xs' m* $\subseteq \{ks.\ set\ ks \subseteq \{..m\} \wedge length\ ks = length\ xs'\}$
using *member-le-sum-list* **by** *auto*
moreover have *finite* $\{ks.\ set\ ks \subseteq \{..m\} \wedge length\ ks = length\ xs'\}$
using *List.finite-lists-length-eq* **by** *auto*
ultimately have *finite* (*?indices xs' m*)
using *Finite-Set.finite-subset* **by** *auto*
thus *finite s* **unfolding** *s-def* **by** *auto*
qed
have $(\sum ks \in ?indices\ (x \#\ xs')\ n.\ ?v\ (x \#\ xs')\ ks) =$
$(\sum ks \in \bigcup\ ?S.\ ?v\ (x \#\ xs')\ ks)$ **unfolding** *2* **by** *blast*
also have ... $= (\sum s \mid ?P\ s.\ \sum ks \in s.\ ?v\ (x \#\ xs')\ ks)$
using *sum.Union-disjoint-sets*[*of ?S*] *4 5* **by** *auto*
also have ... $= (\sum m \leq n.\ \sum ks \in ((\#)\ (n - m)\ ‘\ ?indices\ xs'\ m).$
?v $(x \#\ xs')$ *ks*) (**is** *?G*) **proof** *cases*
assume *xs'-def*: $xs' = []$
hence *6*: *?indices* $[]\ m = (if\ m = 0\ then\ \{[]\}\ else\ \{\})$ **for** *m* **by** *auto*
have $(\sum s \mid \exists m \leq n.\ s = (\#)\ (n - m)\ ‘\ ?indices\ xs'\ m.$
$\sum ks \in s.\ ?v\ (x \#\ xs')\ ks) =$
$(\sum s \mid \exists m \leq n.\ s = (\#)\ (n - m)\ ‘\ ?indices\ []\ m.\ \sum ks \in s.\ ?v\ [x]\ ks)$
using *xs'-def* **by** *auto*
also have ... $=$
$(\sum s \mid s = (\#)\ n\ ‘\ ?indices\ []\ 0 \vee$
$(\exists m \leq n.\ m \neq 0 \wedge s = (\#)\ (n - m)\ ‘\ ?indices\ []\ m).\ \sum ks \in s.\ ?v\ [x]\ ks)$
by (*metis* (*no-types*, *lifting*) *Collect-cong diff-zero le0*)
also have ... $=$
$(\sum s \mid s = (\#)\ n\ ‘\ \{[]\} \vee$
$(\exists m \leq n.\ m \neq 0 \wedge s = (\#)\ (n - m)\ ‘\ \{\}).\ \sum ks \in s.\ ?v\ [x]\ ks)$
unfolding *6* **by** (*simp*; *metis*)
also have ... $=$
$(\sum s \mid s = \{[n]\} \vee (n \neq 0 \wedge s = \{\}).\ \sum ks \in s.\ ?v\ [x]\ ks)$
by (*metis* (*no-types*, *lifting*) *dual-order.refl image-empty*
image-insert le-zero-eq)
also have ... $=$
sum $(\lambda\ s.\ \sum ks \in s.\ ?v\ [x]\ ks)\ (\{s\ .\ s = \{[n]\}\} \cup \{s\ .\ n \neq 0 \wedge s = \{\}\})$

```
    by (rule sum.cong; auto)
  also have ... = ?v [x] [n] by auto
  finally have 7: (∑ s | ∃ m≤n. s = (#) (n − m) ‘ ?indices xs′ m.
    ∑ ks∈s. ?v (x # xs′) ks) = ?v [x] [n] .
  have (∑ m≤n. ∑ ks∈((#) (n − m) ‘ ?indices xs′ m). ?v (x # xs′) ks) =
    (∑ m≤n. ∑ ks∈((#) (n − m) ‘ ?indices [] m). ?v [x] ks)
    using xs′-def by auto
  also have ... = (if n = 0
    then (∑ ks∈((#) (n − 0) ‘ ?indices [] 0). ?v [x] ks)
    else (∑ m≤Suc (n − 1). ∑ ks∈((#) (n − m) ‘ ?indices [] m). ?v [x] ks))
    by auto
  also have ... = (if n = 0
    then (∑ ks∈((#) (n − 0) ‘ ?indices [] 0). ?v [x] ks)
    else (∑ ks∈((#) (n − 0) ‘ ?indices [] 0). ?v [x] ks) +
    (∑ m≤(n − 1). ∑ ks∈((#) (n − Suc m) ‘ ?indices [] (Suc m)). ?v [x] ks))
    unfolding sum.atMost-Suc-shift by auto
  also have ... = (if n = 0
    then (∑ ks∈((#) (n − 0) ‘ {[]}). ?v [x] ks)
    else (∑ ks∈((#) (n − 0) ‘ {[]}). ?v [x] ks) +
    (∑ m≤(n − 1). ∑ ks∈((#) (n − Suc m) ‘ {}). ?v [x] ks))
    unfolding 6 by auto
  also have ... = ?v [x] [n] by auto
  finally have 8: (∑ m≤n. ∑ ks∈((#) (n − m) ‘ ?indices xs′ m).
    ?v (x # xs′) ks) = ?v [x] [n] .
  show ?G unfolding 7 8 by simp
next
  assume xs′-non-empty: xs′ ≠ []
  have indices-non-empty: ?indices xs′ m ≠ {} for m proof −
    let ?indices-instance = m # List.replicate (length xs′ − 1) 0
    have ?indices-instance ∈ ?indices xs′ m
      using xs′-non-empty by auto
    thus ?thesis by blast
  qed
  show ?G proof (rule sum-exists)
    show inj (λm. (#) (n − m) ‘ ?indices xs′ m) proof
      fix m1 m2
      assume 9: (#) (n − m1) ‘ ?indices xs′ m1 = (#) (n − m2) ‘ ?indices xs′
m2
      from indices-non-empty obtain indices-instance-1
        where indices-instance-1 ∈ ?indices xs′ m1 by blast
      hence 10: (n − m1) # indices-instance-1 ∈ (#) (n − m1) ‘ ?indices xs′
m1
        by blast
      from indices-non-empty obtain indices-instance-2
        where indices-instance-2 ∈ ?indices xs′ m2 by blast
      hence 11: (n − m2) # indices-instance-2 ∈ (#) (n − m2) ‘ ?indices xs′
m2
        by blast
      from 9 10 11 show m1 = m2 by auto
```

```
    qed
  qed
 qed
 also have ... = (∑ m≤n. ∑ ks∈?indices xs' m. ?v (x # xs') ((n − m) # ks))
  by (rule sum.cong; simp add: sum.reindex)
 also have ...
  = (∑ m≤n. ∑ ks∈?indices xs' m. of-nat ((n choose (n − m)) * multinomial' ks) *
  (x ^ (n − m) * mpow xs' ks)) using Cons by auto
 also have ... = (∑ m≤n. ∑ ks∈?indices xs' m.
  (of-nat (n choose (n − m)) * x ^ (n − m)) *
  (of-nat (multinomial' ks) * mpow xs' ks)) by (simp add: algebra-simps)
 also have ... =
  (∑ m≤n. of-nat (n choose (n − m)) * x ^ (n − m) *
  (∑ ks∈?indices xs' m. of-nat (multinomial' ks) * mpow xs' ks))
  by (simp add: sum-distrib-left)
 also have ... = (∑ m≤n. of-nat (n choose (n − m)) * x ^ (n − m) *
  sum-list xs' ^ m) using Cons by auto
 also have ... = (∑ m≤n. of-nat (n choose m) * sum-list xs' ^ m * x ^ (n −
m))
  apply (rule sum.cong)
  apply simp
  subgoal for m
   apply (simp add: Binomial.binomial-symmetric[of m n] algebra-simps)
   done
  done
 also have ... = (sum-list xs' + x) ^n
  by (rule Binomial.binomial-ring[symmetric])
 also have ... = (sum-list (x # xs')) ^n by (simp add: algebra-simps)
 finally have (∑ ks∈?indices (x # xs') n. ?v (x # xs') ks) =
  (sum-list (x # xs')) ^n .
 thus ?case by auto
qed
```

This version of the multinomial theorem is also useful.

```
corollary multinomial-ring-alt:
 fixes xs :: 'a::comm-semiring-1 list
 shows (1 + sum-list xs) ^n = (∑ ks | length ks = length xs ∧ sum-list ks ≤ n.
  of-nat (n mchoose ks) * mpow xs ks)
proof −
 have (1 + sum-list xs) ^n = (sum-list (1 # xs)) ^n by auto
 also have ... = (∑ ks | length ks = 1 + length xs ∧ sum-list ks = n.
  of-nat (multinomial' ks) * mpow (1 # xs) ks)
  unfolding multinomial-ring by auto
 also have ... = (∑ ks | length ks = length xs ∧ sum-list ks ≤ n.
  of-nat (n mchoose ks) * mpow xs ks)
 proof (rule sum.reindex-cong[of tl, symmetric])
  show inj-on tl {ks. length ks = 1 + length xs ∧ sum-list ks = n}
   unfolding inj-on-def
```

by (*smt* (*verit*, *del-insts*) *One-nat-def add.commute add-left-cancel*
list.collapse list.size(*3*) *mem-Collect-eq nat.simps*(*3*) *plus-nat.simps*(*2*)
sum-list.Cons)
show {*ks*. *length ks* = *length xs* ∧ *sum-list ks* ≤ *n*} =
tl ‘ {*ks*. *length ks* = *1* + *length xs* ∧ *sum-list ks* = *n*}
unfolding *image-Collect*
apply (*rule Collect-cong*)
subgoal for *ks*
apply *standard*
subgoal
by (*metis One-nat-def add.commute less-eqE list.sel*(*3*) *list.size*(*4*)
sum-list.Cons)
subgoal
by (*metis Suc-eq-plus1 add.commute add-diff-cancel-left′ le-add2 length-0-conv*
length-tl
list.collapse nat.simps(*3*) *sum-list.Cons*)
done
done
fix *ks*
assume *0*: *ks* ∈ {*ks*. *length ks* = *1* + *length xs* ∧ *sum-list ks* = *n*}
from *0* **obtain** *k ks′* **where** *ks-def*: *ks* = *k* # *ks′*
by (*simp*; *metis Suc-length-conv*)
hence *n* − *sum-list ks′* = *k* **using** *0* **by** *auto*
thus *of-nat* (*n mchoose tl ks*) ∗ *mpow xs* (*tl ks*) =
of-nat (*multinomial′ ks*) ∗ *mpow* (*1* # *xs*) *ks*
unfolding *ks-def multinomial-def* **by** *simp*
qed
finally show *?thesis* **.**
qed

end
theory *Lemma-1-8-Defs*
imports *Main ../MPoly-Utils/More-More-MPoly-Type Bit-Counting*
Utils Multinomial
begin

2.4 Expressing polynomial solutions in terms of carry counting

2.4.1 Preliminary definitions

locale *Lemma-1-8-Defs* =
fixes *P* :: *int mpoly*
and $\mathcal{B}$ :: *nat*
and *L* :: *int*
and *z* :: *nat list*
begin

abbreviation σ **where** $\sigma \equiv$ *count-bits*
abbreviation τ **where** $\tau \equiv$ *count-carries*

definition δ::*nat* **where** δ = *total-degree P*
definition ν::*nat* **where** ν = *max-vars P*
definition *b*::*nat* **where** *b* ≡ $\mathcal{B}$ *div 2*
definition *n*::*nat* ⇒ *nat* **where** *n j* = (δ+*1*) ^*j*

definition *X*::*int mpoly* **where** *X* = *Var 0*

The are assignments of variables, used to evaluate (multivariable) polynomials.

definition *z-assign* **where** *z-assign* = ((!$_0$) (*map int z*))
definition $\mathcal{B}$*-assign* **where** $\mathcal{B}$*-assign* = (λ*i*. (*int* $\mathcal{B}$) *when i* = *0*)

We will often use this set as indices of sums.

definition δ*-tuples* :: (*nat list*) *set* **where**
  δ*-tuples* ≡ {*i*. *length i* = ν + *1* ∧ *sum-list i* ≤ δ}

definition *P-coeff* :: *nat list* ⇒ *int* **where**
  P-coeff i ≡ *coeff P* (*Abs-poly-mapping* ((!$_0$) *i*))

definition *D-exponent* :: *nat list* ⇒ *nat* **where**
  D-exponent i ≡ *n* (ν+*1*) − (∑ *s*≤ν. *i*!*s* ∗ *n s*)
definition *D-precoeff* :: *nat list* ⇒ *int* **where**
  D-precoeff i ≡ *int* (*mfact i* ∗ *fact* (δ − *sum-list i*))

This is really a univariate polynomial

definition *D*::*int mpoly* **where**
  D ≡ ∑ *i*∈δ*-tuples*. *of-int* (*D-precoeff i* ∗ *P-coeff i*) ∗ *X*^(*D-exponent i*)

This is really a univariate polynomial

definition *c*::*int mpoly* **where**
  c ≡ (∑ *i*≤ν. *of-nat* (*z*!*i*) ∗ *X*^(*n i*))

Definition of the constant K

definition *R*::*int mpoly* **where** *R* ≡ (*1*+*c*) ^δ ∗ *D*
definition *S*::*nat* **where** *S* ≡ (∑ *i*≤(*2*∗δ+*1*) ∗ *n* ν. *b* ∗ $\mathcal{B}$ ^*i*)
definition *K*::*int* **where** *K* ≡ *insertion* $\mathcal{B}$*-assign R* + *int S*

Some more notation used in the proofs : (e j) is the coefficient of X^j in R

definition *e*::*nat* ⇒ *int*
  where *e j* = *coeff R* (*Poly-Mapping.single 0 j*)

end

end
theory *Lemma-1-8-Coding*
  imports *Lemma-1-8-Defs*
begin

2.4.2 Bounds on the defined variables

locale *K-Nonnegative* = *Lemma-1-8-Defs* +
  assumes *δ-pos*: $\delta > 0$
    and *L-pos*: $L > 0$
    and *L-lower-bound*: $L \geq$ *max-coeff P*
    and *len-z*: *length z* = ν+1
    and $\mathcal{B}$*-even*: 2 *dvd* $\mathcal{B}$
    and $\mathcal{B}$*-lower-bound*: $\mathcal{B} > 2 * fact\ \delta * (nat\ L) * (1 + sum\text{-}list\ z)\ \hat{}\ \delta$
begin

This is for convenience in proofs, but the following lemma is strictly stronger.

lemma $\mathcal{B}$*-ge-1*[*simp*]: $\mathcal{B} \geq 1$
  using $\mathcal{B}$*-lower-bound δ-pos L-pos* **by** *auto*

lemma $\mathcal{B}$*-gt-2*[*simp*]: $\mathcal{B} > 2$
  using $\mathcal{B}$*-lower-bound*
 by (*smt* (*verit*, *ccfv-threshold*) *L-pos* $\mathcal{B}$*-ge-1 fact-gt-zero less-2-cases less-numeral-extra*(3)
    less-one linorder-neqE-nat mult-eq-0-iff nat-1-eq-mult-iff nat-le-iff-add not-one-le-zero

    power-not-zero zero-less-nat-eq)

Also for convenience.

lemma $\mathcal{B}$*-ge-2*[*simp*]: $\mathcal{B} \geq 2$
  using $\mathcal{B}$*-gt-2* **by** *linarith*

lemma *b-def-reverse*: 2 * *b* = $\mathcal{B}$ **using** $\mathcal{B}$*-even b-def* **by** *simp*

lemma *b-ge-1*[*simp*]: $b \geq 1$
  using *b-def-reverse* $\mathcal{B}$*-ge-1* **by** *fastforce*

lemma *n-ge-1*[*simp*]: $n\ j \geq 1$
  unfolding *n-def* **by** *auto*

lemma *L-lower-bound-specialize*: $L \geq abs\ (P\text{-}coeff\ i)$
proof *cases*
  assume *assm*: *P-coeff i* ≠ 0
  hence *P-coeff i* ∈ *coeffs P*
    unfolding *P-coeff-def coeffs-def* **by** (*simp add*: *coeff-def coeff-keys*)
  hence *0*: *abs* (*P-coeff i*) ∈ *abs* ‘ *coeffs P*
    by *auto*

  have $L \geq Max\ (abs\ `\ coeffs\ P)$
    using *L-lower-bound max-coeff-def* **by** *metis*
  thus *?thesis*
    using *0* **by** (*meson Max-ge finite-coeffs finite-imageI order-trans*)
next
  assume ¬ *P-coeff i* ≠ 0
  hence *P-coeff i* = 0 **by** *auto*
  thus *?thesis* **using** *L-pos* **by** *auto*

qed

lemma *δ-tuples-finite*[*simp*]: *finite δ-tuples*
proof −
have *stronger*: *finite* {*i*::*nat list*. *length i* = *n* ∧ *sum-list i* ≤ *K* }
(**is** *finite* (*?S n*)) **for** *n K*
proof (*induction n*)
case *0* **thus** *?case* **by** *auto*
next
case (*Suc n*)
{ fix *i* **assume** *assm-i*: *i* ∈ *?S* (*Suc n*)
then obtain *j0 j* **where** *i-def*: *i* = *j0*#*j*
by (*smt* (*verit*, *best*) *Suc-length-conv mem-Collect-eq*)
hence (*j0*, *j*) ∈ ({..*K*} × *?S n*) **using** *assm-i* **by** *auto*
hence *i* ∈ (λ(*j0*, *j*). *j0*#*j*) ‘ ({..*K*} × *?S n*) **using** *i-def* **by** *auto* **}**
hence *?S* (*Suc n*) ⊆ (λ(*j0*, *j*). *j0*#*j*) ‘ ({..*K*} × *?S n*)
by *blast*
moreover have *finite* ((λ(*j0*, *j*). *j0*#*j*) ‘ ({..*K*} × *?S n*))
using *Suc.IH* **by** *blast*
ultimately have *finite* (*?S* (*Suc n*))
by (*meson finite-subset*)
thus *?case* **.**
qed
thus *?thesis* **unfolding** *δ-tuples-def* **.**
qed

lemma *P-z-insertion*: *insertion z-assign P* = (∑ *i*∈*δ-tuples*. *P-coeff i* ∗ *mpow z i*)
proof −
have *mpow-eq*: *i*∈*δ-tuples* ⟹ (∏ *s*≤*ν*. (*z-assign s*) ^(*i*!*s*)) = *mpow z i* **for** *i*
proof −
assume *i-def*: *i*∈*δ-tuples*
have *0*: *s* < *length z* ⟹ *z-assign s* = *int* (*z*!*s*) **for** *s*
proof −
assume *s* < *length z*
hence *s* < *length* (*map int z*)
using *length-map* **by** *simp*
hence *z-assign s* = (*map int z*) ! *s*
unfolding *z-assign-def* **using** *nth0-nth* **by** *blast*
also have ... = *int* (*z* ! *s*)
using ‹*s* < *length z*› *nth-map* **by** *blast*
finally show *?thesis* **.**
qed

have *z-i-length*: *min* (*length z*) (*length i*) = *length z*
using *δ-tuples-def len-z i-def* **by** *auto*

have (∏ *s*≤*ν*. (*z-assign s*) ^(*i*!*s*)) = (∏ *s*<*length z*. (*z-assign s*) ^(*i*!*s*))
using *Suc-eq-plus1 len-z lessThan-Suc-atMost* **by** *presburger*
also have ... = (∏ *s*<*length z*. (*int* (*z*!*s*)) ^(*i*!*s*))

```
    using 0 by simp
  finally show (∏ s≤ν. (z-assign s) ^(i!s)) = int (mpow z i)
    unfolding mpow-def using z-i-length by auto
qed

have P = (∑ i∈δ-tuples. of-int (P-coeff i) * (∏ s=0..ν. (Var s) ^(i!s)))
  unfolding δ-tuples-def ν-def δ-def of-int-Const P-coeff-def
  using mpoly-multivariate-expansion by auto
hence insertion z-assign P = (∑ i∈δ-tuples. insertion z-assign (of-int (P-coeff
i)) *
  insertion z-assign (∏ s≤ν. (Var s) ^(i!s)))
  using insertion-sum insertion-mult δ-tuples-finite
  by (smt (verit, del-insts) sum.cong atLeast0AtMost)
 also have ... = (∑ i∈δ-tuples. P-coeff i * insertion z-assign (∏ s≤ν. (Var
s) ^(i!s)))
  using of-int-Const by simp
also have ... = (∑ i∈δ-tuples. P-coeff i * mpow z i)
  by (simp add: mpow-eq)
finally show ?thesis .
qed
```

This is essentially an instance of the multinomial theorem.

```
lemma c-delta-expansion:
  (1+c) ^δ = (∑ i∈δ-tuples. of-nat ((δ mchoose i) * mpow z i) * X^(∑ s≤ν. i!s
* n s))
proof −
  define F::(int mpoly) list where
    F = map (λs. of-nat (z!(nat s)) * X^(n (nat s))) [0..ν]

  have F-length: length F = ν+1
    unfolding F-def by simp

  have F-get: s < length F ⟹ F!s = of-nat (z!s) * X^(n s) for s
    unfolding F-def using F-length by simp

  have F-mpow: length i = ν + 1 ⟹
    mpow F i = of-nat (mpow z i) * X ^ (∑ s<ν + 1. i ! s * n s) for i
  proof −
    assume length i = ν + 1
    hence F-i-length: min (length F) (length i) = ν + 1 using F-length by auto
    have mpow F i = (∏ s<ν+1. (of-nat (z!s) * X^(n s)) ^ i!s)
      using F-i-length by (simp add: mpow-def F-get)
    also have ... = (∏ s<ν+1. (of-nat (z!s)) ^ i!s) * (∏ s<ν+1. (X^(n s)) ^ i!s)
     by (simp add: power-mult-distrib prod.distrib)
    also have ... = of-nat (mpow z i) * (∏ s<ν+1. (X^(n s)) ^ i!s)
      unfolding mpow-def using len-z F-i-length by auto
    also have ... = of-nat (mpow z i) * (∏ s<ν+1. X^(i!s * n s))
      by (metis mult.commute power-mult)
    also have ... = of-nat (mpow z i) * X^(∑ s<ν+1. i!s * n s)
```

```
    by (smt (verit, ccfv-SIG) power-sum prod.cong)
  finally show ?thesis .
 qed

 have (1 + c) ^δ = (1 + (∑ s≤ν. F!s)) ^δ
  by (simp add : c-def F-get F-length)
 also have ... = (1 + sum-list F) ^δ
  using F-length by (metis Suc-eq-plus1 atLeastLessThanSuc-atLeastAtMost sum-list-sum-nth

                     atLeast0AtMost)
 also have ... = (∑ i | length i = length F ∧ sum-list i ≤ δ. of-nat (δ mchoose
i) * mpow F i)
  by (simp only: multinomial-ring-alt)
 also have ... = (∑ i∈δ-tuples. of-nat (δ mchoose i) * mpow F i)
  unfolding δ-tuples-def by (simp add: F-length)
 also have ... = (∑ i∈δ-tuples. of-nat (δ mchoose i) *
  of-nat (mpow z i) * X^(∑ s≤ν. i!s * n s))
  by (auto simp add: algebra-simps F-mpow δ-tuples-def lessThan-Suc-atMost)
 finally show ?thesis by auto
qed

lemma n-νp1-ge-sum: i∈δ-tuples ⟹ n (ν+1) ≥ (∑ s≤ν. i!s * n s)
proof −
 assume i-def: i∈δ-tuples

 { fix s assume s ∈ {..ν}
  hence i!s * n s ≤ i!s * n ν
   unfolding n-def by (simp add: power-increasing) }
 note 0 = this

 have (∑ s≤ν. i!s * n s) ≤ (∑ s≤ν. i!s * n ν)
  using 0 by (meson atLeastAtMost-iff atMost-iff sum-mono zero-le)
 also have ... = n ν * (∑ s≤ν. i!s)
  using sum-distrib-right by (metis mult.commute)
 also have ... = n ν * (∑ s<length i. i!s)
  using i-def unfolding δ-tuples-def using lessThan-Suc-atMost by force
 also have ... = n ν * sum-list i
  by (simp add: sum-list-sum-nth atLeast0LessThan)
 also have ... ≤ n ν * δ
  using i-def unfolding δ-tuples-def by simp
 also have ... ≤ n (ν+1)
  unfolding n-def by auto
 finally show ?thesis .
qed

lemma D-exponent-inj: inj-on D-exponent δ-tuples
proof
 have digit: i ∈ δ-tuples ⟹
  nth-digit (∑ s≤ν. i!s * (δ+1) ^s) s (δ+1) = (if s ≤ ν then i!s else 0) for i s
```

```
  proof −
    assume i-def: i ∈ δ-tuples
    have s ≤ ν ⟹ i!s < δ+1 for s
    proof −
      assume s ≤ ν
      hence s < length i using i-def δ-tuples-def by auto
      hence i!s ≤ sum-list i using elem-le-sum-list by auto
      thus ?thesis using i-def δ-tuples-def by auto
    qed
    hence nth-digit (∑ s=0..ν. i!s * (δ+1) ^ s) s (δ+1) = (if s ≤ ν then i!s else
0)
      using nth-digit-gen-power-series-general[of δ+1 ν] δ-pos by simp
    thus ?thesis
      using atMost-atLeast0 by presburger
  qed

  fix i j assume i-def: i∈δ-tuples and j-def: j∈δ-tuples
    and D-exponent i = D-exponent j
  hence (∑ s≤ν. i!s * (δ+1) ^ s) = (∑ s≤ν. j!s * (δ+1) ^ s)
    unfolding D-exponent-def using n-νp1-ge-sum n-def
    by (metis (no-types, lifting) diff-diff-cancel sum.cong)
  hence s ≤ ν ⟹ i!s = j!s for s
    using digit i-def j-def by (metis (full-types))
  hence s < length i ⟹ i!s = j!s for s
    using i-def unfolding δ-tuples-def by auto
  thus i = j
    apply (simp add: List.list-eq-iff-nth-eq)
    using δ-tuples-def i-def j-def by force
qed

definition R-exponent :: nat list ⇒ nat list ⇒ nat where
  R-exponent i j = D-exponent j + (∑ s≤ν. i!s * n s)

lemma R-expansion:
  R = (∑ i∈δ-tuples. (∑ j∈δ-tuples.
    of-int ((δ mchoose i) * mfact j * fact (δ − sum-list j) * mpow z i * P-coeff j) *
    X^(R-exponent i j)))
proof −
  define A::nat list ⇒ int mpoly where A = (λi. of-nat ((δ mchoose i) * mpow z
i))
  define B::nat list ⇒ int mpoly where B = (λi. of-int (D-precoeff i * P-coeff i))


  have AB-rewrite: A i * B j = of-int ((δ mchoose i) * mfact j *
    fact (δ − sum-list j) * mpow z i * P-coeff j) for i j
  using A-def B-def D-precoeff-def by simp

  have R = (1+c) ^δ * D by (simp add: R-def)
  also have ... = (∑ i∈δ-tuples. A i * X^(∑ s≤ν. i!s * n s)) *
```

```
    (∑j∈δ-tuples. B j * X^(D-exponent j))
    by (simp add: D-def c-delta-expansion A-def B-def)
  also have ... = (∑i∈δ-tuples. (∑j∈δ-tuples.
    A i * X^(∑s≤ν. i!s * n s) * (B j * X^(D-exponent j))))
    using sum-product by simp
  also have ... = (∑i∈δ-tuples. (∑j∈δ-tuples.
    (A i * B j) * (X^(∑s≤ν. i!s * n s) * X^(D-exponent j))))
    by (simp add: algebra-simps)
  also have ... = (∑i∈δ-tuples. (∑j∈δ-tuples.
    A i * B j * X^(R-exponent i j)))
    unfolding R-exponent-def by (simp add: mult.commute power-add)
  finally show ?thesis by (simp add: AB-rewrite)
qed

lemma c-degree-bound: degree c 0 ≤ n ν
proof −
  have 0: s∈{..ν} ⟹ degree (of-nat (z!s) * X^(n s)) 0 ≤ n ν for s
  proof −
    assume Hs: s ∈ {..ν}
    have degree (of-nat (z!s) * X^(n s)) 0 ≤ degree (X^(n s)) 0
      using degree-mult degree-Const of-nat-Const by (metis add-0)
    also have ... ≤ n s * degree X 0
      using degree-pow by auto
    also have ... = n s
      unfolding X-def using degree-Var by (metis nat-mult-1-right)
    also have ... ≤ n ν
      unfolding n-def using Hs
      by (simp add: power-increasing)
    finally show ?thesis .
  qed

  have degree c 0 ≤ Max (insert 0 ((λs. degree (of-nat (z!s) * X^(n s)) 0) ‘
{..ν}))
    unfolding c-def using degree-sum by blast
  also have ... ≤ n ν
    by (simp add: 0)
  finally show ?thesis .
qed

lemma D-degree-bound: degree D 0 ≤ n (ν+1)
proof −
  have 0: i∈δ-tuples ⟹
    degree (of-int (D-precoeff i * P-coeff i) * X^(D-exponent i)) 0 ≤ n (ν+1) for i
  proof −
    assume i∈δ-tuples
    have degree (of-int (D-precoeff i * P-coeff i) * X^(D-exponent i)) 0 ≤
      degree (X^(D-exponent i)) 0
      using degree-mult degree-Const of-int-Const by (metis add-0)
    also have ... ≤ D-exponent i
```

```
    unfolding X-def using degree-pow degree-Var by (metis mult.right-neutral)
   also have ... ≤ n (ν+1)
    unfolding D-exponent-def by auto
   finally show ?thesis .
  qed

  have degree D 0 ≤ Max (insert 0 (
   (λi. degree (of-int (D-precoeff i * P-coeff i) * X^(D-exponent i)) 0) ` δ-tuples))
   unfolding D-def using degree-sum δ-tuples-finite by blast
  also have ... ≤ n (ν+1)
   using 0 by auto
  finally show ?thesis .
qed

lemma R-degree-bound: degree R 0 ≤ (2*δ+1) * n ν
proof −
  have c-1-deg: degree (1 + c) 0 ≤ degree c 0
  proof cases
   assume degree c 0 ≥ 1
   thus ?thesis using degree-add degree-one by (metis max-def zero-le)
  next
   assume ¬ 1 ≤ degree c 0
   hence degree c 0 = 0 by auto
   moreover have degree (1 + c) 0 ≤ 0
    by (metis degree-add MPoly-Type.degree-one calculation max-0L)
   ultimately show ?thesis by auto
  qed

  have degree R 0 ≤ degree ((1+c)^δ) 0 + degree D 0
   unfolding R-def by (simp add: degree-mult)
  also have ... ≤ δ * degree (1+c) 0 + degree D 0
   by (simp add: degree-pow)
  also have ... ≤ δ * degree c 0 + degree D 0
   by (simp add: c-1-deg)
  also have ... ≤ δ * n ν + n (ν+1)
   using c-degree-bound D-degree-bound by (simp add: add-le-mono)
  finally show ?thesis unfolding n-def by auto
qed


lemma R-univariate: vars R ⊆ {0}
proof −
  define x where x = (λi j. of-int
   (int ((δ mchoose i) * mfact j * fact (δ − sum-list j) * mpow z i) * P-coeff j)
   * X ^ R-exponent i j)
  have vars-0: vars (x i j) ⊆ {0} for i j
  proof −
   have vars (x i j) ⊆ vars (X^R-exponent i j)
    using vars-Const vars-mult of-int-Const x-def by fastforce
```

```
    also have ... ⊆ vars X
      using vars-pow by auto
    also have ... = {0}
      using vars-Var X-def by metis
    finally show ?thesis .
  qed

  have vars R ⊆ (⋃ i∈δ-tuples. vars (∑ j∈δ-tuples. x i j))
    unfolding R-expansion x-def using vars-setsum δ-tuples-finite by blast
  also have ... ⊆ (⋃ i∈δ-tuples. (⋃ j∈δ-tuples. vars (x i j)))
    using vars-setsum δ-tuples-finite by fast
  also have ... ⊆ (⋃ i∈δ-tuples. (⋃ j∈δ-tuples. {0}))
    using vars-0 by auto
  also have ... ⊆ {0}
    by auto
  finally show ?thesis .
qed

lemma R-sum-e: R = (∑ i≤(2*δ+1) * n ν. (of-int (e i)) * X^i)
proof −
  have 0: i ∉ {..(2*δ+1) * n ν} ⟹ of-int (e i) * X^i = 0 for i
  proof −
    assume i∉{..(2*δ+1) * n ν}
    hence i > (2*δ+1) * n ν
      by auto
    hence i > degree R 0
      using R-degree-bound by auto
    hence i ∉ ((λm. lookup m 0) ' keys (mapping-of R))
      unfolding degree-def by auto
    hence Poly-Mapping.single 0 i ∉ keys (mapping-of R)
      by (metis lookup-single-eq rev-image-eqI)
    hence e i = 0
      unfolding e-def coeff-def by (simp add: in-keys-iff)
    thus ?thesis by auto
  qed

  have R = Sum-any (λi. monom (Poly-Mapping.single 0 i) (e i))
    using mpoly-univariate-expansion R-univariate e-def by auto
  also have ... = Sum-any (λi. Const (e i) * monom (Poly-Mapping.single 0 i)
1)
    using cst-poly-times-monom-one by (metis (mono-tags, lifting) Const.abs-eq)
  also have ... = Sum-any (λi. of-int (e i) * X^i)
    unfolding of-int-Const e-def X-def
     by (metis (no-types, lifting) Var.abs-eq Var0-def monom.abs-eq monom-pow
mult-1 power-one)
  also have ... = Sum-any (λi. if i∈{..(2*δ+1) * n ν} then of-int (e i) * X^i else
0)
    using 0 by meson
  also have ... = (∑ i∈{..(2*δ+1) * n ν}. of-int (e i) * X^i)
```

by (*simp add*: *Sum-any.conditionalize*)
finally show *?thesis* **.**
qed

lemma *e-expression*:
e p = (∑ *i*∈*δ-tuples*. (∑ *j*∈*δ-tuples* ∩ {*j*. *R-exponent i j* = *p*}.
(*δ mchoose i*) * *mfact j* * *fact* (*δ* − *sum-list j*) * *P-coeff j* * *mpow z i*))
proof −
define *m* **where** *m* = *Poly-Mapping.single* (*0*::*nat*) *p*
define *c* **where** *c* = (λ*i j*.
(*δ mchoose i*) * *mfact j* * *fact* (*δ* − *sum-list j*) * *mpow z i* * *P-coeff j*)

{ fix *i j*
have *X*^(*R-exponent i j*) = *monom* (*Poly-Mapping.single 0* (*R-exponent i j*))
1
unfolding *X-def*
by (*metis Var.abs-eq Var*$_0$*-def monom.abs-eq monom-pow nat-mult-1 power-one*)
hence *of-int* (*c i j*) * *X*^(*R-exponent i j*) = *monom* (*Poly-Mapping.single 0*
(*R-exponent i j*)) (*c i j*)
unfolding *of-int-Const* **using** *cst-poly-times-monom-one* **by** (*metis Const.abs-eq*)
hence *coeff* (*of-int* (*c i j*) * *X*^(*R-exponent i j*)) *m* =
coeff (*monom* (*Poly-Mapping.single 0* (*R-exponent i j*)) (*c i j*)) *m*
by *simp*
also have ... = (*c i j when m* = *Poly-Mapping.single 0* (*R-exponent i j*))
using *More-MPoly-Type.coeff-monom* **by** *blast*
also have ... = (*c i j when R-exponent i j* = *p*)
unfolding *m-def* **by** (*smt* (*verit*, *best*) *lookup-single-eq*)
finally have *coeff* (*of-int* (*c i j*) * *X*^(*R-exponent i j*)) *m* =
(*c i j when R-exponent i j* = *p*) **. }**
note *coeff-interior* = *this*

have *0*: (∑ *j*∈*δ-tuples*. (*c i j when R-exponent i j* = *p*)) =
(∑ *j*∈*δ-tuples* ∩ {*j*. *R-exponent i j* = *p*}. *c i j*) **for** *i*
using *δ-tuples-finite*
by (*smt* (*verit*, *ccfv-SIG*) *mem-Collect-eq sum.cong sum.inter-restrict when-def*)

have *e p* = *coeff* (∑ *i*∈*δ-tuples*. (∑ *j*∈*δ-tuples*. *of-int* (*c i j*) * *X*^(*R-exponent*
i j))) *m*
unfolding *e-def m-def c-def* **using** *R-expansion* **by** *simp*
also have ... = (∑ *i*∈*δ-tuples*. (∑ *j*∈*δ-tuples*. *coeff* (*of-int* (*c i j*) * *X*^(*R-exponent*
i j)) *m*))
using *coeff-sum δ-tuples-finite* **by** (*smt* (*verit*, *best*) *sum.cong*)
also have ... = (∑ *i*∈*δ-tuples*. (∑ *j*∈*δ-tuples*. (*c i j when R-exponent i j* = *p*)))
using *coeff-interior* **by** *simp*
also have ... = (∑ *i*∈*δ-tuples*. (∑ *j*∈*δ-tuples* ∩ {*j*. *R-exponent i j* = *p*}. *c i j*))
using *0* **by** *simp*
finally show *?thesis* **unfolding** *c-def*
by (*smt* (*verit*, *ccfv-SIG*) *ab-semigroup-mult-class.mult-ac*(*1*)

mult.commute of-nat-mult sum.cong)
qed

This is a key step of the proof.

lemma *e-n-ν1-expression*: *e* (*n* (*ν*+*1*)) = *fact* *δ* ∗ *insertion z-assign P*
proof −
have *simp-indices*: *i*∈*δ-tuples* ⟹ *δ-tuples* ∩ {*j*. *R-exponent i j* = *n* (*ν*+*1*)} = {*i*}
(**is** - ⟹ *?S i* = {*i*}) **for** *i*
proof −
assume *i-def*: *i*∈*δ-tuples*
have *0*: *i* ∈ *?S i*
unfolding *R-exponent-def D-exponent-def* **using** *n-νp1-ge-sum i-def* **by** *simp*
have *1*: *j1* ∈ *?S i* ⟹ *j2* ∈ *?S i* ⟹ *j1* = *j2* **for** *j1 j2*
unfolding *R-exponent-def* **using** *D-exponent-inj*
by (*simp add*: *inj-on-def*)

have {*i*} ⊆ *?S i* **using** *0* **by** *auto*
moreover have *card* (*?S i*) ≤ *1* **using** *1* **by** (*simp add*: *card-le-Suc0-iff-eq*)
ultimately show *?thesis*
by (*metis One-nat-def δ-tuples-finite card.empty card.insert*
card-seteq empty-iff finite.emptyI finite-Int)
qed

have *simp-coeff*: *i*∈*δ-tuples* ⟹ (*δ mchoose i*) ∗ *mfact i* ∗ *fact* (*δ* − *sum-list i*) = *fact δ* **for** *i*
unfolding *δ-tuples-def multinomial-def multinomial'-def* **using** *mchoose-dvd* **by** *fastforce*

have *e* (*n* (*ν*+*1*)) = (∑ *i*∈*δ-tuples*. (∑ *j*∈*δ-tuples* ∩ {*j*. *R-exponent i j* = *n* (*ν*+*1*)}.
(*δ mchoose i*) ∗ *mfact j* ∗ *fact* (*δ* − *sum-list j*) ∗ *P-coeff j* ∗ *mpow z i*))
by (*simp add*: *e-expression*)
also have ... = (∑ *i*∈*δ-tuples*.
(*δ mchoose i*) ∗ *mfact i* ∗ *fact* (*δ* − *sum-list i*) ∗ *P-coeff i* ∗ *mpow z i*)
using *simp-indices* **by** *auto*
also have ... = (∑ *i*∈*δ-tuples*. *fact δ* ∗ *P-coeff i* ∗ *mpow z i*)
using *simp-coeff* **by** *auto*
also have ... = *fact δ* ∗ (∑ *i*∈*δ-tuples*. *P-coeff i* ∗ *mpow z i*)
using *sum-distrib-left*
by (*smt* (*verit*) *ab-semigroup-mult-class.mult-ac*(*1*) *sum.cong*)
also have ... = *fact δ* ∗ *insertion z-assign P*
using *P-z-insertion* **by** *simp*
finally show *?thesis* **.**
qed

lemma *abs-int*[*simp*]: *abs* (*int x*) = *int x*
using *abs-of-nat* **by** *auto*

This is a key step of the proof.

lemma *e-upper-bound*:
abs (*e p*) ≤ *fact* δ ∗ *L* ∗ (*1* + *sum-list z*) ^ δ
proof −
{
fix *i* **assume** *i*∈δ-*tuples*
define *S-i* **where** *S-i* = δ-*tuples* ∩ {*j*. *R-exponent i j* = *p*}

have *finite S-i* **unfolding** *S-i-def* **using** δ-*tuples-finite* **by** *simp*

{ fix *j1 j2* **assume** *j1* ∈ *S-i* **and** *j2* ∈ *S-i*
hence *D-exponent j1* = *D-exponent j2*
unfolding *S-i-def R-exponent-def* **by** *auto*
hence *j1* = *j2*
using ‹*j1* ∈ *S-i*› ‹*j2* ∈ *S-i*› *D-exponent-inj* **unfolding** *S-i-def*
by (*meson IntD1 inj-on-contraD*) **}**
hence *card S-i* ≤ *1* **by** (*simp add*: ‹*finite S-i*› *card-le-Suc0-iff-eq*)

have *0*: (∑ *j*∈*S-i*. *mfact j* ∗ *fact* (δ − *sum-list j*) ∗ *abs* (*P-coeff j*)) ≤
fact δ ∗ *L* ∗ *card S-i*
proof −
have *j*∈*S-i* ⟹ *mfact j* ∗ *fact* (δ − *sum-list j*) ≤ *fact* δ **for** *j*
unfolding *S-i-def* δ-*tuples-def* **using** *mchoose-le* **by** *blast*
hence *j*∈*S-i* ⟹ *mfact j* ∗ *fact* (δ − *sum-list j*) ∗ *abs* (*P-coeff j*) ≤ *fact* δ ∗
L **for** *j*
using *L-lower-bound-specialize*
by (*metis abs-ge-zero mult-mono′ of-nat-0-le-iff of-nat-fact zle-int*)
thus *?thesis* **by** (*simp add*: *mult.commute sum-bounded-above*)
qed

have *abs* (∑ *j*∈*S-i*. (δ *mchoose i*) ∗ *mfact j* ∗ *fact* (δ − *sum-list j*) ∗ *P-coeff j*
∗ *mpow z i*)
≤ (∑ *j*∈*S-i*. *abs* ((δ *mchoose i*) ∗ *mfact j* ∗ *fact* (δ − *sum-list j*) ∗ *P-coeff j*
∗ *mpow z i*))
using *sum-abs* **by** *simp*
also have ... = (∑ *j*∈*S-i*. (δ *mchoose i*) ∗
mfact j ∗ *fact* (δ − *sum-list j*) ∗ *abs* (*P-coeff j*) ∗ *mpow z i*)
using *abs-mult abs-int* **by** (*metis* (*no-types*, *opaque-lifting*))
also have ... = (∑ *j*∈*S-i*. *int* (δ *mchoose i*) ∗ *mpow z i* ∗
(*mfact j* ∗ *fact* (δ − *sum-list j*) ∗ *abs* (*P-coeff j*)))
by (*simp add*: *algebra-simps*)
also have ... = *int* (δ *mchoose i*) ∗ *mpow z i* ∗
(∑ *j*∈*S-i*. *mfact j* ∗ *fact* (δ − *sum-list j*) ∗ *abs* (*P-coeff j*))
by (*simp only*: *sum-distrib-left*)
also have ... ≤ *int* (δ *mchoose i*) ∗ *mpow z i* ∗ (*fact* δ ∗ *L* ∗ *card S-i*)
using *0* **by** (*simp add*: *mult-left-mono*)
also have ... ≤ *fact* δ ∗ *L* ∗ *int* (δ *mchoose i*) ∗ *mpow z i*
using ‹*card S-i* ≤ *1*› *L-pos order-le-less* **by** *fastforce*
finally have *abs* (∑ *j*∈*S-i*. (δ *mchoose i*) ∗ *mfact j* ∗ *fact* (δ − *sum-list j*) ∗
P-coeff j ∗ *mpow z i*) ≤ *fact* δ ∗ *L* ∗ *int* (δ *mchoose i*) ∗ *mpow z i* .

```
  } note bound-coeff = this

  have abs (e p) ≤ (∑ i∈δ-tuples. abs (∑ j∈δ-tuples ∩ {j. R-exponent i j = p}.
    (δ mchoose i) * mfact j * fact (δ − sum-list j) * P-coeff j * mpow z i))
    using sum-abs e-expression by simp
  also have ... ≤ (∑ i∈δ-tuples. fact δ * L * (int (δ mchoose i) * mpow z i))
    using bound-coeff
    by (simp add: ab-semigroup-mult-class.mult-ac(1) sum-mono)
  also have ... = fact δ * L * (∑ i∈δ-tuples. int (δ mchoose i) * mpow z i)
    by (simp only: sum-distrib-left)
  also have ... = fact δ * L * int (∑ i∈δ-tuples. (δ mchoose i) * mpow z i)
    using int-sum by simp
  also have ... = fact δ * L * int ((1 + sum-list z) ^δ)
    unfolding δ-tuples-def
    by (simp only: multinomial-ring-alt len-z of-nat-id)
   finally show ?thesis by simp
qed

lemma e-b-bound: shows 0 < e j + b and e j + b < B
proof −
  have 0: abs (e j) < b
  proof −
    have 2 * abs (e j) ≤ 2 * fact δ * L * (1 + sum-list z) ^δ
      using e-upper-bound by (simp add: algebra-simps)
    also have ... = int ( 2 * fact δ * (nat L) * (1 + sum-list z) ^δ)
      using int-ops L-pos by auto
    also have ... < int B using B-lower-bound
      by presburger
    finally show ?thesis using b-def-reverse by auto
  qed

  have e j < b using 0 by linarith
  thus e j + b < B using b-def-reverse by linarith

  have − e j < b using 0 by linarith
  thus 0 < e j + b by linarith
qed
```

This is a key step of the proof.

```
lemma K-expression: K = (∑ i≤(2*δ+1) * n ν. (e i + int b) * B ^i)
proof −
  have 0: insertion B-assign R = (∑ i≤(2*δ+1) * n ν. (e i) * B ^i)
  proof −
    have insertion B-assign R = (∑ i≤(2*δ+1) * n ν. (e i) * (insertion B-assign
X) ^i)
      by (simp add: R-sum-e)
    also have ... = (∑ i≤(2*δ+1) * n ν. (e i) * B ^i)
      unfolding X-def B-assign-def by simp
    finally show ?thesis .
```

```
  qed

  have K = insertion ℬ-assign R + S
    by (simp add: K-def)
  also have ... = (∑ i≤(2*δ+1) * n ν. e i * ℬ^i) + (∑ i≤(2*δ+1) * n ν. b *
ℬ^i)
    by (simp add: S-def 0)
  also have ... = (∑ i≤(2*δ+1) * n ν. (e i + int b) * ℬ^i)
    by (simp add: sum.distrib algebra-simps)
  finally show ?thesis .
qed

lemma δ-n-positive: 0 < (2*δ+1) * n ν
  using n-ge-1
  by (metis add-gr-0 mult-pos-pos not-gr0 not-one-le-zero zero-less-one)

lemma K-lower-bound: K > int ℬ^((2*δ+1) * n ν)
proof −
 from δ-n-positive have (∑ i=0..<(2*δ+1) * n ν. ℬ^i) = 1 + (∑ i=1..<(2*δ+1)
* n ν. ℬ^i)
    using sum.atLeast-Suc-lessThan power-0 by (metis One-nat-def)
  hence 0: (∑ i<(2*δ+1) * n ν. ℬ^i) > 0
    by (metis Suc-eq-plus1 add.commute atLeast0LessThan zero-less-Suc)

  have j ∈ {..(2*δ+1) * n ν} ⟹ 1 ≤ e j + b for j
    using int-one-le-iff-zero-less e-b-bound by blast
  hence 1: j ∈ {..(2*δ+1) * n ν} ⟹ ℬ^j ≤ (e j + b) * ℬ^j for j
    using ℬ-ge-1 by simp

  have int ℬ^((2*δ+1) * n ν) < int (∑ i<(2*δ+1) * n ν. ℬ^i) + ℬ^((2*δ+1)
* n ν)
    using 0 by (simp add: nat-int-comparison(2))
  also have ... = (∑ i<(2*δ+1) * n ν. int ℬ^i) + ℬ^((2*δ+1) * n ν)
    using Int.int-sum by simp
  also have ... = (∑ i≤(2*δ+1) * n ν. int ℬ^i)
   by (metis lessThan-Suc-atMost of-nat-eq-of-nat-power-cancel-iff sum.lessThan-Suc)
  also have ... ≤ K unfolding K-expression
    using sum-mono 1 by (metis (no-types, lifting) of-nat-power)
  finally show ?thesis .
qed

corollary K-gt-0: K > 0
  using ℬ-gt-2 δ-n-positive K-lower-bound
  by (smt (verit) of-nat-zero-less-power-iff power-eq-0-iff zero-less-power2)

end

end
theory Lemma-1-8
```

imports *Lemma-1-8-Coding*
begin

2.4.3 Proof of the equivalence

locale *Lemma-1-8* = *K-Nonnegative* +
assumes $\mathcal{B}$*-power2*: *is-power2* (*int* $\mathcal{B}$)
begin

lemma *b-power2*: *is-power2* (*int b*)
using *b-def int-ops is-power2-div2* $\mathcal{B}$*-power2* $\mathcal{B}$*-gt-2*
by (*smt* (*verit*) *Suc-1 of-nat-less-iff*)

lemma *K-upper-bound*: $K <$ *int* $\mathcal{B}\hat{}((2*\delta+1) * n\ \nu + 1)$
proof −
have $j \in \{..(2*\delta+1) * n\ \nu\} \Longrightarrow e\ j + b \leq \mathcal{B} - 1$ **for** j
using *e-b-bound* $\mathcal{B}$*-ge-1* **by** (*simp add*: *of-nat-diff*)
hence *0*: $j \in \{..(2*\delta+1) * n\ \nu\} \Longrightarrow (e\ j + b) * \mathcal{B}\hat{}j \leq (\mathcal{B} - 1) * \mathcal{B}\hat{}j$ **for** j
using $\mathcal{B}$*-ge-1* **by** *simp*

have *1*: $(\mathcal{B} - 1) * (\sum i{\leq}n.\ \mathcal{B}\hat{}i) = \mathcal{B}\hat{}(n+1) - 1$ **for** n
proof (*induction n*)
case *0* **show** *?case* **by** *simp*
next
case (*Suc n*)
have *int* $((\mathcal{B} - 1) * (\sum i{\leq}Suc\ n.\ \mathcal{B}\hat{}i)) =$
int $((\mathcal{B} - 1) * (\sum i{\leq}n.\ \mathcal{B}\hat{}i) + (\mathcal{B} - 1) * \mathcal{B}\hat{}(Suc\ n))$
by (*auto simp add*: *algebra-simps*)
also have ... = *int* $\mathcal{B}\hat{}(Suc\ n) - (1::int) + (\mathcal{B} - (1::int)) *$ *int* $\mathcal{B}\hat{}(Suc\ n)$
using *int-ops Suc.IH* $\mathcal{B}$*-ge-1* **by** *simp*
also have ... = *int* $\mathcal{B} *$ $\mathcal{B}\hat{}(Suc\ n) - 1$
using *int-ops* **by** (*simp add*: *algebra-simps*)
also have ... = *int* $(\mathcal{B}\hat{}(Suc\ (Suc\ n)) - 1)$
using $\mathcal{B}$*-ge-1 int-ops* **by** *simp*
finally show *?case*
by (*metis Suc-eq-plus1 of-nat-eq-iff*)
qed

have $K \leq (\sum i{\leq}(2*\delta+1) * n\ \nu.\ (\mathcal{B} - 1) *$ *int* $\mathcal{B}\hat{}i)$
unfolding *K-def* **using** *sum-mono 0*
by (*metis* (*no-types*, *lifting*) *K-def K-expression of-nat-mult of-nat-power*)
also have ... = *int* $((\mathcal{B} - 1) * (\sum i{\leq}(2*\delta+1) * n\ \nu.\ \mathcal{B}\hat{}i))$
by (*simp add*: *sum-distrib-left*)
also have ... = *int* $(\mathcal{B}\hat{}((2*\delta+1) * n\ \nu + 1) - 1)$
using *1* **by** *simp*
also have ... = $\mathcal{B}\hat{}((2*\delta+1) * n\ \nu + 1) - (1::int)$
using $\mathcal{B}$*-ge-1 int-ops*(*6*) **by** *auto*

```
  also have ... < int \<B>^((2*δ+1) * n ν + 1) by auto
  finally show ?thesis .
qed

lemma direct-implication:
  insertion z-assign P = 0 ⟹ τ (nat K) ((b−1) * \<B>^(n (ν+1))) = 0
proof −
  assume insertion z-assign P = 0
  hence assm: e (n (ν+1)) = 0 using e-n-ν1-expression by simp

  define deg where deg = (2*δ+1) * n ν
  define x where x = (λi. nat (e i + int b))
  define y where y = (λi. if i = n (ν+1) then b−1 else 0)

  have x-sum: nat K = (∑ i<deg+1. x i * \<B>^i)
  proof −
    have 0: (e i + int b) * \<B>^i ≥ 0 for i
      using e-b-bound(1) by (simp add: dual-order.strict-implies-order)

    have nat K = nat (∑ i≤deg. (e i + int b) * \<B>^i)
      unfolding K-expression deg-def by auto
    also have ... = (∑ i≤deg. nat ((e i + int b) * \<B>^i))
      using nat-sum-distrib 0 by auto
    also have ... = (∑ i≤deg. x i * \<B>^i)
      unfolding x-def
      by (metis (no-types, opaque-lifting) dual-order.strict-implies-order
        e-b-bound(1) int-ops(7) nat-0-le nat-int)
    finally show ?thesis
      using Suc-eq-plus1 lessThan-Suc-atMost by presburger
  qed

  have y-sum: (b−1) * \<B>^(n (ν+1)) = (∑ i<deg+1. y i * \<B>^i)
  proof −
    have 0: n (ν+1) ≤ deg unfolding deg-def n-def by auto

    have (∑ i∈{..deg}. y i * \<B>^i) =
      (∑ i∈{..deg} − {n (ν+1)}. y i * \<B>^i) + (∑ i∈{n (ν+1)}. y i * \<B>^i)
    by (meson 0 atMost-iff empty-subsetI finite-atMost insert-subset sum.subset-diff)
    also have ... = (∑ i∈{n (ν+1)}. y i * \<B>^i)
      unfolding y-def by simp
    also have ... = y (n (ν+1)) * \<B>^(n (ν+1))
      by simp
    also have ... = (b−1) * \<B>^(n (ν+1))
      unfolding y-def by simp
    finally show ?thesis
      using Suc-eq-plus1 lessThan-Suc-atMost by presburger
  qed

  have digit-cond: x i + y i < \<B> for i
```

```
  proof cases
    assume i = n (ν+1)
    hence x i + y i ≤ b + (b − 1)
      unfolding x-def y-def using assm by auto
    also have ... = ℬ − 1
      using b-def-reverse by auto
    also have ... < ℬ
      using ℬ-ge-1 by auto
    finally show ?thesis .
  next
    assume i ≠ n (ν+1)
    hence x i + y i ≤ nat (e i + int b)
      unfolding x-def y-def by auto
    also have ... < ℬ
      using e-b-bound by (smt (verit) nat-less-iff)
    finally show ?thesis .
  qed

  have τ-digits: τ (x i) (y i) = 0 for i
  proof cases
    assume i-def: i = n (ν+1)
    obtain k where b = 2^k using b-power2 is-power2-def by auto
    thus ?thesis using assm unfolding x-def y-def i-def
      using nat-int plus-int-code(2) count-carries-pow2-block-ones count-carries-0n
by presburger
  next
    assume i ≠ n (ν+1)
    thus ?thesis unfolding y-def by simp
  qed

  obtain k where k-def: ℬ = 2^k
    using ℬ-power2 is-power2-def by auto
  have k-ge-1: 1 ≤ k
    using k-def ℬ-ge-2
    by (metis ℬ-gt-2 antisym linorder-linear one-le-numeral order-less-le
        power-increasing power-one-right)

  have τ (nat K) ((b−1) * ℬ^(n (ν+1))) = (∑ i<deg+1. τ (x i) (y i))
    using count-carries-digitwise-no-overflow[OF k-ge-1] digit-cond k-def
    unfolding x-sum y-sum by blast
  also have ... = 0
    using τ-digits by simp
  finally show ?thesis .
qed


lemma reverse-implication:
  τ (nat K) ((b−1) * ℬ^(n (ν+1))) = 0 ⟹ insertion z-assign P = 0
proof −
```

assume *assm*: $\tau\ (nat\ K)\ ((b-1) * \mathcal{B}\hat{}(n\ (\nu+1))) = 0$

define *deg* **where** $deg = (2*\delta+1) * n\ \nu$
define *x* **where** $x = (\lambda i.\ nat\ (e\ i + int\ b))$
define *y* **where** $y = (\lambda i.\ if\ i = n\ (\nu+1)\ then\ b-1\ else\ 0)$

have *x-sum*: $nat\ K = (\sum i{<}deg+1.\ x\ i * \mathcal{B}\hat{}i)$
proof −
  have *0*: $(e\ i + int\ b) * \mathcal{B}\hat{}i \geq 0$ **for** *i*
    using *e-b-bound*(*1*) **by** (*simp add*: *dual-order.strict-implies-order*)

  have $nat\ K = nat\ (\sum i{\leq}deg.\ (e\ i + int\ b) * \mathcal{B}\hat{}i)$
    unfolding *K-expression deg-def* **by** *auto*
  also have $... = (\sum i{\leq}deg.\ nat\ ((e\ i + int\ b) * \mathcal{B}\hat{}i))$
    using *nat-sum-distrib 0* **by** *auto*
  also have $... = (\sum i{\leq}deg.\ x\ i * \mathcal{B}\hat{}i)$
    unfolding *x-def*
    by (*metis* (*no-types*, *opaque-lifting*) *dual-order.strict-implies-order*
      e-b-bound(*1*) *int-ops*(*7*) *nat-0-le nat-int*)
  finally show *?thesis*
    using *Suc-eq-plus1 lessThan-Suc-atMost* **by** *presburger*
qed

have *y-sum*: $(b-1) * \mathcal{B}\hat{}(n\ (\nu+1)) = (\sum i{<}deg+1.\ y\ i * \mathcal{B}\hat{}i)$
proof −
  have *0*: $n\ (\nu+1) \leq deg$ **unfolding** *deg-def n-def* **by** *auto*

  have $(\sum i{\in}\{..deg\}.\ y\ i * \mathcal{B}\hat{}i) =$
    $(\sum i{\in}\{..deg\} - \{n\ (\nu+1)\}.\ y\ i * \mathcal{B}\hat{}i) + (\sum i{\in}\{n\ (\nu+1)\}.\ y\ i * \mathcal{B}\hat{}i)$
  by (*meson 0 atMost-iff empty-subsetI finite-atMost insert-subset sum.subset-diff*)
  also have $... = (\sum i{\in}\{n\ (\nu+1)\}.\ y\ i * \mathcal{B}\hat{}i)$
    unfolding *y-def* **by** *simp*
  also have $... = y\ (n\ (\nu+1)) * \mathcal{B}\hat{}(n\ (\nu+1))$
    by *simp*
  also have $... = (b-1) * \mathcal{B}\hat{}(n\ (\nu+1))$
    unfolding *y-def* **by** *simp*
  finally show *?thesis*
    using *Suc-eq-plus1 lessThan-Suc-atMost* **by** *presburger*
qed

have *xy-bound*: $x\ i < \mathcal{B} \wedge y\ i < \mathcal{B}$ **for** *i*
  unfolding *x-def y-def* **using** *e-b-bound b-def*
 by (*smt* (*verit*, *best*) *$\mathcal{B}$-ge-1 div-less-dividend less-imp-diff-less less-numeral-extra*(*1*)

    nat-less-iff one-less-numeral-iff order-less-le-trans semiring-norm(*76*))

obtain *k* **where** *k-def*: $\mathcal{B} = 2\hat{}k$
  using *$\mathcal{B}$-power2 is-power2-def* **by** *auto*
have *k-ge-1*: $1 \leq k$

using *k-def* $\mathcal{B}$*-ge-2*
by (*metis* $\mathcal{B}$*-gt-2 antisym linorder-linear one-le-numeral order-less-le power-increasing power-one-right*)

have $n\ (\nu{+}1) < deg{+}1$
unfolding *deg-def n-def* **by** *auto*
hence $\tau\ (x\ (n\ (\nu{+}1)))\ (y\ (n\ (\nu{+}1))) \leq \tau\ (\sum i{<}deg{+}1.\ x\ i * \mathcal{B}\hat{}i)\ (\sum i{<}deg{+}1.\ y\ i * \mathcal{B}\hat{}i)$
using *count-carries-digitwise-specific*[*OF k-ge-1*] *xy-bound*
unfolding *k-def* **by** *meson*
hence $\tau\ (x\ (n\ (\nu{+}1)))\ (y\ (n\ (\nu{+}1))) = 0$
using *x-sum y-sum assm* **by** *auto*
hence $\tau\ (b{-}1)\ (nat\ (e\ (n\ (\nu{+}1)) + int\ b)) = 0$
using *x-def y-def n-def deg-def count-carries-sym* **by** *simp*
hence *b dvd nat* $(e\ (n\ (\nu{+}1)) + int\ b)$
using *count-carries-divisibility-pow2 b-power2* **unfolding** *is-power2-def* **by** *force*
hence $(int\ b)$ *dvd* $(e\ (n\ (\nu{+}1)) + int\ b)$
by (*metis dual-order.strict-iff-not e-b-bound*(*1*) *nat-0-le of-nat-dvd-iff*)
hence *b-dvd-e*: *b dvd e* $(n\ (\nu{+}1))$
by *auto*
hence $e\ (n\ (\nu{+}1)) = 0$
proof *cases*
assume $e\ (n\ (\nu{+}1)) = 0$ **thus** *?thesis* **.**
next
assume $e\ (n\ (\nu{+}1)) \neq 0$
hence $b \leq abs\ (e\ (n\ (\nu{+}1)))$
using *b-dvd-e* **by** (*simp add*: *zdvd-imp-le*)
hence $\mathcal{B} \leq 2 * abs\ (e\ (n\ (\nu{+}1)))$
using *b-def-reverse* **by** *auto*
hence $fact\ \delta * (nat\ L) * (1 + sum\text{-}list\ z)\hat{}\delta < abs\ (e\ (n\ (\nu{+}1)))$
using $\mathcal{B}$*-lower-bound* **by** *linarith*
thus *?thesis*
using *e-upper-bound L-pos dual-order.strict-iff-not*
by *fastforce*
qed
thus *?thesis* **using** *e-n-*ν*1-expression* **by** *auto*
qed

lemma *tau-rewrite*:
$\tau\ (2 * nat\ K)\ ((\mathcal{B}{-}2) * \mathcal{B}\hat{}(n\ (\nu{+}1))) = \tau\ (nat\ K)\ ((b{-}1) * \mathcal{B}\hat{}(n\ (\nu{+}1)))$
using *count-carries-even-even b-def-reverse*
by (*smt* (*verit*, *best*) *ab-semigroup-mult-class.mult-ac*(*1*) *mult-numeral-1-right numerals*(*1*) *right-diff-distrib'*)

lemma *lemma-1-8*:
shows *insertion z-assign* $P = 0 \longleftrightarrow \tau\ (2 * nat\ K)\ ((\mathcal{B}{-}2) * \mathcal{B}\hat{}(n\ (\nu{+}1))) = 0$

and $K > int\ \mathcal{B}\hat{}((2*\delta+1) * n\ \nu)$
and $K < int\ \mathcal{B}\hat{}((2*\delta+1) * n\ \nu + 1)$
using *K-lower-bound K-upper-bound direct-implication reverse-implication n-def tau-rewrite*
by *simp-all blast*

end

end
theory *Diophantine-Definition*
imports *MPoly-Utils/More-More-MPoly-Type*
begin

definition *is-nonnegative* :: $(nat \Rightarrow int) \Rightarrow bool$ **where**
$is\text{-}nonnegative\ f \equiv \forall i.\ f\ i \geq 0$

definition *is-diophantine-over-Z* :: *nat set* $\Rightarrow$ *bool* **where**
is-diophantine-over-Z A = $(\exists P.$
$(\forall a.\ (a \in A) \longleftrightarrow (\exists f.\ insertion\ (f(0 := int\ a))\ P = 0)))$

definition *is-diophantine-over-Z-with* :: *nat set* $\Rightarrow$ *int mpoly* $\Rightarrow$ *bool* **where**
is-diophantine-over-Z-with A P =
$(\forall a.\ (a \in A) \longleftrightarrow (\exists f.\ insertion\ (f(0 := int\ a))\ P = 0))$

definition *is-diophantine-over-N* :: *nat set* $\Rightarrow$ *bool* **where**
is-diophantine-over-N A = $(\exists P.$
$(\forall a.\ (a \in A) \longleftrightarrow (\exists f.\ insertion\ (f(0 := int\ a))\ P = 0 \wedge is\text{-}nonnegative\ f)))$

definition *is-diophantine-over-N-with* :: *nat set* $\Rightarrow$ *int mpoly* $\Rightarrow$ *bool* **where**
is-diophantine-over-N-with A P =
$(\forall a.\ (a \in A) \longleftrightarrow (\exists f.\ insertion\ (f(0 := int\ a))\ P = 0 \wedge is\text{-}nonnegative\ f))$

lemma *is-diophantine-finite-vars*:
assumes *is-diophantine-over-N-with A P*
shows $a \in A \longleftrightarrow (\exists f.\ insertion\ (f(0 := int\ a))\ P = 0 \wedge is\text{-}nonnegative\ f \wedge (\forall i > max\text{-}vars\ P.\ f\ i = 0))$
proof (*cases* $a \in A$)
case *True*
with *assms* **obtain** *f* **where** *f-def*: $insertion\ (f(0 := int\ a))\ P = 0 \wedge is\text{-}nonnegative\ f$
unfolding *is-diophantine-over-N-with-def* **by** *auto*
define f' **where** $f' \equiv (\lambda i.\ if\ i \leq max\text{-}vars\ P\ then\ f\ i\ else\ 0)$
have *1*: $\bigwedge v.\ v \in vars\ P \Longrightarrow (f(0 := int\ a))\ v = (f'(0 := int\ a))\ v$
unfolding *f'-def max-vars-def*
by *simp* (*subst Max.coboundedI*, *auto simp*: *vars-finite*)
have $insertion\ (f'(0 := int\ a))\ P = 0$
apply (*subst insertion-irrelevant-vars*[*of P - f*(0 := *int a*)])
unfolding *1* **by** (*auto simp*: *f-def*)
thus *?thesis*

apply (*simp add*: *True*)
apply (*rule exI*[*of* - *f'*])
using *f-def* **by** (*auto simp*: *f'-def is-nonnegative-def*)
next
case *False*
then show *?thesis*
using *assms* **unfolding** *is-diophantine-over-N-with-def* **by** *auto*
qed

end
theory *Total-Degree-Env*
imports *Total-Degree Substitutions*
begin

3 Bottom-up total_degree under a substitution-degree environment

lift-definition *total-degree-env*
:: (*nat* ⇒ *nat*) ⇒ 'a::*zero-neq-one mpoly* ⇒ *nat*
is λ*env p*. *Max* (*insert 0*
((λ*m*. *sum* (λ*i*. *env i* ∗ *lookup m i*) (*keys m*)) ‘
(*keys p*))) .

total-degree-env env p walks over the monomial representation of p, and whenever it sees $(Var\ i)^m$, it contributes $m * env\ i$ instead of m.

lemma *total-degree-env-id*:
total-degree-env (λ-. *1*) *p* = *total-degree p*
by *transfer simp*

lemma *total-degree-env-zero*[*simp*]: *total-degree-env f 0* = *0*
by (*simp add*: *total-degree-env.rep-eq zero-mpoly.rep-eq*)

lemma *total-degree-env-one*[*simp*]: *total-degree-env f 1* = *0*
by (*simp add*: *total-degree-env.rep-eq one-mpoly.rep-eq*)

lemma *total-degree-env-Const*[*simp*]: *total-degree-env f* (*Const c*) = *0*
by (*simp add*: *total-degree-env.rep-eq Const.rep-eq* $Const_0$*-def*)

lemma *total-degree-env-Const-le*: *total-degree-env f* (*Const c*) ≤ *0*
by *simp*

lemma *total-degree-env-Var*[*simp*]:
total-degree-env f (*Var i*) = *f i*
by (*simp add*: *total-degree-env.rep-eq Var.rep-eq* Var_0*-def*)

lemma *total-degree-env-Var-le*: *total-degree-env f* (*Var i*) ≤ *f i*
by *simp*

lemma *total-degree-env-neg*: *total-degree-env f* $(-P)$ = *total-degree-env f P*
by (*simp add*: *total-degree-env.rep-eq uminus-mpoly.rep-eq*)

lemma *total-degree-env-mult*: *total-degree-env f* $(P * Q) \leq$ *total-degree-env f P* + *total-degree-env f Q*
proof −
have $m \in$ *keys* (*mapping-of* $(P * Q)) \Longrightarrow$
sum $(\lambda i.\ f\ i *$ *lookup m i*) (*keys m*) $\leq$ *total-degree-env f P* + *total-degree-env f Q* **for** *m*
proof −
assume $m \in$ *keys* (*mapping-of* $(P * Q)$)
hence $m \in \{a + b \mid a\ b.\ a \in$ *keys* (*mapping-of P*) $\wedge\ b \in$ *keys* (*mapping-of Q*)$\}$
unfolding *times-mpoly.rep-eq* **using** *keys-mult* **by** *blast*
then obtain *a b* **where** *m-def*: $m = a + b$
and *a-key*: $a \in$ *keys* (*mapping-of P*)
and *b-key*: $b \in$ *keys* (*mapping-of Q*)
by *blast*
have *a-bound*: *sum* $(\lambda i.\ f\ i *$ *lookup a i*) (*keys a*) $\leq$ *total-degree-env f P*
unfolding *total-degree-env.rep-eq*
by (*rule Max-ge*, *simp-all add*: *a-key*)
have *b-bound*: *sum* $(\lambda i.\ f\ i *$ *lookup b i*) (*keys b*) $\leq$ *total-degree-env f Q*
unfolding *total-degree-env.rep-eq*
by (*rule Max-ge*, *simp-all add*: *b-key*)
show *sum* $(\lambda i.\ f\ i *$ *lookup m i*) (*keys m*) $\leq$ *total-degree-env f P* + *total-degree-env f Q*
unfolding *m-def*
apply (*subst setsum-keys-plus-distrib*, *simp-all add*: *algebra-simps*)
using *a-bound b-bound* **by** *linarith*
qed
thus *?thesis*
unfolding *total-degree-env.rep-eq* **by** *simp*
qed

lemma *total-degree-env-pow*: *total-degree-env f* $(P\ \hat{}\ n) \leq n *$ *total-degree-env f P*
by (*induction n*, *simp-all add*: *le-trans*[*OF total-degree-env-mult*])

lemma *total-degree-env-add*: *total-degree-env f* $(P + Q) \leq$ *max* (*total-degree-env f P*) (*total-degree-env f Q*)
proof −
have $m \in$ *keys* (*mapping-of* $(P + Q)) \Longrightarrow$
sum $(\lambda i.\ f\ i *$ *lookup m i*) (*keys m*) $\leq$ *max* (*total-degree-env f P*) (*total-degree-env f Q*) **for** *m*
proof −
assume $m \in$ *keys* (*mapping-of* $(P + Q)$)
hence *in-union*: $m \in$ *keys* (*mapping-of P*) $\cup$ *keys* (*mapping-of Q*)
unfolding *plus-mpoly.rep-eq* **using** *Poly-Mapping.keys-add* **by** *fastforce*
show *sum* $(\lambda i.\ f\ i *$ *lookup m i*) (*keys m*) $\leq$ *max* (*total-degree-env f P*) (*total-degree-env f Q*)
unfolding *total-degree-env.rep-eq*

by (*metis* (*mono-tags*, *lifting*) *Max-ge UnE finite-imageI finite-insert finite-keys imageI*
in-union insertCI le-max-iff-disj)
qed
thus *?thesis* **unfolding** *total-degree-env.rep-eq* **by** *simp*
qed

lemma *total-degree-env-diff*:
fixes $P :: 'a::\{ab\text{-}group\text{-}add,zero\text{-}neq\text{-}one\}$ *mpoly*
shows *total-degree-env f* $(P - Q) \leq$ *max* (*total-degree-env f P*) (*total-degree-env f Q*)
unfolding *diff-conv-add-uminus total-degree-env-neg*[*of f Q*, *symmetric*]
by (*rule total-degree-env-add*)

lemma *total-degree-env-sum*:
fixes $P :: 'a \Rightarrow 'b::\{ab\text{-}group\text{-}add,zero\text{-}neq\text{-}one\}$ *mpoly*
assumes *S-fin*: *finite S*
shows *total-degree-env ctxt* (*sum P S*) $\leq$ *Max* (*insert 0* ((λ*i*. *total-degree-env ctxt* (*P i*)) ‘ *S*))
apply (*induct rule*: *finite-induct*[*OF S-fin*], *simp-all*)
apply (*rule le-trans*[*OF total-degree-env-add*], *simp*)
using *order.trans* **by** *fastforce*

lemma *total-degree-env-prod*:
assumes *S-fin*: *finite S*
shows *total-degree-env ctxt* (*prod P S*) $\leq$ *sum* (λ*i*. *total-degree-env ctxt* (*P i*)) *S*
by (*induct rule*: *finite-induct*[*OF S-fin*], *simp-all add*: *le-trans*[*OF total-degree-env-mult*])

lemma *total-degree-env-poly-subst-monom*:
defines *degree-monom* ≡ (λ*m t*. (*lookup m*) *t*)
shows *total-degree-env ctxt* (*poly-subst-monom f m*)
$\leq$ ($\sum$ *t*∈*keys m*. *degree-monom m t* ∗ *total-degree-env ctxt* (*f t*))
unfolding *poly-subst-monom-alt degree-monom-def*
by (*simp add*: *le-trans*[*OF total-degree-env-prod*[*OF finite-keys*]]
le-trans[*OF sum-mono*[*OF total-degree-env-pow*]])

lemma *total-degree-env-poly-subst-list*:
fixes $p :: 'a::comm\text{-}ring\text{-}1$ *mpoly*
shows *total-degree-env ctxt* (*poly-subst-list fs p*)
$\leq$ *total-degree-env* (λ*m*. *total-degree-env ctxt* (*fs* $!_0$ *m*)) *p*
proof −
have *total-degree-env ctxt* (*poly-subst-list fs p*)
= *total-degree-env ctxt* ($\sum$ *m*. *Const* (*coeff p m*) ∗ *poly-subst-monom* (($!_0$) *fs*) *m*)
by (*simp add*: *poly-subst-list-def poly-subst-def*)
also have ... $\leq$ *Max* (*insert 0*
((λ*m*. *total-degree-env ctxt* (*Const* (*coeff p m*)
∗ *poly-subst-monom* (($!_0$) *fs*) *m*))
‘ (*keys* (*mapping-of p*))))

using
total-degree-env-sum
by (*metis* (*no-types*) *finite-keys poly-subst-alt poly-subst-def total-degree-env-sum*)
also have ... ≤ *Max* (*insert 0*
((λ*m*. *total-degree-env ctxt* (*Const* (*coeff p m*))
+ *total-degree-env ctxt* (*poly-subst-monom* ((!$_0$) *fs*) *m*))
‘ (*keys* (*mapping-of p*))))
using
total-degree-env-mult
by (*smt* (*verit*, *best*) *Max.insert Max-function-mono Max-mono empty-not-insert finite-imageI finite-insert finite-keys image-is-empty max-nat.left-neutral subsetI*)
also have ... ≤ *Max* (*insert 0*
((λ*m*. *total-degree-env ctxt* (*poly-subst-monom* ((!$_0$) *fs*) *m*))
‘ (*keys* (*mapping-of p*))))
by *simp*
also have ... ≤ *Max* (*insert 0*
((λ*m*. (∑ *i*∈*keys m*. *lookup m i* ∗ *total-degree-env ctxt* (*fs* !$_0$ *i*)))
‘ (*keys* (*mapping-of p*))))
using
total-degree-env-poly-subst-monom[*of* - λ*m*. *fs* !$_0$ *m*]
by (*metis* (*no-types*, *lifting*) *Max.insert Max-function-mono Orderings.order-eq-iff finite-imageI finite-keys image-is-empty max-nat.left-neutral*)
finally show *?thesis* **by** (*simp add*: *total-degree-env-def mult.commute*)
qed

lemma *total-degree-poly-subst-list-env*:
fixes *p* :: *'a*::*comm-ring-1 mpoly*
shows *total-degree* (*poly-subst-list fs p*)
≤ *total-degree-env* (λ*m*. *total-degree* (*fs* !$_0$ *m*)) *p*
using *total-degree-env-poly-subst-list total-degree-env-id*
by (*metis* (*lifting*) *ext*)

lemma *total-degree-env-Var-list-bound*: *total-degree-env* (λ-. *1*) ((*map Var ls*) !$_0$ *i*) ≤ *1*
by (*cases i* < *length ls*, *auto simp*: *nth0-def*)

lemma *total-degree-env-Var-list*:
total-degree-env (λ-. *1*) ((*map Var ls*) !$_0$ *i*) = (*if i* < *length ls then 1 else 0*)
by (*simp add*: *nth0-def*)

lemma *total-degree-map-Var*:
total-degree ((*map Var ls*) !$_0$ *j* :: *'a*::*comm-semiring-1 mpoly*) ≤ *1*
by (*cases j* < *length ls*; *auto simp add*: *nth0-def*)

lemma *total-degree-map-Var-int*:
total-degree ((*map Var ls*) !$_0$ *j* :: *int mpoly*) ≤ *Suc 0*
using *total-degree-map-Var* **by** *auto*

lemma *total-degree-env-mono3-map-Var*:
(⋀*i*. *env i* ≤ *1*) ⟹ *total-degree-env env* ((*map Var ls*) !$_0$ *j*) ≤ *1*
by (*cases j* < *length ls*, *simp-all add*: *nth0-def*)

lemma *total-degree-env-reduce*: *i* < *length ls*
⟹ *total-degree-env env* ((*ls* @ *xs*) !$_0$ *i*) = *total-degree-env env* (*ls* !$_0$ *i*)
unfolding *nth0-def* **by** (*simp add*: *nth-append-left*)

lemma *total-degree-env-mono*:
fixes *P* :: *int mpoly*
assumes ∀ *i*≤*max-vars P*. *env1 i* ≤ *env2 i*
shows *total-degree-env env1 P* ≤ *total-degree-env env2 P*
unfolding *total-degree-env.rep-eq* **apply** *simp*
apply (*rule*)
subgoal premises *h* **for** *m*
apply (*rule le-trans*[*of* - (∑ *i*∈*keys m*. *env2 i* ∗ *lookup m i*)])
subgoal
apply (*rule sum-mono*)
using *h assms after-max-vars*
by (*metis Suc-eq-plus1 in-keys-iff mult-le-mono1 not-less-eq-eq*)
subgoal
by (*meson Max-ge finite-imageI finite-insert finite-keys h image-eqI insert-iff*)
done
done

lemma *total-degree-env-mono2*:
fixes *P* :: *int mpoly*
shows *total-degree P* ≤ *rhs1* ⟹ (⋀*i*. *i* ≤ *max-vars P* ⟹ *env i* ≤ *1*) ⟹ *rhs1* = *rhs2*
⟹ *total-degree-env env P* ≤ *rhs2*
unfolding *total-degree-env-id*[*symmetric*]
by (*meson dual-order.trans total-degree-env-mono*)

lemma *total-degree-env-mono3-bounded*:
fixes *ls* :: *int mpoly list*
shows *j* ≤ *bound* ⟹ (⋀*i*. *i* ≤ *bound* ⟹ *env i* ≤ *1*) ⟹ *max-vars* (*ls* !$_0$ *j*) ≤ *bound*
⟹ *total-degree* (*ls* !$_0$ *j*) ≤ *Suc 0* ⟹ *total-degree-env env* (*ls* !$_0$ *j*) ≤ *Suc 0*
proof −
assume *a*: *j* ≤ *bound* **and** *b*: ⋀*i*. *i* ≤ *bound* ⟹ *env i* ≤ *1*
and *c*: *max-vars* (*ls* !$_0$ *j*) ≤ *bound* **and** *d*: *total-degree* (*ls* !$_0$ *j*) ≤ *Suc 0*

{
fix *a*

```
    assume a ∈ keys (mapping-of (ls !0 j))
    with c have ∀ i∈keys a. i ≤ bound
      unfolding max-vars-def vars-def by simp

    with b have (∑ i∈keys a. env i * lookup a i) ≤ sum (lookup a) (keys a)
      by (subst sum-mono, simp-all)
  } note e = this

  from d e show ?thesis
    apply (cases j < length ls, simp-all add: nth0-def)
    unfolding total-degree-env-id[symmetric] total-degree-env.rep-eq
    using dual-order.trans by (auto, blast)
qed

lemma total-degree-env-mono3:
  (⋀i. env i ≤ 1) ⟹ total-degree (ls !0 j) ≤ 1
    ⟹ total-degree-env env (ls !0 j) ≤ 1
proof −
  assume a: (⋀i. env i ≤ 1) and b: total-degree (ls !0 j) ≤ 1

  {
    fix a
    assume a ∈ keys (mapping-of (ls ! j))
    have (∑ i∈keys a. env i * lookup a i) ≤ sum (lookup a) (keys a)
      apply (rule sum-mono)
      using a by simp
  } note c = this

  from a b show ?thesis
    apply (cases j < length ls, simp-all add: nth0-def )
    unfolding total-degree-env-id[symmetric] total-degree-env.rep-eq
    using c dual-order.trans by (auto, blast)
qed

lemma total-degree-env-mono3':
  (⋀i. env i ≤ Suc 0) ⟹ total-degree (ls !0 j) ≤ Suc 0
    ⟹ total-degree-env env (ls !0 j) ≤ Suc 0
  using total-degree-env-mono3 by auto

lemma total-degree-env-mono4:
  (⋀i. env i ≤ 1) ⟹ total-degree-env (λ-. 1) (ls !0 j) ≤ 1
    ⟹ total-degree-env env (ls !0 j) ≤ 1
  unfolding total-degree-env-id by (rule total-degree-env-mono3, simp-all)

end
theory Suitable-For-Coding
 imports ../Diophantine-Definition HOL−Library.Rewrite MPoly-Utils/Total-Degree-Env
begin
```

3.1 Polynomials suitable for coding

definition *fresh-var* :: *int mpoly* ⇒ *nat* **where**
fresh-var P = (*if vars P* = {} *then 1 else max-vars P* + *1*)

definition *suitable-for-coding* :: *int mpoly* ⇒ *int mpoly* **where**
suitable-for-coding P ≡ P^2 + (*Var* (*fresh-var P*) − *1*)^2

lemma *suitable-for-coding-degree-vars*:
shows *degree* (*suitable-for-coding P*) (*fresh-var P*) > *0*
vars (*suitable-for-coding P*) = *insert* (*fresh-var P*) (*vars P*)
proof −
have *Suc* (*Max* (*vars P*)) ∉ *vars P*
by (*meson Max-ge Suc-n-not-le-n vars-finite*)

hence *fresh-var P* ∉ *vars P*
proof (*cases vars P* = {})
case *True*
then show *?thesis* **by** *auto*
next
case *False*
show *?thesis* **unfolding** *fresh-var-def*
unfolding *max-vars-of-nonempty*[*OF False*]
using ‹*Suc* (*Max* (*vars P*)) ∉ *vars P*› **by** *simp*
qed
hence *0*: *degree P* (*fresh-var P*) = *0*
unfolding *in-vars-non-zero-degree* **by** *blast*

have *1*: *degree* (P^2) (*fresh-var P*) = *0*
apply (*subst degree-pow-positive*)
subgoal by *simp*
subgoal using *0* **by** *simp*
done

have *2*: *degree* ((*Var* (*fresh-var P*) :: *int mpoly*)^2) (*fresh-var P*) = *2*
by (*subst degree-pow-positive*; *simp*)

have *3*: *degree* ((*Const 1* :: *int mpoly*)^2) (*fresh-var P*) = *0*
by (*subst degree-pow-positive*; *simp*)

have *4*: *degree* ((*Const 2* :: *int mpoly*) ∗ *Var* (*fresh-var P*) ∗ *Const 1*) (*fresh-var P*) = *1*
apply (*subst degree-mult-non-zero*)
subgoal by (*simp-all add*: *Const-numeral Var-neq-zero*)
subgoal by (*simp add*: *Const-one*)
subgoal
apply (*subst degree-mult-non-zero*)
subgoal by (*simp add*: *Const-numeral*)
subgoal by (*simp add*: *Var-neq-zero*)
subgoal by *simp*

```
    done
  done

 have 5: degree ((Var (fresh-var P) :: int mpoly)^2 + (Const 1)^2) (fresh-var P) =
2
  apply (subst degree-add-different-degree)
  unfolding 2 3 apply auto
  done

 have 6: degree ((Var (fresh-var P) :: int mpoly)^2 + (Const 1)^2 − Const 2 ∗ Var
(fresh-var P) ∗ Const 1) (fresh-var P) = 2
  apply (subst degree-diff-different-degree)
  unfolding 4 5 apply auto
  done

 show degree (suitable-for-coding P) (fresh-var P) > 0
  unfolding suitable-for-coding-def power2-diff
  unfolding Const-numeral[symmetric] Const-one[symmetric]
  apply (subst degree-add-different-degree[of - fresh-var P])
  subgoal unfolding 1 6 by auto
  subgoal unfolding 1 6 by auto
  done

 have 7: vars (P^2) = vars P
  by (subst vars-pow-positive; auto)

 have 8: vars ((Const 1)^2 :: int mpoly) = {}
  by (subst vars-pow-positive; auto)

 have 9: vars ((Var (fresh-var P) :: int mpoly)^2) = {fresh-var P}
  by (subst vars-pow-positive; auto)

 have 10: vars ((Var (fresh-var P) :: int mpoly)^2 + (Const 1)^2) = {fresh-var P}
  apply (subst vars-add-different-degree)
  unfolding 8 9 apply auto
  done

 have 11: vars ((Const 2 :: int mpoly) ∗ Var (fresh-var P) ∗ Const 1) = {fresh-var
P}
  apply (subst vars-mult-non-zero)
  subgoal by (simp add: Const-numeral Var-neq-zero)
  subgoal by (simp add: Const-one)
  subgoal
    apply (subst vars-mult-non-zero)
    subgoal by (simp add: Const-numeral)
    subgoal by (simp add: Var-neq-zero)
    subgoal by simp
    done
  done
```

have *12*: *vars* ((*Var* (*fresh-var P*) :: *int mpoly*)^2 + (*Const 1*)^2 − *Const 2* ∗ *Var* (*fresh-var P*) ∗ *Const 1*) = {*fresh-var P*}
apply (*subst vars-diff-different-degree*)
unfolding *10 11*
using *4 5* **apply** *auto*
done

show *vars* (*suitable-for-coding P*) = *insert* (*fresh-var P*) (*vars P*)
unfolding *suitable-for-coding-def power2-diff*
unfolding *Const-numeral*[*symmetric*] *Const-one*[*symmetric*]
apply (*subst vars-add-different-degree*)
subgoal unfolding *7 12* **using** *1 6* **by** *auto*
subgoal
apply (*subst vars-diff-different-degree*)
subgoal unfolding *10 11* **using** *4 5* **by** *auto*
subgoal
apply (*subst vars-add-different-degree*)
subgoal unfolding *8* **by** *auto*
subgoal
apply (*subst vars-pow-positive*[*of* - *Var* (*fresh-var P*)])
subgoal by *simp*
subgoal unfolding *7 8 11* **by** *simp*
done
done
done
done
qed

lemma *suitable-for-coding-coeff0*:
fixes *P*
defines *n* ≡ *max-vars* (*suitable-for-coding P*)
defines *m0* ≡ *Abs-poly-mapping* ((!$_0$) (*replicate* (*n*+*1*) *0*))
shows *coeff* (*suitable-for-coding P*) *m0* > *0*
proof −
have *h0*: *nth-default 0* (*replicate* (*n*+*1*) *0*) = (λ*x*. *0*)
unfolding *nth-default-def*
using *List.nth-replicate* **by** *fastforce*

have *h1*: *m0* = *0*
unfolding *m0-def*
apply (*subst zero-poly-mapping-def*)
apply (*subst Abs-poly-mapping-inject*)
unfolding *nth0-def when-def* **apply** *simp-all*
using *List.nth-replicate* **by** *fastforce*

have *insertion* (*nth-default 0* (*replicate* (*n*+*1*) *0*)) (*suitable-for-coding P*) = *coeff* (*suitable-for-coding P*) *m0*

```
    using insertion-trivial[of suitable-for-coding P]
    by (subst h0, subst h1, simp)

  moreover have insertion (nth-default 0 (replicate (n+1) 0)) (suitable-for-coding
P) > 0
    unfolding suitable-for-coding-def fresh-var-def nth-default-def
    by (simp add: suitable-for-coding-degree-vars(2) fresh-var-def n-def vars-finite)

  ultimately show ?thesis
    by auto
qed


lemma suitable-for-coding-max-vars:
  assumes vars P ≠ {}
  shows max-vars (suitable-for-coding P) = max-vars P + 1
proof −
  have 0: vars (suitable-for-coding P) ⊆ vars P ∪ {fresh-var P}
    unfolding suitable-for-coding-degree-vars by simp

  have 1: Max (vars P ∪ {fresh-var P}) = fresh-var P
    unfolding max-vars-def
    apply (subst Max-Un)
    unfolding max-vars-def fresh-var-def
    apply (auto simp: assms vars-finite)
    done

  have 2: Max (insert 0 (vars (suitable-for-coding P))) = Max (vars (suitable-for-coding
P))
    by (simp add: suitable-for-coding-degree-vars(2) vars-finite)

  have max-vars (suitable-for-coding P) ≤ fresh-var P
    unfolding max-vars-def
    apply (subst 1[symmetric]) unfolding 2
    apply (rule Max-mono[OF 0])
    using suitable-for-coding-degree-vars(2)[of P]
    apply (auto simp: vars-finite)
    done

  thus ?thesis
    using assms suitable-for-coding-degree-vars(2)[of P]
    unfolding fresh-var-def max-vars-def
    by (simp add: order-antisym-conv vars-finite)
qed


lemma suitable-for-coding-diophantine-equivalent:
  fixes P :: int mpoly
  assumes insertion (z(0 := a)) (suitable-for-coding P) = 0 and is-nonnegative
```

```
z
  shows ∃ y. insertion (y(0 := a)) P = 0 ∧ is-nonnegative y
proof (rule exI[of - z], rule conjI)
  show insertion (z(0 := a)) P = 0
  proof cases
    assume vars P = {}
    thus ?thesis
      using assms(1) unfolding suitable-for-coding-def by auto
  next
    assume vars P ≠ {}
    thus ?thesis
      using assms(1) unfolding suitable-for-coding-def by auto
  qed

  show is-nonnegative z
    using assms(2) by auto
qed

lemma suitable-for-coding-exists-positive-unknown:
  fixes P :: int mpoly
  assumes dioph: is-diophantine-over-N-with A P
  assumes a: a ∈ A
  assumes insertion (y(0 := a)) P = 0 and is-nonnegative y
  shows ∃ z. insertion (z(0 := a)) (suitable-for-coding P) = 0
        ∧ (∃ i∈{1..fresh-var P}. z i > 0)
        ∧ (∀ i > fresh-var P. z i = 0)
        ∧ is-nonnegative z
proof cases
  assume 0: vars P = {}

  define z where z ≡ (λi. if i > 1 then 0 else (y(1 := 1)) i)

  show ?thesis
    unfolding suitable-for-coding-def
    apply (rule exI[of - z])
    using assms 0
    unfolding fresh-var-def is-nonnegative-def z-def
    apply simp
    apply (subst insertion-irrelevant-vars[of P - y(0 := a)])
    apply auto
    done
next
  assume 1: vars P ≠ {}

  define z :: nat ⇒ int where z ≡ (λi. if i > fresh-var P then 0 else (y(fresh-var
P := 1)) i)

  have 2: insertion (z(0 := a)) P = 0
  proof (subst insertion-irrelevant-vars[of P z(0 := a) y(0 := a)])
```

```
    show v ∈ vars P ⟹ (z(0 := int a)) v = (y(0 := int a)) v for v
      unfolding fresh-var-def max-vars-of-nonempty[OF 1] z-def
      using 1 by (simp; metis Max-ge le-eq-less-or-eq not-less-eq-eq vars-finite)
    show insertion (y(0 := int a)) P = 0
      by (auto simp: assms)
  qed

  show ?thesis
  proof (rule exI[of - z]; rule conjI[OF - conjI[OF - conjI]])
    show insertion (z(0 := a)) (suitable-for-coding P) = 0
      using 1 2
      unfolding suitable-for-coding-def fresh-var-def z-def
      by auto

    show ∃ i∈{1..fresh-var P}. z i > 0
      apply (rule bexI[of - fresh-var P])
      unfolding fresh-var-def max-vars-def z-def
      apply auto
      done

    show ∀ i>fresh-var P. z i = 0
      unfolding z-def by auto

    show is-nonnegative z
      using assms(4) by (auto simp: is-nonnegative-def z-def)
  qed
qed

lemma suitable-for-coding-total-degree:
  shows total-degree (suitable-for-coding P) > 0
  using total-degree-zero-degree-zero suitable-for-coding-degree-vars(1)[of P]
  by fastforce


lemma suitable-for-coding-total-degree-bound:
  assumes total-degree P > 0
  shows total-degree (suitable-for-coding P) ≤ 2 * total-degree P
  unfolding suitable-for-coding-def
  unfolding total-degree-env-id[symmetric]
  apply (rule le-trans[OF total-degree-env-add])
  apply (rule max.boundedI)
  subgoal by (simp add: total-degree-env-pow)
  subgoal
    apply (rule le-trans[OF total-degree-env-pow])
    apply (rule mult-le-mono2)
    apply (rule le-trans[OF total-degree-env-diff], simp) using total-degree-env-id
    by (metis One-nat-def assms le-zero-eq less-imp-neq not-less-eq-eq)
  done
```

end
theory *Poly-Degree*
imports *More-More-MPoly-Type Total-Degree-Env Poly-Extract*
keywords *poly-degree* :: *thy-defn* **and** |
begin

ML‹
```
(* intermediate representation of degree expressions *)
structure Degree-Expr = struct
  datatype expr =
    Const of int
  | Var   of string
  | Add   of expr * expr
  | Max   of expr * expr
  | Mul   of expr * expr
  | Term  of term (* New constructor for Isabelle terms *)

  fun c k       = Const k
  fun v name    = Var name
  fun add (x,y) = Add (x,y)
  fun max (x,y) = Max (x,y)
  fun mul (x,y) = Mul (x,y)
  fun exp tm    = Term tm (* Constructor function for exponents *)

  (* convert expr into a HOL term of type nat *)
  fun to-hol expr =
    let
      fun numeral k = HOLogic.mk-number @{typ nat} k
      val plus = Term.Const (@{const-name Groups.plus-class.plus}, @{typ nat ⇒
nat ⇒ nat})
      val times = Term.Const (@{const-name Groups.times-class.times}, @{typ nat
⇒ nat ⇒ nat})
      val maxc = Term.Const (@{const-name Orderings.ord-class.max}, @{typ nat
⇒ nat ⇒ nat})
      fun conv (Const k)       = numeral k
        | conv (Var x)         = Free (deg- ^ x, @{typ nat})
        | conv (Add(a,b))      = plus $ conv a $ conv b
        | conv (Mul(a,b))      = times $ conv a $ conv b
        | conv (Max(a,b))      = maxc $ conv a $ conv b
        | conv (Term tm)       = tm
    in conv expr end

  (* simplify expr *)
  fun norm (Add (Const a, Const b)) = Const (a + b)
    | norm (Max (Const a, Const b)) = Const (Int.max (a, b))
    | norm (Mul (Const a, Const b)) = Const (a * b)
    | norm (Add (x, y)) = norm (Add (norm x, norm y))
    | norm (Max (x, y)) = norm (Max (norm x, norm y))
    | norm (Mul (x, y)) = norm (Mul (norm x, norm y))
```

```
    | norm e = e
end

fun to-sexpr-untyped (t: term) =
  case t of
     f $ x => (apply  ^ to-sexpr-untyped f ^   ^ to-sexpr-untyped x ^ )
   | Const (n, -) => (const  ^ n ^  )
   | Free (n, -) => (free  ^ n ^ )
   | Var (n, -) => (var  ^  @{make-string} n ^ )
   | Bound n => (bound  ^ @{make-string} n ^ )
   | Abs (n, -, e) => (bound  ^ n ^   ^ to-sexpr-untyped e ^   )

(* recursively compute the degree expression of a term*)
fun compute-degree-expr ctxt deg-env deg-dict tm =
  let
    val (f, args) = Term.strip-comb tm

    fun schematic-to-free tm = case tm of
        Var ((name, -), @{typ int mpoly}) => Free (name, @{typ int mpoly})
      | a $ b => (schematic-to-free a) $ (schematic-to-free b)
      | x => x
    fun close exp = schematic-to-free exp

    fun dest-nat-numeral t =
      let val (-, k) = HOLogic.dest-number t in k end
  in
    case (f, args) of
      (* constants → degree 0 *)
        (* handle MPoly constants *)
      (Const (MPoly-Type.Const, -), -) =>
        Degree-Expr.c 0
        (* handle literal numerals *)
   | (Const (@{const-name Num.numeral-class.numeral}, -), -) =>
        Degree-Expr.c 0
   | (Const (@{const-name Groups.zero-class.zero}, -), -) =>
        Degree-Expr.c 0
   | (Const (@{const-name Groups.one-class.one}, -), -) =>
        Degree-Expr.c 0

      (* variables → look up index and apply deg-env *)
   | (Const (MPoly-Type.Var, -), [n]) =>
        let val i = dest-nat-numeral n in
          deg-env i (* default: Degree-Expr.c 1 *)
        end

      (* addition/subtraction → max of degrees *)
   | (Const (@{const-name Groups.plus-class.plus}, -), [s, t]) =>
        Degree-Expr.max (compute-degree-expr ctxt deg-env deg-dict s,
                         compute-degree-expr ctxt deg-env deg-dict t)
```

```
  | (Const (@{const-name Groups.minus-class.minus}, -), [s, t]) =>
      Degree-Expr.max (compute-degree-expr ctxt deg-env deg-dict s,
                       compute-degree-expr ctxt deg-env deg-dict t)

    (* multiplication → sum of degrees *)
  | (Const (@{const-name Groups.times-class.times}, -), [s, t]) =>
      Degree-Expr.add (compute-degree-expr ctxt deg-env deg-dict s,
                       compute-degree-expr ctxt deg-env deg-dict t)

    (* power → multiply degree of base by exponent *)
  | (Const (@{const-name Power.power-class.power}, -), [base, exp]) =>
      let
        val d-base = compute-degree-expr ctxt deg-env deg-dict base
        val (f-exp, args-exp) = Term.strip-comb exp
        val d-exp =
          case (f-exp, args-exp) of
            (Const (@{const-name Num.numeral-class.numeral}, -), []) =>
              Degree-Expr.c (dest-nat-numeral exp)
            | - => Degree-Expr.exp (close exp)
      in
        Degree-Expr.mul (d-exp, d-base)
      end

    (* substitution → deg-env updated for each substitution term *)
  | (Const (Substitutions.poly-subst-list, -), [subs, body]) =>
      let
        val subst-terms = HOLogic.dest-list subs

          val subst-degrees = map (compute-degree-expr ctxt deg-env deg-dict)
subst-terms

        fun new-env i =
          (List.nth (subst-degrees, i))
          handle Subscript => error
              Referencing an ill−defined substitution! Given list has less elements
than referred to.
      in
        compute-degree-expr ctxt new-env deg-dict body
      end

    (* dependent MPoly → use dependency theorem or unfold definition *)
  | (Const (hd, -), -) =>
      let
          val closed-args = map close args
          val deg-opt = Symtab.lookup deg-dict hd
      in
        (case deg-opt of
         (* Case 1: fall back to a ...-degree-correct dependency if one exists*)
```

```
        SOME deg-thm =>
          let
            val rhs-term = (case Thm.prop-of deg-thm of
             Const (@{const-name HOL.Trueprop}, -) $ (Const (@{const-name
Orderings.less-eq}, -) $ - $ r) => r      (* c ≤ rhs *)
              | Const (Pure.imp, -) $ - $ (Const (@{const-name HOL.Trueprop},
-) $ (Const (@{const-name Orderings.less-eq}, -) $ - $ r)) => r
                | t => raise TERM (poly-deg-simps is not an inequality, [t]))
              |> Term.close-schematic-term
            val full-rhs = Term.betapplys (rhs-term, closed-args)
           in
            Degree-Expr.exp full-rhs
          end
        (* Case 2: find and unfold the definition of the dependent MPoly*)
        | NONE =>
          let
            val def-thm = Proof-Context.get-thm ctxt (hd ^ -def)
         val rhs-term = Thm.rhs-of def-thm |> Thm.term-of |> Term.close-schematic-term
            val full-rhs = Term.betapplys (rhs-term, closed-args)
          in
            compute-degree-expr ctxt deg-env deg-dict full-rhs
          end)
      end
    (* unrecognized → error*)
  | (hd, -) => Degree-Expr.v (to-sexpr-untyped hd)
 end

(* generate theorem statement for poly-degree-correct *)
fun make-corr-thm-term (poly-term : term) (poly-degree-term : term) : term =
 let
   val T = fastype-of poly-term
    val total-degree-const = Const (@{const-name total-degree}, T --> @{typ
nat})
   val lhs = total-degree-const $ poly-term
   val rhs = poly-degree-term
 in
    HOLogic.mk-Trueprop (HOLogic.mk-binrel @{const-name Orderings.less-eq}
(lhs, rhs))
 end

(* get definition theorems of dependent MPoly *)
fun get-deps ctxt exclude input =
 let
   val (hd, args) = strip-comb input

   fun get-def-term name =
     (case Thm.prop-of (Proof-Context.get-thm ctxt (name ^ -def)) of
       Const (@{const-name Pure.eq}, -) $ - $ r => r      (* c ≡ rhs *)
       | Const (@{const-name HOL.eq}, -) $ - $ r => r      (* c  = rhs *)
```

```
      | t => raise TERM (poly-degree: not a definition, [t]))

    fun is-allowed s = not
                       (fold (fn a => fn b => a orelse b)
                        (map (fn r => String.isSubstring r s) exclude)
                        false)

    val args-deps = maps (get-deps ctxt exclude) args

    val hd-deps =
      case hd of
        Bound - => []
      | Const (Num.num.One, -) => []
      | Const (Num.num.Bit0, -) => []
      | Const (Num.num.Bit1, -) => []
      | Const (Groups.zero-class.zero, -) => []
      | Const (Groups.one-class.one, -) => []
      | Const (Num.numeral-class.numeral, -) => []
      | Const (Nat.semiring-1-class.of-nat, -) => []
      | Const (cname, T) =>
          if is-allowed cname andalso
             (T = @{typ int mpoly} orelse T = @{typ int mpoly ⇒ int mpoly})
          then
            cname :: get-deps ctxt exclude (get-def-term cname)
          else []
      | - => []
  in
    hd-deps @ args-deps
  end

fun prove-total-degree-var-list context N =
  let
    (* Returns the term for n in unary notation, i.e., Suc (Suc ... 0) *)
     fun mk-unary-nat 0 = Const (@{const-name Groups.zero-class.zero}, @{typ
nat})
       | mk-unary-nat n = Const (@{const-name Nat.Suc}, @{typ nat ⇒ nat}) $
mk-unary-nat (n − 1);

    (* Generates a list of terms [0, Suc 0, ..., Suc ... (Suc 0)] up to limit−1 *)
    fun unary-nat-list limit = HOLogic.mk-list @{typ nat} (map mk-unary-nat (0
upto (limit − 1)))
                               |> Thm.cterm-of context

    val lists = (map unary-nat-list (0 upto N))
  in
    map (Simplifier.simplify context)
      (map (fn ls => Drule.infer-instantiate′ context [SOME ls, NONE] @{thm
total-degree-map-Var})
       lists)
```

```
  end

fun unfold-nth0 (context : Proof.context) =
  Local-Defs.unfold-tac context ([@{thm nth0-Cons-0}, @{thm nth0-Cons-Suc}] @
prove-nat-normal context)

val PREPROCESSING-THMS : thm list = [
  @{thm Notation.Const-numeral[symmetric]},
  @{thm total-degree-env-id[symmetric]},
  @{thm nth0-Cons-0},
  @{thm nth0-Cons-Suc}]


val DEG-ENV-THMS : thm list = [
  @{thm le-trans[OF total-degree-env-Const-le]},
  @{thm le-trans[OF total-degree-env-Var-le]},
  @{thm total-degree-neg},
  @{thm le-trans[OF total-degree-env-add max.mono]},
  @{thm le-trans[OF total-degree-env-diff max.mono]},
  @{thm le-trans[OF total-degree-env-mult add-mono]},
  @{thm le-trans[OF total-degree-env-pow mult-le-mono2]},
  @{thm le-trans[OF total-degree-env-poly-subst-list]}]

(* prove correctness theorem *)
fun prove-corr-thm lthy' (corr-thm : term) def-deps deg-deps (auxthms : thm list)
=
   Goal.prove lthy' [] [] corr-thm
     (fn {context: Proof.context, prems = -: thm list} =>
       let
         val mk-le-trans2 = (fn th => th RSN (2, @{thm le-trans}))
         fun mk-env-mono th = @{thm total-degree-env-mono2} OF [th]
         val deg-rules  = map mk-env-mono deg-deps

         val trans-auxiliary-theorems = map mk-le-trans2 auxthms
       in
         Local-Defs.unfold-tac context (def-deps @ PREPROCESSING-THMS)
         THEN REPEAT (
           REPEAT-FIRST (resolve-tac context (
             deg-rules @ DEG-ENV-THMS @ [@{thm impI}]
           ))
           THEN unfold-nth0 context
         )
       THEN TRY (REPEAT-SOME (dresolve-tac context trans-auxiliary-theorems))
         THEN ((
           SOLVE (
         TRY (REPEAT-SOME (resolve-tac context [@{thm total-degree-env-mono3'}]))
             THEN ALLGOALS (simp-tac (context addsimps
                       (@{thms algebra-simps} @ prove-total-degree-var-list context
20)))
```

```
      )
    ) ORELSE (
   TRY (REPEAT-SOME (resolve-tac context [@{thm total-degree-env-mono3-bounded}]))
      THEN (auto-tac (context addsimps auxthms
        @ @{thms algebra-simps}
        @ prove-total-degree-var-list context 20
      ))
    ))
  end
  )

(* register the local-theory command *)
val - =
  Outer-Syntax.local-theory command-keyword ‹poly-degree›
    find the upper bound of the total degree of a multivariate polynomial and prove
correctness
   (Parse.name -- (Scan.option (keyword ‹|› |-- Parse.thms1)) >> (fn (raw-poly-str,
args) => fn lthy : local-theory =>
      let
        (* read & typecheck term in this theory context *)
        val input-thms = case args of
          NONE => []
        | SOME args => let
            val input-facts = map fst args
         val input-thms = List.concat (map (Proof-Context.get-fact lthy) input-facts)
            in
              input-thms
            end

        val poly-tm = Syntax.read-term lthy raw-poly-str

        (* peel off the head constant to build names, extract body *)
        val (hd, -) = Term.strip-comb poly-tm
        val short =
            (case hd of
               Const (s, -) => Binding.name-of (Binding.qualified-name s)
             | Free  (s, -) => Binding.name-of (Binding.qualified-name s)
             | -            => error poly-degree: expected a constant or free at head)
        val poly-def-thm =
            (case Term.strip-comb poly-tm of
              (Const (cname, -), -) => Proof-Context.get-thm lthy (cname ^ -def)
              | - => raise TERM (Could not locate a corresponding definition, []))
        val poly-body =
            (case Thm.prop-of poly-def-thm of
              Const (@{const-name Pure.eq}, -) $ - $ r => r     (* c ≡ rhs *)
            | Const (@{const-name HOL.eq},  -) $ - $ r => r     (* c  = rhs *)
            | t => raise TERM (poly-degree: not a definition, [t]))

        fun fully-qualify-xstring ctxt xstr =
```

```
        let
         val const = Proof-Context.read-const {proper = false, strict = false} ctxt
xstr
          val (qname, -) = Term.dest-Const const
        in
          qname
        end

      (* get dependent theorems *)
      val deg-dep-names = [coding-variables.D-poly, coding-variables.K-poly] @
                          [combined-variables.R-poly, combined-variables.S-poly,
combined-variables.T-poly,
                           combined-variables.A3-poly, combined-variables.A2-poly,
combined-variables.A1-poly,
                     combined-variables.X-poly, combined-variables.Y-poly]

      val f = try (fn n => Proof-Context.get-thm lthy (n ^ -degree-correct))
    val deg-dict-opt = fold (fn key => Symtab.update (key, f key)) deg-dep-names
Symtab.empty

      val g = fully-qualify-xstring lthy
       val deg-dict-aux = Symtab.fold (fn (k, SOME v) => Symtab.update (g k,
v) | (-, NONE) => I)
                         deg-dict-opt Symtab.empty

      fun u name =
        let val key = f name in
          if is-some key then Symtab.update (g name, the key) else I
        end

      (* for calculation *)
      val deg-dict = deg-dict-aux |> u coding-variables.M-poly
      val deg-thms = map #2 (Symtab.dest (deg-dict))

      (* for proving correctness *)
      val deg-thms-for-proof = (let val x = f combined-variables.M-poly in
                                  if is-some x then [the x] else [] end)
        @ map #2 (Symtab.dest deg-dict-aux)

      val disallowed-deps = P1 :: suitable-for-coding :: Symtab.keys deg-dict

       val (-, poly-body-content) = strip-abs poly-body; (* if poly-body might be an
Abs *)
      val deps = get-deps lthy disallowed-deps poly-body-content;
      val deps-unique = Library.distinct (op =) deps

      val - = Pretty.writeln (Pretty.str-list Definitions:   deps-unique);

    val def-deps-opt = (deps-unique) |> map (try (fn n => Proof-Context.get-thm
```

```
lthy (n ^ -def)))
      val def-deps = map the (List.filter is-some def-deps-opt)

      val - = Pretty.writeln (Pretty.big-list Dependency Theorems:
                        (map (Thm.pretty-thm lthy) (def-deps @ deg-thms)));

      (* compute the symbolic degree AST & lower to a HOL term *)
      fun init-env (-: int) = Degree-Expr.Const 1
      val expr   = compute-degree-expr lthy init-env deg-dict poly-body
      val deg-tm = Degree-Expr.to-hol expr

      (* build and add the '-degree' definition *)
      val def-name  = short ^ -degree
      val def-const = Free (def-name, @{typ nat})
      val eqn       = Logic.mk-equals (def-const, deg-tm)

      (* generate definition and correctness lemma for poly-degree *)
      val ((-, (-, def-thm)), lthy') = gen-def eqn lthy

      val corr-name = short ^ -degree-correct
      val corr-cterm = make-corr-thm-term poly-tm def-const
      (* val corr-cterm = make-corr-thm-term poly-tm deg-tm *)

      (* prove the correctness lemma *)
      val corr-thm = prove-corr-thm lthy' corr-cterm (poly-def-thm :: def-deps @
[def-thm]) deg-thms-for-proof input-thms
      val (-, lthy'') = lthy' |> gen-theorems (corr-name, [corr-thm]);
    in
    (* return the updated theory with definition and correctness theorem in context
*)
      lthy''
    end));
›


end
theory Coding-Theorem-Definitions
 imports ../Coding/Multinomial ../Coding/Bit-Counting Digit-Expansions.Bits-Digits
         ../MPoly-Utils/More-More-MPoly-Type ../MPoly-Utils/Poly-Extract
         ../MPoly-Utils/Total-Degree-Env ../MPoly-Utils/Poly-Degree
begin
```

4 The Coding Theorem

```
lemma series-bound:
  fixes b :: int
  assumes b ≥ 2
  shows (∀k≤q. f k < b) ⟹ (∑k = 0..q. f k * b ^ k) < b^(Suc q)
proof (induct q)
```

```
  case 0
  then show ?case
    by simp
next
  case (Suc q)
  hence f-le-bound: ∀ k≤Suc q. f k ≤ b−1 using assms
    by auto

  from Suc show ?case
  proof −
    assume ∀ k≤q. f k < b ⟹ (∑ k = 0..q. f k * b ^ k) < b ^ Suc q
    with Suc have asm: (∑ k = 0..q. f k * b ^ k) < b ^ Suc q
      by auto

    have (∑ k = 0..q. f k * b ^ k) + f (Suc q) * (b * b ^ q)
          ≤ (∑ k = 0..q. f k * b ^ k) + (b−1) * (b * b ^ q)
      using f-le-bound assms by auto
    also have ... < b^(Suc q) + (b−1) * (b * b ^ q)
      using asm by auto
    also have ... ≤ b * b * b^q
      by (simp add: left-diff-distrib)
    finally show ?thesis
      using asm by auto
  qed
qed
```

4.1 Definition of polynomials required in the Coding Theorem

```
locale coding-variables =
  fixes P :: int mpoly
    and a :: int
    and f :: int
begin
```

Notation for working with P

```
definition δ :: nat where δ ≡ total-degree P
definition ν :: nat where ν ≡ max-vars P


definition P-coeff :: nat list ⇒ int where
  P-coeff i ≡ coeff P (Abs-poly-mapping ((!0) i))
```

Notation used in the proofs

```
definition n :: nat ⇒ nat where n i ≡ (δ+1) ^i
definition δ-tuples :: (nat list) set where
  δ-tuples ≡ {i. length i = ν + 1 ∧ sum-list i ≤ δ}

lemma δ-tuples-finite[simp]: finite δ-tuples
```

proof −
  have *stronger*: *finite* {*i*::*nat list*. *length i* = *n* ∧ *sum-list i* ≤ *K* }
    (**is** *finite* (*?S n*)) **for** *n K*
  proof (*induction n*)
    case *0* **thus** *?case* **by** *auto*
  next
    case (*Suc n*)
    { fix *i* **assume** *assm-i*: *i* ∈ *?S* (*Suc n*)
      then obtain *j0 j* **where** *i-def*: *i* = *j0*#*j*
        by (*smt* (*verit*, *best*) *Suc-length-conv mem-Collect-eq*)
      hence (*j0*, *j*) ∈ ({..*K*} × *?S n*) **using** *assm-i* **by** *auto*
      hence *i* ∈ (λ(*j0*, *j*). *j0*#*j*) ‘ ({..*K*} × *?S n*) **using** *i-def* **by** *auto* **}**
    hence *?S* (*Suc n*) ⊆ (λ(*j0*, *j*). *j0*#*j*) ‘ ({..*K*} × *?S n*)
     by *blast*
    moreover have *finite* ((λ(*j0*, *j*). *j0*#*j*) ‘ ({..*K*} × *?S n*))
      using *Suc.IH* **by** *blast*
    ultimately have *finite* (*?S* (*Suc n*))
      by (*meson finite-subset*)
    thus *?case* **.**
  qed
  thus *?thesis* **unfolding** *δ-tuples-def* **.**
qed

abbreviation σ **where** σ ≡ *count-bits*
abbreviation τ **where** τ ≡ *count-carries*

The variables of Definition 2.2

This is not the same $\mathcal{L}$ as in Lemma 1.8

definition $\mathcal{L}$ :: *nat* **where** $\mathcal{L}$ ≡ *nat* (∑ *i*∈*δ-tuples*. *abs* (*P-coeff i*))

We have to use Inf instead of Min to define r because the set is infinite

definition *r* :: *nat* **where** *r* ≡ *Inf* {*y*. *4*^*y* > *2* ∗ *fact δ* ∗ $\mathcal{L}$ ∗ (ν + *3*)^*δ*}
definition β :: *nat* **where** β ≡ *4*^*r*
definition γ :: *nat* **where** γ ≡ β^(*n* ν)
definition α :: *nat* **where** α ≡ *δ* ∗ *n* ν + *1*

lemma *α-gt-0*: α > *0* **unfolding** *α-def* **by** *auto*
lemma *γ-gt-0*: γ > *0* **unfolding** *γ-def β-def n-def* **by** *auto*

definition *b* :: *int* **where**
  b ≡ *1* + *3*∗(*2*∗*a* + *1*) ∗ *f*
definition $\mathcal{B}$ :: *int* **where**
  $\mathcal{B}$ ≡ (*of-nat* β) ∗ *b*^*δ*
definition *N0* :: *int* **where**
  N0 ≡ $\mathcal{B}$^((*δ*+*1*)^ν + *1*)
definition *N1* :: *int* **where**
  N1 ≡ *4* ∗ $\mathcal{B}$^((*2*∗*δ*+*1*) ∗ (*δ*+*1*)^ν + *1*)
definition *N* :: *int* **where**

$N \equiv N0 * N1$
definition $c :: int \Rightarrow int$ **where**
$c\ g \equiv 1 + a*\mathcal{B} + g$

poly-extract b
poly-degree $b\text{-}poly$

poly-extract $\mathcal{B}$
poly-degree $\mathcal{B}\text{-}poly$

poly-extract $N0$
poly-extract $N1$
poly-extract N
poly-extract c
poly-degree $c\text{-}poly$

The M polynomial.

definition $m :: nat \Rightarrow int$ **where**
$m\ j \equiv (if\ j \in n\ `\ \{1..\nu\}\ then\ \mathcal{B} - b\ else\ \mathcal{B} - 1)$
definition $M :: int$ **where**
$M \equiv (\sum j{=}0..n\ \nu.\ m\ j * \mathcal{B}\hat{}j)$

definition $m\text{-}poly :: nat \Rightarrow int\ mpoly$ **where**
$m\text{-}poly\ j = (if\ j \in n\ `\ \{1..\nu\}\ then\ \mathcal{B}\text{-}poly - b\text{-}poly\ else\ \mathcal{B}\text{-}poly - 1)$

definition $M\text{-}poly :: int\ mpoly$ **where**
$M\text{-}poly \equiv (\sum\ j{=}0..n\ \nu.\ m\text{-}poly\ j * \mathcal{B}\text{-}poly\ \hat{}\ j)$

lemma *m-correct*: *insertion fn* (*m-poly j*) = *coding-variables.m P* (*fn 0*) (*fn* (*Suc 0*)) *j*
unfolding *coding-variables.m-def coding-variables.m-poly-def*
by (*cases j* ∈ *n* ‘ {*1..ν*}, *simp-all add*: $\mathcal{B}$*-correct b-correct*)

lemma *m-poly-degree-env-correct*: *total-degree-env ctxt* (*m-poly j*)
≤ *max* (*total-degree-env ctxt* $\mathcal{B}$*-poly*) (*total-degree-env ctxt b-poly*)
unfolding *m-poly-def* **by** (*auto simp*: *le-trans*[*OF total-degree-env-diff*])

lemma *M-correct*:
insertion fn (*coding-variables.M-poly P*) = *coding-variables.M P* (*fn 0*) (*fn 1*)
unfolding *M-poly-def coding-variables.M-def* **apply** *simp*
unfolding $\mathcal{B}$*-correct m-correct* **by** *auto*

lemma *m-poly-degree-correct*:
shows $\delta > 0 \Longrightarrow$ *total-degree* (*m-poly j*) ≤ *2*δ
unfolding *total-degree-env-id*[*symmetric*]
apply (*rule le-trans*[*OF coding-variables.m-poly-degree-env-correct*])
apply (*unfold total-degree-env-id*, *simp*, *intro conjI*)
using *coding-variables.*$\mathcal{B}$*-poly-degree-correct*[*unfolded coding-variables.*$\mathcal{B}$*-poly-degree-def*]

```
 using coding-variables.b-poly-degree-correct[unfolded coding-variables.b-poly-degree-def]
 by (auto simp add: mult-2)

lemma M-poly-degree-correct:
 assumes asm: δ > 0
 shows total-degree M-poly ≤ (1+(δ+1) ^ν) * 2*δ
proof −
 have fin0: finite {0..(δ + 1) ^ν} by simp
 {
  fix a
  assume a ∈ {0..Suc δ ^ν}
  hence a: a ≤ (δ+1) ^ν by auto
  have total-degree (coding-variables.m-poly P a * B-poly ^ a)
        ≤ total-degree (coding-variables.m-poly P a) + a * total-degree B-poly
   apply (rule le-trans[OF total-degree-mult add-mono])
   by (auto simp: le-trans total-degree-mult total-degree-pow)
  also have ... ≤ 2*δ + (δ+1) ^ν * (2*δ)
   apply (rule add-mono, simp add: m-poly-degree-correct[OF asm])
   apply (rule mult-mono, simp-all add: a[unfolded Suc-eq-plus1[symmetric]])
  using coding-variables.B-poly-degree-correct[unfolded coding-variables.B-poly-degree-def]
   by (auto simp add: mult-2)
  finally have total-degree (coding-variables.m-poly P a * B-poly ^ a)
            ≤ 2*δ + (δ+1) ^ν * (2*δ).
 } note h0 = this

 show ?thesis
  unfolding coding-variables.M-poly-def coding-variables.n-def
  apply (rule le-trans[OF total-degree-sum[OF fin0]], simp)
  using h0 by (auto simp: mult.assoc)
qed


definition D-exponent :: nat list ⇒ nat where
 D-exponent i ≡ n (ν+1) − (∑ s≤ν. i!s * n s)
definition D-precoeff :: nat list ⇒ int where
 D-precoeff i ≡ int (mfact i * fact (δ − sum-list i))
definition D :: int where
 D ≡ ∑ i∈δ-tuples. of-int (D-precoeff i * P-coeff i) * B ^(D-exponent i)

definition D-poly :: int mpoly where
 D-poly ≡ ∑ i∈δ-tuples. Const (D-precoeff i * P-coeff i) * B-poly ^ (D-exponent
i)

lemma D-correct:
 insertion fn (coding-variables.D-poly P) = coding-variables.D P (fn 0) (fn 1)
 unfolding D-poly-def coding-variables.D-def apply simp
 using insertion-sum[OF δ-tuples-finite, of fn] apply simp
 unfolding B-correct by auto
```

lemma *D-poly-degree-env-correct*:
shows *total-degree-env fv D-poly* $\leq$ *n* (ν+1) * *total-degree-env fv* $\mathcal{B}$*-poly*
proof −
{
fix *a*
assume *a* $\in$ δ*-tuples*
hence *d*: *D-exponent a* $\leq$ *n* (ν+1)
unfolding *D-exponent-def* **by** *simp*
have *total-degree-env fv* (*Const* (*D-precoeff a* * *P-coeff a*) * $\mathcal{B}$*-poly* ^ *D-exponent a*)
$\leq$ *D-exponent a* * *total-degree-env fv* ($\mathcal{B}$*-poly*)
by (*metis add-0 le-trans total-degree-env-Const total-degree-env-mult total-degree-env-pow*)
with *d* **have** *total-degree-env fv* (*Const* (*D-precoeff a* * *P-coeff a*) * $\mathcal{B}$*-poly* ^ *D-exponent a*)
$\leq$ *n* (ν+1) * *total-degree-env fv* ($\mathcal{B}$*-poly*)
by (*simp add*: *le-trans*)
} note *h0* = *this*

show *?thesis*
unfolding *D-poly-def*
apply (*rule le-trans*[*OF total-degree-env-sum*[*OF* δ*-tuples-finite*]])
using *h0* **by** *simp*
qed

lemma *D-poly-degree-correct*: *total-degree* (*coding-variables.D-poly P*) $\leq$ (δ+1) ^ (ν+1) * (2*δ)
unfolding *total-degree-env-id*[*symmetric*]
apply (*rule le-trans*[*OF D-poly-degree-env-correct*], *unfold total-degree-env-id n-def*)
apply (*rule mult-le-mono*)
using $\mathcal{B}$*-poly-degree-correct*[*unfolded* $\mathcal{B}$*-poly-degree-def*] **by** *auto*

definition *K* :: *int* $\Rightarrow$ *int* **where**
K g $\equiv$ (*c g*) ^ δ * *D* + ($\sum$ *i*=0..(2*δ+1)*n* ν. *of-nat* (β *div* 2) * *b* ^ δ * $\mathcal{B}$ ^ *i*)

definition *K-poly* :: *int mpoly* **where**
K-poly $\equiv$ *c-poly* ^ δ * *D-poly* + ($\sum$ *i*=0..(2*δ+1)*n* ν. *of-nat* (β *div* 2) * *b-poly* ^ δ * $\mathcal{B}$*-poly* ^ *i*)

lemma *K-correct*:
insertion fn (*coding-variables.K-poly P*) = *coding-variables.K P* (*fn 0*) (*fn 1*) (*fn 2*)
unfolding *K-poly-def coding-variables.K-def* **apply** *simp*
unfolding *c-correct D-correct b-correct* $\mathcal{B}$*-correct* **by** *auto*

lemma *K-poly-degree-correct*:
shows *total-degree* (*coding-variables.K-poly P*)
$\leq$ *max* (δ*(1+2*δ) + (δ+1) ^ (ν+1) * 2*δ) ((1 + (2*δ+1)*(δ+1) ^ ν) * 2*δ)

```
proof −
  have fin0: finite {0..(2 * δ + 1) * n ν} by simp

  {
    fix a
    assume a∈{0..Suc δ ^ ν + 2 * δ * Suc δ ^ ν}
    hence a: a ≤ Suc δ ^ ν + 2 * δ * Suc δ ^ ν by simp
    have total-degree (of-nat (β div 2) * b-poly ^ δ * ℬ-poly ^ a)
       ≤ 2*δ + (Suc δ ^ ν + 2 * (δ * Suc δ ^ ν)) * (2*δ)
      apply (rule le-trans[OF total-degree-mult add-mono])
      subgoal
        apply (rule le-trans[OF total-degree-mult], simp add: of-nat-Const)
        apply (rule le-trans[OF total-degree-pow])
        using b-poly-degree-correct[unfolded b-poly-degree-def]
        by simp
      subgoal
        apply (rule le-trans[OF total-degree-pow], rule mult-le-mono)
        using ℬ-poly-degree-correct[unfolded ℬ-poly-degree-def]
        using a by simp-all
      done
  } note h0 = this

  show ?thesis
    unfolding K-poly-def
    apply (rule le-trans[OF total-degree-add max.mono]
        | rule le-trans[OF total-degree-diff max.mono]
        | rule le-trans[OF total-degree-mult add-mono]
        | rule le-trans[OF total-degree-pow mult-le-mono2])+
    subgoal using c-poly-degree-correct[unfolded c-poly-degree-def] by simp
    subgoal unfolding mult.assoc by (rule D-poly-degree-correct)
    apply (rule le-trans[OF total-degree-sum[OF fin0]])
    using h0 by (simp add: n-def algebra-simps)
qed


definition T where T ≡ M + (ℬ−2) * ℬ^((δ+1)^(ν+1)) * N0
definition S :: int ⇒ int where S g ≡ g + 2 * K g * N0
definition R :: int ⇒ int where R g ≡ (S g + T + 1)*N + T + 1
definition X :: int ⇒ int where X g ≡ (N−1) * R g
definition Y :: int where Y ≡ N^2

poly-extract T
poly-extract S
poly-extract R
poly-extract X
poly-extract Y
```

These are the statements that make up theorem I.

```
definition statement1-weak where
```

statement1-weak y ≡ (*y 0* = *a*) ∧ (∀ *i*. *0* ≤ *y i* ∧ *y i* < *b*) ∧ *insertion y P* = *0*
definition *statement1-strong* **where**
statement1-strong y ≡ *statement1-weak y* ∧ (∃ *i*∈{*1*..*ν*}. *y i* ≠ *0*)

We evaluate Y in 0 because it doesn't depend on g.

definition *statement2-strong* **where**
statement2-strong g ≡ *b* ≤ *g* ∧ *g* < (*int γ*) ∗ *b*^*α* ∧ *Y dvd* (*2* ∗ *nat* (*X g*) *choose nat* (*X g*))
definition *statement2-weak* **where**
statement2-weak g ≡ *0* ≤ *g* ∧ *g* < *2* ∗ (*int γ*) ∗ *b*^*α* ∧ *Y dvd* (*2* ∗ *nat* (*X g*) *choose nat* (*X g*))

lemma *δ-tuples-nonempty*: *δ-tuples* ≠ {}
proof −
define *i* **where** *i* ≡ *replicate* (*ν*+*1*) (*0*::*nat*)
have *i* ∈ *δ-tuples*
unfolding *δ-tuples-def i-def*
by (*smt* (*verit*, *best*) *in-set-replicate length-replicate less-eq-nat.simps*(*1*)
mem-Collect-eq sum-list-eq-0-iff)
thus *?thesis* **by** *auto*
qed

corollary *x-series-bound*:
assumes *0* < *δ*
assumes *x* ∈ *δ-tuples*
shows (∑ *s*≤*ν*. *x* ! *s* ∗ *Suc δ* ^ *s*) ≤ (*δ*+*1*)^(*ν*+*1*)
proof −
from *assms* **have** *2* ≤ *int* (*Suc δ*)
by *auto*
from *assms* **have** *length x* = *Suc ν*
unfolding *δ-tuples-def* **by** *simp*
have *x-bound*: ∀ *s*≤*ν*. (*x* ! *s*) < *Suc δ*
using *assms* **unfolding** *δ-tuples-def*
by *simp* (*smt* (*verit*) *elem-le-sum-list le-imp-less-Suc order-trans*)
hence *x-bound2*: ∀ *k*≤*ν*. *int* (*x* ! *k*) < *int* (*Suc δ*)
by *auto*
show *?thesis*
using *series-bound*[*of Suc δ ν* (!) *x*, *OF* ‹*2* ≤ *int* (*Suc δ*)› *x-bound2*]
apply (*subst atMost-atLeast0*, *subst less-imp-le*, *simp-all*)
apply (*subst nat-int-comparison*(*2*))
by *auto* (*smt* (*verit*, *del-insts*) *of-nat-Suc of-nat-add of-nat-mult*
of-nat-power times-nat.simps(*2*))
qed

lemma *D-exponent-injective*:
assumes *0* < *δ*
shows *inj-on D-exponent δ-tuples*
proof −
from *assms* **have** *1* < *δ* + *1*

by *auto*

{
fix *x y*
assume *a1*: $x \in \delta$-*tuples* **and** *a2*: $y \in \delta$-*tuples*
hence *l1*: *length* $x = Suc\ \nu$ **and** *l2*: *length* $y = Suc\ \nu$
unfolding δ-*tuples-def* **by** *auto*
from *a1 a2* **have** *l3*: $\forall s{\leq}\nu.\ x\ !\ s < \delta + 1$ **and** *l4*: $\forall s{\leq}\nu.\ y\ !\ s < \delta + 1$
unfolding δ-*tuples-def* **using** *l1 l2 elem-le-sum-list*
by (*metis* (*no-types*, *lifting*) *Suc-eq-plus1 le-imp-less-Suc mem-Collect-eq order-trans*)+

assume $(\sum s{\leq}\nu.\ x\ !\ s * (\delta + 1)\ \hat{}\ s) = (\sum s{\leq}\nu.\ y\ !\ s * (\delta + 1)\ \hat{}\ s)$
hence $x = y$
apply (*subst* (*asm*) *digit-wise-gen-equiv*[*OF* ‹*1* < δ + *1*›])
apply (*subst* (*asm*) *atMost-atLeast0*)+
apply (*subst* (*asm*) *nth-digit-gen-power-series-general*[*OF* ‹*1* < δ + *1*› *l3*])
apply (*subst* (*asm*) *nth-digit-gen-power-series-general*[*OF* ‹*1* < δ + *1*› *l4*])
using *l1 l2 l3 l4* **by** (*metis less-Suc-eq-le nth-equalityI*)
}

then show *?thesis*
unfolding *inj-on-def D-exponent-def n-def*
using *x-series-bound*[*OF* ‹*0* < δ›]
proof −
{ fix *nns* :: *nat list* **and** *nnsa* :: *nat list*
have *ff1*: $\bigwedge ns.\ ns \notin \delta$-*tuples* $\vee$ $(\sum n{\leq}\nu.\ ns\ !\ n * Suc\ \delta\ \hat{}\ n) \leq Suc\ \delta\ \hat{}\ Suc\ \nu$
using ‹$\bigwedge x.\ x \in \delta$-*tuples* $\Longrightarrow$ $(\sum s{\leq}\nu.\ x\ !\ s * Suc\ \delta\ \hat{}\ s) \leq (\delta + 1)\ \hat{}\ (\nu + 1)$› **by** *force*
have $\forall n\ na.\ \exists nb.\ \neg\ (na{::}nat) \leq n \vee na + nb = n$
using *le-Suc-ex* **by** *blast*
then obtain *nn* :: *nat* ⇒ *nat* ⇒ *nat* **where**
ff2: $\bigwedge n\ na.\ \neg\ n \leq na \vee n + nn\ na\ n = na$
by *metis*
then have *ff3*: $\bigwedge n\ na.\ \neg\ n \leq na \vee na - nn\ na\ n = n$
by (*metis* (*no-types*) *add-diff-cancel-right'*)
{ assume $(\sum n{\leq}\nu.\ nnsa\ !\ n * Suc\ \delta\ \hat{}\ n) \leq Suc\ \delta\ \hat{}\ Suc\ \nu \wedge (\sum n{\leq}\nu.\ nns\ !\ n * Suc\ \delta\ \hat{}\ n) \neq (\sum n{\leq}\nu.\ nnsa\ !\ n * Suc\ \delta\ \hat{}\ n)$
then have $(\sum n{\leq}\nu.\ nns\ !\ n * Suc\ \delta\ \hat{}\ n) \leq Suc\ \delta\ \hat{}\ Suc\ \nu \longrightarrow (\sum n{\leq}\nu.\ nns\ !\ n * Suc\ \delta\ \hat{}\ n) \leq Suc\ \delta\ \hat{}\ Suc\ \nu \wedge Suc\ \delta\ \hat{}\ Suc\ \nu - nn\ (Suc\ \delta\ \hat{}\ Suc\ \nu)\ (\sum n{\leq}\nu.\ nnsa\ !\ n * Suc\ \delta\ \hat{}\ n) \neq (\sum n{\leq}\nu.\ nns\ !\ n * Suc\ \delta\ \hat{}\ n)$
using *ff3* **by** (*smt* (*z3*))
then have $(\sum n{\leq}\nu.\ nnsa\ !\ n * Suc\ \delta\ \hat{}\ n) \leq Suc\ \delta\ \hat{}\ Suc\ \nu \wedge (\sum n{\leq}\nu.\ nns\ !\ n * Suc\ \delta\ \hat{}\ n) \leq Suc\ \delta\ \hat{}\ Suc\ \nu \longrightarrow nns \notin \delta$-*tuples* $\vee$ $nnsa \notin \delta$-*tuples* $\vee$ $nns = nnsa \vee (\delta + 1)\ \hat{}\ (\nu + 1) - (\sum n{\leq}\nu.\ nns\ !\ n * (\delta + 1)\ \hat{}\ n) \neq (\delta + 1)\ \hat{}\ (\nu + 1) - (\sum n{\leq}\nu.\ nnsa\ !\ n * (\delta + 1)\ \hat{}\ n)$
using *ff2* **by** *fastforce*
then have $(\sum n{\leq}\nu.\ nnsa\ !\ n * Suc\ \delta\ \hat{}\ n) \leq Suc\ \delta\ \hat{}\ Suc\ \nu \longrightarrow nns \notin$

δ-tuples $\vee$ nnsa $\notin$ δ-tuples $\vee$ nns = nnsa $\vee$ $(\delta + 1)$ ^ $(\nu + 1)$ - $(\sum n{\leq}\nu.$ nns ! n * $(\delta + 1)$ ^ n) $\neq$ $(\delta + 1)$ ^ $(\nu + 1)$ - $(\sum n{\leq}\nu.$ nnsa ! n * $(\delta + 1)$ ^ n)
using *ff1* **by** *blast* **}**
then have nns $\notin$ δ-tuples $\vee$ nnsa $\notin$ δ-tuples $\vee$ nns = nnsa $\vee$ $(\delta + 1)$ ^ $(\nu + 1)$ - $(\sum n{\leq}\nu.$ nns ! n * $(\delta + 1)$ ^ n) $\neq$ $(\delta + 1)$ ^ $(\nu + 1)$ - $(\sum n{\leq}\nu.$ nnsa ! n * $(\delta + 1)$ ^ n)
using *ff1* **by** (*metis* (*no-types*) *Suc-eq-plus1* ‹$\bigwedge y\ x.\ [\![x \in \delta\text{-tuples};\ y \in \delta\text{-tuples};\ (\sum s{\leq}\nu.\ x\ !\ s * (\delta + 1)\ \hat{}\ s) = (\sum s{\leq}\nu.\ y\ !\ s * (\delta + 1)\ \hat{}\ s)]\!] \Longrightarrow x = y$›) **}**
then show $\forall$ ns$\in\delta$-tuples. $\forall$ nsa$\in\delta$-tuples. $(\delta + 1)$ ^ $(\nu + 1)$ - $(\sum n{\leq}\nu.$ ns ! n * $(\delta + 1)$ ^ n) = $(\delta + 1)$ ^ $(\nu + 1)$ - $(\sum n{\leq}\nu.$ nsa ! n * $(\delta + 1)$ ^ n) $\longrightarrow$ ns = nsa
by *blast*
qed
qed

corollary *D-exponent-injective'*: 0 < δ $\Longrightarrow$ *inj-on D-exponent* (δ-tuples - {x})
using *D-exponent-injective* **by** (*rule inj-on-diff*)

lemma *D-precoeff-bound*:
assumes 0 < *sum-list i* **and** *sum-list i* $\leq$ δ
shows |*D-precoeff i*| $\leq$ *fact* δ
proof -
have δ *mchoose i* $\geq$ 1
using *assms* **unfolding** *multinomial-def multinomial'-def*
by *simp* (*metis dvd-mult-div-cancel fact-ge-Suc-0-nat mchoose-dvd mult.commute one-le-mult-iff*)
thus *?thesis*
using *assms* **unfolding** *D-precoeff-def multinomial-def multinomial'-def*
by (*metis abs-of-nat mchoose-le of-nat-fact of-nat-le-iff*)
qed

We later assume that $\delta > 0$ i.e. P is not the zero polynomial

lemma *P-coeffs-not-all-zero*:
assumes $\delta > 0$
shows $\exists$ i$\in\delta$-tuples. *P-coeff i* $\neq$ 0
proof -
have $\neg$ ($\forall$ i$\in\delta$-tuples. *P-coeff i* = 0)
proof
assume *coeffs-zero*: $\forall$ i$\in\delta$-tuples. *P-coeff i* = 0
{ fix m **assume** *assm*: m $\in$ *keys* (*mapping-of P*)
then obtain i **where** m = *Abs-poly-mapping* ((!$_0$) i) **and** i$\in\delta$-tuples
using *mpoly-keys-subset* **unfolding** δ-tuples-def δ-def ν-def **by** *auto*
hence *P-coeff i* $\neq$ 0
using *assm P-coeff-def* **by** (*simp add*: *coeff-keys*)
hence *False*
using *coeffs-zero* ‹i$\in\delta$-tuples› **by** *auto* **}**
hence *keys* (*mapping-of P*) = {}
by *blast*

```
    hence P = 0
      by (metis keys-eq-empty mapping-of-inverse zero-mpoly.abs-eq)
    thus False
      using ‹δ > 0› δ-def by simp
  qed
  thus ?thesis by auto
qed

lemma ℒ-pos:
  assumes δ > 0
  shows ℒ > 0
proof −
  have ¬ ℒ = 0
  proof
    assume ℒ = 0
    hence (∑ i∈δ-tuples. abs (P-coeff i)) = 0
      unfolding ℒ-def by (metis int-ops(1) nat-0-le sum-abs-ge-zero)
    hence i∈δ-tuples ⟹ abs (P-coeff i) = 0 for i
    using sum-nonneg-eq-0-iff[OF δ-tuples-finite] by (metis (full-types) abs-ge-zero)
    hence i∈δ-tuples ⟹ P-coeff i = 0 for i
      by auto
    thus False
      using P-coeffs-not-all-zero[OF ‹δ > 0›] by auto
  qed
  thus ?thesis by auto
qed

lemma ℒ-ge-P-coeff: i∈δ-tuples ⟹ abs (P-coeff i) ≤ int ℒ for i
proof −
  assume i∈δ-tuples
  hence abs (P-coeff i) ≤ (∑ i∈δ-tuples. abs (P-coeff i))
    by (meson δ-tuples-finite abs-ge-zero sum-nonneg-leq-bound)
  also have ... = int ℒ
    unfolding ℒ-def by auto
  finally show ?thesis .
qed

lemma ℒ-ge-max-coeff:
  assumes δ > 0
  shows max-coeff P ≤ int ℒ
proof −
  have lookup (mapping-of P) i ∈ (P-coeff ‘ δ-tuples) ∪ {0} for i
  proof (cases lookup (mapping-of P) i = 0)
    case True thus ?thesis by simp
  next
    case False
    hence i∈keys (mapping-of P)
      by (simp add: in-keys-iff)
    thus ?thesis
```

```
    using mpoly-keys-subset unfolding P-coeff-def MPoly-Type.coeff-def δ-tuples-def
δ-def ν-def
      by auto
  qed
  hence 1: coeffs P ⊆ P-coeff ` δ-tuples
    unfolding coeffs-def range-def by auto

  have 2: coeffs P ≠ {}
  proof
    assume coeffs P = {}
    hence δ = 0
      unfolding δ-def using coeffs-empty-iff[of P] by simp
    thus False
      using ‹δ > 0› by auto
  qed

  from ℒ-ge-P-coeff 1 2 show ?thesis
    unfolding max-coeff-def by auto
qed

lemma β-lower-bound: β > 2 * fact δ * ℒ * (ν+3) ^δ
proof −
  define S where S ≡ {y. 4 ^y > 2 * fact δ * ℒ * (ν + 3) ^δ}

  have 2 * fact δ * ℒ * (ν + 3) ^δ ∈ S
    unfolding S-def using power-gt-expt by auto
  hence S ≠ {}
    by auto
  hence Inf S ∈ S
    by (simp add: Inf-nat-def1)
  hence r ∈ S
    using r-def S-def by simp
  thus ?thesis
    unfolding S-def β-def by simp
qed

corollary r-pos:
  assumes δ > 0
  shows r > 0
  using β-lower-bound β-def ℒ-pos[OF ‹δ > 0›] zero-less-iff-neq-zero by fastforce


lemma marcos-state:
  fixes i
  shows total-degree-env
  (λm. total-degree-env (λ-. Suc 0) ([Var 0, Var 1, Var 2] !0 m))
  ([Var 0, Var 1, Var 2] !0 i) ≤ 1
proof −
  have 1: [Var 0, Var 1, Var 2] !0 m = map Var [0, 1, 2] !0 m for m
```

```
  apply auto done

 have aux: (λm. total-degree-env (λ-. Suc 0) (map Var [0, 1, 2] !0 m)) =
      (λm. if m < length [0, 1, 2] then 1 else 0)
   by (metis One-nat-def length-Cons list.size(3) total-degree-env-Var-list)

 show ?thesis
  apply (rule le-trans[of - total-degree-env (λm. if m < length [0, 1, 2] then 1
else 0) ([Var 0, Var 1, Var 2] !0 i)])
  subgoal
   apply simp
   using aux
   by (smt (z3) 1 Suc-1 Suc-eq-plus1 dual-order.refl le-less-Suc-eq less-2-cases
       linorder-not-less list.size(3,4) nth0-0
       nth0-Cons-0 nth0-Cons-Suc total-degree-env-Var total-degree-env-zero)
  subgoal
    by (smt (verit, del-insts) One-nat-def Suc-leI Suc-le-mono add.right-neutral
add-Suc-right antisym-conv1
      bot-nat-0.not-eq-extremum list.size(3,4) nth0-0 nth0-Cons-0 nth0-Cons-Suc
total-degree-env-Var
       total-degree-env-zero)
  done
qed

end

end
theory Lemma-2-2
 imports HOL−Number-Theory.Number-Theory ../Coding/Utils
begin
```

4.2 Increasing the base b appropriately

```
lemma exp-grows:
 fixes a b k :: int
 assumes H: a ≥ 2 ∧ k ≥ 0
 shows ∃ (s::nat). a^s ≥ b ∧ s ≥ k
proof −
 let ?c = nat(abs(b)) + nat(abs(k))
 have nat(a)^?c ≥ b
    by (smt (verit, best) assms nat-le-iff nat-plus-as-int of-nat-1 one-add-one
self-le-ge2-pow
      zless-nat-conj zless-nat-eq-int-zless)
thus ?thesis
  using assms by force
qed

lemma little-fermat:
 fixes a m :: int
```

```
  assumes gcd: coprime a m and posit: a ≥ 1 ∧ m ≥ 2
  shows ∃ k l. a^k−1 = m∗l ∧ k ≥ 1
proof −
  have residues m
    using posit residues-def by auto
  hence a^totient(nat m) mod m = 1
    using  Residues.residues.euler-theorem[of m a] gcd
    by (metis abs-le-zero-iff abs-one mod-pos-pos-trivial
        residues.m-gt-one residues.res-eq-to-cong verit-la-generic)
  hence 0: (a^totient(nat m) − 1) mod m = 0
    by (metis diff-self mod-0 mod-diff-right-eq)

  have totient(nat m) ≥ 1
    using ‹residues m› not-le-imp-less residues.m-gt-one by fastforce
  thus ?thesis
    using 0 by blast
qed


lemma lemma-2-2:
  fixes a Z :: int
  assumes posit:Z > 0 ∧ a ≥ 0
  shows ∃ f r. f ≥ Z ∧ 1 + 3∗(2∗a+1)∗f = 4^r
proof −
  let ?c = 3∗(2∗a+1) and ?d = 1 + 3∗(2∗a+1)∗Z
  have d: ?d ≥1 ∧ ?d ≥ 0 using posit by auto
  have c: ?c ≥ 2 ∧ (4::int) ≥ 1  using posit by auto
  have c-odd: ∃ h. ?c = 2∗h +1 by presburger
  then obtain h where h-def: ?c = 2∗h +1 by auto

  have gcd: coprime (4::int) ?c
    using h-def
    by (metis coprime-add-one-right coprime-mult-left-iff
        mult-2 mult-2-right numeral-Bit0 one-add-one)
  hence 10:∃ k l. (4::int)^k−1 = ?c∗l ∧ k ≥ 1
    using little-fermat[of 4 ?c] c by auto
  then obtain k l where 11: 4^k−1 = ?c∗l ∧ k ≥ 1
    by auto
  obtain s where s-def: s ≥ 4 ∧ 4^s ≥ 1 + 3∗(2∗a+1)∗Z
    using d exp-grows[of 4 4 ?d] by force

  have (4::nat)^(s∗k) ≥ (4::nat)^s
    using 11 by simp
  hence 20: ((4::nat)^(s∗k)) ≥ 1 + 3∗(2∗a+1)∗Z
    using s-def by (meson dual-order.trans numeral-power-le-of-nat-cancel-iff)

  have 21: (∃q. 4 ^ k − 1 =  3 ∗ (2 ∗ a + 1) ∗ q)
    using 11 by auto
  hence (4^k −1) mod ?c = 0
```

```
    using zmod-eq-0-iff[of 4^k-1 ?c] by auto
  hence 4^k mod ?c = 1 mod ?c
    by (smt (verit, ccfv-SIG) 11 mod-mult-self2)
  hence (4^k)^s mod ?c = 1 mod ?c
    by (metis power-mod power-one)
  hence ((4^k)^s -1) mod ?c = 0
    by (smt (z3) mod-eqE zmod-eq-0-iff)
  then obtain f where f-def: (4^k)^s - 1 = f*?c
    using zmod-eq-0-iff by force
  hence 40: 4^(s*k) = 1 + ?c*f
    by (simp add: mult.commute power-mult)
  hence 1 + f*?c ≥ 1 + Z*?c
   using 20 by (metis int-ops(3) mult.commute of-nat-power-eq-of-nat-cancel-iff)
  hence 50: f ≥ Z
    using c by auto
  hence Z ≤ f ∧ 1 + 3*(2*a+1)*f = 4^(s*k)
    using 40 by auto
  thus ?thesis
    by auto
qed



lemma lemma-2-2-useful:
  fixes a min-b :: int
  assumes min-b ≥ 0 ∧ a ≥ 0
  defines b ≡ λf. 1 + 3 * (2*a + 1) * f
  shows ∃f. f > 0 ∧ is-square (b f) ∧ is-power2 (b f) ∧ b f > min-b
proof −
  have 0: f > 0 ⟹ b f > f for f
    unfolding b-def using assms(1)
    by (simp add: add-strict-increasing)

  from lemma-2-2[of min-b + 1 a] obtain f r
    where fr-def: min-b + 1 ≤ f ∧ 1 + 3 * (2 * a + 1) * f = 4 ^ r
    by (auto simp: assms(1))

  show ?thesis
  proof (rule exI[of - f], intro conjI)
    show f > 0
      using fr-def assms by auto
    show is-square (b f)
      unfolding is-square-def b-def conjunct2[OF fr-def] power2-eq-square
      by (metis num-double numeral-times-numeral power-mult-distrib)
    show is-power2 (b f)
      unfolding b-def conjunct2[OF fr-def] by simp
    show min-b < b f
      using assms(1) conjunct1[OF fr-def]
      by (smt (verit) 0)
```

qed
qed

end
theory *Lower-Bounds*
imports *Coding-Theorem-Definitions ../Coding/Lemma-1-8-Coding Digit-Expansions.Bits-Digits*
HOL−Library.Rewrite ../Coding/Suitable-For-Coding
begin

4.3 Lower bounds on the defined variables

lemma (**in** *coding-variables*) *defs-non-negative*:
fixes $g :: int$
assumes $a \geq 0$
assumes $f > 0$
assumes $g \geq 0$
assumes $\delta > 0$
assumes *P-coeff* (*replicate* $(\nu+1)$ *0*) > 0
shows $\mathcal{B} > 0$ **and** $N \geq 2$ **and** $R\ g > 0$ **and** $S\ g \geq 0$ **and** $T \geq 0$ **and** $N0 \geq 1$
proof −
note *simps* = *eval-def insertion-simps comp-def*

have $\beta > 0$
unfolding β-*def* **by** *auto*
hence $\beta \geq 0$
by *auto*
have b: $b > 2$
unfolding b-*def* **apply** (*simp add*: *assms*)
using *assms*(*1−2*) **by** (*smt* (*verit*) *mult-le-cancel-right2*)
moreover have $\mathcal{B} \geq b$
unfolding $\mathcal{B}$-*def*
using ‹$\delta > 0$› ‹$\beta > 0$›
by (*smt* (*verit*, *del-insts*) *calculation comp-apply mult-le-cancel-right1*
of-nat-0-less-iff self-le-power)
ultimately have $\mathcal{B} > 2$
by *auto*
then show *BB*: $\mathcal{B} > 0$
by *auto*
show *N0*: $N0 \geq 1$
unfolding *N0-def*
using ‹$\mathcal{B} > 0$›
by (*simp add*: *int-one-le-iff-zero-less*)
show N: $N \geq 2$
using ‹$N0 \geq 1$› ‹$\mathcal{B} > 0$›
unfolding *N-def N1-def eval-def*
by *simp* (*smt* (*z3*) *mult-pos-pos zero-less-power*)
have m: $(m\ j) \geq 0$ **for** j
unfolding *m-def*
using ‹$\mathcal{B} \geq b$› ‹$\mathcal{B} > 2$›

```
  by (auto simp: eval-def)
have M: M ≥ 0
  unfolding M-def
  using m BB by (simp add: sum-nonneg)
show T ≥ 0
  unfolding T-def
  using N0 M BB ‹2 < ℬ› by force

have K g ≥ 0
proof −
  have 2 ≤ int (Suc δ)
    using ‹δ > 0› by auto
  have 1 < nat ℬ
    using ‹ℬ > 2›
    unfolding ℬ-def eval-def by auto
  hence ℬ ≥ 2
    by auto

  have even β
    by (simp add: β-def r-pos[OF ‹δ > 0›])
  have even ℬ
    unfolding ℬ-def simps β-def
    using r-pos[OF ‹δ > 0›]
    by simp

  have c-bound: c g > 0
    unfolding c-def using BB
    by (simp add: add-strict-increasing assms(1) assms(3))


  have δ-ℒ-bound: fact δ * ℒ < ℬ − 1
  proof −
    have fact δ * ℒ < fact δ * ℒ * (ν + 3) ^ δ * b ^ δ
      using ‹δ > 0› ℒ-pos[OF ‹δ > 0›]
    by simp (smt (verit) b eval-def less-1-mult o-apply of-nat-less-0-iff one-less-power)
    hence fact δ * ℒ < 2 * fact δ * ℒ * (ν + 3) ^ δ * b ^ δ − 1
      by linarith
    moreover have 2 * fact δ * ℒ * (ν + 3) ^ δ * b ^ δ < 4 ^ r * b ^ δ
      using β-lower-bound[unfolded β-def]
    by (smt (z3) b fact-2 int-eq-iff-numeral less-imp-of-nat-less mult-of-nat-commute

        numeral-Bit0 of-nat-fact of-nat-power zero-less-power zmult-zless-mono2)
    ultimately show ?thesis
      unfolding ℬ-def β-def
      by auto
  qed

  have D-bound: D > 0
  proof −
```

```
  let ?i0 = replicate (ν+1) 0
  let ?a0 = P-coeff ?i0

  have ?a0 > 0
    using assms by auto

  have h0: i∈δ-tuples − {replicate (ν+1) 0} ⟹ sum-list i > 0 for i
    apply (rule ccontr)
    unfolding δ-tuples-def using replicate-eqI by fastforce

  have h1: (∑ r=0..(δ+1) ^(ν+1)−1. ℬ ^ r) > 0
    apply (subst sum-pos)
    using ℬ-def BB eval-def by (auto)

  have h2: D-precoeff ?i0 * P-coeff ?i0 * ℬ ^ D-exponent ?i0
          = fact δ * ?a0 * ℬ ^ ((δ+1) ^(ν+1))
  proof −
    have (∑ s≤ν. ?i0 ! s * Suc δ ^ s) = 0
      by auto (metis List.nth-replicate le-imp-less-Suc replicate-Suc)
    thus ?thesis
      unfolding D-precoeff-def D-exponent-def
      using ‹?a0 > 0› apply (simp add: sum-list-replicate)
      by (simp add: mfact-def ℬ-def n-def)
  qed

  have h3: ?i0 ∈ δ-tuples
    by (simp add: sum-list-replicate δ-tuples-def)

  have h4: (∑ i∈δ-tuples − {?i0}. ℬ ^ D-exponent i) ≤ (∑ r=0..(δ+1) ^(ν+1)−1.
ℬ ^ r)
  proof −
    have h41: D-exponent ' (δ-tuples − {?i0}) ⊆ {0..(δ+1) ^(ν+1)−1}
    proof −
      {
        fix x
        assume asm: x ∈ (δ-tuples − {?i0})
        hence length x = Suc ν
          unfolding δ-tuples-def by simp
        have x-nonzero: ∃ s≤ν. x ! s > 0
          apply (rule ccontr)
          using h0[OF asm]
          by (simp add: sum-list-sum-nth ‹length x = Suc ν›)
        then obtain i where i: i ≤ ν ∧ x ! i > 0
          by auto
        have (∑ s≤ν. x ! s * Suc δ ^ s) > 0
           using x-nonzero by (subst sum-pos2[of - i], auto simp: i)
      } note a = this

      from a show ?thesis
```

```
      unfolding D-exponent-def
      apply (simp add: n-def)
      using diff-le-mono2 by fastforce
  qed

  have (∑ i∈δ-tuples − {?i0}. ℬ ^ D-exponent i)
      = sum ((^) ℬ ∘ D-exponent) (δ-tuples − {?i0})
    by auto
  also have ... = sum ((^) ℬ) (D-exponent ' (δ-tuples − {?i0}))
    apply (subst sum.reindex)
    using D-exponent-injective'[OF ‹δ > 0›] by auto
  also have ... ≤ sum ((^) ℬ) {0..(δ+1)^(ν+1)−1}
    apply (subst sum-mono2)
    using h41 ‹ℬ ≥ 2› by auto
  finally show ?thesis
    by auto
 qed

 have |D − fact δ * ?a0 * ℬ ^ ((δ+1)^(ν+1))|
     = |(∑ i∈δ-tuples − {?i0}. D-precoeff i * P-coeff i * ℬ ^ D-exponent i)|
   apply (subst abs-eq-iff, rule disjI1)
   unfolding D-def eval-def apply simp
   apply (subst sum.remove[of δ-tuples ?i0])
   using h2 h3 ‹?a0 > 0› by auto
 also have ... ≤ (∑ i∈δ-tuples − {?i0}. |D-precoeff i * P-coeff i *
                                          ℬ ^ D-exponent i|)
   by auto
 also have ... ≤ (∑ i∈δ-tuples − {?i0}. |D-precoeff i| * |P-coeff i| *
                                          |ℬ| ^ D-exponent i)
   apply (rule sum-mono)
   by (auto simp: abs-mult power-abs)
 also have ... ≤ (∑ i∈δ-tuples − {?i0}. fact δ * |P-coeff i| * |ℬ| ^ D-exponent
i)
   apply (rule sum-mono)
   apply (frule h0)
   unfolding δ-tuples-def
 by auto (smt (verit, best) ℬ-def D-precoeff-bound mult-right-mono zero-le-power)
 also have ... ≤ (∑ i∈δ-tuples − {?i0}. fact δ * ℒ * ℬ ^ D-exponent i)
   apply (rule sum-mono)
   using ℬ-def BB eval-def
   by (simp add: ℒ-ge-P-coeff mult-right-mono)
 also have ... = fact δ * ℒ * (∑ i∈δ-tuples − {?i0}. ℬ ^ D-exponent i)
   by (simp add: sum-distrib-left)
 also have ... ≤ fact δ * ℒ * (∑ r=0..(δ+1)^(ν+1)−1. ℬ ^ r)
   using h4 by (simp add: ℒ-pos[OF ‹δ > 0›])
 also have ... = (∑ r=0..(δ+1)^(ν+1)−1. fact δ * ℒ * ℬ ^ r)
   by (simp add: sum-distrib-left)
 also have ... ≤ (∑ r=0..(δ+1)^(ν+1)−1. (ℬ − 1) * ℬ ^ r)
   apply (rule sum-mono)
```

```
      using δ-ℒ-bound h1 ‹1 < nat ℬ› by fastforce
    also have ... < ℬ ^ ((δ+1) ^(ν+1))
      using series-bound[OF ‹ℬ ≥ 2›, of (δ + 1) ^ (ν + 1) − 1 λx. ℬ − 1] by
auto
   finally have |D − fact δ ∗ ?a0 ∗ ℬ ^ ((δ+1) ^(ν+1))| < ℬ ^ ((δ+1) ^(ν+1))
      by auto

    hence fact δ ∗ ?a0 ∗ ℬ ^ ((δ+1) ^(ν+1)) − D < ℬ ^ ((δ+1) ^(ν+1))
      by auto
    hence D > (fact δ ∗ ?a0 − 1) ∗ ℬ ^ ((δ+1) ^(ν+1))
      by (simp add: int-distrib(3))

    thus ?thesis
      using ‹?a0 > 0› BB
      unfolding ℬ-def eval-def
        by (smt (verit, ccfv-SIG) fact-gt-zero mult-le-0-iff mult-nonneg-nonneg
o-apply zero-le-power)
  qed

  moreover have (∑ i = 0..(2 ∗ δ + 1) ∗ n ν. of-nat (β div 2) ∗ b ^ δ ∗ ℬ ^
i) ≥ 0
    using BB ‹β ≥ 0› b
    unfolding eval-def
    apply simp
    by (rule sum-nonneg, simp)

  ultimately show ?thesis
    using ‹δ > 0› c-bound D-bound
    unfolding K-def eval-def
    by auto
 qed

 show S g ≥ 0
  unfolding S-def using N0
  by (simp add: ‹0 ≤ K g› assms(3))

 show R g > 0
  unfolding R-def
  using ‹T ≥ 0› ‹S g ≥ 0› ‹N ≥ 2› by auto
qed

lemma (in coding-variables) lower-bounds:
 fixes g :: int
 assumes a ≥ 0
 assumes f > 0
 assumes g ≥ 0
 assumes δ > 0
 assumes p0: P-coeff (replicate (ν+1) 0) > 0
 shows X g ≥ 3 ∗ b and Y ≥ 2^8 and Y ≥ b
```

```
proof −
  note nonneg = defs-non-negative[OF assms(1−4)]
  note simps = eval-def insertion-simps comp-def

  have β > 0
    unfolding β-def by auto
  have b ≤ 𝓑
    unfolding 𝓑-def
    using nonneg[OF p0] ‹δ > 0› ‹β > 0›
      by (smt (verit, del-insts) 𝓑-def comp-apply insertion-mult insertion-of-nat
insertion-pow
          mult-le-cancel-right1 of-nat-0-less-iff self-le-power)

  from ‹b ≤ 𝓑› have 3 * b ≤ 4 * 𝓑
    using nonneg(1)[OF p0] by auto
  also have ... ≤ N1
    unfolding N1-def
    using nonneg[OF p0] by auto
  also have ... ≤ N
    unfolding N-def
    using nonneg(2,6)[OF p0] N-def mult-le-cancel-right1 by force
  also have ... ≤ R g
    unfolding R-def
    using nonneg(2,4,5)[OF p0]
    using comp-apply mult-le-cancel-right1
    by (smt (verit) insertion-of-int of-int-1)
  also have ... ≤ X g
    unfolding X-def
    using nonneg[OF p0] by auto
  finally show X g ≥ 3 * b .

  have b ≥ 2
    unfolding b-def apply (simp add: assms)
    using assms(1−2) by (smt (verit) mult-le-cancel-right2)
  with ‹b ≤ 𝓑› have 𝓑 ≥ 2
    by auto
  hence N0 ≥ 2
    unfolding N0-def
    using ‹δ > 0›
    by auto (smt (verit, del-insts) mult-le-cancel-right2 zero-less-power)
  moreover from ‹𝓑 ≥ 2› have N1 ≥ 2^3
    unfolding N1-def
    using ‹δ > 0›
    by auto (smt (verit, del-insts) mult-le-cancel-right2 zero-less-power)
  ultimately have N ≥ 2^4
    unfolding N-def
    by (smt (verit, del-insts) mult-left-mono mult-right-mono not-exp-less-eq-0-int
        power3-eq-cube power4-eq-xxxx power-commuting-commutes)
  hence Y ≥ (2^4)^2
```

```
    unfolding Y-def
    unfolding power2-eq-square
    using mult-mono' by fastforce
  then show Y ≥ 2^8
    by auto

  from ‹b ≤ B› have b ≤ N0
    unfolding N0-def
    using nonneg[OF p0]
    by (smt (verit) add-gr-0 comp-apply self-le-power zero-less-one)
  also have ... ≤ N
    unfolding N-def
    using nonneg(6)[OF p0] ‹2 ^ 3 ≤ N1› by auto
  also have ... ≤ Y
    unfolding Y-def
    by (smt (verit) pos2 power2-less-0 self-le-power)
  finally show Y ≥ b .
qed

end
theory Coding-Theorem
 imports Coding-Theorem-Definitions ../Coding/Tau-Reduction ../Coding/Masking

    ../Coding/Lemma-1-8
begin


lemma digit-sum-bound-int:
  fixes f::nat ⇒ int
  assumes 1 < b and ∀ i∈{0..q}. f i < b
  shows (∑ i=0..q. f i * b^i) < b^(Suc q)
using assms(2)
proof (induct q)
  case 0 thus ?case by simp
next
  case (Suc q)
  have (∑ i=0..Suc q. f i * b^i) = (∑ i=0..q. f i * b^i) + f (Suc q) * b^(Suc q)
    by simp
  also have ... < b^(Suc q) + f (Suc q) * b^(Suc q)
    using Suc by auto
  also have ... ≤ b^(Suc q) + (b-1) * b^(Suc q)
    using Suc.prems assms(1) by auto
  also have ... = b^(Suc (Suc q))
     by (metis add.commute add-diff-eq cancel-comm-monoid-add-class.diff-cancel
group-cancel.rule0
        int-distrib(1) lambda-one power-Suc)
  finally show ?case .
qed
```

4.4 Proof

locale *coding-theorem* = *coding-variables* +
  assumes *a-nonneg*: $a \geq 0$
    and *f-pos*: $f > 0$
    and *b-power2*: *is-power2 b*
    and δ*-pos*: $\delta > 0$
begin

b being a power of 2 implies that the following are also powers of 2:

lemma $\mathcal{B}$*-power2*: *is-power2* $\mathcal{B}$
  unfolding $\mathcal{B}$*-def* β*-def* **using** *b-power2* **by** *simp*
lemma *N0-power2*: *is-power2 N0*
  unfolding *N0-def* **using** $\mathcal{B}$*-power2* **by** *simp*
lemma *N1-power2*: *is-power2 N1*
  unfolding *N1-def* **using** $\mathcal{B}$*-power2* **by** *simp*
lemma *N-power2*: *is-power2 N*
  unfolding *N-def* **using** *N0-power2 N1-power2* **by** *simp*
lemma *Ng-power2*: *is-power2 N*
  by (*simp add*: *N-power2*)

lemma $\mathcal{B}$*-ge-2*: $\mathcal{B} \geq 2$
  unfolding $\mathcal{B}$*-def* β*-def* **using** *r-pos*[*OF* δ*-pos*] *is-power2-ge1*[*OF b-power2*] **apply** *simp*
  by (*smt* (*verit*, *best*) *mult-pos-pos one-le-power pos-zmult-eq-1-iff self-le-power*)

lemma $\mathcal{B}$*-even*: *2 dvd* $\mathcal{B}$
  using $\mathcal{B}$*-power2* **unfolding** *is-power2-def* **using** $\mathcal{B}$*-ge-2*
  by (*metis Suc-eq-plus1 dvd-refl even-power mult-cancel-left1 not-gr-zero numeral-le-iff*
    numerals(*1*) *of-nat-1 of-nat-numeral plus-nat.add-0 power.simps*(*2*) *power-one-right*
    semiring-norm(*69*) *zero-neq-numeral*)

lemma *b-le-*$\mathcal{B}$: $b \leq \mathcal{B}$
  unfolding $\mathcal{B}$*-def* β*-def* **apply** *simp*
  using δ*-pos is-power2-ge1*[*OF b-power2*]
  by (*smt* (*verit*) *mult-le-cancel-right1 self-le-power zero-less-power*)

lemma *M-bound*: $0 \leq M \wedge M < N0$
proof
  show $0 \leq M$
  proof −
    have $m\ i \geq 0$ **for** i
      unfolding *m-def* **using** *b-le-*$\mathcal{B}$ *is-power2-ge1*[*OF* $\mathcal{B}$*-power2*] **by** *simp*
    hence $m\ i * \mathcal{B}\hat{}i \geq 0$ **for** i
      using *is-power2-ge1*[*OF* $\mathcal{B}$*-power2*] **by** *simp*
    thus *?thesis*
      unfolding *M-def* **by** (*meson sum-nonneg*)
  qed

```
  show M < N0
  proof −
    have m i < ℬ for i
      unfolding m-def using is-power2-ge1[OF b-power2] by simp
    thus ?thesis
      unfolding M-def N0-def n-def using digit-sum-bound-int ℬ-ge-2 by simp
  qed
qed

context
  fixes g::int
  assumes g-lower-bound: 0 ≤ g
      and g-upper-bound: g < 2 * b * ℬ ^ (n ν)
begin

lemma g-lt-N0: g < N0
proof −
  have g < 2 * b * ℬ ^ (n ν)
    using g-upper-bound by simp
  also have ... ≤ ℬ * ℬ ^ (n ν)
  proof −
    have 2 ≤ β
    by (metis β-def add-leE numeral-Bit0 one-le-numeral r-pos[OF δ-pos] self-le-power)

    hence 2 * b ≤ ℬ
      unfolding ℬ-def using is-power2-ge1[OF b-power2] δ-pos
    by (metis int-nat-eq int-one-le-iff-zero-less less-irrefl-nat mult-mono' of-nat-0-le-iff

          of-nat-le-iff of-nat-numeral self-le-power zless-nat-eq-int-zless)
    thus ?thesis
      using is-power2-ge1[OF ℬ-power2] by simp
  qed
  also have ... = N0
    unfolding N0-def n-def
    using power-add N0-def by simp
  finally show ?thesis .
qed

lemma c-bound: abs (c g) < 3 * b * ℬ ^ (n ν)
proof −
  have a ≤ 6*a+3
    using a-nonneg by auto
  also have ... ≤ (6*a+3) * f
    using f-pos a-nonneg by auto
  finally have 0: 1 + a ≤ b
    unfolding b-def by simp

  have abs (c g) = abs (1 + a * ℬ + g)
```

```
    using c-def by simp
  also have ... = 1 + a * ℬ + g
    using is-power2-ge1[OF ℬ-power2] g-lower-bound a-nonneg by auto
  also have ... ≤ (1 + a) * ℬ + g
    using is-power2-ge1[OF ℬ-power2]
    by (smt (verit, ccfv-SIG) mult-right-cancel mult-right-mono)
  also have ... ≤ b * ℬ + g
    by (smt (verit, best) 0 ℬ-power2 is-power2-ge1 mult-right-mono)
  also have ... ≤ b * ℬ ^ (n ν) + g
    unfolding n-def using is-power2-ge1 ℬ-power2 0 a-nonneg self-le-power
    by fastforce
  finally show ?thesis
    using g-upper-bound by auto
qed

lemma D-bound: abs D ≤ fact δ * ℒ * ℬ ^ (n (ν+1))
proof −
  have 0: i∈δ-tuples ⟹ D-precoeff i ≤ int (fact δ) for i
   unfolding D-precoeff-def δ-tuples-def using mchoose-le of-nat-mono by fastforce

  have D-exponent i ≤ n (ν+1) for i
    unfolding D-exponent-def by simp
  hence 1: ℬ ^ (D-exponent i) ≤ ℬ ^ (n (ν+1)) for i
    using is-power2-ge1[OF ℬ-power2] power-increasing by blast

  have abs D ≤ (∑ i∈δ-tuples. abs (D-precoeff i * P-coeff i * ℬ ^ (D-exponent i)))
    using D-def δ-tuples-finite by simp
 also have ... = (∑ i∈δ-tuples. abs (D-precoeff i) * abs (P-coeff i) * (abs ℬ) ^ (D-exponent
i))
    using abs-mult by (metis (no-types, opaque-lifting) power-abs)
  also have ... = (∑ i∈δ-tuples. D-precoeff i * abs (P-coeff i) * ℬ ^ (D-exponent
i))
    using is-power2-ge1[OF ℬ-power2] D-precoeff-def by auto
  also have ... ≤ (∑ i∈δ-tuples. fact δ * abs (P-coeff i) * ℬ ^ (n (ν+1)))
  proof −
    { fix i assume assm: i∈δ-tuples
      have 0 ≤ ℬ ^ (D-exponent i)
        using is-power2-ge1[OF ℬ-power2] by auto
      moreover have 0 ≤ D-precoeff i
        using D-precoeff-def by auto
      ultimately have D-precoeff i * abs (P-coeff i) * ℬ ^ (D-exponent i) ≤
        fact δ * abs (P-coeff i) * ℬ ^ (n (ν+1))
        using 0[OF assm] 1 by (simp add: mult-mono') }
    thus ?thesis
      using sum-mono by meson
  qed
  also have ... = fact δ * (∑ i∈δ-tuples. abs (P-coeff i)) * ℬ ^ (n (ν+1))
    using sum-distrib-left sum-distrib-right by metis
  finally show ?thesis
```

```
  unfolding ℒ-def by simp
qed

lemma c-δ-D-bound: 2 * abs ((c g) ^δ * D) ≤ ℬ ^((2*δ+1)*n ν + 1)
proof −
  have 0: β > int (2 * fact δ * ℒ * 3^δ)
  proof −
    have 3^δ ≤ (ν+3) ^δ
      by (simp add: power-mono)
    thus ?thesis
      using β-lower-bound
      by (smt (verit, best) nat-less-as-int nat-mult-less-cancel-disj not-less)
  qed

  have 2 * abs ((c g) ^δ * D) = 2 * (abs (c g)) ^δ * abs D
    by (simp add: abs-mult power-abs)
  also have ... ≤ 2 * (3 * b * ℬ ^(n ν)) ^δ * (fact δ * ℒ * ℬ ^(n (ν+1)))
    using D-bound c-bound by (simp add: δ-pos mult-mono)
  also have ... = 2 * (3^δ * b ^δ * ℬ ^(n ν * δ)) * (fact δ * ℒ * ℬ ^(n (ν+1)))
    using power-mult power-mult-distrib by (metis (no-types, opaque-lifting))
  also have ... = (2 * fact δ * ℒ * 3^δ) * (b ^δ * ℬ ^((2*δ+1) * n ν))
  proof −
    have X^(n ν * δ) * X^(n (ν+1)) = X^((2*δ+1) * n ν) for X::int
      using power-add n-def
     by (smt (verit) add-mult-distrib2 left-add-twice mult.commute power-one-right)
    thus ?thesis
      by simp
  qed
  also have ... ≤ β * (b ^δ * ℬ ^((2*δ+1) * n ν))
    using 0 is-power2-ge1 b-power2 ℬ-power2
    by (smt (verit, del-insts) mult-le-0-iff mult-right-mono zero-less-power)
  also have ... = ℬ * ℬ ^((2*δ+1) * n ν)
    using ℬ-def by simp
  finally show ?thesis
    using power-add by simp
qed

lemma K-bound: 0 ≤ K g ∧ 2 * K g < 3 * ℬ ^((2*δ+1)*n ν + 1)
proof
  define K2 :: int where K2 ≡ (∑ i=0..(2*δ+1)*n ν. of-nat (β div 2) * b ^δ *
ℬ ^i)
  have K-def-alt: K g = (c g) ^δ * D + K2
    using K2-def K-def by simp

  have K2-def-alt: K2 = (∑ i=0..(2*δ+1)*n ν. (ℬ div 2) * ℬ ^i)
  proof −
    have K2 = (∑ i=0..(2*δ+1)*n ν. (β div 2) * b ^δ * ℬ ^i)
      using K2-def by simp
    also have ... = (∑ i=0..(2*δ+1)*n ν. (ℬ div 2) * ℬ ^i)
```

```
    proof −
      have 2 dvd β
        unfolding β-def using r-pos[OF δ-pos] by simp
      hence (β div 2) * b^δ = B div 2
        by (simp add: B-def dvd-div-mult zdiv-int)
      thus ?thesis
        using sum.cong by auto
     qed
     finally show ?thesis .
  qed

  have K2-bound: B^((2*δ+1) * n ν + 1) ≤ 2 * K2 ∧ K2 < B^((2*δ+1) * n ν
+ 1)
  proof
   have K2 = (∑ i=0..<(2*δ+1)*n ν. (B div 2) * B^i) + (B div 2) * B^((2*δ+1)
* n ν)
     unfolding K2-def-alt using n-def
    by (smt (verit) atLeastLessThanSuc-atLeastAtMost sum.atLeast0-lessThan-Suc)
    also have ... ≥ (B div 2) * B^((2*δ+1) * n ν)
    proof −
      have (B div 2) * B^i ≥ 0 for i
        using is-power2-ge1[OF B-power2] by auto
      thus ?thesis
        using sum-nonneg by (smt (verit, ccfv-threshold))
    qed
    finally have 2 * K2 ≥ 2 * (B div 2) * B^((2*δ+1) * n ν)
      by simp
    thus 2 * K2 ≥ B^((2*δ+1) * n ν + 1)
      using B-even power-add by simp

    show K2 < B^((2*δ+1)*n ν + 1)
    proof −
      have B div 2 < B
        using B-ge-2 by simp
      thus ?thesis
        unfolding K2-def-alt using digit-sum-bound-int B-ge-2 by simp
    qed
  qed

  show 0 ≤ K g
    unfolding K-def-alt using c-δ-D-bound K2-bound by simp linarith
  show 2 * (K g) < 3 * B^((2*δ+1)*n ν + 1)
    unfolding K-def-alt using c-δ-D-bound K2-bound by simp linarith
qed
```

technical condition 2.7, first part

```
lemma T-bound: 0 ≤ T ∧ T < N
proof
  show 0 ≤ T
```

```
  proof −
    have T = M + (ℬ − 2) * ℬ^(n (ν+1)) * N0
      using T-def n-def by simp
    thus ?thesis
      using M-bound ℬ-ge-2 is-power2-ge1[OF N0-power2] by auto
  qed

  have 0: 1 + (ℬ − 2) * ℬ^(n (ν+1)) ≤ N1
  proof −
     have 1 + (ℬ − 2) * ℬ^(n (ν+1)) ≤ 2 * ℬ^(n (ν+1)) + (ℬ − 2) * ℬ^(n
(ν+1))
      using is-power2-ge1[OF ℬ-power2] by (smt (z3) one-le-power)
    also have ... = ℬ^(n (ν+1) + 1)
     using ℬ-ge-2 by (metis add.commute diff-add-cancel int-distrib(1) power-add
power-one-right)
    also have ... ≤ 4 * ℬ^((2*δ+1) * n ν + 1)
    proof −
      have n (ν+1) + 1 ≤ (2*δ+1) * n ν + 1
        unfolding n-def by simp
      thus ?thesis
           using is-power2-ge1[OF ℬ-power2] power-increasing by (smt (verit,
ccfv-SIG) K-bound)
    qed
    also have ... = N1
      using N1-def n-def by simp
    finally show ?thesis .
  qed

  show T < N
  proof −
    have T < N0 + (ℬ − 2) * ℬ^(n (ν+1)) * N0
      using T-def M-bound n-def by simp
    also have ... = (1 + (ℬ − 2) * ℬ^(n (ν+1))) * N0
      by (simp add: algebra-simps)
    also have ... ≤ N1 * N0
      using 0 is-power2-ge1[OF N0-power2] by simp
    also have ... = N
      using N-def by simp
    finally show ?thesis .
  qed
qed
```

technical condition 2.7, second part

```
lemma S-bound: 0 ≤ S g ∧ S g < N
proof
  show 0 ≤ S g
  proof −
    have S g = g + 2 * K g * N0
      using S-def by simp
```

```
    thus ?thesis
      using K-bound g-lower-bound is-power2-ge1[OF N0-power2] by simp
  qed

  have 0: g < \<B>^((2*\<delta>+1)*n \<nu> + 1)
  proof −
    have g < \<B>^(n \<nu> + 1)
      using g-lt-N0 N0-def n-def by simp
    also have ... ≤ \<B>^((2*\<delta>+1)*n \<nu> + 1)
    proof −
      have n \<nu> + 1 ≤ (2*\<delta>+1)*n \<nu> + 1
        unfolding n-def by simp
      thus ?thesis
        using is-power2-ge1[OF \<B>-power2] power-increasing by blast
    qed
    finally show ?thesis .
  qed

  show S g < N
  proof −
    have S g = g + 2 * K g * N0
      using S-def by simp
    also have ... ≤ (g + 2 * K g) * N0
    proof −
      have g ≤ g * N0
        using g-lower-bound is-power2-ge1[OF N0-power2]
        by (simp add: mult-le-cancel-left1)
      thus ?thesis
        by (simp add: algebra-simps)
    qed
    also have ... < N1 * N0
    proof −
      have g + 2 * K g < N1
        unfolding N1-def using 0 K-bound n-def by simp
      thus ?thesis
        using is-power2-ge1[OF N0-power2] by auto
    qed
    also have ... = N
      using N-def by simp
    finally show ?thesis .
  qed
qed
```

Technical condition 2.8

```
lemma tau-S-T-decomp: \<tau> (nat (S g)) (nat (T)) =
  \<tau> (nat g) (nat (M)) + \<tau> (nat (2 * K g)) (nat ((\<B> − 2) * \<B>^(n (\<nu>+1))))
proof −
  have 0: nat (S g) = nat g + nat (K g) * nat (2 * N0)
  proof −
```

```
    have nat (S g) = nat (g + 2 * K g * N0)
      using S-def by simp
    also have ... = nat g + 2 * nat (K g) * nat (N0)
      using g-lower-bound K-bound is-power2-ge1[OF N0-power2]
      by (simp add: nat-add-distrib nat-mult-distrib)
    finally show ?thesis
      by auto
  qed

  have 1: nat (T) = nat (M) + nat ((ℬ div 2 - 1) * ℬ^(n (ν+1))) * nat (2 *
N0)
  proof -
    have nat (T) = nat (M + (ℬ - 2) * ℬ^(n (ν+1)) * N0)
      using T-def n-def by simp
    also have ... = nat (M + 2 * (ℬ div 2 - 1) * ℬ^(n (ν+1)) * N0)
      using ℬ-even by simp
   also have ... = nat (M) + nat ((ℬ div 2 - 1) * ℬ^(n (ν+1))) * nat (2 * N0)
      using ℬ-ge-2 M-bound is-power2-ge1[OF N0-power2]
      by (simp add: nat-power-eq nat-add-distrib nat-mult-distrib)
    finally show ?thesis .
  qed

  have τ (nat (S g)) (nat (T)) =
    τ (nat g + nat (K g) * nat (2 * N0))
      (nat (M) + nat ((ℬ div 2 - 1) * ℬ^(n (ν+1))) * nat (2 * N0))
    using 0 1 by metis
  also have ... = τ (nat g) (nat (M)) + τ (nat (K g)) (nat ((ℬ div 2 - 1) * ℬ^(n
(ν+1))))
  proof -
    obtain k where k-def: nat (2 * N0) = 2^k
      using N0-power2 is-power2-def by fastforce
    have nat g + nat (M) < nat (2 * N0)
      using M-bound g-lt-N0 by auto
    thus ?thesis
      using count-carries-add-shift-no-overflow k-def by metis
  qed
  also have ... =  τ (nat g) (nat (M)) + τ (nat (2 * K g)) (nat ((ℬ - 2) * ℬ^(n
(ν+1))))
  proof -
    have 0: 2 * nat (K g) = nat (2 * K g)
      by simp
    have 1: ℬ - 2 = 2 * (ℬ div 2 - 1)
      using ℬ-even by simp
     have 2: 2 * nat ((ℬ div 2 - 1) * ℬ^(n (ν+1))) = nat ((ℬ - 2) * ℬ^(n
(ν+1)))
      unfolding 1 by linarith
    show ?thesis
      using count-carries-even-even 0 2 by metis
  qed
```

finally show *?thesis* **.**
qed

end

Helper lemmas for masking

lemma *n-masking-lemma*[*simp*]:
assumes *masking-lemma* δ ν (*nat* $\mathcal{B}$) (*nat b*) *C*
shows *masking-lemma.n* δ = *n*
unfolding *n-def* **using** *masking-lemma.n-def assms* **by** *auto*

lemma *m-masking-lemma*[*simp*]:
assumes *masking-lemma* δ ν (*nat* $\mathcal{B}$) (*nat b*) *C*
shows *masking-lemma.m* δ ν (*nat* $\mathcal{B}$) (*nat b*) *j* = *m j*
proof −
have *0*: *int* (*nat* $\mathcal{B}$) = $\mathcal{B}$
using *is-power2-ge1*[*OF* $\mathcal{B}$*-power2*] **by** *simp*
have *1*: *int* (*nat b*) = *b*
using *is-power2-ge1*[*OF b-power2*] **by** *simp*
have *2*: *int* (*nat* $\mathcal{B}$ − *nat b*) = ($\mathcal{B}$ − *b*)
proof −
have *int* (*nat* $\mathcal{B}$ − *nat b*) = *int* (*nat* $\mathcal{B}$) − *int* (*nat b*)
using *b-le-*$\mathcal{B}$ **by** *auto*
thus *?thesis*
using *0 1* **by** *simp*
qed
have *3*: *int* (*nat* $\mathcal{B}$ − *1*) = ($\mathcal{B}$ − *1*)
proof −
have *1* ≤ *nat* $\mathcal{B}$
using *is-power2-ge1*[*OF* $\mathcal{B}$*-power2*] **by** *auto*
thus *?thesis*
using *0* **by** *simp*
qed

have *masking-lemma.m* δ ν (*nat* $\mathcal{B}$) (*nat b*) *j* =
(*if j* ∈ *n* ‘ {*1..*ν} *then int* (*nat* $\mathcal{B}$ − *nat b*) *else int* (*nat* $\mathcal{B}$ − *1*))
unfolding *masking-lemma.m-def*[*OF assms*] *n-masking-lemma*[*OF assms*] **by** *auto*
also have ... = (*if j* ∈ *n* ‘ {*1..*ν} *then* ($\mathcal{B}$ − *b*) *else* ($\mathcal{B}$ − *1*))
using *2 3* **by** *simp*
also have ... = *m j*
using *m-def* **by** *simp*
finally show *?thesis* **.**
qed

lemma *M-masking-lemma*[*simp*]:
assumes *masking-lemma* δ ν (*nat* $\mathcal{B}$) (*nat b*) *C*
shows *masking-lemma.M* δ ν (*nat* $\mathcal{B}$) (*nat b*) = *nat M*
proof −

have *masking-lemma.M* δ ν (*nat* $\mathcal{B}$) (*nat* b) = ($\sum j$=*0*..n ν. m j * $\mathcal{B}$^j)
unfolding *masking-lemma.M-def*[*OF assms*] **using** *is-power2-ge1*[*OF* $\mathcal{B}$*-power2*]

unfolding *n-masking-lemma*[*OF assms*] *m-masking-lemma*[*OF assms*, *symmetric*] **by** *auto*
thus *?thesis*
using *M-def* **by** *auto*
qed

Helper lemmas to apply Lemma 1.8

We can only apply Lemma 1.8 when g can be decomposed in base $\mathcal{B}$ with digits $<b$

context
fixes g :: *int*
and z :: *nat* ⇒ *int*
assumes *g-sum*: $g = (\sum i$=*1*..ν. z i * $\mathcal{B}$^(n i))
and *z-bound*: ∀ i. *0* ≤ z i ∧ z i < b
begin

This is quite verbose but justified by the following lemma

definition *z-list* :: *nat list* **where**
z-list ≡ *nat a* # *map* (*nat* ∘ z ∘ *nat*) [*1*..ν]

lemma *z-list-nth-head*: *z-list*!*0* = *nat a*
unfolding *z-list-def* **by** *simp*

lemma *z-list-nth-tail*: *1* ≤ i ⟹ i ≤ ν ⟹ *z-list*!i = *nat* (z i)
proof −
assume *1* ≤ i **and** i ≤ ν
then obtain j **where** *j-def*: i = j + *1* **and** j < ν
using *nat-le-iff-add* **by** *fastforce*

have *z-list*!i = (*map* (*nat* ∘ z ∘ *nat*) [*1*..ν])!j
unfolding *z-list-def j-def* **by** *simp*
also have ... = (*nat* ∘ z ∘ *nat*) ([*1*..ν]!j)
using *List.nth-map* ‹j < ν› **by** *simp*
also have ... = *nat* (z (j+*1*))
by (*metis Suc-eq-plus1* ‹i ≤ ν› *add.commute comp-apply int-ops*(*4*) *j-def nat-int nth-upto*
of-nat-mono)
finally show *?thesis*
using *j-def* **by** *simp*
qed

lemma *length-z-list*[*simp*]: *length z-list* = ν+*1*
unfolding *z-list-def* **by** *auto*

lemma δ*-lemma-1-8*[*simp*]: *Lemma-1-8-Defs.*δ P = δ

using *Lemma-1-8-Defs.δ-def δ-def* **by** *simp*

lemma *ν-lemma-1-8*[*simp*]: *Lemma-1-8-Defs.ν P* = *ν*
using *Lemma-1-8-Defs.ν-def ν-def* **by** *simp*

lemma *n-lemma-1-8*[*simp*]: *Lemma-1-8-Defs.n P* = *n*
using *Lemma-1-8-Defs.n-def n-def δ-lemma-1-8* **by** *auto*

lemma *insertion-z-assign*[*simp*]:
insertion (*Lemma-1-8-Defs.z-assign z-list*) *P* = *insertion* (*z*(*0* := *a*)) *P*
proof −
have *i*∈*vars P* ⟹ *Lemma-1-8-Defs.z-assign z-list i* = (*z*(*0* := *a*)) *i* **for** *i*
proof −
assume *i*∈*vars P*
hence *i-bound*: *i* ≤ *ν*
unfolding *ν-def max-vars-def* **by** (*simp add*: *vars-finite*)

have *Lemma-1-8-Defs.z-assign z-list i* = *int* (*z-list* !$_0$ *i*)
unfolding *Lemma-1-8-Defs.z-assign-def*
by (*metis Suc-eq-plus1 i-bound le-imp-less-Suc length-map length-z-list nth0-nth nth-map*)
also have ... = *int* (*z-list* ! *i*)
using *length-z-list i-bound* **by** (*simp add*: *nth0-nth*)
also have ... = *int* (*if i* = *0 then nat a else nat* (*z i*))
using *z-list-nth-head z-list-nth-tail i-bound* **by** *auto*
also have ... = (*if i* = *0 then a else z i*)
using *a-nonneg z-bound* **by** *auto*
also have ... = (*z*(*0* := *a*)) *i*
by *auto*
finally show *?thesis* **.**
qed
thus *?thesis*
using *insertion-irrelevant-vars* **by** *metis*
qed

lemma *S-lemma-1-8*[*simp*]:
int (*Lemma-1-8-Defs.S P* (*nat ℬ*)) = (∑ *i*=*0*..(*2*∗*δ*+*1*)∗*n ν*. *int* (*β div 2*) ∗ *b*^*δ* ∗ *ℬ*^*i*)
proof −
have *int* (*Lemma-1-8-Defs.S P* (*nat ℬ*)) =
(∑ *i*≤(*2*∗*δ*+*1*) ∗ *n ν*. *int* (((*nat ℬ*) *div 2*) ∗ (*nat ℬ*)^*i*))
unfolding *Lemma-1-8-Defs.S-def Lemma-1-8-Defs.b-def δ-lemma-1-8 n-lemma-1-8 ν-lemma-1-8* **by** *simp*
also have ... = (∑ *i*≤(*2*∗*δ*+*1*) ∗ *n ν*. (*ℬ div 2*) ∗ *ℬ*^*i*)
proof −
have (*nat ℬ*) *div 2* = *nat* (*ℬ div 2*)
using *ℬ-even* **by** *auto*
thus *?thesis*
using *is-power2-ge1*[*OF ℬ-power2*] **by** *auto*

```
  qed
  also have ... = (∑ i=0..(2*δ+1) * n ν. (ℬ div 2) * ℬ ^ i)
    using atLeast0AtMost by presburger
  also have ... = (∑ i=0..(2*δ+1)*n ν. int (β div 2) * b ^ δ * ℬ ^ i)
  proof −
    have ℬ div 2 = int (β div 2) * b ^ δ
      unfolding ℬ-def β-def by (simp add: dvd-div-mult r-pos[OF δ-pos] zdiv-int)
    thus ?thesis
      by auto
  qed
  finally show ?thesis .
qed

lemma c-lemma-1-8[simp]:
  insertion (Lemma-1-8-Defs.ℬ-assign (nat ℬ)) (Lemma-1-8-Defs.c P z-list) = a
* ℬ + g
  (is ?lhs = ?rhs)
proof −
  have ?lhs = (∑ i≤ν. int (z-list!i) * (Lemma-1-8-Defs.ℬ-assign (nat ℬ) (0::nat)) ^ (n
i))
    unfolding Lemma-1-8-Defs.c-def Lemma-1-8-Defs.X-def n-lemma-1-8 ν-lemma-1-8

    by simp
  also have ... = (∑ i≤ν. int (z-list!i) * ℬ ^ (n i))
    unfolding Lemma-1-8-Defs.ℬ-assign-def using ℬ-ge-2 by simp
  also have ... = int (z-list!0) * ℬ ^ (n 0) + (∑ i=1..ν. int (z-list!i) * ℬ ^ (n i))
   by (simp add: atLeastSucAtMost-greaterThanAtMost atMost-atLeast0 sum.head)
  also have ... = a * ℬ + (∑ i=1..ν. int (z-list!i) * ℬ ^ (n i))
    using n-def z-list-def a-nonneg by auto
  also have ... = a * ℬ + g
    unfolding g-sum using z-bound z-list-nth-tail by auto
  finally show ?thesis .
qed

lemma D-lemma-1-8[simp]:
  insertion (Lemma-1-8-Defs.ℬ-assign (nat ℬ)) (Lemma-1-8-Defs.D P) = D
  (is ?lhs = ?rhs)
proof −
  have 0: Lemma-1-8-Defs.D-precoeff P i = D-precoeff i for i
    unfolding Lemma-1-8-Defs.D-precoeff-def D-precoeff-def by simp
  have 1: Lemma-1-8-Defs.P-coeff P i = P-coeff i for i
    unfolding Lemma-1-8-Defs.P-coeff-def P-coeff-def by simp
  have 2: Lemma-1-8-Defs.D-exponent P i = D-exponent i for i
    unfolding Lemma-1-8-Defs.D-exponent-def D-exponent-def by simp
  have 3: Lemma-1-8-Defs.δ-tuples P = δ-tuples
    unfolding Lemma-1-8-Defs.δ-tuples-def δ-tuples-def by simp

  have ?lhs = (∑ i∈δ-tuples. D-precoeff i * P-coeff i * (Lemma-1-8-Defs.ℬ-assign
(nat ℬ) (0::nat)) ^ (D-exponent i))
```

```
  unfolding Lemma-1-8-Defs.D-def Lemma-1-8-Defs.X-def by (simp add: 0 1 2
3)
 also have ... = D
  unfolding D-def Lemma-1-8-Defs.ℬ-assign-def using ℬ-ge-2 by simp
 finally show ?thesis .
qed

lemma R-lemma-1-8[simp]:
 insertion (Lemma-1-8-Defs.ℬ-assign (nat ℬ)) (Lemma-1-8-Defs.R P z-list) = (c
g) ^δ * D
 (is ?lhs = ?rhs)
proof −
 have ?lhs = (1 + a * ℬ + g) ^δ * D
  unfolding Lemma-1-8-Defs.R-def by (simp add: algebra-simps)
 also have ... = (c g) ^δ * D
  unfolding c-def by simp
 finally show ?thesis .
qed

lemma K-lemma-1-8[simp]:
 Lemma-1-8-Defs.K P (nat ℬ) z-list = K g
 unfolding Lemma-1-8-Defs.K-def K-def by simp

lemma lemma-1-8-helper:
 shows insertion (z(0 := a)) P = 0 ⟷ τ (nat (2 * K g)) (nat ((ℬ − 2) *
ℬ ^(n (ν+1)))) = 0
  and K g > ℬ ^((2*δ+1) * n ν)
  and K g < ℬ ^((2*δ+1) * n ν + 1)
proof −
 have z-sum-bound: sum-list z-list < (ν+1) * nat b
 proof −
  have sum-list z-list = (∑ i=0..ν. z-list!i)
   using sum-list-sum-nth length-z-list
   by (metis Suc-eq-plus1 atLeastLessThanSuc-atLeastAtMost)
  also have ... = z-list!0 + (∑ i=1..ν. z-list!i)
   by (simp add: atLeastSucAtMost-greaterThanAtMost sum.head)
  also have ... = nat a + (∑ i=1..ν. nat (z i))
   using z-list-nth-head z-list-nth-tail by auto
  also have ... ≤ nat a + (∑ i=1..ν. nat b)
   using z-bound by (smt (verit, del-insts) add-left-mono nat-le-eq-zle sum-mono)
  also have ... = nat a + card {1..ν} * nat b
   using sum-constant by auto
  also have ... = nat a + ν * nat b
   by auto
  also have ... < nat b + ν * nat b
   unfolding b-def by simp (smt (verit) mult-less-cancel-left2 a-nonneg f-pos)
  finally show ?thesis
   by auto
 qed
```

```
  have B-lower-bound: 2 * fact δ * L * (sum-list z-list + 1) ^δ < nat B
  proof −
    have 0: (sum-list z-list + 1) ^δ < (ν+3) ^δ * (nat b) ^δ
    proof −
      have sum-list z-list + 1 ≤ (ν+1) * nat b
        using z-sum-bound by auto
      also have ... < (ν+3) * nat b
        using is-power2-ge1[OF b-power2]
      by (smt (verit, ccfv-threshold) add-less-cancel-left mult.commute nat-mult-less-cancel-disj

            one-less-numeral-iff semiring-norm(77) zero-less-nat-eq)
      finally show ?thesis
        using power-mult-distrib zero-le by (metis δ-pos power-strict-mono)
    qed

    have 2 * fact δ * L * (sum-list z-list + 1) ^δ < 2 * fact δ * L * (ν+3) ^δ *
(nat b) ^δ
      using 0 fact-ge-1 L-pos[OF δ-pos] by auto
    also have ... ≤ β * (nat b) ^δ
      using β-lower-bound by auto
    also have ... = nat B
    unfolding B-def using is-power2-ge1[OF b-power2] by (simp add: nat-mult-distrib
nat-power-eq)
    finally show ?thesis .
  qed

  have 0: Lemma-1-8 P (nat B) (int L) z-list
    unfolding Lemma-1-8-def K-Nonnegative-def
   using B-power2 is-power2-ge1 L-ge-max-coeff[OF δ-pos] δ-pos L-pos[OF δ-pos]
         B-lower-bound Lemma-1-8-axioms.intro B-even even-nat-iff by auto
  show K g > B ^((2*δ+1) * n ν)
    using Lemma-1-8.lemma-1-8(2)[OF 0] B-ge-2 by simp
  show K g < B ^((2*δ+1) * n ν + 1)
    using Lemma-1-8.lemma-1-8(3)[OF 0] B-ge-2 by simp
  show insertion (z(0 := a)) P = 0 ⟷ τ (nat (2 * K g)) (nat ((B − 2) * B ^(n
(ν+1)))) = 0
    using Lemma-1-8.lemma-1-8(1)[OF 0] B-ge-2 nat-diff-distrib nat-mult-distrib
nat-power-eq
    by auto
qed

end

lemma aux-sum-bound-reindex-n:
  0 ≤ (x::int) ⟹ (∑ i=0..q. x^(n i)) ≤ (∑ i=0..n q. x^i)
proof (induct q)
  case 0 thus ?case apply simp
   by (metis member-le-sum atLeastAtMost-iff dual-order.refl finite-atLeastAtMost
```

```
zero-le
        zero-le-power)
next
  case (Suc q)

  have 0: Suc (n q) ≤ n (Suc q)
    unfolding n-def using δ-pos by auto

  have (∑ i=0..Suc q. x^(n i)) = (∑ i=0..q. x^(n i)) + x^(n (Suc q))
    by simp
  also have ... ≤ (∑ i=0..n q. x^i) + x^(n (Suc q))
    using Suc by simp
  also have ... ≤ (∑ i=0..n q. x^i) + (∑ i=Suc(n q)..n (Suc q). x^i)
    using 0 Suc.prems member-le-sum
   by (metis add.commute add-le-cancel-right atLeastAtMost-iff finite-atLeastAtMost
order.refl
         zero-le-power)
  also have ... = (∑ i=0..n (Suc q). x^i)
  proof −
    have {0..n q} ∪ {Suc (n q)..n (Suc q)} = {0..n (Suc q)}
      using 0 by auto
    moreover have {0..n q} ∩ {Suc (n q)..n (Suc q)} = {}
      by auto
    ultimately show ?thesis
      using sum.union-disjoint by (metis finite-atLeastAtMost)
  qed
  finally show ?case .
qed

lemma coding-theorem-direct:
  statement1-strong y ⟹ (∃ g. statement2-strong g)
proof
  assume assm: statement1-strong y

  have 1 < b
    using assm unfolding statement1-strong-def statement1-weak-def
    by (smt (verit, del-insts))

  have y-bound: 0 ≤ y i ∧ y i < b for i
    using assm unfolding statement1-strong-def statement1-weak-def by auto

  define g where g ≡ (∑ i=1..ν. y i * ℬ^(n i))

  have g-lower-bound: b ≤ g
  proof −
    from assm obtain j where 0: y j > 0 and j-1ν: j ∈ {1..ν}
      unfolding statement1-strong-def statement1-weak-def
      by (meson leD linorder-neqE-linordered-idom)
```

```
    have g ≥ y j * ℬ^(n j)
     using assm member-le-sum[OF j-1ν] unfolding statement1-strong-def statement1-weak-def

     by (smt (verit, del-insts) b-le-ℬ finite-atLeastAtMost g-def mult-nonneg-nonneg
mult-zero-left zero-le-power)
    hence g ≥ ℬ^(n j) using 0 is-power2-ge1[OF ℬ-power2]
      by (smt (verit, ccfv-SIG) mult-le-cancel-right1 zero-less-power)
    hence g ≥ ℬ
      unfolding n-def using is-power2-ge1[OF ℬ-power2]
      by (smt (verit, ccfv-SIG) δ-pos add-gr-0 self-le-power zero-less-power)
    thus g ≥ b
      using b-le-ℬ by simp
  qed

  have g-upper-bound: g < b * ℬ^(n ν)
  proof −
    have 0: y i * ℬ^(n i) ≤ (b − 1) * ℬ^(n i) for i
      using y-bound is-power2-ge1[OF ℬ-power2] by auto

    have g = (∑ i∈{0<..ν}. y i * ℬ^(n i))
      unfolding g-def by (simp add: atLeastSucAtMost-greaterThanAtMost)
    also have ... ≤ y 0 * ℬ^(n 0) + (∑ i∈{0<..ν}. y i * ℬ^(n i))
      using is-power2-ge1 ℬ-power2 b-power2 y-bound
      by (smt (verit, del-insts) mult-nonneg-nonneg zero-le-power)
    also have ... = (∑ i∈{0..ν}. y i * ℬ^(n i))
      using sum.head[of 0 ν] by (smt (verit) zero-le)
    also have ... ≤ (∑ i∈{0..ν}. (b − 1) * ℬ^(n i))
      using 0 sum-mono by meson
    also have ... = (b − 1) * (∑ i=0..ν. ℬ^(n i))
      using sum-distrib-left by metis
    also have ... ≤ (b − 1) * (∑ i=0..n ν. ℬ^i)
     using aux-sum-bound-reindex-n is-power2-ge1[OF b-power2] ℬ-ge-2 by (simp
add: mult-left-mono)
    also have ... ≤ (b − 1) * (∑ i=0..<n ν. ℬ^i) + (b − 1) * ℬ^(n ν)
     by (metis atLeastLessThanSuc-atLeastAtMost distrib-left order-refl sum.atLeast0-lessThan-Suc)
    also have ... ≤ (ℬ − 1) * (∑ i=0..<n ν. ℬ^i) + (b − 1) * ℬ^(n ν)
       by (smt (verit, ccfv-SIG) b-le-ℬ ℬ-power2 is-power2-ge1 mult-right-mono
sum-nonneg
         zero-le-power)
    also have ... = ℬ^(n ν) − 1 + (b − 1) * ℬ^(n ν)
      by (simp add: ℬ-ge-2 atLeast0LessThan power-diff-1-eq)
    also have ... = b * ℬ^(n ν) − 1
      by (simp add: algebra-simps)
    also have ... < b * ℬ^(n ν)
      by auto
    finally show ?thesis .
  qed

  have tau-KB: τ (nat (2 * K g)) (nat ((ℬ − 2) * ℬ^(n (ν+1)))) = 0
```

```
  proof −
    have insertion (y(0 := a)) P = 0
      using assm unfolding statement1-strong-def statement1-weak-def by auto
    thus ?thesis
      using lemma-1-8-helper(1) g-def y-bound by auto
  qed

  have tau-gM: τ (nat g) (nat (M)) = 0
  proof −
    define C where C ≡ b * B^(n ν)

  interpret masking: masking-lemma δ ν (nat B) (nat b) (nat C)
     unfolding masking-lemma-def C-def n-def using δ-pos B-power2 b-power2
B-ge-2 b-le-B is-power2-ge1
   by (smt (verit, ccfv-SIG) B-even dvd-imp-le even-nat-iff mult-le-mono1 nat-eq-iff2

      nat-less-eq-zless nat-mult-distrib nat-power-eq order-le-less zero-less-nat-eq)

  have 1: nat g = (∑ i=1..ν. nat (y i) * (nat B)^(n i))
    proof −
      have 0 ≤ y i * B^(n i) for i
        using y-bound is-power2-ge1[OF B-power2] by auto
      hence nat g = (∑ i=1..ν. nat (y i * B^(n i)))
        unfolding g-def using nat-sum-distrib by auto
      also have ... = (∑ i=1..ν. nat (y i) * (nat B)^(n i))
         using y-bound is-power2-ge1[OF B-power2] nat-mult-distrib nat-power-eq
by auto
      finally show ?thesis .
    qed

    show ?thesis
    using masking.masking-lemma 1 y-bound
    unfolding n-masking-lemma[OF masking.masking-lemma-axioms]
            M-masking-lemma[OF masking.masking-lemma-axioms]
      by (meson nat-less-eq-zless)
  qed

  have tau-ST: τ (nat (S g)) (nat (T)) = 0
   using tau-S-T-decomp g-lower-bound g-upper-bound is-power2-ge1[OF b-power2]
tau-KB tau-gM
    by auto

  have dvd-XY: Y dvd (2 * nat (X g) choose nat (X g))
  proof −
    have 0: 0 ≤ T ∧ T < N
      using T-bound g-lower-bound g-upper-bound is-power2-ge1[OF b-power2] by
force
    have 1: 0 ≤ S g ∧ S g < N
      using S-bound g-lower-bound g-upper-bound is-power2-ge1[OF b-power2] by
```

```
force
    have 2: Tau-Reduction (nat (N)) (nat (S g)) (nat (T))
      unfolding Tau-Reduction-def using N-power2 0 1
    by (metis int-one-le-iff-zero-less is-power2-ge1 nat-0-le order-le-less zless-nat-conj)
   have 3: Tau-Reduction.R (nat (N)) (nat (S g)) (nat (T)) = nat (R g) (is ?lhs
= ?rhs)
    proof −
      have ?lhs = nat ((S g + T + 1) * N + T + 1)
     unfolding Tau-Reduction.R-def[OF 2] using 0 1 by (simp add: nat-add-distrib
nat-mult-distrib)
      also have ... = nat (R g)
        unfolding R-def by simp
      finally show ?thesis unfolding Tau-Reduction.R-def[OF 2] by simp
    qed

    have (nat N)^2 dvd (2 * (nat N − 1) * (nat (R g)) choose (nat N − 1) *
nat (R g))
    using Tau-Reduction.tau-as-binomial-coefficient[OF 2] 3 tau-ST by presburger
    hence nat ((N)^2) dvd (2 * nat ((N − 1) * R g) choose nat ((N − 1) * R
g))
      using is-power2-ge1[OF N-power2]
    by (simp add: ab-semigroup-mult-class.mult-ac(1) nat-diff-distrib nat-mult-distrib
nat-power-eq)
    hence nat (Y) dvd (2 * nat (X g) choose nat (X g))
      unfolding X-def Y-def by simp
    thus ?thesis
      by auto
  qed

  have b * B^(n ν) = (int γ) * b^α
    unfolding B-def α-def γ-def by (simp add: power-mult power-mult-distrib)
  thus statement2-strong g
    unfolding statement2-strong-def using g-lower-bound g-upper-bound dvd-XY
by auto
qed

lemma coding-theorem-reverse:
  statement2-weak g ⟹ (∃ y. statement1-weak y)
proof −
  assume assm: statement2-weak g

  have g-bound: 0 ≤ g ∧ g < 2 * b * B^(n ν)
  proof −
    have b * B^(n ν) = (int γ) * b^α
      unfolding B-def γ-def α-def by (simp add: power-mult power-mult-distrib)
    thus ?thesis
      using assm unfolding statement2-weak-def by auto
  qed
```

```
  have tau-ST: τ (nat (S g)) (nat (T)) = 0
  proof −
    have 0: 0 ≤ T ∧ T < N
      using T-bound g-bound is-power2-ge1[OF b-power2] by force
    have 1: 0 ≤ S g ∧ S g < N
      using S-bound g-bound is-power2-ge1[OF b-power2] by force
    have 2: Tau-Reduction (nat (N)) (nat (S g)) (nat (T))
      unfolding Tau-Reduction-def using N-power2 0 1
     by (metis int-one-le-iff-zero-less is-power2-ge1 nat-0-le order-le-less zless-nat-conj)
    have 3: Tau-Reduction.R (nat (N)) (nat (S g)) (nat (T)) = nat (R g) (is ?lhs
= ?rhs)
    proof −
      have ?lhs = nat ((S g + T + 1) * N + T + 1)
     unfolding Tau-Reduction.R-def[OF 2] using 0 1 by (simp add: nat-add-distrib
nat-mult-distrib)
      also have ... = nat (R g)
        unfolding R-def by simp
      finally show ?thesis .
    qed

    have nat Y = (nat N)^2
    unfolding Y-def using is-power2-ge1[OF N-power2] by (simp add: nat-power-eq)
    moreover have 2 * nat (X g) = 2 * (nat N − 1) * nat (R g)
      unfolding X-def using is-power2-ge1[OF N-power2]
      by simp (smt (verit, best) nat-1 nat-diff-distrib nat-mult-distrib)
    moreover have nat (X g) = (nat N − 1) * nat (R g)
      unfolding X-def using is-power2-ge1[OF N-power2]
      by simp (smt (verit, best) nat-1 nat-diff-distrib nat-mult-distrib)
    ultimately show ?thesis
       using Tau-Reduction.tau-as-binomial-coefficient[OF 2] 3 assm unfolding
statement2-weak-def
      by (metis 0 bot-nat-0.extremum-strict nat-dvd-iff nat-eq-iff2
            nat-less-eq-zless zero-eq-power2)
  qed

  have tau-gM: τ (nat g) (nat (M)) = 0
   and tau-KB: τ (nat (2 * K g)) (nat ((B − 2) * B ^(n (ν+1)))) = 0
    using tau-S-T-decomp g-bound tau-ST by simp-all

  define C :: int where C ≡ B * B ^(n ν)

  interpret masking: masking-lemma δ ν (nat B) (nat b) (nat C)
  unfolding masking-lemma-def C-def n-def using δ-pos B-power2 b-power2 B-ge-2
b-le-B is-power2-ge1
  by (smt (verit, ccfv-SIG) B-even dvd-imp-le even-nat-iff mult-le-mono1 nat-eq-iff2

      nat-less-eq-zless nat-mult-distrib nat-power-eq order-le-less zero-less-nat-eq)

  have 1: nat g < nat C
```

unfolding *C-def* **using** *g-bound N0-def g-lt-N0 n-def* **by** *force*

obtain *y*::*nat*⇒*nat*
where *g-sum-y*: *nat g* = (∑ *i*=*1*..*ν*. *y i* ∗ (*nat* $\mathcal{B}$) ^(*n i*))
and *y-bound*: ∀ *i*. *y i* < *nat b*
using *masking.masking-lemma 1 tau-gM*
unfolding *n-masking-lemma*[*OF masking.masking-lemma-axioms*] *M-masking-lemma*[*OF masking.masking-lemma-axioms*]
by *blast*

define *z*::*nat*⇒*int* **where** *z* ≡ (λ*i*. *if i* = *0 then a else int* (*y i*))

have *g-sum*: *g* = (∑ *i*=*1*..*ν*. *z i* ∗ $\mathcal{B}$ ^(*n i*))
proof −
have *i*∈{*1*..*ν*} ⟹ *z i* ∗ $\mathcal{B}$ ^(*n i*) = *int* (*y i* ∗ (*nat* $\mathcal{B}$) ^(*n i*)) **for** *i*
using $\mathcal{B}$*-ge-2 z-def* **by** *auto*
hence (∑ *i*=*1*..*ν*. *z i* ∗ $\mathcal{B}$ ^(*n i*)) = (∑ *i*=*1*..*ν*. *int* (*y i* ∗ (*nat* $\mathcal{B}$) ^(*n i*)))
by *simp*
also have ... = *int* (∑ *i*=*1*..*ν*. *y i* ∗ (*nat* $\mathcal{B}$) ^(*n i*))
by *simp*
also have ... = *int* (*nat g*)
using *g-sum-y* **by** *simp*
also have ... = *g*
using *g-bound* **by** *auto*
finally show *?thesis*
by *auto*
qed

have *z-bound*: *0* ≤ *z i* ∧ *z i* < *b* **for** *i*
proof (*cases i* = *0*)
case *True* **thus** *?thesis*
unfolding *z-def b-def* **using** *a-nonneg f-pos*
by *simp* (*smt* (*verit*, *ccfv-SIG*) *mult-less-cancel-left2*)
next
case *False* **thus** *?thesis*
unfolding *z-def* **apply** *simp* **using** *y-bound zless-nat-eq-int-zless* **by** *blast*
qed

have *z-0*: *z 0* = *a*
unfolding *z-def* **by** *simp*

have *insertion z P* = *0*
using *lemma-1-8-helper*(*1*)[*OF g-sum*] *z-bound tau-KB z-0* **by** *auto*
thus *?thesis*
unfolding *statement1-weak-def* **using** *z-bound z-0* **by** *auto*
qed

lemma *coding-theorem-reverse′*:
assumes ∃ *g*. *0* ≤ *g* ∧ *g* < *2* ∗ (*int* γ) ∗ *b* ^α ∧ *Y dvd* (*2* ∗ *nat* (*X g*) *choose*

nat (*X g*))
shows $\exists z.\ (z\ 0 = a) \wedge (\forall i.\ 0 \leq z\ i \wedge z\ i < b) \wedge$ *insertion z P* = *0*
using *coding-theorem-reverse*[*unfolded statement2-weak-def statement1-weak-def*]
using *assms* **by** *auto*

end

end
theory *Lucas-Sequences*
imports *Main HOL.Parity*
begin

5 Lucas Sequences

fun ψ :: *int* $\Rightarrow$ *nat* $\Rightarrow$ *int* **where**
$\psi\ A\ 0 = 0 |$
$\psi\ A\ (Suc\ 0) = 1 |$
$\psi\ A\ (Suc\ (Suc\ n)) = A * (\psi\ A\ (Suc\ n)) - (\psi\ A\ n)$

fun χ :: *int* $\Rightarrow$ *nat* $\Rightarrow$ *int* **where**
$\chi\ A\ 0 = 2 |$
$\chi\ A\ (Suc\ 0) = A |$
$\chi\ A\ (Suc\ (Suc\ n)) = A * (\chi\ A\ (Suc\ n)) - (\chi\ A\ n)$

5.1 Elementary properties

theorem ψ-*induct* [*consumes 0*, *case-names 0 1 sucsuc*]:
$P\ 0 \Longrightarrow P\ 1 \Longrightarrow (\bigwedge n.\ P\ (n + 1) \Longrightarrow P\ n \Longrightarrow P\ (n + 2)) \Longrightarrow P\ (n{::}nat)$
apply (*induct rule*: ψ.*induct*)
by *simp-all*

theorem ψ-*induct-strict* [*consumes 0*, *case-names 0 1 2 sucsuc*]:
$P\ 0 \Longrightarrow P\ 1 \Longrightarrow P\ 2 \Longrightarrow (\bigwedge n.\ n{>}0 \Longrightarrow P\ (n + 1) \Longrightarrow P\ n \Longrightarrow P\ (n + 2))$
$\Longrightarrow\ P\ (n{::}nat)$
apply (*induct rule*: χ.*induct*)
apply *simp*
apply (*metis One-nat-def*)
by (*metis One-nat-def Suc-1 Suc-eq-plus1 add-2-eq-Suc' bot-nat-0.not-eq-extremum*)

lemma *lem0*: $n{\geq}2 \Longrightarrow \exists m.\ n = Suc\ (Suc\ m)$
by (*simp add*: *nat-le-iff-add*)

lemma ψ-*reverse*:
assumes $n{\geq}1$
shows $\psi\ A\ (n{-}1) = A * (\psi\ A\ n) - (\psi\ A\ (n{+}1))$
proof −
obtain *m* **where** *Suc* (*Suc m*) = *n*+*1*
by (*metis Suc-eq-plus1 add-eq-if assms not-one-le-zero*)
then show *?thesis* **by** *auto*

qed

Strict monotonicity

lemma *lucas-strict-monotonicity*: $A>1 \Longrightarrow \psi\ A\ (Suc\ n) > \psi\ A\ n \wedge \psi\ A\ (Suc\ n) > 0$
proof (*induction n*)
  case *0*
  then show *?case* **by** *auto*
next
  case (*Suc n*)
  from *Suc* **have** $A*\psi\ A\ (Suc\ n)>(\psi\ A\ n+\psi\ A\ (Suc\ n))$
    proof −
      from *Suc* **have** *b1*:$A*\psi\ A\ (Suc\ n)\geq 2*\psi\ A\ (Suc\ n)$
        by *simp*
      from *Suc* **have** *b2*:$2*\psi\ A\ (Suc\ n)>\psi\ A\ n+\psi\ A\ (Suc\ n)$
        by *simp*
      from *b1 b2* **show** *?thesis*
        using *less-le-trans* **by** *blast*
    qed
  from *this* **and** *Suc* **show** *?case*
    by *auto*
qed

lemma *lucas-monotone1*:
  fixes *A*
  assumes *A>1*
  shows $n\geq 2 \longrightarrow \psi\ A\ n \geq A$
proof (*induction n*)
  case *0*
  then show *?case* **by** *simp*
next
  case (*Suc n*)
  note *HR* = *this*
  consider (*0*) *n=0* | (*1*) *n=1* | (*2*)*n≥2* **by** *fastforce*
  then show *?case*
  proof *cases*
    case *0*
    then show *?thesis* **by** *simp*
  next
    case *1*
    then show *?thesis* **by** *simp*
  next
    case *2*
    then show *?thesis* **using** *HR lucas-strict-monotonicity*[*of A n*] *assms* **by** *simp*
  qed
qed

lemma *lucas-monotone2*:
  fixes *A n m*

assumes *A>1*
shows $\psi\ A\ n \leq \psi\ A\ (n+m)$
proof (*induction m*)
case *0*
then show *?case* **using** *assms* **by** *auto*
next
case (*Suc m*)
then show *?case* **using** *lucas-strict-monotonicity*[*of A n+m*] *assms* **by** *auto*
qed

lemma *lucas-monotone3*:
fixes *A n*
assumes $A > 1$
shows $\psi\ A\ n \geq int\ n$
proof (*induction n*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc n*)
then show *?case* **using** *lucas-strict-monotonicity*[*of A n*] *assms* **by** *auto*
qed

lemma *lucas-monotone4*:
fixes *A n m*
assumes *A>1* **and** $n \leq m$
shows $\psi\ A\ n \leq \psi\ A\ m$
proof −
obtain *k* **where** $m = n + k$ **using** *assms less-eqE* **by** *blast*
thus *?thesis* **using** *lucas-monotone2*[*of A n k*] *assms* **by** *auto*
qed

lemma *lucas-exp-growth-lt*:
fixes *A*::*int* **and** *n*::*nat*
assumes *A>1*
shows $\psi\ A\ (Suc\ (Suc\ (Suc\ n))) < A\ \hat{}\ (n+2)$
proof (*induction n*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc n*)
note $t = this$
have $\psi\ A\ m \geq 0$ **for** *m*
proof (*cases m*)
case *0*
then show *?thesis* **by** *auto*
next
case (*Suc nat*)

```
    then show ?thesis using assms lucas-strict-monotonicity[of A nat] by auto
  qed
  then have maj1 : ⋀m. m>0 ⟹ ψ A (Suc (Suc m)) ≤ A * ψ A (Suc m)
    by (simp add: lucas-strict-monotonicity)
  have triv : A^(Suc n +2) = A* A^(n+2) by auto
  have maj2 : ψ A (Suc (Suc (Suc (Suc n))))
    ≤ A * ψ A (Suc (Suc (Suc n)))
    apply (rule maj1[of Suc (Suc n)]) by auto
  have maj3 : A * ψ A (Suc (Suc (Suc n))) < A* A^(n+2)
    using t assms by auto
  have maj4 : ψ A (Suc (Suc (Suc (Suc n)))) < A* A^(n+2)
    using maj2 maj3 by auto
then show ?case using maj4 triv by auto
qed

lemma lucas-exp-growth-le:
  fixes A::int and n::nat
  assumes A>1
  shows ψ A (Suc (Suc n)) ≤ A^(n+1)
proof (induction n)
  case 0
  then show ?case by auto
next
  case (Suc n)
  note t = this
  have  ψ A m ≥ 0 for m
  proof (cases m)
    case 0
    then show ?thesis by auto
  next
    case (Suc nat)
    then show ?thesis using assms lucas-strict-monotonicity[of A nat] by auto
  qed
  then have maj1 : ⋀m. m>0 ⟹ ψ A (Suc (Suc m)) ≤ A * ψ A (Suc m)
    by (simp add: lucas-strict-monotonicity)
  have triv : A^(Suc n +2) = A* A^(n+2) by auto
  have maj2 : ψ A (Suc  (Suc (Suc n)))
    ≤ A * ψ A (Suc (Suc n))
    apply (rule maj1[of Suc n]) by auto
  have maj3 : A * ψ A  (Suc (Suc n)) ≤ A* A^(n+1)
    using t assms by auto
  have maj4 : ψ A (Suc (Suc (Suc n))) ≤ A* A^(n+1)
    using maj2 maj3 by auto
then show ?case using maj4 triv by auto
qed

lemma lucas-exp-growth-gt:
  fixes A::int and n::nat
  assumes A>1
```

shows $\psi\ A\ (Suc\ (Suc\ n)) > (A-1)\hat{}(n+1)$
proof (*induction n rule*: ψ-*induct*)
case *0*
then show *?case* **by** *auto*
next
case *1*
have $1 - 2*A < -1$ **using** *assms* **by** *auto*
then have *m1*: $A*A - 2*A + 1 < A*A - 1$ **by** *auto*
moreover have *m2*: $(A-1)\hat{}(1+1) = A*A - 2 * A + 1$
apply *simp*
by (*smt* (*verit*, *ccfv-SIG*) *mult-cancel-left2 square-diff-square-factored*)
moreover have $\psi\ A\ (Suc\ (Suc\ 1)) = A*A - 1$ **by** *auto*
then show *?case* **using** *m1 m2* **by** *auto*
next
case (*sucsuc n*)
note $t = this$
have *m1*: $\bigwedge m.\ \psi\ A\ (m+1) \geq\ \psi\ A\ m$
using *assms lucas-strict-monotonicity*[*of A m*]
by (*metis Suc-eq-plus1-left add.commute lucas-strict-monotonicity not-le-imp-less not-less-iff-gr-or-eq*)
have *m2*: $\psi\ A\ (Suc\ (Suc\ (n+1))) \geq\ \psi\ A\ (Suc\ (Suc\ n))$
using *m1*[*of Suc* (*Suc n*)] **by** *auto*
then have *m3*: $\psi\ A\ (Suc\ (Suc\ (n+1))) - \psi\ A\ (Suc\ (Suc\ (n)))\geq 0$
by *auto*
have *m4*: $\bigwedge m.\ m \geq 0 \Longrightarrow m + (A-1) * \psi\ A\ (Suc\ (Suc\ (n+1))) \geq (A-1) * \psi\ A\ (Suc\ (Suc\ (n+1)))$
by *auto*
have $A*\psi\ A\ (Suc\ (Suc\ (n+1))) - \psi\ A\ (Suc\ (Suc\ n)) \geq (A-1) * \psi\ A\ (Suc\ (Suc\ (n+1)))$
using *m4*[*of* $\psi\ A\ (Suc\ (Suc\ (n+1))) - \psi\ A\ (Suc\ (Suc\ (n)))$]
by (*smt* (*z3*) *left-diff-distrib' m3 mult-cancel-right1*)
then have *m5*: $\psi\ A\ (Suc\ (Suc\ (n+2))) \geq (A-1) * \psi\ A\ (Suc\ (Suc\ (n+1)))$
by *auto*
have *triv*: $\bigwedge m\ k.\ m \geq k \Longrightarrow (A-1)*m \geq (A-1)*k$
using *assms* **by** *auto*
have $(A-1) * \psi\ A\ (Suc\ (Suc\ (n+1))) \geq (A-1) * (A-1)\ \hat{}\ (n + 1 + 1)$
using *triv*[*of* $\psi\ A\ (Suc\ (Suc\ (n+1)))\ (A-1)\ \hat{}(n + 1 + 1)$] *t assms* **by** *auto*
then have *m6*: $\psi\ A\ (Suc\ (Suc\ (n+2))) \geq (A-1)*(A - 1)\ \hat{}\ (n + 1 + 1)$
using *m5* **by** *auto*
then have *m7*: $(A-1)*(A - 1)\ \hat{}\ (n + 1 + 1) = (A-1)\hat{}(n+2+1)$
by *auto*
then show *?case* **using** *m6 m7*
by (*smt* (*verit*, *best*) *assms m5 mult-strict-left-mono t*(*1*))
qed

lemma *lucas-symmetry-A*:
fixes *A*::*int* **and** *n*::*nat*
assumes $A \geq 2$

```
  shows (ψ A n) = (if (odd n) then ψ (−A) n else − ψ (−A) n)
proof −
  consider (0) n=0 | (1) n=1 | (suc) n≥2 by fastforce
  then show ?thesis
  proof cases
    case 0
    then show ?thesis by simp
  next
    case 1
    then show ?thesis by simp
  next
    case suc
    then show ?thesis
    proof (induction n rule: full-nat-induct)
      case (1 n)
      then show ?case
        using assms lem0[of n] 1.prems
        using le-Suc-eq by auto
      qed
  qed
qed

lemma lucas-symmetry-A2: −ψ A n = (−1::int)^n * ψ (−A) n
proof(induction n rule: ψ-induct)
  case 0
  then show ?case by simp
next
  case 1
  then show ?case by simp
next
  case (sucsuc n)
  then show ?case
    by (auto simp add: algebra-simps)
qed

lemma lucas-symmetry-A-abs: assumes abs A > 1 shows abs (ψ A n) = ψ (abs
A) n
proof (cases n=0)
  case True
  then show ?thesis by simp
next
  case False
 then show ?thesis using lucas-strict-monotonicity[of abs A n−1] lucas-symmetry-A2[of
A n] assms
    by (smt (z3) One-nat-def Suc-pred lucas-symmetry-A not-gr-zero)
qed


lemma lucas-A-eq-2:
```

```
  fixes n::nat
  shows (ψ 2 n) = (int n)
proof −
  consider (0) n=0 | (1) n=1 | (suc) n≥2 by fastforce
  then show ?thesis
  proof cases
    case 0
      then show ?thesis by simp
    next
      case 1
      then show ?thesis by simp
    next
      case suc
      then show ?thesis
      proof (induction n rule: full-nat-induct)
        case (1 n)
        then show ?case
          using lem0[of n]
            using 1.prems ψ.simps(2) eq-numeral-Suc le-Suc-eq le-simps(1) lessI
numeral-Bit0
                of-nat-0
          by fastforce
      qed
   qed
 qed


lemma lucas-periodic-modN:
  fixes N::int
  assumes N > 0
  shows ∃ T≥1. ∀ n. (ψ A (T + n)) mod N = (ψ A n) mod N
proof −
  define f where f ≡ (λr. ((ψ A r) mod N, (ψ A (r+1)) mod N))
  then have 0: f ‘ {1..(nat(N^2)+1)} ⊆ ({0..(N−1)} × {0..(N−1)})
    using pos-mod-bound pos-mod-sign assms
    by auto
  have finite ({0..(N−1)} × {0..(N−1)})
    by auto
 then have 1: card(f ‘ {1..(nat(N^2)+1)}) ≤ card (({0..(N−1)} × {0..(N−1)}))
    using 0 Finite-Set.card-mono
    by blast
  have card ({0..(N−1)} × {0..(N−1)}) = nat(N^2)
    apply simp
    by (smt (verit, del-insts) assms nat-power-eq power2-eq-square)
  then have 2: card(f ‘ {1..(nat(N^2)+1)}) < card({1..(nat(N^2)+1)})
    using 0 1
    by auto
  then have ¬ inj-on f {1..(nat(N^2)+1)}
    using Finite-Set.pigeonhole
```

```
    by blast
  then have 3: ∃ r∈{1..(nat(N^2)+1)}. ∃ s∈{1..(nat(N^2)+1)}. (f r = f s) ∧ r
≠ s
    using inj-on-def
    by blast
  then obtain r s where def-rs:(f r = f s) ∧ r ≠ s by auto
  define b a where def-b: b ≡ max r s and def-a: a ≡ min r s
  then have prop-ab:f a = f b ∧ a < b
    using def-rs
    by (simp add: max-def min-def)
  have prop-rec-desc: ∀ k. k≤a ⟶ f (b−k) = f(a−k)
  proof
    fix k
    show k≤a ⟶ f (b−k) = f(a−k)
    proof (induction k)
      case 0
      then show ?case using prop-ab by simp
    next
      case (Suc k)
      then show ?case

      proof (cases k=a)
        case True
        then show ?thesis by auto
      next
        case False
        assume HR: k ≤ a ⟶ f (b − k) = f (a − k)
        assume k≠a
        have Suc k ≤ a ⟹ f (b − Suc k) = f (a − Suc k)
        proof −
          assume k-small-a: (Suc k) ≤ a

          then have k-small: a−k ≥ 1 b−k ≥ 1 using prop-ab by auto
          have k≤a using k-small-a by simp

            then have eg1: (ψ A (a−k)) mod N = (ψ A (b−k)) mod N ∧ ψ A
(a−k+1) mod N = (ψ A (b−k+1)) mod N
            using HR f-def by force
            have eg2: ψ A (a−k−1) = A ∗ ψ A (a−k) − ψ A (a−k+1) ∧ ψ A
(b−k−1) = A ∗ ψ A (b−k) − ψ A (b−k+1)
            using k-small ψ-reverse[of a−k A] ψ-reverse[of b−k A]
            by auto
          then have (ψ A (a−k−1)) mod N = (A ∗ ψ A (a−k) − ψ A (a−k+1))
mod N
            by presburger
          also have ... = (A ∗ ψ A (b−k) − ψ A (b−k+1)) mod N
            using eg1 by (metis mod-diff-cong mod-mult-cong)
          finally have eg3: (ψ A (a−k−1)) mod N = (ψ A (b−k−1)) mod N
            using eg2 by presburger
```

```
        then show  f (b − Suc k) = f (a − Suc k) using eg1 k-small
         by (smt (z3) diff-Suc-eq-diff-pred diff-commute f-def le-add-diff-inverse2)
       qed
       then show ?thesis by (rule impI)
     qed
   qed
 qed
 define T where def-T: T ≡ b−a
 then have T1: T≥1 using prop-ab
   by (simp add: Suc-leI)
 have f T = f 0 using prop-rec-desc def-T by auto
 then have 0: (ψ A T) mod N = (ψ A 0) mod N ∧ (ψ A (T+1)) mod N = (ψ
A 1) mod N
   using f-def by simp
 have ∀k. (ψ A (T+k)) mod N = (ψ A k) mod N
 proof
   fix k
   show (ψ A (T+k)) mod N = (ψ A k) mod N using 0
   proof (induction k rule:ψ-induct)
   case 0
     then show ?case by simp
   next
     case 1
     then show ?case by simp
   next
     case (sucsuc n)
     then have n:ψ A (T+n) mod N = ψ A (n) mod N
       using sucsuc.IH(1) sucsuc.prems by simp
     then have n1:ψ A (T+(Suc n)) mod N = ψ A (Suc n) mod N
       using sucsuc.IH(1) sucsuc.prems by simp
     have ψ A (T+(Suc (Suc n))) mod N = (A ∗ ψ A (Suc (T + n)) − ψ A (T
+ n)) mod N
       by simp
      also have ... = (A ∗ ψ A (Suc (T + n)) mod N − ψ A (T + n) mod N)
mod N
       by (metis mod-diff-eq)
     also have ... = (A ∗ ψ A (Suc (n)) mod N − ψ A (n) mod N) mod N
       using n n1 by (metis add-Suc-right mod-mult-cong)
     also have ... = (A ∗ ψ A (Suc (n)) − ψ A (n)) mod N
       by (metis mod-diff-eq)
     finally have ψ A (T+(Suc (Suc n))) mod N = ψ A ((Suc (Suc n))) mod N
by simp
     then show ?case using add-2-eq-Suc′ by presburger
   qed
 qed
 then show ?thesis using T1 by auto
qed
```

lemma *lucas-modN*:
fixes N::*int*
assumes $N>0$
shows $\forall n.\ \exists k{\geq}n.\ \psi\ A\ k\ mod\ N = 0$
proof
fix n
have $\exists T{\geq}1.\ \forall n.\ (\psi\ A\ (T + n))\ mod\ N = (\psi\ A\ n)\ mod\ N$
using *lucas-periodic-modN assms*
by *blast*
then obtain T **where** *def-T*: $\forall n.\ (\psi\ A\ (T{+}n))\ mod\ N = (\psi\ A\ n)\ mod\ N \wedge T{\geq}1$
by *auto*
have *0*: $\forall k.\ (\psi\ A\ (T * k))\ mod\ N = 0$
proof
fix k
show $(\psi\ A\ (T * k))\ mod\ N = 0$
proof (*induction k*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc k*)
then show *?case* **using** *def-T* **by** *simp*
qed
qed
define K **where** *def-K* : $K \equiv T * n$
then have *1*: $K{\geq}n$ **using** *def-T* **by** *auto*
have $(\psi\ A\ K)\ mod\ N = 0$ **using** *0 def-K* **by** *auto*
then show $\exists k{\geq}n.\ \psi\ A\ k\ mod\ N = 0$ **using** *1* **by** *auto*
qed

lemma *lucas-parity*:
fixes A::*int* **and** B::*nat*
assumes *even A*
shows *even* $(\psi\ A\ B)$ = *even B*
proof(*induction B rule*: *ψ-induct*)
case *0*
then show *?case*
by *simp*
next
case *1*
then show *?case*
by *simp*
next
case (*sucsuc n*)
then show *?case*
by (*auto simp add*: *assms*)
qed

corollary *lucas-parity2*:
fixes *A*::*int* **and** *B*::*nat*
assumes *even A*
shows *even* (ψ *A B* − *int B*)
by (*simp add*: *assms lucas-parity*)

lemma *lucas-monotone-A*:
assumes *1*<*A A*≤*A'*
shows ψ *A n* ≤ ψ *A' n*
proof (*induction n rule*:ψ*-induct*)
case *0*
then show *?case* **by** *simp*
next
case *1*
then show *?case* **by** *simp*
next
case (*sucsuc n*)
note *HR* = *this*
consider (*equal*) *A*=*A'* | (*strict*) *A*<*A'* **using** *assms* **by** *fastforce*
then show *?case*
proof *cases*
case *equal*
then show *?thesis* **by** *simp*
next
case *strict*
have ψ *A n* ≥ *0*
proof (*cases n*=*0*)
case *True*
then show *?thesis* **by** *simp*
next
case *False*
then show *?thesis* **using** *lucas-strict-monotonicity*[*of A n*−*1*] *assms* **by** *auto*
qed
then have ψ *A* (*n* + *2*) = *A* ∗ ψ *A* (*n*+*1*) − ψ *A n* **by** *simp*
also have ... ≤ *A* ∗ ψ *A* (*n*+*1*) **using** ‹ψ *A n* ≥ *0*› **by** *simp*
also have ... ≤ (*A'* − *1*) ∗ ψ *A* (*n*+*1*) **using** ‹*A*<*A'*› *assms*(*1*) *lucas-strict-monotonicity* **by** *auto*
also have ... ≤ (*A'* − *1*) ∗ ψ *A'* (*n*+*1*) **using** *assms HR* **by** *simp*
also have ... ≤ *A'* ∗ ψ *A'* (*n*+*1*) − ψ *A' n* **using** *lucas-strict-monotonicity*[*of A' n*] *assms Suc-eq-plus1 int-distrib*(*3*) **by** *fastforce*
finally have ψ *A* (*n* + *2*) ≤ ψ *A'* (*n*+*2*) **by** *simp*
then show *?thesis* **by** *simp*
qed
qed

lemma *lucas-congruence*:

fixes A::*int* **and** B::*int* **and** n::*int*
assumes $n{=}n \wedge A \ mod\ n = B \ mod\ n$
shows $(\psi\ A\ m) \ mod\ n = (\psi\ B\ m) \ mod\ n$
proof (*induction m rule*: ψ-*induct*)
case *0*
then show *?case* **by** *auto*
next
case *1*
then show *?case* **by** *auto*
next
case (*sucsuc m*)
note $t = this$
have *e1*: $(A * \psi\ A\ (m + 1)) \ mod\ n = (B * \psi\ B\ (m + 1)) \ mod\ n$
using *assms mod-mult-cong sucsuc.IH*(*1*) **by** *blast*
then have *e2*: $\bigwedge k\ l.\ k \ mod\ n = l \ mod\ n \Longrightarrow (A * \psi\ A\ (m + 1) - k) \ mod\ n = (B * \psi\ B\ (m + 1) - l) \ mod\ n$
using *mod-diff-cong* **by** *blast*
have *e3*: $(A * \psi\ A\ (m + 1) - \psi\ A\ m) \ mod\ n = (B * \psi\ B\ (m + 1) - \psi\ B\ m) \ mod\ n$
using t *e2*[*of* $\psi\ A\ m\ \psi\ B\ m$] **by** *blast*
then show *?case* **by** *auto*
qed

corollary *lucas-congruence2*:
fixes α::*int* **and** m::*nat*
shows $\psi\ \alpha\ m \ mod\ (\alpha - 2) = int\ m \ mod\ (\alpha - 2)$
proof −
have *fac1*: $\psi\ 2\ a = int\ a$ **for** a
proof (*induction a rule*: ψ-*induct*)
case *0*
then show *?case* **by** *auto*
next
case *1*
then show *?case* **by** *auto*
next
case (*sucsuc n*)
then show *?case* **by** *auto*
qed
have $\alpha \ mod\ (\alpha - 2) = 2 \ mod\ (\alpha - 2)$
by (*smt* (*z3*) *minus-mod-self2*)
then have *fac2*: $\psi\ \alpha\ m \ mod\ (\alpha - 2) = \psi\ 2\ m \ mod\ (\alpha - 2)$
using *lucas-congruence*[*of* $\alpha - 2\ \alpha\ 2\ m$] **by** *auto*
then show *?thesis* **using** *fac2 fac1* **by** *auto*
qed

end
theory *Pell-Equation*
imports *Lucas-Sequences Complex-Main ../Coding/Utils*

begin

5.2 The Pell Equation

5.2.1 Auxiliary facts

named-theorems *real-of-int*

lemma *floor-of-real-of-int*[*real-of-int*]: $\lfloor$*real-of-int* $x\rfloor = x$
by *auto*

lemma *floor-of-real-of-int-sub2*[*real-of-int*]: $\lfloor x -$ *real-of-int* $y\rfloor = \lfloor x\rfloor - y$
by *auto*

lemma *floor-of-real-of-int-mult*[*real-of-int*]: $\lfloor$*real-of-int* $x *$ *real-of-int* $y\rfloor = x * y$
by (*metis floor-of-int of-int-mult*)

lemma *real-of-int-inequality*: $X \leq Y \longleftrightarrow$ *real-of-int* $X \leq$ *real-of-int* Y **by** *auto*
lemma *real-of-int-strict-inequality*: $X < Y \longleftrightarrow$ *real-of-int* $X <$ *real-of-int* Y **by** *auto*

lemma *evenX2k*:
fixes X::*int*
assumes *evenX*:*even* X
shows $\exists\ k.\ X = 2*k$
proof −
obtain k **where** $X=2*k$ **using** *evenX* **by** *auto*
then have $X=2*k$ **by** *auto*
thus *?thesis* **by** (*rule exI*)
qed

lemma *distrib-add-diff*:
fixes $a\ b\ c\ d$::*real*
shows $(a+b)*(c-d) = a*c - a*d + b*c - b*d$
by (*simp add*: *algebra-simps*)

lemma *floor-even*:
fixes X::*int*
assumes *Xeven*: *even* X
shows *real-of-int* $\lfloor$(*real-of-int* X)/2$\rfloor$ = (*real-of-int* X)/2
proof −
obtain k **where** $X = 2*k$ **using** *Xeven* **by** *auto*
thus *?thesis* **by** *auto*
qed

lemma *even-to-mod2*:
fixes $X\ Y$::*int*
assumes *even* X = *even* Y
shows X *mod* 2 = Y *mod* 2
proof (*cases even* X)

```
  case True
  have 0:X mod 2 = 0 using ‹even X› by auto
  have even Y using assms ‹even X› by auto
  then have Y mod 2 = 0 by auto
  thus X mod 2 = Y mod 2 using 0 by auto
next
  case False
  have 1:X mod 2 = 1 using odd-iff-mod-2-eq-one ‹odd X› by auto
  have odd Y using assms ‹odd X› by auto
  then have Y mod 2 = 1 using odd-iff-mod-2-eq-one by auto
  thus X mod 2 = Y mod 2 using 1 by auto
qed

lemma oddA-to-mod:
  fixes X Y A::int
  assumes odd A
  shows A^2 mod 4 = 1
proof −
    have even (A−1) using assms by auto
    then obtain k where (A−1)=2∗k using evenX2k[of A−1] by auto
    then have A = 2∗k+1 by auto
    then have 1:A^2 = 4∗k^2 + 4∗k +1  using power2-sum[of 2∗k 1] by auto
    have (4∗k^2 + 4∗k) mod 4 = 0 by auto
    then have (4∗k^2 + 4∗k +1) mod 4 = 1 by fastforce
    thus A^2 mod 4 = 1 using 1 by auto
  qed

lemma sol-non-zero:
  fixes X Y A::int
  assumes sol: X^2 − (A^2−4)∗Y^2 = 4 and Alarge: A^2>4
  shows X+sqrt(A^2−4)∗Y ≠ 0
proof(rule ccontr)
    assume 0: ¬ X + sqrt(A^2−4)∗Y ≠ 0
   have A4:(real-of-int A)^2 >4 using Alarge real-of-int-strict-inequality by auto
    have 0=(X+sqrt(A^2−4)∗Y)∗(X−sqrt(A^2−4)∗Y) using 0 by auto
    also have ... = (X^2 − (sqrt(A^2−4)∗Y)^2) unfolding power-def
    using square-diff-square-factored[of X sqrt(A^2−4)∗Y] by (simp add: algebra-simps)
    also have ... = (X^2 − (A^2−4)∗Y^2) using A4 unfolding power-def by
auto
    also have ... = 4 using sol by auto
    finally show False by auto
  qed

lemma conj-inversion:
fixes X::int and Y::int and A::int
assumes A4:A^2 > 4 and sol:X^2 − (A^2 − 4)∗Y^2 = 4
shows 1/2∗(X−sqrt(A^2−4)∗Y) = 2∗inverse(X+sqrt(A^2−4)∗Y)
proof −
  define E where E ≡ sqrt(A^2−4)
```

have *not0*: $X+sqrt(A\hat{}2-4)*Y\neq 0$ **using** *A4 sol sol-non-zero* **by** *auto*
have $4 \leq A\hat{}2$ **using** *A4* **by** *auto*
then have $4 \leq$ *real-of-int* $(A\hat{}2)$
using *real-of-int-inequality*[*of* 4 $A\hat{}2$] **by** *auto*
then have *0*:$4 \leq (real\text{-}of\text{-}int\ A)^2$ **by** *auto*
have *E2*:$E\hat{}2 = A\hat{}2 - 4$ **unfolding** *E-def* **using** *0* **by** *auto*
have *Enot0*:$X+E*Y \neq 0$ **using** *not0* **unfolding** *E-def* **by** *auto*
have $(X-E*Y)*(X+E*Y) = X\hat{}2 - (E*Y)\hat{}2$
by (*simp add*: *power2-eq-square square-diff-square-factored*)
also have ... $=X\hat{}2 - E\hat{}2*Y\hat{}2$ **by** (*simp add*: *algebra-simps*)
also have ... $= X\hat{}2 - (A\hat{}2 - 4)*Y\hat{}2$ **using** *E2* **by** *auto*
also have ... $= 4$ **using** *sol* **by** (*simp add*: *algebra-simps*)
finally have $(X-E*Y)*(X+E*Y) = 4$ **by** *auto*
then have $1/2*((X-E*Y)*(X+E*Y))*(inverse\ (X+E*Y))$
$= 2*(inverse\ (X+E*Y))$ **by** (*simp add*: *algebra-simps*)
then have $1/2*(X-E*Y)*((X+E*Y)*(inverse\ (X+E*Y))) = 2*(inverse\ (X+E*Y))$
by (*simp add*: *algebra-simps*)
then have $1/2*(X-E*Y) = 2*(inverse\ (X+E*Y))$
using *right-inverse*[*of* $X+E*Y$] **using** *Enot0* **by** *auto*
thus *?thesis* **unfolding** *E-def* **by** *auto*
qed

5.2.2 Group structure of the solutions

lemma *group-structure*:
fixes *X1 X2 Y1 Y2 A*::*int*
assumes *A4*:$A\hat{}2 > 4$
shows $(X1\hat{}2 - (A\hat{}2-4)*Y1\hat{}2 = 4) \wedge (X2\hat{}2 - (A\hat{}2-4)*Y2\hat{}2 = 4)$
$\Longrightarrow (X1*X2 + (A\hat{}2-4)*Y1*Y2)\hat{}2 - (A\hat{}2-4)*(X1*Y2 + X2*Y1)\hat{}2$
$= 16$
proof −
assume *asm*: $X1\hat{}2 - (A\hat{}2-4)*Y1\hat{}2 = 4 \wedge\ X2\hat{}2 - (A\hat{}2-4)*Y2\hat{}2 = 4$
define $X1'$ **where** $X1' \equiv real\text{-}of\text{-}int\ X1$
define $X2'$ **where** $X2' \equiv real\text{-}of\text{-}int\ X2$
define $Y1'$ **where** $Y1' \equiv real\text{-}of\text{-}int\ Y1$
define $Y2'$ **where** $Y2' \equiv real\text{-}of\text{-}int\ Y2$
define A' **where** $A' \equiv real\text{-}of\text{-}int\ A$
define D'::*real* **where** $D'=A'\hat{}2 - 4$
define D **where** $D=A\hat{}2 - 4$
define $X3'$::*real* **where** $X3' \equiv X1'*X2' + D'*Y1'*Y2'$
define $Y3'$::*real* **where** $Y3' \equiv X1'*Y2' + X2'*Y1'$
define *X3* **where** $X3 \equiv X1*X2 + D*Y1*Y2$
define *Y3* **where** $Y3 \equiv X1*Y2 + X2*Y1$

have *int1*:$X1'\hat{}2-D'*Y1'\hat{}2=real\text{-}of\text{-}int\ (X1\hat{}2-D*Y1\hat{}2)$
unfolding *power-def X1'-def Y1'-def D'-def D-def A'-def* **by** *auto*
have *int2*:$X2'\hat{}2-D'*Y2'\hat{}2=real\text{-}of\text{-}int\ (X2\hat{}2-D*Y2\hat{}2)$
unfolding *power-def D'-def D-def A'-def X2'-def Y2'-def* **by** *auto*
have *int3*:$X3'\hat{}2-D'*Y3'\hat{}2=real\text{-}of\text{-}int\ (X3\hat{}2-D*Y3\hat{}2)$

unfolding *power-def D'-def D-def A'-def X3'-def X3-def Y3'-def Y3-def X1'-def X2'-def Y1'-def Y2'-def* **by** *auto*

have *A'^2>4* **unfolding** *A'-def* **using** *A4 real-of-int-strict-inequality* **by** *auto*
then have *d*:$|D'| = D'$ **unfolding** *D'-def* **by** *auto*
then have *0*:$(X1' + (sqrt\ D') * Y1')*(X2' + (sqrt\ D') * Y2') = X3' + (sqrt\ D')*Y3'$
unfolding *X3'-def Y3'-def* **by** (*simp add*: *algebra-simps*)
have *1*:$(X1' - (sqrt\ D') * Y1')*(X2' - (sqrt\ D') * Y2') = X3' - (sqrt\ D')*Y3'$
unfolding *X3'-def Y3'-def* **using** *d* **by** (*simp add*: *algebra-simps*)

have $X3'^2 - D'*Y3'^2 = (X3' + (sqrt\ D')*Y3')*(X3' - (sqrt\ D')*Y3')$
unfolding *power-def* **by** (*simp add*: *algebra-simps d*)
also have ... $= (X1' + (sqrt\ D') * Y1')*(X2' + (sqrt\ D') * Y2')$
$*(X1' - (sqrt\ D') * Y1')*(X2' - (sqrt\ D') * Y2')$ **using** *0 1* **by** *auto*
also have ... $= (X1'^2 - D'*Y1'^2)* (X2'^2 - D'*Y2'^2)$
unfolding *power-def* **by** (*simp add*: *algebra-simps d*)
finally have $X3'^2 - D'*Y3'^2 = (X1'^2 - D'*Y1'^2)* (X2'^2 - D'*Y2'^2)$
by *auto*
then have $X3^2 - D*Y3^2 = \lfloor(X1'^2 - D'*Y1'^2)*(X2'^2 - D'*Y2'^2)\rfloor$
using *floor-of-real-of-int*[*of* $X3^2 - D * Y3^2$] *int1 int2 int3* **by** *auto*
then have $X3^2 - D*Y3^2 = (X1^2 - D*Y1^2)* (X2^2 - D*Y2^2)$
using *int1 int2 int3 floor-of-real-of-int-mult*[*of* $X1^2 - D*Y1^2$ $X2^2 - D*Y2^2$] **by** *auto*
then have $X3^2 - D*Y3^2 = (X1^2 - D*Y1^2)* (X2^2 - D*Y2^2)$ **by** *auto*
then have $X3^2 - D*Y3^2 = 16$ **using** *asm D-def* **by** *auto*
thus *?thesis* **unfolding** *X3-def Y3-def D-def* **by** (*simp add*: *algebra-simps*)
qed

lemma *group-structure-evenXi*:
fixes *X Y A*::*int*
assumes *sol*:$(X^2 - (A^2-4)*Y^2 = 4)$ **and** *even A*
shows *even X*
proof −
obtain *k* **where** $A=2*k$ **using** ‹*even A*› **by** *auto*
then have $A^2-4 = 4*(k^2-1)$ **by** *auto*
then have *0*: $((A^2-4)*Y^2 - 4)\ mod\ 4 = 0$
using *dvd-eq-mod-eq-0* **by** *auto*

have $X^2 =((A^2-4)*Y^2 + 4)$ **using** *sol* **by** *auto*
then have $(X^2)\ mod\ 4 =((A^2-4)*Y^2 + 4)\ mod\ 4$ **by** *auto*
then have $(X^2)\ mod\ 4 = 0$ **using** *0* **by** *auto*
thus *even X* **using** *dvd-eq-mod-eq-0*[*of 2* (X^2)] **by** *auto*
qed

lemma *XimodYi*:
fixes *X Y A*::*int*
assumes *A4*:$A^2 > 4$ **and** *sol*:$(X^2 - (A^2-4)*Y^2 = 4)$ **and** *odd A*

```
  shows X mod 2 = Y mod 2
proof −
  have 0:A^2 mod 4 = 1 using oddA-to-mod A4 sol ‹odd A› by auto
  have X^2 = A^2*Y^2− 4*Y^2 +4 using sol by (simp add: algebra-simps)
  then have X^2 mod 4 = (A^2*Y^2 − 4*Y^2) mod 4 by auto
  then have 1:X^2 mod 4 = A^2*Y^2 mod 4
    using mod-diff-right-eq[of A^2*Y^2 4*Y^2 4] by (simp add: mod-simps)

  have A^2*Y^2 = (4*(A^2 div 4) + (A^2 mod 4))*Y^2
    using mult-div-mod-eq[of A^2 4] by auto
  then have A^2*Y^2 = 4*(A^2 div 4)*Y^2 + (A^2 mod 4)*Y^2
     using distrib-right [of 4*(A^2 div 4) (A^2 mod 4) Y^2] by auto
  then have A^2*Y^2 mod 4 = (A^2 mod 4)*Y^2 mod 4
    using mod-mult-self4 by (simp add: algebra-simps)
  then have X^2 mod 4 = Y^2 mod 4 using 0 1 by auto
  then have 2:4 dvd (X^2 − Y^2) using mod-eq-dvd-iff by auto

  have (X+Y)*(X−Y) = X^2 −Y^2 unfolding power-def
    using square-diff-square-factored by (simp add: algebra-simps)
  then have 3:4 dvd (X+Y)*(X−Y) using 2 by auto
  then have 2 dvd (X+Y)*(X−Y)
    using dvd-trans[of 2 4 (X+Y)*(X−Y)] by auto
  then have even X=even Y by auto
  thus X mod 2 = Y mod 2 using even-to-mod2[of X Y] by auto
qed


lemma group-structure-int:
  fixes X1 X2 Y1 Y2 A::int
  assumes A4:A^2 > 4 and sol1:(X1^2 − (A^2−4)*Y1^2 = 4)
    and sol2:(X2^2 − (A^2−4)*Y2^2 = 4)
  shows even(X1*X2 + (A^2−4)*Y1*Y2) ∧ even(X1*Y2 + X2*Y1)
proof (cases even A)
  case True
  have evenX1X2:even (X1*X2) using group-structure-evenXi[of X1 A Y1] sol1
‹even A› by auto
  have even ((A^2−4)*Y1*Y2) using ‹even A› by auto
  then have con1:even(X1*X2 + (A^2−4)*Y1*Y2) using evenX1X2 by auto

  have evenX1Y2:even (X1*Y2) using group-structure-evenXi[of X1 A Y1] sol1
‹even A› by auto
   have even (X2*Y1) using group-structure-evenXi[of X2 A Y2] sol2 ‹even A›
by auto
  then have even(X1*Y2 + X2*Y1) using evenX1Y2 by auto
  thus even(X1*X2 + (A^2−4)*Y1*Y2) ∧ even(X1*Y2 + X2*Y1) using con1
by auto
next
  case False
  have 1:X1 mod 2 = Y1 mod 2 using XimodYi A4 sol1 ‹odd A› by auto
```

have *2*:*X2 mod 2 = Y2 mod 2* **using** *XimodYi A4 sol2* ‹*odd A*› **by** *auto*
have *A^2 mod 4 = 1* **using** *oddA-to-mod A4 sol1* ‹*odd A*› **by** *auto*
then have *mod4*:*(A^2−4) mod 4 = 1* **by** *auto*

have *0*:*(A^2−4) mod 2 = 1*
proof −
have *(A^2 − 4) mod 2 = (2∗2∗((A^2 − 4) div 4) + 1) mod 2*
using *mult-div-mod-eq*[*of 4 A^2−4*] *mod4*
arg-cong[*of A^2 − 4 2∗2∗((A^2 − 4) div 4) + 1*] **by** (*simp add*: *mod-simps*)
also have *... = ((2∗2∗((A^2 − 4) div 4) mod 2) + 1) mod 2*
using *mod-add-left-eq*[*of 2∗2∗((A^2 − 4) div 4) 2 1*] **by** (*simp add*: *mod-simps*)
also have *... = 1* **by** *auto*
finally show *(A^2 − 4) mod 2 = 1* **by** *auto*
qed

have *((A^2−4)∗Y1∗Y2) mod 2 = ((A^2−4) mod 2)∗Y1∗Y2 mod 2*
using *mod-mult-left-eq*[*of A^2 −4 2 Y1∗Y2*] **by** (*simp add*: *algebra-simps*)
also have *... = Y1∗Y2 mod 2* **using** *0* **by** *auto*
also have *... = ((Y1 mod 2)∗(Y2 mod 2)) mod 2* **by** (*simp add*: *mod-simps*)
also have *... = ((X1 mod 2)∗(X2 mod 2)) mod 2* **using** *1 2* **by** *auto*
also have *... = X1∗X2 mod 2* **by** (*simp add*: *mod-simps*)
finally have *3*:*((A^2−4)∗Y1∗Y2) mod 2 = X1∗X2 mod 2* **by** *auto*

have *(X1∗X2 −(A^2−4)∗Y1∗Y2) mod 2 = (X1∗X2 mod 2 −((A^2−4)∗Y1∗Y2 mod 2)) mod 2*
by (*simp add*: *mod-simps*)
also have *... = (X1∗X2 mod 2 − (X1∗X2 mod 2)) mod 2*
using *3* **by** (*simp add*: *mod-simps*)
also have *... = 0* **by** *auto*
finally have *(X1∗X2 −(A^2−4)∗Y1∗Y2) mod 2 = 0* **by** *auto*
then have *con1*:*even*(*X1∗X2 + (A^2−4)∗Y1∗Y2*)
using *dvd-eq-mod-eq-0*[*of 2 X1∗X2 − (A^2−4)∗Y1∗Y2*] **by** *auto*

have *(X1∗Y2 + X2∗Y1) mod 2 = (X1∗Y2 mod 2 + (X2∗Y1 mod 2)) mod 2*
using *mod-add-left-eq*[*of X2∗Y1 2 X1∗Y2*]
mod-add-right-eq[*of (X2∗Y1 mod 2) X1∗Y2 2*] **by** (*simp add*: *algebra-simps*)
also have *... = (X1∗(Y2 mod 2) mod 2 + X2∗(Y1 mod 2) mod 2) mod 2*
using *mod-mult-left-eq*[*of Y1 2 X2*]
mod-mult-left-eq[*of Y2 2 X1*] **by** (*simp add*: *algebra-simps*)
also have *... = ((X1 mod 2)∗(Y2 mod 2) mod 2 + (X2 mod 2)∗(Y1 mod 2) mod 2) mod 2*
using *mod-mult-right-eq*[*of Y2 mod 2 X1 2*]
mod-mult-right-eq[*of Y1 mod 2 X2 2*] **by** (*simp add*: *algebra-simps*)
also have *... = ((X1 mod 2)∗(X2 mod 2) mod 2 + (X2 mod 2)∗(X1 mod 2) mod 2) mod 2*
using *1 2* **by** *auto*
also have *... = 0* **by** (*simp add*: *mod-simps*)

```
  finally have (X1*Y2 + X2*Y1) mod 2=0 by auto
  then have even (X1*Y2 + X2*Y1)
    using dvd-eq-mod-eq-0[of 2  (X1*Y2 -X2*Y1)] by auto
  thus even(X1*X2 + (A^2-4)*Y1*Y2) ∧ even(X1*Y2 + X2*Y1) using con1
by auto
qed

lemma group-structure-sol4:
  fixes X1 X2 Y1 Y2 A::int
  assumes Alarge:A^2>4 and sol1:(X1^2 - (A^2-4)*Y1^2 = 4)
    and sol2:(X2^2 - (A^2-4)*Y2^2 = 4)
  defines X3 ≡ X1*X2 + (A^2-4)*Y1*Y2 and Y3 ≡ X1*Y2 + X2*Y1
  shows (floor (X3/2)) ^2 - (A^2-4)*(floor(Y3/2)) ^2 = 4
proof -
   have evX3: even X3 using group-structure-int Alarge sol1 sol2  unfolding
X3-def by auto
  have evY3: even Y3 using group-structure-int [of A X1 Y1 X2 Y2] Alarge sol1
sol2
    unfolding Y3-def by auto
  have floorY3: (A^2-4)*(floor (Y3/2)) ^2 = (A^2-4)*(Y3/2) ^2
    using evY3 floor-even[of Y3] by auto
  have X3^2 - (A^2-4)*Y3^2 = 16 unfolding X3-def Y3-def
    using group-structure Alarge sol1 sol2  by auto
  then have 4 = (X3^2 - (A^2-4)*Y3^2)/4 by auto
   then have 4 = (X3/2) ^2  - (A^2-4)*(Y3/2) ^2 unfolding power-def by
auto
  then have 4 = floor ((X3/2) ^2  - real-of-int((A^2-4)*(floor(Y3/2)) ^2))
    using evY3 floor-even[of Y3] by auto
  then show ?thesis  using floor-of-real-of-int-mult[of floor(real-of-int X3/2)]
   floor-of-real-of-int-sub2[of (X3/2) ^2 (A^2-4)*(floor(Y3/2)) ^2] evX3 floor-even[of
X3]
    unfolding power-def by auto
qed
```

5.2.3 Smallest solution

```
lemma smallest-sol-sublemma:
  fixes X Y A::int
  assumes Alarge: A^2>4 and XYsol: X^2 -(A^2-4)*Y^2 = 4
    and X≥0 and Y>0
  shows X + Y*sqrt(A^2-4) ≥ A + sqrt(A^2 -4)
proof -
  have sqrt-D-nonneg:sqrt(A^2-4)≥0 using Alarge
      real-of-int-strict-inequality[of 4 A^2] by auto
  have Y1: Y ≥ 1 using ‹Y>0› by auto
  then have X^2 - (A^2-4)*Y^2 ≤ X^2 - A^2 + 4
    using Y1 nat-le-eq-zle[of A^2-4 (A^2-4)*Y^2] ‹A^2>4› by auto
  then have 2: X ≥ A using XYsol ‹X≥0› power2-le-imp-le by auto
  have Y*sqrt(A^2-4) ≥ sqrt(A^2 -4)
```

using *mult-mono*[*of 1 real-of-int Y sqrt*(*A^2 −4*) *sqrt*(*A^2 −4*)]
sqrt-D-nonneg Y1 **by** *auto*
thus *?thesis* **using** *2 add-mono*[*of A X^2 sqrt*(*A^2 −4*) *sqrt*(*A^2−4*)∗*Y*] **by** *auto*
qed

lemma *binomial-form-sol*:
fixes *X Y A*::*int*
assumes *Alarge*: *A^2>4* **and** *XYsol*: *X^2 −*(*A^2−4*)∗*Y^2 = 4*
shows (*X*+*Y*∗*sqrt*(*A^2−4*))∗(*X*−*Y*∗*sqrt*(*A^2−4*)) = *4*
proof −
have *real-int-A4*: *real-of-int A^2 > 4*
using *Alarge real-of-int-strict-inequality*[*of 4 A^2*] **by** *auto*
then have *real-of-int Y ∗ sqrt* ((*real-of-int A*)^2 − *4*) ∗
(*real-of-int Y ∗ sqrt* ((*real-of-int A*)^2 − *4*)) = *real-of-int Y^2 ∗* ((*real-of-int A*)^2 − *4*)
unfolding *power-def* **by** (*simp add*:*algebra-simps*)
then have (*X*+*Y*∗*sqrt*(*A^2−4*))∗(*X*−*Y*∗*sqrt*(*A^2−4*)) = *X^2 −* (*A^2 −4*)∗*Y^2*
unfolding *power-def*
using *real-int-A4 square-diff-square-factored*[*of X Y*∗*sqrt*(*A^2−4*)]
by (*simp add*:*algebra-simps*)
thus *?thesis* **using** *XYsol* **by** *auto*
qed

lemma *smallest-sol*:
fixes *X Y A*::*int*
assumes *Alarge*: *A^2>4* **and** *XYsol*: *X^2 −*(*A^2−4*)∗*Y^2 = 4*
and *lowerbound*: *2 < X + Y*∗*sqrt*(*A^2−4*)
and *upperbound*: *X +Y*∗*sqrt*(*A^2−4*) *< A + sqrt*(*A^2−4*)
shows *False*
proof −
have *real-int-A4*: (*real-of-int A^2*) *> 4*
using *Alarge real-of-int-strict-inequality*[*of 4 A^2*] **by** *auto*
then have *sqrt-D-pos*:*sqrt*(*A^2−4*)*>0* **by** *auto*
then have *sqrt-D-nonneg*:*sqrt*(*A^2−4*)≥*0* **by** *auto*

have *disj*: *Y=0* ∨ (*X*≥*0* ∧ *Y>0*) ∨ (*X*≥*0* ∧ *Y<0*) ∨ (*X<0* ∧ *Y>0*) ∨ (*X<0* ∧ *Y<0*)
by *auto*

then consider
(*case0*) *Y=0*
| (*case1*) *X*≥*0* ∧ *Y>0*
| (*case2*) *X*≥*0* ∧ *Y<0*
| (*case3*) *X<0* ∧ *Y>0*
| (*case4*) *X<0* ∧ *Y<0*
by *auto*

```
  then show False
  proof (cases)
    assume Y0: Y = 0
    then have 0: X^2 = 2^2 using XYsol by auto
    thus ?thesis
    proof (cases X≥0)
      assume X ≥ 0
      then have 2 = X + Y*sqrt(A^2−4) using power2-eq-imp-eq[of X 2] 0 Y0
by auto
      thus ?thesis using lowerbound by auto
    next
      assume ¬ X ≥ 0
      then have X + Y*sqrt(A^2−4) = −2 using power2-eq-imp-eq[of −X 2] 0
Y0 by auto
      thus False using lowerbound by auto
    qed
  next

    assume assm: X≥0 ∧ Y>0
    thus False using smallest-sol-sublemma[of A X Y] Alarge XYsol upperbound
by auto
  next

    assume assm: X<0 ∧ Y<0
    then have (−Y)*sqrt(A^2−4) > 0
      using mult-strict-mono'[of 0 − real-of-int Y 0 sqrt(A^2−4)] sqrt-D-pos by
auto
    thus False using sqrt-D-nonneg assm
      add-strict-mono[of real-of-int X 0 Y*sqrt(A^2−4) 0] lowerbound by auto
  next

    assume assm: X<0 ∧ Y>0
    then have −Y*sqrt(A^2−4) < 0 using sqrt-D-pos assm
      mult-strict-mono[of 0 − real-of-int Y 0 sqrt(A^2−4)] by auto
    then have 0: X − Y*sqrt(A^2−4) < 0 using assm
      add-le-less-mono[of 0 X 0 −Y*sqrt(A^2−4)] by auto
   then have 1: 2*(X − Y*sqrt(A^2−4)) > (X+Y*sqrt(A^2−4))*(X−Y*sqrt(A^2−4))
    using lowerbound mult-strict-right-mono-neg[of 2 X+Y*sqrt(A^2−4)X−Y*sqrt(A^2−4)]
by auto
    thus False using binomial-form-sol[of A X Y] Alarge XYsol 0
      by (simp add: algebra-simps)

  next
    assume assm: 0≤ X ∧ Y<0
    then have 1: X − Y*sqrt(A^2−4) ≥ A + sqrt(A^2 −4)
      using smallest-sol-sublemma[of A X −Y] XYsol Alarge assm by auto
    have (−Y)*sqrt(A^2−4) > 0 using mult-strict-mono'[of 0 − real-of-int Y 0
sqrt(A^2−4)]
```

sqrt-D-pos assm **by** *auto*
then have $X - Y*sqrt(A\hat{}2-4) \geq 0$ **using** *assm*
add-le-less-mono[*of 0 X 0* $-Y*sqrt(A\hat{}2-4)$] **by** *auto*
then have $2*(X - Y*sqrt(A\hat{}2-4)) \leq 4$ **using** *lowerbound mult-right-mono*[*of 2* $X+Y*sqrt(A\hat{}2-4)$
$X - Y*sqrt(A\hat{}2-4)$] *binomial-form-sol*[*of A X Y*] *XYsol Alarge* **by** *auto*
then have *5*: $X - Y*sqrt(A\hat{}2-4) \leq 2$ **by** *auto*

thus *False*
proof (*cases* $A<0$)
assume *assm*: $A<0$
have $A+ sqrt(A\hat{}2-4) \leq A+sqrt(A\hat{}2)$ **using** *real-sqrt-le-mono*[*of* $A\hat{}2-4$ $A\hat{}2$] **by** *auto*
thus *False* **using** *upperbound lowerbound assm* **by** *auto*
next
assume *assm*: $\neg A<0$
then have $A>2$ **using** *power2-less-imp-less*[*of 2 A*] *Alarge* **by** *auto*
thus *False* **using** *sqrt-D-pos*
add-strict-mono[*of 2 A 0 sqrt*$(A\hat{}2-4)$] *5 1* **by** *auto*
qed
qed
qed

5.2.4 Finite generation of solutions

lemma *finite-generation-nat*:
fixes *X Y A*::*int* **and** *n*::*nat*
assumes *sol*: $X\hat{}2 - (A\hat{}2-4)*Y\hat{}2 = 4$ **and** *Alarge*: $A\hat{}2 > 4$
shows $\exists X3.\ \exists\ Y3.\ 2*(((X + sqrt(A\hat{}2-4)*Y)/2)\hat{}n) = X3 + sqrt(A\hat{}2-4)*Y3 \wedge$
$X3\hat{}2 - (A\hat{}2-4)*Y3\hat{}2 = 4$ (**is** *?P n*)
proof (*induct n*)
case *0*
have *1*: $\exists\ Y1.\ 2*(((X + sqrt(A\hat{}2-4)*Y)/2)\hat{}0) = 2 + sqrt(A\hat{}2-4)*Y1 \wedge$
$2\hat{}2 - (A\hat{}2-4)*Y1\hat{}2 = 4$
apply (*rule exI*[*of - 0*])
by *auto*
show *?case*
apply (*rule exI*[*of - 2*])
using *1* **by** *auto*
next
case (*Suc n*)
then have *n*: *?P n* **using** *Suc.prems* **by** *simp*
then obtain *X1 Y1* **where** *X1Y1def*: $2*(((X + sqrt\ (A^2 - 4)*Y)/2)\hat{}n = X1 + sqrt(A^2 - 4)*Y1 \wedge$
$X1^2 - (A^2 - 4) * Y1^2 = 4$ **by** *blast*
define *X3* **where** $X3 \equiv X*X1 + (A\hat{}2-4)*Y*Y1$
define *Y3* **where** $Y3 \equiv X*Y1 + X1*Y$
then have *X3Y3sol*: $(floor\ (X3/2))\hat{}2 - (A\hat{}2-4)*(floor(Y3/2))\hat{}2 = 4$ **un-**

folding *X3-def Y3-def*
using *group-structure-sol4 X1Y1def sol Alarge* **by** *auto*
have *evenX3*: *even X3* **unfolding** *X3-def* **using** *Alarge sol X1Y1def group-structure-int*
by *auto*
have *evenY3*: *even Y3* **unfolding** *Y3-def* **using** *Alarge sol X1Y1def group-structure-int*
by *auto*

have *realA4*: (*real-of-int A*) ^2 ≥ *4* **using** *Alarge real-of-int-strict-inequality* **by**
auto
have *2*∗(((*X* + *sqrt*(*A* ^2−*4*)∗*Y*)/*2*) ^(*n*+*1*)) =
2∗*1*/*2*∗(((*X* + *sqrt*(*A* ^2−*4*)∗*Y*)/*2*) ^*n*)∗(*X* + *sqrt*(*A* ^2−*4*)∗*Y*) **by** *auto*
also have ... = *1*/*2*∗((*X1* + *sqrt*(*A*^{2}−*4*)∗*Y1*)∗(*X*+*sqrt*(*A* ^2−*4*)∗*Y*)) **using**
X1Y1def **by** *auto*
also have ... = *1*/*2*∗(*X1*∗*X* + *sqrt*(*A* ^2−*4*)∗*Y1*∗*X* + *X1*∗*sqrt*(*A* ^2−*4*)∗*Y* +
(*A* ^2−*4*)∗*Y1*∗*Y*)
using *realA4* **by** (*simp add*: *algebra-simps*)
also have ... =*1*/*2*∗(*X3* + *Y3*∗*sqrt*(*A* ^2−*4*)) **unfolding** *X3-def Y3-def* **by**
(*simp add*: *algebra-simps*)
also have ... =*floor*(*X3*/*2*) + *floor*(*Y3*/*2*)∗*sqrt*(*A* ^2−*4*) **using** *evenX3 floor-even*
evenY3 floor-even **by** *auto*
finally have *2*∗(((*X* + *sqrt*(*A* ^2−*4*)∗*Y*)/*2*) ^(*n*+*1*)) = *floor*(*X3*/*2*) + *floor*(*Y3*/*2*)∗*sqrt*(*A* ^2−*4*)
by *auto*

then have *final*: *2*∗(((*X* + *sqrt*(*A* ^2−*4*)∗*Y*)/*2*) ^(*n*+*1*)) = *floor*(*X3*/*2*) +
floor(*Y3*/*2*)∗*sqrt*(*A* ^2−*4*)
∧ (*floor* (*X3*/*2*)) ^2 − (*A* ^2−*4*)∗(*floor*(*Y3*/*2*)) ^2 = *4* **using** *X3Y3sol* **by**
auto
have *4*: ∃ *Y2*. *2*∗(((*X* + *sqrt*(*A* ^2−*4*)∗*Y*)/*2*) ^(*n*+*1*)) = *floor*(*X3*/*2*) + *Y2*∗*sqrt*(*A* ^2−*4*)
∧ (*floor* (*X3*/*2*)) ^2 − (*A* ^2−*4*)∗*Y2* ^2 = *4*
apply (*rule exI*[*of* - *floor*(*Y3*/*2*)])
using *final* **by** *auto*
show *?case*
apply (*rule exI*[*of* - *floor*(*X3*/*2*)])
using *4* **by** *auto*
qed

lemma *finite-generation*:
fixes *X Y A*::*int* **and** *n*::*nat*
assumes *sol*: *X* ^2 − (*A* ^2−*4*)∗*Y* ^2 = *4* **and** *A* ^2>*4*
shows ∃ *X1*. ∃ *Y1*. *2*∗ (*inverse* ((*X* + *sqrt*(*A* ^2−*4*)∗*Y*)/*2*) ^*n*) = *X1* + *sqrt*(*A* ^2−*4*)∗*Y1*
∧
X1 ^2 − (*A* ^2−*4*)∗*Y1* ^2 = *4*
proof (*cases n*=*0*)
assume *n*=*0*
thus *?thesis* **using** *finite-generation-nat*[*of X A Y 0*] *sol* ‹*A* ^2>*4*› **by** *auto*
next
assume ¬ *n*=*0*
have *X* + *sqrt*(*A* ^2−*4*)∗*Y* ≠ *0* **using** ‹*A* ^2>*4*› *sol sol-non-zero* **by** *auto*

```
 then have 2*(inverse ((X+sqrt(A^2-4)*Y)/2))^n = 2*(2*inverse (X+sqrt(A^2-4)*Y))^n
   using divide-inverse nonzero-inverse-inverse-eq[of 2]
         nonzero-inverse-mult-distrib[of inverse 2 X+sqrt(A^2-4)*Y] by auto
 then have 2*(inverse ((X+sqrt(A^2-4)*Y)/2))^n = 2*((X+sqrt(A^2-4)*(-Y))/2)^n
   using conj-inversion ‹A^2>4› sol by auto
 thus ?thesis using finite-generation-nat[of X A -Y n] ‹A^2>4› sol by auto
qed

lemma real-arch-power:
 fixes x::real and y::real
 assumes x1: x>1 and y1: y ≥ 1
 shows ∃ n. x^n ≤ y ∧ y < x^(n+1)
proof -
 define P::nat ⇒ bool where P ≡ (λn. y < x^n)
 obtain m where mdef: y < x^m using x1 real-arch-pow by auto
 then have Pm: P m unfolding P-def by auto
 have ¬ P 0 using y1 unfolding P-def by auto
 then have ∃k<m. (∀ i≤k. ¬ P i) ∧ P (Suc k)
   using Pm ex-least-nat-less[of P m] by auto
 then obtain n where (∀ i≤n. ¬ P i) ∧ P (Suc n) by auto
 then have x^n ≤ y ∧ (y<x^(n+1)) unfolding P-def by auto
 thus ?thesis
    by (rule exI[of - n])
qed

lemma finite-gen-all-sol:
 fixes X::int and Y::int and A::int
 defines rho ≡ |A| + sqrt(A^2-4)
       and Z ≡ X + sqrt(A^2-4)*Y and D ≡ A^2-4
 assumes Alarge: A^2>4 and XYsol: X^2-(A^2-4)*Y^2 = 4
 shows ∃ n. Z ∈ {2*(rho/2)^n, -2*(rho/2)^n, 2*inverse(rho/2)^n, -2*inverse(rho/2)^n}
∧
     Y ∈ {1/sqrt(D)*((rho/2)^n - inverse(rho/2)^n), -1/sqrt(D)*((rho/2)^n
- inverse(rho/2)^n)}
proof-
 define eta where eta ≡ |X| + sqrt(A^2-4)*|Y|
 have etasol: |X|^2 - (A^2-4)*|Y|^2 =4 using XYsol by auto
 have A2: real-of-int |A| > 2 using Alarge power2-less-imp-less[of 2 |A|] by auto
  have realA4: 4 < (real-of-int A)^2 using Alarge real-of-int-strict-inequality by
auto
 then have rho2: rho > 2 unfolding rho-def
   using A2 less-trans[of 2 |A| |A| + sqrt(A^2-4)] by auto
 then have rho1: 1/2 * rho > 1 by auto

 have eta2: eta ≥ 2
 proof (cases |Y| = 0)
   assume |Y| = 0
    thus ?thesis unfolding eta-def using etasol power2-eq-imp-eq[of 2 |X|] by
auto
```

next
assume $|Y| \neq 0$
then have *rho≤eta* **unfolding** *eta-def rho-def*
using *smallest-sol-sublemma*[*of* $|A|$ $|X|$ $|Y|$] *Alarge etasol* **by** (*simp add*: *algebra-simps*)
thus *?thesis* **using** *rho2* **by** *auto*
qed
then have *eta1*: $1/2 * eta \geq 1$ **by** *auto*

obtain *n* **where** *ineq1*: $(1/2 * rho)\hat{}n \leq 1/2 * eta \ \wedge\ 1/2 * eta < (1/2 * rho)\hat{}(n+1)$
using *real-arch-power*[*of* $1/2 * rho$ $1/2 * (|X| + sqrt(A\hat{}2-4)*|Y|)$]
eta1 rho1 **unfolding** *eta-def* **by** *auto*
have *lhspos*: $2*(1/2 * rho)\hat{}n > 0$ **using** *rho1* **by** *auto*
have *0*: $inverse((1/2 * rho)\hat{}n) > 0$ **using** *rho1* **by** *auto*
have *1*: $(1/2*rho)\hat{}n \neq 0$ **using** *rho1* **by** *auto*
then have *ineq4*: $2 \leq eta* inverse(rho/2)\hat{}n$ **using** *ineq1 lhspos*
mult-mono'[*of* $2*(1/2 *rho)\hat{}n$ *eta inverse*$((1/2 * rho)\hat{}n)$ *inverse*$((1/2 * rho)\hat{}n)$]
right-inverse[*of* $(1/2 * rho)\hat{}n$] *power-inverse*[*of* $1/2*rho$ *n*] **by** (*simp add*: *algebra-simps*)
have *eta** *inverse*$((1/2 * rho)\hat{}n) < rho*(1/2*rho)\hat{}n*inverse((1/2*rho)\hat{}n)$ **using** *eta2 0*
ineq1 mult-strict-right-mono[*of eta* $2*(1/2 *rho)\hat{}(n+1)$ *inverse*$((1/2 * rho)\hat{}n)$] **by** *auto*
then have *ineq5*: *eta** *inverse*$(rho/2)\hat{}n < rho$ **using** *right-inverse*[*of* $(1/2*rho)\hat{}n$] *1*
power-inverse[*of rho*/2 *n*] **by** (*simp add*: *algebra-simps*)

have *rhosol*: $|A|\hat{}2 - (A\hat{}2-4) = 4$ **by** *auto*
then obtain *X1 Y1* **where** *X1Y1def*: $2* (inverse\ (rho/2)\hat{}n) = X1 + sqrt(A\hat{}2-4)*Y1 \wedge X1\hat{}2 - (A\hat{}2-4)*Y1\hat{}2 = 4$ **unfolding** *rho-def*
using *Alarge finite-generation*[*of* $|A|$ *A 1 n*] **by** *auto*
define *X3* **where** $X3 \equiv |X|*X1 + (A\hat{}2-4)*|Y|*Y1$
define *Y3* **where** $Y3 \equiv |X|*Y1 + X1*|Y|$
have *X3Y3sol*: $4 = (floor(X3/2))\hat{}2 - (A\hat{}2-4)*(floor(Y3/2))\hat{}2$ **unfolding** *X3-def Y3-def*
using *group-structure-sol4*[*of A* $|X|$ $|Y|$ *X1 Y1*] *X1Y1def Alarge etasol*
unfolding *X3-def Y3-def* **by** *auto*
have *evX3*: *even X3* **using** *group-structure-int*[*of A* $|X|$ $|Y|$ *X1 Y1*]
Alarge X1Y1def etasol **unfolding** *X3-def* **by** *auto*
have *evY3*: *even Y3* **using** *group-structure-int*[*of A* $|X|$ $|Y|$ *X1 Y1*]
Alarge X1Y1def etasol **unfolding** *Y3-def* **by** *auto*

have $eta*inverse(rho/2)\hat{}n = 1/2*eta*(2*inverse(rho/2)\hat{}n)$ **by** *auto*
also have ... $= 1/2*((|X| + sqrt(A\hat{}2-4)*|Y|)*(X1+sqrt(A\hat{}2-4)*Y1))$
unfolding *eta-def* **using** *X1Y1def* **by** *auto*
also have ... $= 1/2*(|X|*X1 + sqrt(A\hat{}2-4)*|Y|*X1 + |X|*sqrt(A\hat{}2-4)*Y1$

$+ (A\hat{}2-4)*|Y|*Y1)$
using *realA4* **by** (*simp add*: *algebra-simps*)
also have ... $=1/2*(X3 + Y3*sqrt(A\hat{}2-4))$ **unfolding** *X3-def Y3-def* **by** (*simp add*: *algebra-simps*)
also have ... $=floor(X3/2) + floor(Y3/2)*sqrt(A\hat{}2-4)$ **using** *evX3 floor-even evY3 floor-even*
by *auto*
finally have *eq*: $eta*inverse(rho/2)\hat{}n = floor(X3/2) + floor(Y3/2)*sqrt(A\hat{}2-4)$ **by** *auto*

then have *lowerbound*: $floor(X3/2) + floor(Y3/2)*sqrt(A\hat{}2-4) \geq 2$ **using** *ineq4* **by** *auto*
have *upperbound*: $floor(X3/2) + floor(Y3/2)*sqrt(A\hat{}2-4) < rho$ **using** *eq ineq5* **by** *auto*

have $eta*inverse(rho/2)\hat{}n = 2$
proof(*rule ccontr*)
assume $eta*inverse(rho/2)\hat{}n \neq 2$
then have *strict-lowerbound*: $floor(X3/2) + floor(Y3/2)*sqrt(A\hat{}2-4) > 2$
using *eq lowerbound* **by** *auto*
thus *False* **using** *smallest-sol*[*of* $|A|$ $floor(X3/2)$ $floor(Y3/2)$] *Alarge upperbound X3Y3sol*
unfolding *rho-def* **by** *auto*
qed
then have $eta*(inverse((rho/2)\hat{}n)*(rho/2)\hat{}n) = 2*(rho/2)\hat{}n$
using *power-inverse*[*of rho/2 n*] **by** *auto*
then have *etan*: $eta = 2*(rho/2)\hat{}n$ **using** *left-inverse*[*of* $(rho/2)\hat{}n$] *1* **by** *fastforce*

have $|X| - sqrt(A\hat{}2-4)*|Y| = 4*inverse(eta)$ **using** *Alarge etasol conj-inversion*[*of A* $|X|$ $|Y|$] **unfolding** *eta-def* **by** *auto*
then have *etaconj*: $|X| - sqrt(A\hat{}2-4)*|Y| = 2*inverse((rho/2))\hat{}n$
using *power-inverse*[*of rho/2 n*] *etan* **by** *auto*

define *Zset* **where** $Zset \equiv \{2*(rho/2)\hat{}n, -2*(rho/2)\hat{}n, 2*inverse(rho/2)\hat{}n, -2*inverse(rho/2)\hat{}n\}$
define *Yset* **where** $Yset \equiv \{1/sqrt(D)*((rho/2)\hat{}n - inverse(rho/2)\hat{}n), -1/sqrt(D)*((rho/2)\hat{}n - inverse(rho/2)\hat{}n)\}$

have *Yel*: $Y \in Yset$
proof (*cases* $Y \geq 0$)
assume $Y \geq 0$
then have $Y = 1/(2*sqrt(D))*(eta - (|X| - sqrt(D)*|Y|))$
using *realA4* **unfolding** *eta-def D-def* **by** *auto*
then have $Y = 1/(2*sqrt(D))*(2*(rho/2)\hat{}n-2*inverse(rho/2)\hat{}n)$
using *etan etaconj* **unfolding** *D-def* **by** *auto*
thus *?thesis* **unfolding** *Yset-def* **by** (*simp add*:*algebra-simps*)
next
assume $\neg\ 0 \leq Y$

```
  then have Y = -1/(2*sqrt(D))*(eta - (|X| - sqrt(D)*|Y|))
    using realA4 unfolding eta-def D-def by auto
  then have Y = -1/(2*sqrt(D))*(2*(rho/2) ^n-2*inverse(rho/2) ^n)
    using etan etaconj D-def by auto
  thus ?thesis unfolding Yset-def by (simp add:algebra-simps)
 qed


 have disj: (X≥0 ∧ Y≥0) ∨ (X<0 ∧ Y≥0) ∨ (X≥0 ∧ Y<0) ∨ (X<0 ∧ Y<0)
by auto


 then consider
    (case-pos-pos) X≥0 ∧ Y≥0
    | (case-neg-pos) X<0 ∧ Y≥0
    | (case-pos-neg) X≥0 ∧ Y<0
    | (case-neg-pos) X<0 ∧ Y<0
  by auto

 then have Zel: Z ∈ Zset
 proof (cases)
  assume assm:X≥0 ∧ Y≥0
  thus ?thesis using etan unfolding eta-def Z-def Zset-def by auto
 next
  assume assm: X<0 ∧ Y<0
  then show ?thesis using etan unfolding eta-def Z-def Zset-def by auto
 next
  assume assm: X<0 ∧ Y≥0
  then show ?thesis using etaconj unfolding Z-def Zset-def by auto
 next
  assume assm: X≥0 ∧ Y<0
  then show ?thesis using etaconj unfolding Z-def Zset-def by auto
 qed

 have Z ∈ {2*(rho/2) ^n, -2*(rho/2) ^n, 2*inverse(rho/2) ^n, -2*inverse(rho/2) ^n}
∧
     Y ∈ {1/sqrt(D)*((rho/2) ^n - inverse(rho/2) ^n), -1/sqrt(D)*((rho/2) ^n
- inverse(rho/2) ^n)}
   using Zel Yel unfolding Zset-def Yset-def by auto
 then show ?thesis
    by (rule exI[of - n::nat])
   qed
```

5.2.5 Link between Pell equation and Lucas sequences

```
lemma link-to-lucas:
 fixes A::int and n::nat
 assumes A4:A^2 > 4
 shows inverse(sqrt(A^2-4))*((1/2*((real-of-int A)+sqrt(A^2-4))) ^n -(2*inverse((real-of-int
```

```
A)+sqrt(A^2−4)))^(n))= ψ A n
proof (induction n rule: ψ-induct)
  case 0
  show ?case by auto
next
  case 1
  define A' where A' ≡ real-of-int A
  define E where E ≡ sqrt(A^2−4)
  have A'^2 > 4 using A4 real-of-int-strict-inequality unfolding A'-def by auto
  then have E>0 unfolding E-def A'-def by (simp add: algebra-simps)
  have sol:A^2 − (A^2 − 4)*1 = 4 by (simp add: algebra-simps)
  have not0:A+E ≠ 0 using sol-non-zero[of A A 1]  A4 sol unfolding E-def by
auto

  have 2*inverse(A+E) = 1/2*(A−E)
    using conj-inversion[of A A 1] A4 sol not0 unfolding E-def by auto
  then have 2*inverse(A'+E) = 1/2*(A'−E) unfolding A'-def by auto
  then have 1/2*(A'+E) − 2*inverse(A'+E) = 1/2*(A'+E)−1/2*(A'−E) by
auto
  also have ... = E using right-diff-distrib'[of 1/2 A'+E A'−E]
    by (simp add: algebra-simps)
  finally have 1/2*(A'+E) − 2*inverse(A'+E) =E by auto
  then have (inverse E)*(1/2*(A'+E) − 2*inverse(A'+E)) = 1
    using left-inverse[of E] ‹E>0› by auto
  thus ?case unfolding  E-def A'-def by auto
next
  case (sucsuc n)
  define A' where A' ≡ real-of-int A
  define E where E ≡ sqrt(A^2−4)
 define Yn where Yn≡(inverse E)*((1/2*(A'+ E ))^n −(2*inverse(A'+E))^n)
 define Yn1 where Yn1≡(inverse E)*((1/2*(A'+ E ))^(Suc n) −(2*inverse(A'+E))^(Suc
n))
 define Yn2 where Yn2≡(inverse E)*((1/2*(A'+ E ))^(Suc (Suc n)) −(2*inverse(A'+E))^(Suc
(Suc n)))
  have Yn-IH:Yn = ψ A n unfolding Yn-def A'-def E-def using sucsuc.IH(2)
by auto
  have Yn1-IH:Yn1 = ψ A (n+1) unfolding Yn1-def A'-def E-def using suc-
suc.IH(1) by auto
  have sol:A^2 − (A^2 − 4)*1 = 4 by (simp add: algebra-simps)
  have not0:A+E ≠ 0 using sol-non-zero[of A A 1]  A4 sol unfolding E-def by
auto
  have 1:1/2*(A'+E)*(1/2*(A'+ E))^(Suc n) = (1/2*(A'+ E))^(Suc (Suc n))
    by auto
 have 2:2*inverse(A'+E)*(2*inverse(A'+E))^(Suc n) = (2*inverse(A'+E))^(Suc
(Suc n))
    by auto
  have 3:2*inverse(A'+E)*(1/2*(A'+ E))^(Suc n) = (1/2*(A'+E))^n
    using not0 right-inverse[of A'+E] unfolding A'-def by auto
  have 4:1/2*(A'+E)*(2*inverse(A'+E))^(Suc n) = (2*inverse (A'+E))^n
```

using *not0 right-inverse*[*of* $A'+E$] **unfolding** A'*-def* **by** *auto*

have $2*inverse(A'+E) = 1/2*(A'-E)$
using *conj-inversion*[*of A A 1*] *A4 sol not0* **unfolding** *E-def* A'*-def* **by** *auto*
then have $1/2*(A'+E) + 2*inverse(A'+E) = 1/2*(A'+E) + 1/2*(A'-E)$
by *auto*
also have $... = A'$
using *distrib-left*[*of 1/2* $A'+E$ $A'-E$] **by** (*simp add*: *algebra-simps*)
finally have $A'=1/2*(A'+E) + 2*inverse(A'+E)$ **by** *auto*
then have $A'*Yn1 = (inverse\ E)*(1/2*(A'+E) + 2*inverse(A'+E))$
$*((1/2*(A'+ E))\ \hat{}(Suc\ n) -(2*inverse(A'+E))\ \hat{}(Suc\ n))$
unfolding *Yn1-def* **by** *auto*
also have $... = (inverse\ E)*(1/2*(A'+E)*(1/2*(A'+ E))\ \hat{}(Suc\ n)$
$-(1/2*(A'+E))*(2*inverse(A'+E))\ \hat{}(Suc\ n)$
$+2*inverse(A'+E)*(1/2*(A'+ E))\ \hat{}(Suc\ n)$
$-2*inverse(A'+E)*(2*inverse(A'+E))\ \hat{}(Suc\ n))$
using *distrib-add-diff* **by** *auto*
also have $... = (inverse\ E)*((1/2*(A'+ E))\ \hat{}(Suc\ (Suc\ n))$
$-(2*inverse(A'+E))\ \hat{}n+ (1/2*(A'+ E\ ))\ \hat{}n\ -(2*inverse(A'+E))\ \hat{}(Suc\ (Suc\ n)))$
using *1 4 2 3* **by** *metis*
also have $... = Yn2 + Yn$ **unfolding** *Yn2-def Yn-def* **by** (*simp add*:*algebra-simps*)
finally have $A'*Yn1 = Yn2 + Yn$ **by** *auto*
then have $Yn2 = A'*Yn1 - Yn$ **by** *auto*
also have $... = A'*(\psi\ A\ (n + 1)) - (\psi\ A\ n)$ **using** *Yn-IH Yn1-IH* **by** *auto*
also have $... = \psi\ A\ (n+2)$ **unfolding** A'*-def* **by** *auto*
finally have $Yn2 = \psi\ A\ (n+2)$ **by** *auto*
thus *?case* **unfolding** *Yn2-def E-def* A'*-def* **by** *auto*
qed

5.2.6 Special cases

lemma *lucas-pell-sublemmaA2*:
fixes Y::*int*
shows $\exists m.\ Y = \psi\ 2\ m \vee Y = -\ \psi\ 2\ m$
proof (*cases* $Y \geq 0$)
assume $Y \geq 0$
then have $Y = \psi\ 2\ (nat\ Y) \vee Y = -\ \psi\ 2\ (nat\ Y)$ **using** *lucas-A-eq-2*[*of* (*nat* Y)] **by** *auto*
thus *?thesis* **by** *auto*
next
assume *Yneg*: $\neg\ Y \geq 0$
then have $Y = \psi\ 2\ (nat(-Y)) \vee Y = -\ \psi\ 2\ (nat\ (-Y))$ **using** *lucas-A-eq-2*[*of* $nat(-Y)$] **by** *auto*
thus *?thesis* **by** *auto*
qed

lemma *lucas-pell-sublemmaAmin2*:
fixes Y::*int*

```
  shows ∃m. Y = ψ (−2) m ∨ Y = − ψ (−2) m
proof −
  obtain m where Y = ψ 2 m ∨ Y = − ψ 2 m using lucas-pell-sublemmaA2
by auto
  then show ?thesis
  proof
    assume Y2: Y = ψ 2 m
    thus ?thesis
    proof (cases odd m)
      assume odd m
    then have Y = ψ (−2) m ∨ Y = − ψ (−2) m using Y2 lucas-symmetry-A2[of
−2 m] by auto
      thus ?thesis by auto
    next
      assume meven: ¬ (odd m)
    then have Y = ψ (−2) m ∨ Y = − ψ (−2) m using Y2 lucas-symmetry-A2[of
−2 m] by auto
      thus ?thesis by auto
    qed
  next
    assume Ymin2: Y = − ψ 2 m
    thus ?thesis
    proof (cases odd m)
      assume modd: odd m
     then have Y = ψ (−2) m ∨ Y = − ψ (−2) m using lucas-symmetry-A2[of
−2 m] Ymin2 by auto
      thus ?thesis by auto
    next
      assume ¬ (odd m)
      then have meven: even m by auto
     then have Y = ψ (−2) m ∨ Y = − ψ (−2) m using lucas-symmetry-A2[of
−2 m] Ymin2 by auto
      thus ?thesis by auto
    qed
 qed
qed

lemma lucas-pell-sublemmaA0:
  fixes Y::int
  assumes assm: ∃k. (−4)∗Y^2 + 4 = k^2
  shows ∃m. Y = ψ 0 m ∨ Y = − ψ 0 m
proof−
  obtain k where kdef: −4∗Y^2 + 4 = k^2 using assm by auto


  have smallY: Y ∈ {0,1,−1}
  proof−
    have 1:4∗(1− Y^2) ≥ 0 using kdef by auto
    then have 2: 1≥Y using power2-le-imp-le[of Y 1] by auto
```

```
  have Y ≥ −1 using 1 power2-le-imp-le[of −Y 1] by (simp add: algebra-simps)
   thus ?thesis  using 2 by auto
 qed


 then consider
     (case0) Y=0
   | (case1) Y=1
   | (casemin1) Y=−1
   by auto

 then show ?thesis
 proof (cases)
   assume Y = 0
   then have Y = ψ 0 0 ∨ Y = − ψ 0 0 by auto
   then show ∃ m. Y = ψ 0 m ∨ Y = − ψ 0 m
     by (rule exI[of - 0])
 next
   assume Y=1
   then have Y = ψ 0 1 ∨ Y = − ψ 0 1 by auto
   thus ?thesis
     by (rule exI[of - 1])
 next
   assume Y=−1
   then have Y=ψ 0 1 ∨ Y=−ψ 0 1 by auto
   thus ?thesis
     by (rule exI[of - 1])
 qed
qed


lemma lucas-pell-sublemmaA1:
 fixes Y::int
 assumes assm: ∃ k. (1^2−4)*Y^2 + 4 = k^2
 shows ∃ m. Y = ψ 1 m ∨ Y = − ψ 1 m
proof−
 obtain k where k-def: −3*Y^2 + 4 = k^2 using assm by auto


 have smallY: Y ∈ {0,1,−1}
 proof −
   have 1:4 −3*Y^2≥0 using k-def by auto
   then have 2: 1≥Y using power2-le-imp-le[of Y 1] by auto
  have Y ≥ −1 using 1 power2-le-imp-le[of −Y 1] by (simp add: algebra-simps)
   then show ?thesis using 2 by auto
 qed


 then consider
```

```
    (case0) Y=0
  | (case1) Y=1
  | (casemin1) Y=−1
    by auto

  then show ?thesis
  proof (cases)
    assume Y = 0
    then have Y = ψ 1 0 ∨ Y = − ψ 1 0 by auto
    then show ∃ m. (Y = ψ 1 m ∨ Y = − ψ 1 m)
      by (rule exI[of - 0])
  next
    assume Y=1
    then have Y = ψ 1 1 ∨ Y = − ψ 1 1 by auto
    thus ?thesis
      by (rule exI[of - 1])
  next
    assume Y=−1
    then have Y=ψ 1 1 ∨ Y=−ψ 1 1 by auto
    thus ?thesis
      by (rule exI[of - 1])
    qed
qed

lemma lucas-pell-sublemmaAmin1:
  fixes Y::int
  assumes assm: ∃ k. ((−1) ^2−4)∗Y^2 + 4 = k^2
  shows ∃ m. Y = ψ (−1) m ∨ Y = − ψ (−1) m
proof −
  obtain m where m-def: Y = ψ 1 m ∨ Y = − ψ 1 m
    using lucas-pell-sublemmaA1 assm by auto

  then consider
    (caseplus) Y = ψ 1 m
    | (caseminus) Y = − ψ 1 m
    by auto

  then show ?thesis
  proof (cases)
    assume Y = ψ 1 m
    then have 1: Y = (−1) ^(m+1)∗(ψ (−1) m) using lucas-symmetry-A2[of 1
m] by auto
    then show (∃ m. Y = ψ (−1) m ∨ Y = − ψ (−1) m)
    proof (cases even m)
      assume even m
      then have Y = ψ (−1) m ∨ Y = − ψ (−1) m using 1 by auto
      thus ?thesis
        by (rule exI[of - m])
    next
```

```
    assume odd m
    then have Y = ψ (−1) m ∨ Y = − ψ (−1) m using 1 by auto
    thus ?thesis
      by (rule exI[of - m])
  qed
 next
  assume Y = − ψ 1 m
   then have 1: Y = (−1)^m*(ψ (−1) m) using lucas-symmetry-A2[of 1 m]
by auto
  then show (∃ m. Y = ψ (−1) m ∨ Y = − ψ (−1) m)
  proof (cases even m)
    assume even m
    then have Y = ψ (−1) m ∨ Y = − ψ (−1) m using 1 by auto
    thus ?thesis
      by (rule exI[of - m])
  next
    assume odd m
    then have Y = ψ (−1) m ∨ Y = − ψ (−1) m using 1 by auto
    thus ?thesis
      by (rule exI[of - m])
  qed
 qed
qed
```

5.2.7 The main equivalence

```
lemma lucas-pell-part1:
 fixes A Y::int
 shows (∃ k. (A^2−4)*Y^2 + 4 = k^2) ⟹ (∃ m. Y = ψ A m ∨ Y = − ψ A
m)
proof −
 assume assumption: ∃ k. (A^2−4)*Y^2 + 4 = k^2
 then obtain k where defm: (A^2−4)*Y^2 + 4 = k^2 by auto

 have disj: A=2 ∨ A=−2 ∨ A=0 ∨ A=1 ∨ A=−1 ∨ (A^2 > 4)
 proof −
  have 1:A^2 ≤ 4 ⟹ A ≤ 2 using power2-le-imp-le[of A 2] by auto
  have A^2 ≤ 4 ⟹ A ≥−2 using power2-le-imp-le[of −A 2] by (simp add:
algebra-simps)
  thus A=2 ∨ A=−2 ∨ A=0 ∨ A=1 ∨ A=−1 ∨ (A^2 > 4) using 1 by auto
 qed


 then consider
    (case0) A=0
    | (case1) A=1
    | (casemin1) A=−1
    | (case2) A=2
    | (casemin2) A=−2
```

```
    | (caseLarge) A^2 > 4
   by auto

  then show ?thesis
  proof (cases)
   assume A0: A = 0
   show ?thesis using A0 lucas-pell-sublemmaA0 assumption by auto
  next
   assume Amin1: A = −1
   show ?thesis using Amin1 lucas-pell-sublemmaAmin1 assumption by auto
  next
   assume A1: A = 1
   show ?thesis using A1 lucas-pell-sublemmaA1 assumption by auto
  next
   assume Amin2: A = −2
   show ?thesis using Amin2 lucas-pell-sublemmaAmin2 assumption by auto
  next
   assume A2: A = 2
   show ?thesis using A2 lucas-pell-sublemmaA2 assumption by auto
  next
   assume A4:A ^ 2 > 4
   define D where D≡ A^2 − 4
   define rho where rho ≡ |A| + sqrt(A^2−4)
   have realA4: 4 < (real-of-int A)^2 using A4 real-of-int-strict-inequality by auto

   obtain n where ndef: Y ∈ {1/sqrt(D)∗((rho/2)^n − inverse(rho/2)^n),
      −1/sqrt(D)∗((rho/2)^n − inverse(rho/2)^n)}
     using finite-gen-all-sol[of A k Y] A4 defm unfolding rho-def D-def by auto
   have psiabsA: 1/sqrt(D)∗((rho/2)^n − (inverse(rho/2))^n) = ψ |A| n
      using link-to-lucas[of |A| n] A4 inverse-eq-divide[of sqrt(|A|^2 − 4)]
      nonzero-inverse-mult-distrib[of rho 1/2] sol-non-zero[of |A| A 1]
      unfolding rho-def D-def by auto
   then consider (plus) Y = 1/sqrt(D)∗((rho/2)^n − inverse(rho/2)^n) |
     (minus) Y = −1/sqrt(D)∗((rho/2)^n − inverse(rho/2)^n) using ndef by
auto
   then show ?thesis
   proof cases
    case plus
    then have YabsA: Y = ψ |A| n  using psiabsA plus by auto
    then show ?thesis
    proof (cases A≥0)
     assume Apos: A≥0
     then have Y = ψ A n ∨ Y = − ψ A n using YabsA by auto
     thus ?thesis
      by (rule exI[of - n])
    next
     assume Aneg: ¬ A≥0
     thus ?thesis
     proof (cases even n)
```

```
        assume evn: even n
       then have Y = ψ A n ∨ Y = - ψ A n using YabsA lucas-symmetry-A2[of
|A| n] Aneg by auto
        thus ?thesis
          by (rule exI[of - n])
       next
        assume oddn: ¬ even n
       then have Y = ψ A n ∨ Y = - ψ A n using YabsA lucas-symmetry-A2[of
|A| n] Aneg by auto
        thus ?thesis
          by (rule exI[of - n])
       qed
      qed
    next
      case minus
      then have YabsA: Y = - ψ |A| n  using psiabsA minus by auto
      then show ?thesis
      proof (cases A≥0)
        assume Apos: A≥0
        then have Y = ψ A n ∨ Y = - ψ A n using YabsA by auto
        thus ?thesis
          by (rule exI[of - n])
      next
        assume Aneg: ¬ A≥0
        thus ?thesis
        proof (cases even n)
          assume evn: even n
        then have Y = ψ A n ∨ Y = - ψ A n using YabsA lucas-symmetry-A2[of
|A| n] Aneg by auto
          thus ?thesis
            by (rule exI[of - n])
        next
          assume oddn: ¬ even n
        then have Y = ψ A n ∨ Y = - ψ A n using YabsA lucas-symmetry-A2[of
|A| n] Aneg by auto
          thus ?thesis
            by (rule exI[of - n])
        qed
      qed
    qed
  qed
qed

lemma lucas-pell-part3:
  fixes A::int and m::nat
  shows (A^2-4)*(ψ A m)^2+4 = (χ A m)^2
proof -
  have (A^2-4)*(ψ A m)^2+4 = (χ A m)^2
    ∧ (A^2-4)*ψ A m * ψ A (m+1) +2*A  = χ A m * χ A (m+1)
```

proof (*induction m rule*: *ψ-induct*)
case *0*
then show *?case* **by** *auto*
next
case *1*
have *(A^2−4)∗(ψ A 1)^2+4 = (χ A 1)^2*
by *auto*
have $(A^2 - 4) * \psi\ A\ 1 * \psi\ A\ (1 + 1) + 2 * A = (A\hat{}2-4) * A + 2*A$
by *auto*
also have *... = A^2∗A−4∗A+2∗A*
by (*simp add*: *left-diff-distrib′*)
also have *... = A∗A∗A−2∗A*
by (*simp add*: *power2-eq-square*)
also have *... = A∗(A∗A−2)*
by (*simp add*: *right-diff-distrib′*)
also have *... = χ A 1 ∗ χ A (1+1)*
by *auto*
finally have $(A^2 - 4) * \psi\ A\ 1 * \psi\ A\ (1 + 1) + 2 * A = \chi\ A\ 1 * \chi\ A\ (1+1)$**.**
then show *?case* **by** *auto*
next
case (*sucsuc m*)
note *t = this*
have $(A^2 - 4) * (\psi\ A\ (m + 2))^2 + 4 = (A\hat{}2- 4) * (A * \psi\ A\ (m+1) - \psi\ A\ m)\hat{}2 + 4$
by *auto*
also have $... = (A\hat{}2- 4) * (A * \psi\ A\ (m+1) - \psi\ A\ m)*(A * \psi\ A\ (m+1) - \psi\ A\ m) + 4$
using *power2-eq-square* **by** *auto*
also have $... = (A\hat{}2- 4)*(A*\psi\ A\ (m+1) *(A * \psi\ A\ (m+1) - \psi\ A\ m) - \psi\ A\ m *(A * \psi\ A\ (m+1)$
$- \psi\ A\ m))+4$
by (*simp add*: *left-diff-distrib′*)
also have $... = (A\hat{}2-4)*(A*\psi\ A\ (m+1) * A * \psi\ A\ (m+1) - A*\psi\ A\ (m+1)*\psi\ A\ m$
$- (\psi\ A\ m * A*\psi\ A\ (m+1)- \psi\ A\ m * \psi\ A\ m)) + 4$
by (*simp add*: *right-diff-distrib′*)
also have $... = (A\hat{}2-4)*(A*\psi\ A\ (m+1)*A*\psi\ A\ (m+1)$
$- 2 * A * \psi\ A\ (m+1) * \psi\ A\ m + \psi\ A\ m * \psi\ A\ m)+4$
by *auto*
also have $... = (A\hat{}2-4)*(A\hat{}2*(\psi\ A\ (m+1))\hat{}2 - 2 * A * \psi\ A\ (m+1) * \psi\ A\ m + (\psi\ A\ m)\hat{}2)+4$
by (*simp add*: *power2-eq-square*)
also have $... = (A\hat{}2-4)*(A\hat{}2*(\psi\ A\ (m+1))\hat{}2) + (A\hat{}2-4)*(- 2 * A * \psi\ A\ (m+1) * \psi\ A\ m\ )$
$+ (A\hat{}2-4)*(\psi\ A\ m)\hat{}2+4$
by (*smt* (*verit*, *best*) *add.commute distrib-left mult-minus-left uminus-add-conv-diff*)
also have $... = A\hat{}2*((A\hat{}2-4)*(\psi\ A\ (m+1))\hat{}2) -2*A*(A\hat{}2-4)*\psi\ A\ (m+1) * \psi\ A\ m$
$+ (A\hat{}2-4)*(\psi\ A\ m)\hat{}2+4$

by *auto*
also have ... = $A\hat{}2*((A\hat{}2-4)*(\psi\ A\ (m+1))\hat{}2) - 2*A*(A\hat{}2-4)*\psi\ A\ (m+1) * \psi\ A\ m + (\chi\ A\ m)\hat{}2$
using *t* **by** *auto*
also have ... = $A\hat{}2*((A\hat{}2-4)*(\psi\ A\ (m+1))\hat{}2)+A\hat{}2*4-4*A\hat{}2 -2*A*(A\hat{}2-4)*\psi\ A\ (m+1) * \psi\ A\ m + (\chi\ A\ m)\hat{}2$
by *auto*
also have ... = $A\hat{}2*((A\hat{}2-4)*(\psi\ A\ (m+1))\hat{}2+4)-4*A\hat{}2 -2*A*(A\hat{}2-4)*\psi\ A\ (m+1) * \psi\ A\ m + (\chi\ A\ m)\hat{}2$
using *distrib-left*[*of* $A\hat{}2\ ((A\hat{}2-4)*(\psi\ A\ (m+1))\hat{}2)\ 4$] **by** *simp*
also have ... = $A\hat{}2*(\chi\ A\ (m+1))\hat{}2-4*A*A -2*A*(A\hat{}2-4)*\psi\ A\ (m+1) * \psi\ A\ m + (\chi\ A\ m)\hat{}2$
using *t power2-eq-square* **by** *auto*
also have ... = $A\hat{}2*(\chi\ A\ (m+1))\hat{}2 - 2*A*(2*A+(A\hat{}2-4)*\psi\ A\ (m+1) *\psi\ A\ m) + (\chi\ A\ m)\hat{}2$
using *distrib-left*[*of* $2*A\ 2*A\ (A\hat{}2-4)*\psi\ A\ (m+1) *\psi\ A\ m$] **by** *auto*
also have ... = $A\hat{}2*(\chi\ A\ (m+1))\hat{}2 - 2*A*((A\hat{}2-4)*\psi\ A\ m *\psi\ A\ (m+1) + 2*A) + (\chi\ A\ m)\hat{}2$
by *auto*
also have ... = $A\hat{}2*(\chi\ A\ (m+1))\hat{}2 - 2*A*(\chi\ A\ m * \chi\ A\ (m+1)) + (\chi\ A\ m)\hat{}2$
using *t* **by** *auto*
also have ... = $A*A*\chi\ A\ (m+1) * \chi\ A\ (m+1) - A*\chi\ A\ m * \chi\ A\ (m+1) - A* \chi\ A\ m * \chi\ A\ (m+1) + \chi\ A\ m * \chi\ A\ m$
by (*simp add*: *power2-eq-square*)
also have ... = $(A*\chi\ A\ (m+1)) * (A*\chi\ A\ (m+1) - \chi\ A\ m) -\chi\ A\ m * (A*\chi\ A\ (m+1) - \chi\ A\ m)$
by (*simp add*: *right-diff-distrib'*)
also have ... = $(A*\chi\ A\ (m+1) - \chi\ A\ m) * (A*\chi\ A\ (m+1) - \chi\ A\ m)$
by (*simp add*: *left-diff-distrib'*)
also have ... = $\chi\ A\ (m+2) * \chi\ A\ (m+2)$
by *auto*
also have ... = $(\chi\ A\ (m+2))\hat{}2$
by (*simp add*: *power2-eq-square*)
finally have *partie-1*: $(A^2 - 4) * (\psi\ A\ (m + 2))^2 + 4 = (\chi\ A\ (m+2))\hat{}2$.

have $(A^2 - 4) * \psi\ A\ (m + 2) * \psi\ A\ (m + 2 + 1) + 2 * A =$
$(A\hat{}2-4) * \psi\ A\ (m+2) * (A*\psi\ A\ (m+2) - \psi\ A\ (m+1)) + 2*A$
by *auto*
also have ... = $(A\hat{}2-4)*\psi\ A\ (m+2) * (A*\psi\ A\ (m+2)) - (A\hat{}2-4)* \psi\ A\ (m+2)* \psi\ A\ (m+1) + 2*A$
by (*simp add*: *right-diff-distrib'*)
also have ... = $A*(A\hat{}2-4)*\psi\ A\ (m+2)*\psi\ A\ (m+2) - ((A\hat{}2-4)* \psi\ A\ (m+2)* \psi\ A\ (m+1) + 2*A) + 4*A$
by *auto*
also have ... = $A*(A\hat{}2-4)*(\psi\ A\ (m+2))\hat{}2 - ((A\hat{}2-4)* \psi\ A\ (m+2)* \psi\ A\ (m+1) + 2*A) + A*4$

by (*simp add*: *power2-eq-square*)
also have ... = $A*((A\hat{}2-4)*(\psi\ A\ (m+2))\hat{}2 + 4) - ((A\hat{}2-4)*\psi\ A\ (m+1)*\psi\ A\ (m+2) + 2*A)$
by (*simp add*: *distrib-left*)
also have ... = $A*(\chi\ A\ (m+2))\hat{}2 - \chi\ A\ (m+1) * \chi\ A\ (m+2)$
using *t partie-1* **by** *auto*
also have ... = $\chi\ A\ (m+2)*A*\chi\ A\ (m+2) - \chi\ A\ (m+2) * \chi\ A\ (m+1)$
using *power2-eq-square* **by** *auto*
also have ... = $\chi\ A\ (m+2)*(A*\chi\ A\ (m+2) - \chi\ A\ (m+1))$
using *right-diff-distrib*[*of* $\chi\ A\ (m+2)\ A*\chi\ A\ (m+2)\ \chi\ A\ (m+1)$] **by** *auto*
also have ... = $\chi\ A\ (m+2) * \chi\ A\ (m+2+1)$ **by** *auto*
finally have $(A^2 - 4) * \psi\ A\ (m + 2) * \psi\ A\ (m + 2 + 1) + 2 * A = \chi\ A\ (m+2) * \chi\ A\ (m+2+1)$**.**
then show *?case* **using** *partie-1* **by** *simp*
qed
then show *?thesis* **by** *auto*
qed

lemma *lucas-pell-part2*:
fixes $A\ X$::*int*
shows $(\exists m.\ X = \psi\ A\ m \vee X = -\ \psi\ A\ m) \Longrightarrow (\exists k.\ (A\hat{}2-4)*X\hat{}2 + 4 = k\hat{}2)$
proof −
assume *assumption*: $(\exists m.\ X = \psi\ A\ m \vee X = -\ \psi\ A\ m)$
obtain m **where** *defm*: $X = \psi\ A\ m \vee X = -\ \psi\ A\ m$ **using** *assumption* **by** *auto*
consider (*plus*) $X = \psi\ A\ m$ | (*moins*) $X = -\psi\ A\ m$ **using** *defm* **by** *auto*
then show *?thesis*
proof *cases*
case *plus*
have $(A\hat{}2-4)*X\hat{}2 + 4 = (\chi\ A\ m)\hat{}2$
by (*simp add*: *lucas-pell-part3 plus*)
then show *?thesis* **by** *auto*
next
case *moins*
have $(A\hat{}2-4)*X\hat{}2 + 4 = (\chi\ A\ m)\hat{}2$
by (*simp add*: *lucas-pell-part3 moins*)
then show *?thesis* **by** *auto*
qed
qed

lemma *lucas-pell-nat*:
fixes $A\ Y$ :: *int*
shows $(\exists k.\ (A\hat{}2-4)*Y\hat{}2 + 4 = k\hat{}2) = (\exists m.\ Y = \psi\ A\ m \vee Y = -\ \psi\ A\ m)$
and $(A\hat{}2-4)*(\psi\ A\ m)\hat{}2 + 4 = (\chi\ A\ m)\hat{}2$
using *lucas-pell-part1 lucas-pell-part3* **by** *auto*

corollary *lucas-pell-corollary*:

fixes *A*::*int* **and** *X*::*int*
shows *is-square* ((*A* ^ *2*−*1*)∗*X* ^ *2*+*1*) = (∃ *m*. *X* = *ψ* (*2*∗*A*) *m* ∨ *X* = − *ψ* (*2*∗*A*) *m*)
proof −
have *carre-fois-4*: *is-square n* = *is-square* (*4*∗*n*) (**is** *?P1* = *?Q1*) **for** *n*::*int*
proof
assume *?P1*
then obtain *k* **where** *sq-k*: *n* = *k* ^ *2* **using** *is-square-def* **by** *auto*
have *4*∗*n* = (*2*∗*k*) ^ *2* **by** (*simp add*: *algebra-simps sq-k*)
then show *?Q1* **using** *is-square-def* **by** *blast*
next
assume *?Q1*
then obtain *k* **where** *sq-k*: *4*∗*n* = *k* ^ *2* **using** *is-square-def* **by** *auto*
then have *2 dvd k*
by (*metis even-mult-iff even-numeral power2-eq-square*)
then obtain *l* **where** *db*: *k* = *2*∗*l* **by** *auto*
then have *n* = *l* ^ *2* **using** *sq-k* **by** *force*
then show *?P1* **using** *is-square-def* **by** *auto*
qed
have (∃ *m*. *X* = *ψ* (*2*∗*A*) *m* ∨ *X* = − *ψ* (*2*∗*A*) *m*) = *is-square* (((*2*∗*A*) ^ *2*−*4*)∗*X* ^ *2* + *4*)
using *lucas-pell-part1*[*of 2*∗*A X*] *lucas-pell-part2*[*of X 2*∗*A*] **unfolding** *is-square-def* **by** *auto*
also have ... = *is-square* (*4*∗((*A* ^ *2*−*1*)∗*X* ^ *2*+*1*))
by (*simp add*: *algebra-simps*)
also have ... = *is-square* ((*A* ^ *2*−*1*)∗*X* ^ *2*+*1*)
using *carre-fois-4* **by** *blast*
finally have (∃ *m*. *X* = *ψ* (*2*∗*A*) *m* ∨ *X* = − *ψ* (*2*∗*A*) *m*) = *is-square* ((*A* ^ *2*−*1*)∗*X* ^ *2*+*1*).
then show *?thesis* **by** *simp*
qed

end
theory *Lucas-Diophantine*
imports *Lucas-Sequences*
begin

5.3 Lucas Sequences and Exponentiation

Direct implication of lemma 3.12

lemma *lucas-diophantine-dir*:
fixes *A*::*int* **and** *B*::*nat*
shows (*3* ∗ *2* ^ *B* ∗ *ψ A B*) *mod* (*2*∗*A*−*5*) = (*2* ∗ (*2* ^ (*2*∗*B*) − *1*)) *mod* (*2*∗*A* − *5*)
proof (*induction B rule*: *ψ-induct*)
case *0*
then show *?case* **by** *auto*
next
case *1*

then show *?case* **by** *auto*
next
case (*sucsuc B*)
note *t = this*
have *12∗2^B ∗ (A∗(ψ A (B+1)) − (ψ A B)) = 12∗(2^B)∗A∗(ψ A (B+1)) − 12∗(2^B)∗(ψ A B)*
by (*simp add*: *right-diff-distrib*)
then have *12∗2^B ∗ (A∗(ψ A (B+1)) − (ψ A B)) = 6∗A∗(2^(B+1))∗(ψ A (B+1)) − 12∗(2^B)∗(ψ A B)*
by *simp*
then have *e1*: *3 ∗ (2^(B+2)) ∗ (ψ A (B+2)) = 6∗A∗(2^(B+1))∗(ψ A (B+1)) − 12∗(2^B)∗(ψ A B)*
by *auto*

have *e2*: *2 ∗ A ∗ (3 ∗ 2 ^ (B + 1) ∗ ψ A (B + 1)) mod (2 ∗ A − 5) = 2 ∗ A ∗ (2 ∗ (2 ^ (2 ∗ (B+1)) − 1)) mod (2 ∗ A − 5)*
using *t mod-mult-cong*[*of 2∗A 2∗A−5 2∗A 3∗(2^(B+1))∗(ψ A (B+1)) 2∗(2^(2∗(B+1))−1)*]
by *auto*
have *6∗A∗(2^(B+1)) = 2 ∗ A ∗ (3 ∗ 2 ^ (B + 1))*
by *auto*
then have *e3*: *6∗A∗(2^(B+1))∗(ψ A (B+1)) = 2 ∗ A ∗ (3 ∗ 2 ^ (B + 1) ∗ ψ A (B + 1))*
by *auto*
have *4∗A∗(2^(2∗(B+1))−1) = 2 ∗ A ∗ (2 ∗ (2 ^ (2 ∗ (B+1)) − 1))*
by *simp*
then have *e4*: *6∗A∗(2^(B+1))∗(ψ A (B+1)) mod (2∗A−5) = 4∗A∗(2^(2∗(B+1))−1) mod (2∗A−5)*
using *e2 e3* **by** *metis*

have *3 ∗ 2^B ∗ (ψ A B) mod (2∗A−5) = 2∗(2^(2∗B)−1) mod (2∗A−5)*
using *t* **by** *auto*
have *12∗(2^B)∗(ψ A B) = 4∗ (3 ∗ 2^B ∗ (ψ A B))*
by *auto*
have *8∗(2^(2∗B)−1) = (4::int) ∗ (2∗(2^(2∗B)−1))*
by *auto*
then have *e5*: *12∗(2^B)∗(ψ A B) mod (2∗A−5) = 8∗(2^(2∗B)−1) mod (2∗A−5)*
using *t*
by (*metis* (*no-types*) ‹*12 ∗ 2 ^ B ∗ ψ A B = 4 ∗ (3 ∗ 2 ^ B ∗ ψ A B)*› *mod-mult-right-eq*)

then have *(6∗A∗(2^(B+1))∗(ψ A (B+1)) − 12∗(2^B)∗(ψ A B)) mod (2∗A−5) = (4∗A∗(2^(2∗(B+1))−1) − 8∗(2^(2∗B)−1)) mod (2∗A−5)*
using *e4 mod-diff-cong* **by** *blast*
then have *e6*: *3 ∗ (2^(B+2)) ∗ (ψ A (B+2)) mod (2∗A−5) = (4∗A∗(2^(2∗(B+1))−1) − 8∗(2^(2∗B)−1)) mod (2∗A−5)*
using *e1* **by** *auto*

have *⋀k. 4∗A∗k = (4∗A−10)∗k + 10∗k*

```
  by (metis diff-add-cancel mult.commute ring-class.ring-distribs(1))
 then have 4*A*(2^(2*(B+1))-1) = (4*A-10)*(2^(2*(B+1))-1) + 10 *(2^(2*(B+1))-1)
  by auto
 then have 4*A*(2^(2*(B+1))-1) - 8*(2^(2*B)-1)
   = (4*A-10)*(2^(2*(B+1))-1) + 10 *(2^(2*(B+1))-1) -8*(2^(2*B)-1)
  by auto
 then have e7: 3 * (2^(B+2)) * (ψ A (B+2)) mod (2*A-5)
   = ((4*A-10)*(2^(2*(B+1))-1) + 10 *(2^(2*(B+1))-1) -8*(2^(2*B)-1))
mod (2*A-5)
  using e6 by presburger

 have (4*A-10) mod (2*A-5) = 0
  by (smt (z3) minus-mod-self2 mod-self)
 then have (4*A-10)*(2^(2*(B+1))-1) mod (2*A-5) = 0
  using mod-mult-cong by auto
 then have e8: ⋀k. ((4*A-10)*(2^(2*(B+1))-1) +k) mod (2*A-5) = k mod
(2*A-5)
  by force
 have ((4*A-10)*(2^(2*(B+1))-1) + 10 *(2^(2*(B+1))-1) -8*(2^(2*B)-1))
mod (2*A-5)
     = (10 *(2^(2*(B+1))-1) -8*(2^(2*B)-1)) mod (2*A-5)
  using e8[of (10 *(2^(2*(B+1))-1) -8*(2^(2*B)-1))] by (smt (z3))
 then have e9: 3 * (2^(B+2)) * (ψ A (B+2)) mod (2*A-5)
     = (10 *(2^(2*(B+1))-1) -8*(2^(2*B)-1)) mod (2*A-5)
  using e7 by auto

  have e10: 10 *(2^(2*(B+1))-1) -8*(2^(2*B)-1) = 2*(2^(2*(B+2)) -
(1::int))
  by auto

 then show ?case using e9 e10 by auto
qed
```

A few lemmas helping variable changes in sums

```
lemma translation-var-0-to-1:
 fixes f::nat ⇒ int and n::nat
 shows (∑ i=0..n. f (i+1)) = (∑ i=1..n+1. (f i))
proof (induction n)
 case 0
 then show ?case by auto
next
 case (Suc n)
 then show ?case by auto
qed

lemma chang-var2:
 fixes f::nat ⇒ nat ⇒ int and n::nat
 shows (∑ i=0..n. f (i+1) (n-i)) = (∑ i=1..n+1. f i (n+1-i))
proof -
```

obtain *g*::*nat* ⇒ *int* **where** *g-def*: ⋀*i*. *i* ≤ *n*+*1* ⟹ *g i* = *f i* (*n*+*1*−*i*) **by** *auto*
have *i* ≤ *n* ⟹ *f* (*i*+*1*) (*n*−*i*) = *g* (*i*+*1*) **for** *i*
by (*auto simp add*: *g-def*)
then have (∑ *i*=*0*..*n*. *f* (*i*+*1*) (*n*−*i*)) = (∑ *i*=*0*..*n*. *g* (*i*+*1*))
by *auto*
also have ... = (∑ *i*=*1*..*n*+*1*. *g i*)
using *translation-var-0-to-1* **by** *auto*
also have ... = (∑ *i*=*1*..*n*+*1*. *f i* (*n*+*1*−*i*))
using *g-def* **by** *auto*
finally show *?thesis*.
qed

lemma *chang-var3*:
fixes *f*::*nat* ⇒ *nat* ⇒ *int* **and** *n*::*nat*
assumes *n*≥*1*
shows (∑ *i*=*0*..*n*−*1*. *f* (*i*+*1*) (*n*−*i*)) = (∑ *i*=*1*..*n*. *f i* (*n*+*1*−*i*))
proof −
obtain *g*::*nat* ⇒ *int* **where** *g-def*: ⋀*i*. *i* ≤ *n* ⟹ *g i* = *f i* (*n*+*1*−*i*) **by** *auto*
have *i* ≤ *n*−*1* ⟹ *f* (*i*+*1*) (*n*−*i*) = *g* (*i*+*1*) **for** *i*
using *assms* **by** (*auto simp add*: *g-def*)
then have (∑ *i*=*0*..*n*−*1*. *f* (*i*+*1*) (*n*−*i*)) = (∑ *i*=*0*..*n*−*1*. *g* (*i*+*1*))
by *auto*
also have ... = (∑ *i*=*1*..*n*. *g i*)
using *translation-var-0-to-1*[*of g n*−*1*] *assms* **by** *auto*
also have ... = (∑ *i*=*1*..*n*. *f i* (*n*+*1*−*i*))
using *g-def* **by** *auto*
finally show *?thesis*.
qed

Lemma 3.11, requiring no other result, but necessary to the proof of the reciprocal implication

definition *f-38*:: *int* ⇒ *int* ⇒ *nat* ⇒ *nat* ⇒ *int*
where *f-38 U V a b* = *U*^(*2*∗*a*)∗*V*^(*2*∗*b*)

lemma *lucas-exponential-diophantine*:
fixes *A*::*int* **and** *B*::*nat* **and** *U*::*int* **and** *V*::*int*
assumes *B*>*0*
shows (*U*∗*V*)^(*B*−*1*) ∗ *ψ A B mod* (*U*^*2* − *A*∗*U*∗*V* + *V*^*2*)
= (∑ *r*=*0*..(*B*−*1*). (*U*^(*2*∗*r*))∗(*V*^(*2*∗(*B*−*1*−*r*)))) *mod* (*U*^*2* − *A*∗*U*∗*V* + *V*^*2*)
proof −
have *f0*: *b*=*0* ∨ (*U*∗*V*)^(*b*−*1*) ∗ *ψ A b mod* (*U*^*2* − *A*∗*U*∗*V* + *V*^*2*)
= (∑ *r*=*0*..(*b*−*1*). (*U*^(*2*∗*r*))∗(*V*^(*2*∗(*b*−*1*−*r*)))) *mod* (*U*^*2* − *A*∗*U*∗*V* + *V*^*2*) **for** *b*
proof (*induction b rule*: *ψ-induct-strict*)
case *0*
show *?case* **using** *assms* **by** *auto*
case *1*
then show *?case* **by** *auto*

next
case *2*
have $(U * V)\ \hat{}\ (2 - 1) * \psi\ A\ 2\ mod\ (U^2 - A * U * V + V^2) = U*V*\psi\ A\ (1+1)\ mod\ (U\hat{}2 - A*U*V + V\hat{}2)$
by (*metis add-diff-cancel-left' one-add-one power-one-right*)
also have $... = U*V*A\ mod\ (U\hat{}2 - A*U*V + V\hat{}2)$
by *auto*
also have $... = (U\hat{}2+V\hat{}2 - (U\hat{}2 - A*U*V + V\hat{}2))\ mod\ (U\hat{}2 - A*U*V + V\hat{}2)$
by (*simp add*: *algebra-simps*)
also have $... = (U\hat{}2+V\hat{}2)\ mod\ (U\hat{}2 - A*U*V + V\hat{}2)$
using *minus-mod-self2* **by** *blast*
also have $... = (U\hat{}(2*0)*V\hat{}(2*(2-1-0)) + U\hat{}(2*1)*V\hat{}(2*(2-1-1)))\ mod\ (U\hat{}2-A*U*V+V\hat{}2)$
by (*simp add*: *algebra-simps power2-eq-square*)
also have $... = (\sum r=0..(2-1).\ (U\hat{}(2*r))*(V\hat{}(2*(2-1-r))))\ mod\ (U\hat{}2 - A*U*V + V\hat{}2)$
by *auto*
finally have $(U * V)\ \hat{}\ (2 - 1) * \psi\ A\ 2\ mod\ (U^2 - A * U * V + V^2) = (\sum r=0..(2-1).\ (U\hat{}(2*r))*(V\hat{}(2*(2-1-r))))\ mod\ (U\hat{}2 - A*U*V + V\hat{}2)$**.**
then show *?case* **by** *auto*
next
case (*sucsuc b*)
note $t = this$
have $(U * V)\ \hat{}\ (b + 2 - 1) * \psi\ A\ (b + 2)\ mod\ (U^2 - A * U * V + V^2) = (U*V)\hat{}(b+1)*(A*\psi\ A\ (b+1) - \psi\ A\ b)\ mod\ (U\hat{}2 - A*U*V + V\hat{}2)$
by *auto*
also have $... = ((A*U*V)*(U*V)\hat{}(b+1-1)*\psi\ A\ (b+1) - (U*V)\hat{}2*(U*V)\hat{}(b-1)*\psi\ A\ b)\ mod\ (U\hat{}2 - A*U*V + V\hat{}2)$
apply (*simp add*: *algebra-simps*)
by (*smt* (*verit*, *best*) *One-nat-def Suc-1 ab-semigroup-mult-class.mult-ac*(*1*) *mult.left-commute*
not-gr-zero numeral-1-eq-Suc-0 numeral-plus-numeral plus-1-eq-Suc power-Suc0-right
power-add-numeral2 power-eq-if sucsuc.hyps)
finally have *e1*: $(U * V)\ \hat{}\ (b + 2 - 1) * \psi\ A\ (b + 2)\ mod\ (U^2 - A * U * V + V^2) = ((A*U*V)*(U*V)\hat{}(b+1-1)*\psi\ A\ (b+1) - (U*V)\hat{}2*(U*V)\hat{}(b-1)*\psi\ A\ b)\ mod\ (U\hat{}2 - A*U*V + V\hat{}2)$**.**

have *e21*: $(A*U*V)*(U*V)\hat{}(b+1-1)*\psi\ A\ (b+1)\ mod\ (U\hat{}2 - A*U*V + V\hat{}2) = (A*U*V)*(\sum r=0..(b+1-1).\ (U\hat{}(2*r))*(V\hat{}(2*(b+1-1-r))))\ mod\ (U\hat{}2 - A*U*V + V\hat{}2)$
using *t mod-mult-cong*[*of A*U*V U\hat{}2-A*U*V+V\hat{}2 A*U*V (U*V)\hat{}(b+1-1)*\psi A (b+1)*
$\sum r=0..(b+1-1).\ (U\hat{}(2*r))*(V\hat{}(2*(b+1-1-r)))$]
by (*auto simp add*: *algebra-simps*)
also have $A*U*V\ mod\ (U\hat{}2-A*U*V+V\hat{}2) = (U\hat{}2+V\hat{}2)\ mod\ (U\hat{}2-A*U*V+V\hat{}2)$
by (*smt* (*z3*) *minus-mod-self2*)

```
  then have ((A*U*V)*(∑ r=0..(b+1−1). (U^(2*r))*(V^(2*(b+1−1−r)))))
mod (U^2 − A*U*V + V^2)
    = ((U^2+V^2)*(∑ r=0..(b+1−1). (U^(2*r))*(V^(2*(b+1−1−r))))) mod
(U^2 − A*U*V + V^2)
    using mod-mult-cong[of (A*U*V) (U^2 −A*U*V + V^2) (U^2 + V^2)
        (∑ r=0..(b+1−1). (U^(2*r))*(V^(2*(b+1−1−r)))) (∑ r=0..(b+1−1).
(U^(2*r))*(V^(2*(b+1−1−r))))]
    by force
  then have e22: (A*U*V)*(U*V)^(b+1−1)*ψ A (b+1) mod (U^2 − A*U*V
+V^2)
    = ((U^2+V^2)*(∑ r=0..(b+1−1). (U^(2*r))*(V^(2*(b+1−1−r))))) mod
(U^2 − A*U*V + V^2)
    using e21 by auto

  have (U*V)^2*(U*V)^(b−1)*ψ A b mod (U^2−A*U*V+V^2)
    = (U*V)^2*(∑ r=0..b−1. U^(2*r)*V^(2*(b−1−r))) mod (U^2−A*U*V+V^2)
    using t mod-mult-cong[of (U*V)^2 (U^2−A*U*V+V^2) (U*V)^2
        (U*V)^(b−1)*ψ A b (∑ r=0..b−1. U^(2*r)*V^(2*(b−1−r)))]
    by (auto simp add: algebra-simps)

  then have e2: ((A*U*V)*(U*V)^(b+1−1)*ψ A (b+1) − ((U*V)^2*(U*V)^(b−1)*ψ
A b)) mod (U^2 − A*U*V +V^2)
    = (((U^2+V^2)*(∑ r=0..(b+1−1). (U^(2*r))*(V^(2*(b+1−1−r)))))
    − (U*V)^2*(∑ r=0..b−1. U^(2*r)*V^(2*(b−1−r)))) mod (U^2 − A*U*V
+V^2)
   using mod-diff-cong[of (A*U*V)*(U*V)^(b+1−1)*ψ A (b+1) (U^2 − A*U*V
+V^2)
     ((U^2+V^2)*(∑ r=0..(b+1−1). (U^(2*r))*(V^(2*(b+1−1−r))))) ((U*V)^2*(U*V)^(b−1)*ψ
A b)
       (U*V)^2*(∑ r=0..b−1. U^(2*r)*V^(2*(b−1−r)))] e22 by auto

  have U-sq-f: a^2*f-38 a b i j = f-38 a b (i+1) j for a b i j
  proof −
    have a^2*f-38 a b i j = a^2*a^(2*i)*b^(2*j) using f-38-def by auto
    also have ... = a^(2*(i+1))*b^(2*j)
      by (metis distrib-left-numeral mult.commute nat-mult-1-right power-add)
    also have ... = f-38 a b (i+1) j using f-38-def by auto
    finally show ?thesis.
  qed
  have f-comm: f-38 a b i j = f-38 b a j i for a b i j
  proof −
    have f-38 a b i j = a^(2*i)*b^(2*j) using f-38-def by auto
    also have ... = f-38 b a j i using f-38-def by auto
    finally show ?thesis.
  qed
  have V-sq-f: b^2*f-38 a b i j = f-38 a b i (j+1) for a b i j
  proof −
    have b^2*f-38 a b i j = b^2*f-38 b a j i
      using f-comm by auto
```

also have ... = *f-38 b a (j+1) i*
using *U-sq-f* **by** *auto*
also have ... = *f-38 a b i (j+1)* **using** *f-comm* **by** *auto*
finally show *?thesis*.
qed

have *e31*: ((*U^2+V^2*)*(∑ r=0..(b+1−1). (U^(2*r))*(V^(2*(b+1−1−r)))))
= ((*U^2+V^2*)*(∑ r=0..(b+1−1). *f-38 U V r (b−r)*))
using *f-38-def* **by** *auto*
have *e32*: ... = (∑ r=0..(b+1−1). ((*U^2+V^2*)*f-38 U V r (b−r)))
using *Groups-Big.semiring-0-class.sum-distrib-left* **by** *auto*
have *e33*: ... = (∑ r=0..(b+1−1). (*U^2*f-38 U V r (b−r) + V^2*f-38 U V r (b−r)))
by (*auto simp add*: *algebra-simps*)
have *e34*: ... = (∑ r=0..(b+1−1). *U^2*f-38 U V r (b−r)) + (∑ r=0..(b+1−1). *V^2*f-38 U V r (b−r))
by (*auto simp add*: *algebra-simps Groups-Big.comm-monoid-add-class.sum.distrib*)
have *r* ≤ *b* ⟹ *Suc b − r* = *Suc* (*b−r*) **for** *r*
by *auto*
then have (∑ *r* = *0..b*. *f-38 U V r* (*Suc* (*b* − *r*))) = (∑ *r* = *0..b*. *f-38 U V r* (*Suc b* − *r*))
by *auto*
then have *e35*: (∑ r=0..(b+1−1). *U^2*f-38 U V r (b−r)) + (∑ r=0..(b+1−1). *V^2*f-38 U V r (b−r))
= (∑ r=0..(b+1−1). *f-38 U V (r+1) (b−r)*) + (∑ r=0..(b+1−1). *f-38 U V r (b+1−r)*)
by (*auto simp add*: *U-sq-f V-sq-f*)
have *e36*: ... = (∑ r=1..b+1. *f-38 U V r (b+1−r)*) + (∑ r=0..b. *f-38 U V r (b+1−r)*)
using *chang-var2* **by** *auto*
have *e37*: ... = (∑ r=1..b+1. *f-38 U V r (b+1−r)*) + (∑ r=0..b+1. *f-38 U V r (b+1−r)*)
− *f-38 U V (b+1) 0*
by *auto*
have *e310*: ... = (∑ r=1..b. *f-38 U V r (b+1−r)*) + (∑ r=0..b+1. *f-38 U V r (b+1−r)*)
by (*auto simp add*: *f-38-def algebra-simps*)

have *e3*: ((*U^2+V^2*)*(∑ r=0..(b+1−1). (U^(2*r))*(V^(2*(b+1−1−r)))))
= (∑ r=1..b. *f-38 U V r (b+1−r)*) + (∑ r=0..b+1. *f-38 U V r (b+1−r)*)
using *e31 e32 e33 e34 e35 e36 e37 e310* **by** *auto*

have *e41*: (*U*V*) ^2*(∑ r=0..b−1. *U^(2*r)*V^(2*(b−r−1))) = *U^2*V^2*(∑ r=0..b−1. *f-38 U V r (b−r−1)*)
by (*auto simp add*: *algebra-simps f-38-def*)
have *e42*: ... = (∑ r=0..b−1. *U^2*V^2*f-38 U V r (b−r−1))
using *Groups-Big.semiring-0-class.sum-distrib-left* **by** *auto*
have *r*≤ *b−1* ⟹ (*Suc*(*b−r−1*) = *b−r*) **for** *r*
using *t* **by** *auto*

then have $r \leq b-1 \Longrightarrow U\hat{}2*V\hat{}2*f\text{-}38\ U\ V\ r\ (b-r-1) = f\text{-}38\ U\ V\ (r+1)\ (b-r)$ **for** r
by (*simp add*: *U-sq-f V-sq-f ab-semigroup-mult-class.mult-ac*(1))
then have *e43*: $(\sum r{=}0..b-1.\ U\hat{}2*V\hat{}2*f\text{-}38\ U\ V\ r\ (b-r-1)) = (\sum r{=}0..b-1.\ f\text{-}38\ U\ V\ (r+1)\ (b-r))$
by *auto*
have $... = (\sum r{=}1..b.\ f\text{-}38\ U\ V\ r\ (b+1-r))$
using *chang-var3 t* **by** *auto*

then have *e4*: $(U*V)\hat{}2*(\sum r{=}0..b-1.\ U\hat{}(2*r)*V\hat{}(2*(b-r-1)))$
$= (\sum r{=}1..b.\ f\text{-}38\ U\ V\ r\ (b+1-r))$
using *e41 e42 e43* **by** *auto*

have $(((U\hat{}2+V\hat{}2)*(\sum r{=}0..(b+1-1).\ (U\hat{}(2*r))*(V\hat{}(2*(b+1-1-r)))))$
$-\ (U*V)\hat{}2*(\sum r{=}0..b-1.\ U\hat{}(2*r)*V\hat{}(2*(b-1-r)))) = (\sum r{=}0..b+1.\ f\text{-}38\ U\ V\ r\ (b+1-r))$
using *e3 e4* **by** *auto*
then have $((A*U*V)*(U*V)\hat{}(b+1-1)*\psi\ A\ (b+1) - ((U*V)\hat{}2*(U*V)\hat{}(b-1)*\psi\ A\ b))\ mod\ (U\hat{}2 - A*U*V + V\hat{}2)$
$= (\sum r{=}0..b+1.\ f\text{-}38\ U\ V\ r\ (b+1-r))\ mod\ (U\hat{}2 - A*U*V + V\hat{}2)$
using *e2* **by** *auto*
then have $(U * V)\ \hat{}\ (b + 2 - 1) * \psi\ A\ (b + 2)\ mod\ (U^2 - A * U * V + V^2)$
$= (\sum r{=}0..b+1.\ f\text{-}38\ U\ V\ r\ (b+1-r))\ mod\ (U\hat{}2 - A*U*V + V\hat{}2)$
using *e1* **by** *auto*
then have $(U * V)\ \hat{}\ (b + 2 - 1) * \psi\ A\ (b + 2)\ mod\ (U^2 - A * U * V + V^2)$
$= (\sum r{=}0..b+1.\ U\hat{}(2*r)*V\hat{}(2*(b+1-r)))\ mod\ (U\hat{}2 - A*U*V + V\hat{}2)$
using *f-38-def* **by** *auto*
then show *?case* **by** *auto*
qed
show *?thesis* **using** *f0*[*of B*] *assms* **by** *auto*
qed

corollary *lucas-diophantine-aux*:
fixes *B*::*nat* **and** *A*::*int*
assumes $B>0$
shows $2\hat{}(B-1)*\psi\ A\ B\ mod\ (2*A-5) = (\sum r{=}0..B-1.\ 2\hat{}(2*r))\ mod\ (2*A-5)$
proof −
have *f1*: $2\hat{}(B-1)*\psi\ A\ B\ mod\ (5-2*A) = (\sum r{=}0..B-1.\ 2\hat{}(2*r))\ mod\ (5-2*A)$
using *lucas-exponential-diophantine*[*of B 2 1 A*] *assms* **by** (*auto simp add*: *algebra-simps*)
have *f2*: $a\ mod\ c = b\ mod\ c \Longrightarrow a\ mod\ (-c) = b\ mod\ (-c)$ **for** *a*::*int* **and** *b*::*int* **and** *c*::*int*
by (*metis mod-minus-eq mod-minus-right*)
then show *?thesis* **using** *f2*[*of* $2\hat{}(B-1)*\psi\ A\ B\ (5-2*A)\ (\sum r{=}0..B-1.\ 2\hat{}(2*r))$] *f1* **by** *auto*
qed

Reciprocal implication of lemma 3.12

lemma *lucas-diophantine-rec*:
fixes *B*::*nat* **and** *A*::*int* **and** *W*::*int*
assumes *B>0* ∧ *abs A* > *W^4* ∧ *abs A* > *2^(4*B)* ∧ *3*W*ψ A B mod (2*A−5)* = *2*(W^2−1) mod (2*A−5)*
shows *W* = *2^B*
proof −
have *2^n* ≥ (*1*::*int*) **for** *n*::*nat*
by *simp*
have *B* ≥ *1*
using *assms* **by** *auto*
then have *4*B* ≥ *4*
by *auto*
then have *4*(B−1)* + *4* = (*4*::*nat*)**B*
by *auto*
then have *2^(4*B)* = *2^(4*(B−1))*(2*::*int*)*^4*
using *power-add*[*of 2*::*int 4*(B−1) 4*] **by** *auto*
then have *2^(4*B)* ≥ (*2*::*int*)*^4* **by** *simp*
then have *2^(4*B)* ≥ (*16*::*int*) **by** *simp*
then have *A-grand*: *abs A > 16* **using** *assms* **by** *auto*

have *W-not-0*: *W* ≠ *0*
proof (*rule ccontr*)
assume ¬ (*W* ≠ *0*)
then have *0 mod (2*A−5)* = *−2 mod (2*A−5)*
using *assms* **by** *fastforce*
then have *2*A−5 dvd −2*
by (*simp add*: *mod-eq-0-iff-dvd*)
then have *abs (2*A−5)* ≤ *2*
by (*metis abs-numeral add-eq-0-iff dbl-def dbl-simps*(*3*)
dvd-imp-le-int one-neq-neg-one uminus-dvd-conv(*2*))
then have *2* abs A − 5* ≤ *2*
by *auto*
then have *a-max*: *abs A* ≤ *3*
by *auto*
then show *False* **using** *a-max A-grand* **by** *auto*
qed
then have *abs-W-sup-1*: *abs W* ≥ *1* **by** *auto*

have *2^(B−1)*ψ A B mod (2*A−5)* = (∑ *r*=*0*..*B−1*. *2^(2*r)*) *mod (2*A−5)*
using *assms lucas-diophantine-aux* **by** *auto*
then have *e1*: *2^(B−1)*3*W*ψ A B mod (2*A−5)* = *3*W*(*∑ *r*=*0*..*B−1*. *2^(2*r)*) *mod (2*A−5)*
using *mod-mult-cong*[*of 3*W 2*A−5 3*W 2^(B−1)*ψ A B* (∑ *r*=*0*..*B−1*. *2^(2*r)*)]
by (*auto simp add*: *algebra-simps*)
have *2^(B−1)*2*(W^2−1) mod (2*A−5)* = *2^(B−1)*3*W*ψ A B mod (2*A−5)*
using *assms mod-mult-cong*[*of 2^(B−1) 2*A−5 2^(B−1) 3*W*ψ A B 2*(W^2−1)*]
by (*auto simp add*: *algebra-simps*)

then have *e2*: $3*W*(\sum r{=}0..B{-}1.\ 2\hat{}(2*r))\ mod\ (2*A{-}5) = 2\hat{}(B{-}1)*2*(W\hat{}2{-}1)\ mod\ (2*A{-}5)$
using *e1* **by** *auto*
have *somme-geo*: $(3::int)*(\sum r{=}0..n.\ 2\hat{}(2*r)) = 2\hat{}(2*(n{+}1))\ -\ 1$ **for** n
proof $(induction\ n)$
case *0*
then show *?case* **by** *auto*
next
case $(Suc\ n)$
then show *?case* **by** *auto*
qed
have $3*W*(\sum r{=}0..B{-}1.\ 2\hat{}(2*r)) = W*(2\hat{}(2*B){-}1)$
using *assms somme-geo*[*of* $B{-}1$] **by** *auto*
then have $W*(2\hat{}(2*B){-}1)\ mod\ (2*A{-}5) = 2\hat{}(B{-}1)*2*(W\hat{}2{-}1)\ mod\ (2*A{-}5)$
using *e2* **by** *presburger*
then have $W*(2\hat{}(2*B){-}1)\ mod\ (2*A{-}5) = 2\hat{}B*(W\hat{}2{-}1)\ mod\ (2*A{-}5)$
using *assms* **by** (*metis power-minus-mult*)
then have *e3*: $(W*(2\hat{}(2*B){-}1)\ -\ 2\hat{}B*(W\hat{}2{-}1))\ mod\ (2*A{-}5) = 0\ mod\ (2*A{-}5)$
using *mod-diff-cong*[*of* $2\hat{}B*(W\hat{}2{-}1)\ 2*A{-}5\ W*(2\hat{}(2*B){-}1)\ 2\hat{}B*(W\hat{}2{-}1)\ 2\hat{}B*(W\hat{}2{-}1)$]
by *auto*
then have $(2\hat{}B*(W\hat{}2{-}1) - W*(2\hat{}(2*B){-}1))\ mod\ (2*A{-}5) = 0\ mod\ (2*A{-}5)$
by *presburger*
have *e4*: $2\hat{}B*(W\hat{}2{-}1) - W*(2\hat{}(2*B){-}1) = (2\hat{}B*W{+}1)*(W{-}2\hat{}B)$
by (*auto simp add*: *algebra-simps power2-eq-square power-mult*)

have $2\hat{}(2*B){-}1 \geq (0::int)$
by *auto*
then have *i0*: $abs\ (W*(2\hat{}(2*B){-}1)) = abs\ W\ *\ (2\hat{}(2*B){-}1)$
by (*simp add*: *abs-mult-pos*)
have *i1*: $...\ =\ abs\ W\ *2\hat{}(2*B)\ -\ abs\ W$
by (*auto simp add*: *algebra-simps*)
have *i2*: $...\ = abs\ W\ *\ 2\hat{}B\ *\ 2\hat{}B\ -\ abs\ W$
by (*simp add*: *power-add*[*of* 2 B B] *mult-2*)
have *i-plus*: $(2::int)\hat{}B*2\hat{}B \geq 0 \wedge 2\hat{}B \leq (max\ (2\hat{}B)\ (abs\ W)) \wedge abs\ W \leq (max\ (2\hat{}B)\ (abs\ W))$
by *auto*
then have *i3*: $abs\ W\ *\ 2\hat{}B\ *\ 2\hat{}B \leq (max\ (2\hat{}B)\ (abs\ W))\ *\ 2\hat{}B\ *\ 2\hat{}B$
by *auto*
then have *i4*: $...\leq (max\ (2\hat{}B)\ (abs\ W))*(max\ (2\hat{}B)\ (abs\ W))*(max\ (2\hat{}B)\ (abs\ W))$
by (*auto simp add*: *mult-mono*)
then have *i5*: $abs\ (W*(2\hat{}(2*B){-}1)) \leq (max\ (2\hat{}B)\ (abs\ W))*(max\ (2\hat{}B)\ (abs\ W))*(max\ (2\hat{}B)\ (abs\ W))\ -\ abs\ W$
using *i0 i1 i2 i3 i4* **by** *linarith*

have *h0*: $2\hat{}B*(W\hat{}2{-}1) = 2\hat{}B*W\hat{}2\ -\ 2\hat{}B$

by (*auto simp add*: *algebra-simps*)
have *h1*: ... = *2^B*(abs W)*(abs W) - 2^B*
using *power2-eq-square* **by** *simp*
have *h-plus*: *(abs W)*(abs W) ≥ 0 ∧ 2^B ≤ (max (2^B) (abs W)) ∧ abs W ≤ (max (2^B) (abs W))*
∧ 0 ≤ (max (2^B) (abs W))
by *auto*
then have *h2*: *2^B*(abs W)*(abs W) ≤ (max (2^B) (abs W))*(abs W)*(abs W)*
by (*simp add*: *mult-right-mono*)
moreover have *h3*: ... ≤ *(max (2^B) (abs W))*(max (2^B) (abs W))*(max (2^B) (abs W))*
by (*simp add*: *mult-mono*)
then have *h4*: *2^B*(W^2−1) ≤ (max (2^B) (abs W))*(max (2^B) (abs W))*(max (2^B) (abs W)) - 2^B*
using *h0 h1 h2 h3* **by** *linarith*

have *h51*: *(max (2^B) (abs W))*(max (2^B) (abs W))*(max (2^B) (abs W)) = (max (2^B) (abs W))^3*
by (*simp add*: *h4 power3-eq-cube*)
have *h52*: ... = *max ((2^B)^3) ((abs W)^3)*
using *h-plus* **by** (*smt* (*verit*, *del-insts*) *power-mono zero-le-power*)
have ... = *max (2^(3*B)) ((abs W)^3)*
by (*simp add*: *mult.commute power-mult*)
then have *h53*: *(max (2^B) (abs W))*(max (2^B) (abs W))*(max (2^B) (abs W)) = max (2^(3*B)) ((abs W)^3)*
using *h51 h52* **by** *auto*
have *abs A > 2^(3*B+B)*
using *assms* **by** (*auto simp add*: *algebra-simps*)
then have *abs A > 2^(3*B) * 2^B*
using *power-add*[*of 2 3*B B*] **by** (*smt* (*verit*, *best*))
then have *h54*: *abs A > 2^(3*B)*
by (*smt* (*z3*) ‹*2 ^ (3 * B + B) < |A|*› *not-add-less1 power-less-imp-less-exp*)
have *abs A > (abs W)^4*
using *assms* **by** *auto*
then have *abs A > (abs W)^3 * abs W*
using *power-add*[*of abs W 3 1*] **by** *auto*
then have *abs A > (abs W)^3*
using *abs-W-sup-1*
by (*smt* (*verit*, *del-insts*) *one-power2 power2-eq-square*
power3-eq-cube power-commuting-commutes power-less-power-Suc)
then have *h55*: *max (2^(3*B)) ((abs W)^3) < abs A*
using *h54* **by** *auto*
then have *h5*: *(max (2^B) (abs W))*(max (2^B) (abs W))*(max (2^B) (abs W)) < abs A*
using *h53* **by** *auto*

then have *h*: *2^B*(W^2−1) < abs A - 2^B*
using *h4 h5* **by** *auto*

```
  moreover have i: abs (W*(2^(2*B)-1)) < abs A - abs W
    using i5 h5 by auto

have W-not-1: abs W ≠ 1
  proof (rule ccontr)
    assume ¬ (abs W ≠ 1)
    note t = this
    then have 2^B*(W^2-1) = 0
      by (metis cancel-comm-monoid-add-class.diff-cancel mult.commute
          mult-zero-left power2-abs power-one)
    then have j1: W*(2^(2*B)-1) mod (2*A-5) = 0 mod (2*A-5)
      using e3 by auto
    have j-20:  abs (W*(2^(2*B)-1)) < abs A -1
      using t i by auto
    have abs A -1 < abs (2*A -5)
      using A-grand by auto
    then have j2: abs (W*(2^(2*B)-1)) < abs (2*A-5)
      using j-20 by auto
    then have j3: W*(2^(2*B)-1) = 0
      using j1
      by (smt (z3) ‹(2 ^ B * (W^2 - 1) - W * (2 ^ (2 * B) - 1)) mod (2 * A
- 5) = 0 mod (2 * A - 5)›
          ‹2 ^ B * (W^2 - 1) = 0› mod-neg-neg-trivial mod-pos-pos-trivial)
    then have j4: 2^(2*B)-(1::int) = 0
      using t
        by (smt (verit) ‹16 ≤ 2 ^ (4 * B)› mult-cancel-right1 no-zero-divisors
num-double
          power2-eq-square power-mult power-mult-distrib power-mult-numeral)
    have 2*B ≥ 2 using assms by auto
    then have 2^(2*B) -1 ≥ (3::int)
      by (smt (verit, ccfv-threshold) j4
          one-power2 power2-eq-iff-nonneg power2-less-eq-zero-iff power-increasing)
    then show False using j4
     by (metis ‹1 ≤ B› abs-numeral abs-square-eq-1 add.left-neutral diff-add-cancel
              numeral-One numeral-le-iff one-le-numeral power-abs power-even-eq
power-increasing
          power-one-right semiring-norm(69))
  qed
  then have W-sup-2: abs W ≥ 2
    using W-not-0 by auto
  then have abs (W*(2^(2*B)-1)) < abs A - 2
    using i by auto
  then have i-fin: abs (W*(2^(2*B)-1)) ≤ abs A -3
    by auto

  have B≥1 using assms by auto
  then have maj-2-B: 2^B ≥ (2::int)
    by (metis  one-le-numeral power-increasing power-one-right)
  then have h-fin: 2^B*(W^2-1) < abs A - 2
```

using *h* **by** *auto*

have *2^B∗(W^2−1) ≥ 0*
by (*simp add*: *W-not-0*)
then have *abs (W∗(2^(2∗B)−1) − 2^B∗(W^2−1)) ≤ abs (W∗(2^(2∗B)−1)) + 2^B∗(W^2−1)*
by *auto*
then have *abs (W∗(2^(2∗B)−1) − 2^B∗(W^2−1)) < 2∗abs A −5*
using *i-fin h-fin* **by** *auto*
then have *W∗(2^(2∗B)−1) − 2^B∗(W^2−1) = 0* **using** *e3*
by (*smt* (*verit*) ‹*(2 ^ B ∗ (W² − 1) − W ∗ (2 ^ (2 ∗ B) − 1)) mod (2 ∗ A − 5) = 0 mod (2 ∗ A − 5)*›
mod-neg-neg-trivial mod-pos-pos-trivial)

then have *k0*: *(2^B∗W+1)∗(W−2^B) = 0* **using** *e4* **by** *auto*
have *k1*: *abs (2^B∗W+1) ≥ 2^B∗abs W −1* **by** *auto*
have *2^B∗abs W −1 ≥ 3* **using** *W-sup-2 maj-2-B*
by (*smt* (*verit*, *ccfv-SIG*) ‹*(2 ^ B ∗ W + 1) ∗ (W − 2 ^ B) = 0*› ‹*0 ≤ 2 ^ B ∗ (W² − 1)*›
h0 h1 mult-cancel-left1 mult-le-0-iff mult-right-mono one-power2 power2-sum)
then have *abs (2^B∗W+1) ≥ 3* **using** *k1* **by** *auto*
then have *2^B∗W+1 ≠ 0* **by** *auto*
then have *W−2^B = 0* **using** *k0* **by** *auto*
then show *?thesis* **by** *auto*
qed

end
theory *Lemma-4-4*
imports *Lucas-Sequences HOL.Real*
begin

5.4 Bounds on expressions involving Lucas Sequences

lemma *bernoulli-ineq*:
fixes *a*::*int* **and** *n*::*nat*
assumes *a ≥ 1*
shows *(a−1)^(Suc n) ≥ a^(Suc n) − int (n+1)∗a^n*
proof (*induction n*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc n*)
note *t = this*
have *triv*: *int (n+1) ∗ a^n ≥ 0* **using** *assms* **by** *auto*
have *(a−1)^(Suc (Suc n)) = (a−1) ∗ (a−1)^(Suc n)* **by** *auto*
hence *(a−1)^(Suc (Suc n)) ≥ (a−1) ∗ (a^(Suc n) − int (n+1)∗a^n)*
using *assms t mult-left-mono*[*of (a^(Suc n) − int (n+1)∗a^n) (a−1)^(Suc n) a−1*] **by** *auto*
hence *(a−1)^(Suc (Suc n)) ≥ a^(Suc (Suc n)) − int (Suc n +1)∗a^(Suc n) +*

```
int (n+1)*a^n
    by (auto simp add: algebra-simps)
  thus ?case using triv by force
qed

lemma lemma-4-4:
  fixes U::int and V::int and X::nat
  assumes U ≥ 2*int X and V ≥ 1 and X ≥ 1
  shows −2*int X*(V+1)^(2*X)*ψ (U^2*V) (X+1)
      ≤ U*V*(V^X * ψ (U*(V+1)) (2*X +1) − (V+1)^(2*X) * ψ (U^2*V) (X +1))
    ∧ U*V*(V^X * ψ (U*(V+1)) (2*X +1) − (V+1)^(2*X) * ψ (U^2*V) (X +1))
      ≤ 2*int X*(V+1)^(2*X)*ψ (U^2*V) (X+1)
proof −


  have minU: U ≥ 2 using assms by auto
 have min-UV: U*(V+1) > 1 using assms by (smt (verit) int-ops(2) less-1-mult of-nat-mono)
  hence Suc (Suc (2*X−1)) = 2*X+1 ∧ 2*X−1+1 = 2*X ∧ 1 < U*(V+1) using assms by auto
  hence min-2: ψ (U*(V+1)) (2*X+1) ≥ (U*(V+1)−1)^(2*X)
    using lucas-exp-growth-gt[of U*(V+1) 2*X−1] by auto
  have Suc (Suc (Suc (2*X−2))) = 2*X+1 ∧ 2*X−2+2 = 2*X using assms by auto
  hence maj-2: ψ (U*(V+1)) (2*X+1) ≤ (U*(V+1))^(2*X)
    using lucas-exp-growth-lt[of U*(V+1) 2*X−2] min-UV by auto
  have minU2V: U^2*V > 1
   using assms minU less-1-mult mult.right-neutral one-add-one power2-eq-square
       verit-comp-simplify1(3) verit-la-disequality
    by (metis less-iff-succ-less-eq)
  hence min-1: ψ (U^2*V) (X+1) ≥ (U^2*V−1)^X
    using lucas-exp-growth-gt[of U^2*V X−1] assms by auto
  have maj-1: ψ (U^2*V) (X+1) ≤ (U^2*V)^X
    using lucas-exp-growth-le[of U^2*V X−1] assms minU2V by auto
  have pos-1: ψ (U^2*V) (X+1) > 0 using lucas-strict-monotonicity[of U^2*V X] assms minU2V by auto
   have pos-2: ψ (U*(V+1)) (2*X+1) > 0 using lucas-strict-monotonicity[of U*(V+1) 2*X] min-UV assms by auto


   have ineq1: (U*(V+1)−1)^(2*X) * ψ (U^2*V) (X+1) ≤ ψ (U*(V+1)) (2*X+1) * (U^2*V)^X
    using maj-1 min-2 pos-1 pos-2 mult-mono[of (U*(V+1)−1)^(2*X)
       ψ (U*(V+1)) (2*X+1) ψ (U^2*V) (X+1) (U^2*V)^X] by auto
  have (U*(V+1)−1)^(2*X) * ψ (U^2*V) (X+1)
      ≥ ((U*(V+1))^(2*X) − 2*int X*(U*(V+1))^(2*X−1)) * ψ (U^2*V)
```

```
(X+1)
    using pos-1 bernoulli-ineq[of U*(V+1) 2*X-1] assms min-UV by auto
  hence ψ (U*(V+1)) (2*X+1) * (U^2*V)^X
      ≥ ((U*(V+1))^(2*X) - 2*int X*(U*(V+1))^(2*X-1)) * ψ (U^2*V)
(X+1)
    using ineq1 by auto
  hence ineq2: ψ (U*(V+1)) (2*X+1) * (U^2*V)^X - (U*(V+1))^(2*X) *
ψ (U^2*V) (X+1)
     ≥ - 2*int X*(U*(V+1))^(2*X-1) * ψ (U^2*V) (X+1)
    using diff-mono[of ((U*(V+1))^(2*X) - 2*int X*(U*(V+1))^(2*X-1))
* ψ (U^2*V) (X+1)
       ψ (U*(V+1)) (2*X+1) * (U^2*V)^X (U*(V+1))^(2*X) * ψ (U^2*V)
(X+1)
     (U*(V+1))^(2*X) * ψ (U^2*V) (X+1)] by (auto simp add: algebra-simps)
  have fact1: (U^2*V)^X = U^(2*X)*V^X by (simp add: power-mult power-mult-distrib)
  have fact2: U^(2*X) = U*U^(2*X-1)
    using assms semiring-normalization-rules(27)[of U 2*X-1] by auto
  hence (U^2*V)^X = U^(2*X-1)*U*V^X using fact1 by auto
  hence fact3: ψ (U*(V+1)) (2*X+1) * (U^2*V)^X = U^(2*X-1)*U*V^X
* ψ (U*(V+1)) (2*X+1)
    using assms by (auto simp add: algebra-simps)
  have (U*(V+1))^(2*X) * ψ (U^2*V) (X+1) = U^(2*X-1)*U*(V+1)^(2*X)*
ψ (U^2*V) (X+1)
    using fact2 power-mult-distrib[of U V+1 2*X] by auto
  hence fact4: ψ (U*(V+1)) (2*X+1) * (U^2*V)^X - (U*(V+1))^(2*X) *
ψ (U^2*V) (X+1) =
       U^(2*X-1)*U*(V^X * ψ (U*(V+1)) (2*X+1) - (V+1)^(2*X)* ψ
(U^2*V) (X+1))
    using fact3 by (auto simp add: algebra-simps)
  have -2*int X*(U*(V+1))^(2*X-1) * ψ (U^2*V) (X+1)
     = U^(2*X-1) * (- 2*int X * (V+1)^(2*X-1) * ψ (U^2*V) (X+1))
   using power-mult-distrib[of U V+1 2*X-1] by (auto simp add: algebra-simps)
  hence ineq3: U^(2*X-1)*U*(V^X * ψ (U*(V+1)) (2*X+1) - (V+1)^(2*X)*
ψ (U^2*V) (X+1))
     ≥ U^(2*X-1) * (- 2*int X * (V+1)^(2*X-1) * ψ (U^2*V) (X+1))
    using ineq2 fact4 by argo
  have U^(2*X-1) > 0 using assms by auto
  hence fact5: U^(2*X-1)*b ≥ U^(2*X-1)*c ⟹ b ≥ c for b and c
    by simp
  have U*(V^X * ψ (U*(V+1)) (2*X+1) - (V+1)^(2*X)* ψ (U^2*V) (X+1))
     ≥ - 2*int X * (V+1)^(2*X-1) * ψ (U^2*V) (X+1)
    using ineq3 fact5[of U*(V^X * ψ (U*(V+1)) (2*X+1) - (V+1)^(2*X)*
ψ (U^2*V) (X+1))
       - 2*int X * (V+1)^(2*X-1) * ψ (U^2*V) (X+1)] by (metis (no-types,
lifting) fact5 mult.assoc)
   hence ineq4: U*V*(V^X * ψ (U*(V+1)) (2*X+1) - (V+1)^(2*X)* ψ
(U^2*V) (X+1))
     ≥ -2*int X * V * (V+1)^(2*X-1) * ψ (U^2*V) (X+1)
    using assms mult-left-mono[of - 2*int X * (V+1)^(2*X-1) * ψ (U^2*V)
```

$(X+1)$
$U*(V\hat{}X * \psi\ (U*(V+1))\ (2*X+1) - (V+1)\hat{}(2*X)* \psi\ (U\hat{}2*V)\ (X+1))$
$V]$
by (*auto simp add*: *algebra-simps*)
have $2*int\ X * (V+1)\hat{}(2*X-1) * \psi\ (U\hat{}2*V)\ (X+1) \geq 0$ **using** *assms pos-1*
by *auto*
hence *fact6*: $-2*int\ X * V * (V+1)\hat{}(2*X-1) * \psi\ (U\hat{}2*V)\ (X+1)$
$\geq -2*int\ X * (V+1) * (V+1)\hat{}(2*X-1) * \psi\ (U\hat{}2*V)\ (X+1)$
using *assms* **by** (*auto simp add*: *algebra-simps*)
have $(V+1)*(V+1)\hat{}(2*X-1) = (V+1)\hat{}(2*X)$
using *assms semiring-normalization-rules*(27)[*of V+1 2*X−1*] **by** *auto*
hence $-2*int\ X * (V+1) * (V+1)\hat{}(2*X-1) * \psi\ (U\hat{}2*V)\ (X+1) = -2*int$
$X * (V+1)\hat{}(2*X) * \psi\ (U\hat{}2*V)\ (X+1)$
by *auto*
hence *ineq5*: $U*V*(V\hat{}X * \psi\ (U*(V+1))\ (2*X+1) - (V+1)\hat{}(2*X)* \psi$
$(U\hat{}2*V)\ (X+1))$
$\geq -2*int\ X * (V+1)\hat{}(2*X) * \psi\ (U\hat{}2*V)\ (X+1)$
using *fact6 ineq4* **by** (*auto simp add*: *algebra-simps*)

have *ineq6*: $(U\hat{}2*V-1)\hat{}X * \psi\ (U*(V+1))\ (2*X+1) \leq \psi\ (U\hat{}2*V)\ (X+1)$
$* (U*(V+1))\hat{}(2*X)$
using *mult-mono*[*of* $(U\hat{}2*V-1)\hat{}X\ \psi\ (U\hat{}2*V)\ (X+1)\ \psi\ (U*(V+1))\ (2*X+1)$
$(U*(V+1))\hat{}(2*X)$]
maj-2 min-1 pos-1 pos-2 **by** *auto*
have *fact7*: $(U\hat{}2*V+2*int\ X)*(U\hat{}2*V-1)\hat{}X \geq (U\hat{}2*V+2*int\ X)*((U\hat{}2*V)\hat{}X$
$- int\ X * (U\hat{}2*V)\hat{}(X-1))$
using *bernoulli-ineq*[*of* $U\hat{}2*V\ X-1$] *minU2V assms*(3) **by** *auto*
have $Suc\ (X-1) = X$ **using** *assms*(3) **by** *auto*
hence $(U\hat{}2*V)\hat{}X = U\hat{}2*V*(U\hat{}2*V)\hat{}(X-1)$ **by** (*metis power-Suc*)
hence $((U\hat{}2*V)\hat{}X - int\ X * (U\hat{}2*V)\hat{}(X-1)) = (U\hat{}2*V-int\ X)*(U\hat{}2*V)\hat{}(X-1)$
by (*auto simp add*: *algebra-simps*)
hence *fact8*: $(U\hat{}2*V+2*int\ X)*(U\hat{}2*V-1)\hat{}X \geq (U\hat{}2*V+2*int\ X)*(U\hat{}2*V-int$
$X)*(U\hat{}2*V)\hat{}(X-1)$
using *fact7* **by** *auto*
have *fact9*: $(U\hat{}2*V+2*int\ X)*(U\hat{}2*V-int\ X) = (U\hat{}2*V)\hat{}2 + int\ X*(U\hat{}2*V$
$- 2*int\ X)$
by (*auto simp add*: *algebra-simps power2-eq-square*)
have $U*V \geq 1$ **using** *assms*(2) *minU* **by** (*metis dual-order.strict-trans2 less-1-mult*
mult.left-neutral
mult.right-neutral not-le-imp-less not-less-iff-gr-or-eq not-numeral-less-one)
hence $U\hat{}2*V \geq 2*int\ X$ **using** *assms power2-eq-square*[*of U*] *mult-mono*[*of*
$2*int\ X\ U\ 1\ U*V$] **by** *auto*
hence $(U\hat{}2*V)\hat{}2 + int\ X*(U\hat{}2*V - 2*int\ X) \geq (U\hat{}2*V)\hat{}2$ **using** *assms*
by *auto*
hence *fact10*: $(U\hat{}2*V+2*int\ X)*(U\hat{}2*V-int\ X) \geq (U\hat{}2*V)\hat{}2$ **using** *fact9*
by *auto*
have $(U\hat{}2*V)\hat{}(X-1) \geq 0$ **using** *minU2V* **by** *auto*

hence $(U\hat{}2*V+2*int\ X)*(U\hat{}2*V-1)\hat{}X \geq (U\hat{}2*V)\hat{}2 * (U\hat{}2*V)\hat{}(X-1)$
using *fact8 fact10 mult-right-mono*[*of* $(U\hat{}2*V)\hat{}2$ $(U\hat{}2*V+2*int\ X)*(U\hat{}2*V-int\ X)$ $(U\hat{}2*V)\hat{}(X-1)$]
by *auto*
hence *ineq7*: $(U\hat{}2*V+2*int\ X)*(U\hat{}2*V-1)\hat{}X * \psi\ (U*(V+1))\ (2*X+1)$
$\geq (U\hat{}2*V)\hat{}2 * (U\hat{}2*V)\hat{}(X-1)*\psi\ (U*(V+1))\ (2*X+1)$
using *pos-2 mult-right-mono*[*of* $(U\hat{}2*V)\hat{}2 * (U\hat{}2*V)\hat{}(X-1)$ $(U\hat{}2*V+2*int\ X)*(U\hat{}2*V-1)\hat{}X\ \psi\ (U*(V+1))\ (2*X+1)$]
by *auto*
have $(U\hat{}2*V)\hat{}2*(U\hat{}2*V)\hat{}(X-1) = U\hat{}(2*X)*V\hat{}X*U\hat{}2*V$
using *power-add*[*of* $U\hat{}2*V\ 2\ X-1$] *power-add*[*of* $U\hat{}2*V\ X\ 1$] *power-mult-distrib*[*of* $U\hat{}2\ V\ X$]
power-mult[*of* $U\ 2\ X$] *assms*(*3*) **by** *auto*
hence *ineq8*: $(U\hat{}2*V+2*int\ X)*(U\hat{}2*V-1)\hat{}X * \psi\ (U*(V+1))\ (2*X+1)$
$\geq U\hat{}(2*X)*V\hat{}X * U\hat{}2*V * \psi\ (U*(V+1))\ (2*X+1)$
using *ineq7* **by** *auto*
have $U\hat{}2*V+2*int\ X \geq 0$ **using** *assms minU2V* **by** *auto*
hence $(U\hat{}2*V+2*int\ X) * \psi\ (U\hat{}2*V)\ (X+1) * (U*(V+1))\hat{}(2*X)$
$\geq (U\hat{}2*V+2*int\ X)*(U\hat{}2*V-1)\hat{}X * \psi\ (U*(V+1))\ (2*X+1)$
using *ineq6 mult-left-mono*[*of* $(U\hat{}2*V-1)\hat{}X * \psi\ (U*(V+1))\ (2*X+1)\ \psi\ (U\hat{}2*V)\ (X+1) * (U*(V+1))\hat{}(2*X)\ U\hat{}2*V+2*int\ X$]
by *auto*
hence *ineq9*: $(U\hat{}2*V+2*int\ X) * \psi\ (U\hat{}2*V)\ (X+1) * (U*(V+1))\hat{}(2*X)$
$\geq V\hat{}X * U\hat{}2*V * \psi\ (U*(V+1))\ (2*X+1)*U\hat{}(2*X)$
using *ineq8* **by** (*auto simp add*: *algebra-simps*)
have $(U*(V+1))\hat{}(2*X) = (V+1)\hat{}(2*X)*U\hat{}(2*X)$ **using** *power-mult-distrib*[*of* $U\ V+1\ 2*X$] **by** *auto*
hence *ineq10*: $(U\hat{}2*V+2*int\ X) * \psi\ (U\hat{}2*V)\ (X+1) * (V+1)\hat{}(2*X)*U\hat{}(2*X)$
$\geq V\hat{}X * U\hat{}2*V * \psi\ (U*(V+1))\ (2*X+1)*U\hat{}(2*X)$ **using** *ineq9* **by** *auto*
have $U\hat{}(2*X) > 0$ **using** *minU* **by** *auto*
hence $a*U\hat{}(2*X) \leq b*U\hat{}(2*X) \Longrightarrow a \leq b$ **for** a **and** b **by** *simp*
hence $(U\hat{}2*V+2*int\ X) * \psi\ (U\hat{}2*V)\ (X+1) * (V+1)\hat{}(2*X) \geq V\hat{}X * U\hat{}2*V * \psi\ (U*(V+1))\ (2*X+1)$
using *ineq10* **by** *auto*
hence *ineq11*: $2*int\ X * \psi\ (U\hat{}2*V)\ (X+1)\ *(V+1)\hat{}(2*X)$
$\geq U\hat{}2*V*(V\hat{}X * \psi\ (U*(V+1))\ (2*X\ +1) - (V+1)\hat{}(2*X) * \psi\ (U\hat{}2*V)\ (X\ +1))$
using *diff-mono*[*of* $V\hat{}X * U\hat{}2*V * \psi\ (U*(V+1))\ (2*X+1)\ (U\hat{}2*V+2*int\ X) * \psi\ (U\hat{}2*V)\ (X+1) * (V+1)\hat{}(2*X)$
$U\hat{}2*V*(V+1)\hat{}(2*X) * \psi\ (U\hat{}2*V)\ (X\ +1)\ U\hat{}2*V*(V+1)\hat{}(2*X) * \psi\ (U\hat{}2*V)\ (X\ +1)$]
by (*auto simp add*: *algebra-simps*)
have $2*int\ X * \psi\ (U\hat{}2*V)\ (X+1)\ *(V+1)\hat{}(2*X) \geq 0$ **using** *assms*(*3*) *pos-1* **by** *auto*
hence $2*int\ X * \psi\ (U\hat{}2*V)\ (X+1)\ *(V+1)\hat{}(2*X) \leq U*\ 2*int\ X * \psi\ (U\hat{}2*V)\ (X+1)\ *(V+1)\hat{}(2*X)$
using *minU mult-right-mono*[*of* $1\ U\ 2*int\ X * \psi\ (U\hat{}2*V)\ (X+1)\ *(V+1)\hat{}(2*X)$] **by** *auto*
hence *ineq12*: $U*\ (2*int\ X\ *(V+1)\hat{}(2*X))* \psi\ (U\hat{}2*V)\ (X+1)$

$\geq U*(U*V*(V\hat{}X * \psi\ (U*(V+1))\ (2*X+1) - (V+1)\hat{}(2*X) * \psi\ (U\hat{}2*V)\ (X+1)))$
using *ineq11 power2-eq-square*[*of U*] **by** (*auto simp add*: *algebra-simps*)
have *fact11*: $U*a \geq U*b \Longrightarrow a \geq b$ **for** *a* **and** *b* **using** *minU* **by** *auto*
have $U*V*(V\hat{}X * \psi\ (U*(V+1))\ (2*X+1) - (V+1)\hat{}(2*X) * \psi\ (U\hat{}2*V)\ (X+1))$
$\leq 2*int\ X*(V+1)\hat{}(2*X)*\psi\ (U\hat{}2*V)\ (X+1)$
using *ineq12 fact11*[*of* $U*V*(V\hat{}X * \psi\ (U*(V+1))\ (2*X+1) - (V+1)\hat{}(2*X) * \psi\ (U\hat{}2*V)\ (X+1))$
$2*int\ X*(V+1)\hat{}(2*X)*\psi\ (U\hat{}2*V)\ (X+1)$] **by** *auto*

thus *?thesis* **using** *ineq5* **by** *auto*
qed

Corollaries of lemma 3.9 easier to handle for the proof of Theorem 2

lemma *lemma-4-4-cor*:
fixes *U*::*int* **and** *V*::*int* **and** *X*::*nat*
assumes $U \geq 2*int\ X$ **and** $V \geq 1$ **and** $X \geq 1$
shows $abs\ (U*V*(V\hat{}X * \psi\ (U*(V+1))\ (2*X+1) - (V+1)\hat{}(2*X) * \psi\ (U\hat{}2*V)\ (X+1)))$
$\leq 2*int\ X*(V+1)\hat{}(2*X)*\psi\ (U\hat{}2*V)\ (X+1)$
using *assms lemma-4-4*
mult-minus-left **by** *fastforce*

This version condenses all inequalities using absolute values

lemma *lemma-4-4-abs*:
fixes *U*::*int* **and** *V*::*int* **and** *X*::*nat*
assumes $abs\ U \geq 2*int\ X$ **and** $V \geq 1$ **and** $X \geq 1$
shows $-2*int\ X*(V+1)\hat{}(2*X)*\psi\ (U\hat{}2*V)\ (X+1)$
$\leq abs\ U*V*(V\hat{}X * \psi\ (U*(V+1))\ (2*X+1) - (V+1)\hat{}(2*X) * \psi\ (U\hat{}2*V)\ (X+1))$
$\wedge\ abs\ U*V*(V\hat{}X * \psi\ (U*(V+1))\ (2*X+1) - (V+1)\hat{}(2*X) * \psi\ (U\hat{}2*V)\ (X+1))$
$\leq 2*int\ X*(V+1)\hat{}(2*X)*\psi\ (U\hat{}2*V)\ (X+1)$
proof −
have *even*: $\psi\ (abs\ U*(V+1))\ (2*X+1) = \psi\ (U*(V+1))\ (2*X+1)$
using *assms lucas-symmetry-A*[*of abs U* $*$ $(V+1)$ $2*X+1$]
by (*smt* (*z3*) *abs-zmult-eq-1 dvd-triv-left even-plus-one-iff mult-minus-left of-nat-1 of-nat-le-iff zero-le-mult-iff*)
have $-2*int\ X*(V+1)\hat{}(2*X)*\psi\ (U\hat{}2*V)\ (X+1)$
$\leq abs\ U*V*(V\hat{}X * \psi\ (abs\ U*(V+1))\ (2*X+1) - (V+1)\hat{}(2*X) * \psi\ ((abs\ U)\hat{}2*V)\ (X+1))$
$\wedge\ abs\ U*V*(V\hat{}X * \psi\ (abs\ U*(V+1))\ (2*X+1) - (V+1)\hat{}(2*X) * \psi\ ((abs\ U)\hat{}2*V)\ (X+1))$
$\leq 2*int\ X*(V+1)\hat{}(2*X)*\psi\ ((abs\ U)\hat{}2*V)\ (X+1)$
using *lemma-4-4*
[*of X abs U V*] *assms* **by** *simp*
then show *?thesis* **using** *even* **by** *simp*
qed

lemma *lemma-4-4-cor-abs*:
fixes U::*int* **and** V::*int* **and** X::*nat*
assumes *abs* $U \geq 2*int\ X$ **and** $V \geq 1$ **and** $X \geq 1$
shows *abs* $(U*V*(V\hat{}X * \psi\ (U*(V+1))\ (2*X+1) - (V+1)\hat{}(2*X) * \psi\ (U\hat{}2*V)\ (X+1)))$
$\leq 2*int\ X*(V+1)\hat{}(2*X)*\psi\ (U\hat{}2*V)\ (X+1)$
using *assms lemma-4-4-abs mult-minus-left* **by** *fastforce*

This version uses ϱ (defined in the lemma)

lemma *lemma-4-4-cor-rho*:
fixes U::*int* **and** V::*int* **and** X::*nat* **and** ϱ::*real*
assumes $U \geq 2*int\ X$ **and** $V \geq 1$ **and** $X \geq 1$
defines $\varrho \equiv (real\text{-}of\text{-}int\ (V+1)\hat{}(2*X))/(real\text{-}of\text{-}int\ V\hat{}X)$
shows *abs* $(\psi\ (U*(V+1))\ (2*X+1)/\ \psi\ (U\hat{}2*V)\ (X+1) - \varrho) \leq 2*int\ X*\varrho\ /\ (U*V)$
proof −
define $u\ v\ x$ **where** *uvx-def*: $u = real\text{-}of\text{-}int\ U\ v = real\text{-}of\text{-}int\ V\ x = real\text{-}of\text{-}nat\ X$
have $u*v*(v\hat{}X * real\text{-}of\text{-}int\ (\psi\ (U*(V+1))\ (2*X+1)) - (v+1)\hat{}(2*X) * real\text{-}of\text{-}int\ (\psi\ (U\hat{}2*V)\ (X+1)))$
$= real\text{-}of\text{-}int\ (U*V*(V\hat{}X * \psi\ (U*(V+1))\ (2*X+1) - (V+1)\hat{}(2*X) * \psi\ (U\hat{}2*V)\ (X+1)))$
using *uvx-def* **by** *auto*
hence *eq1*: *abs* $(u*v*(v\hat{}X * real\text{-}of\text{-}int\ (\psi\ (U*(V+1))\ (2*X+1))$
$- (v+1)\hat{}(2*X) * real\text{-}of\text{-}int\ (\psi\ (U\hat{}2*V)\ (X+1))))$
$= real\text{-}of\text{-}int\ (abs\ (U*V*(V\hat{}X * \psi\ (U*(V+1))\ (2*X+1) - (V+1)\hat{}(2*X) * \psi\ (U\hat{}2*V)\ (X+1))))$
by *auto*
have *eq2*: $real\text{-}of\text{-}int\ (2*int\ X*(V+1)\hat{}(2*X)*\psi\ (U\hat{}2*V)\ (X+1))$
$= 2*x*(v+1)\hat{}(2*X)*real\text{-}of\text{-}int\ (\psi\ (U\hat{}2*V)\ (X+1))$
using *uvx-def* **by** *auto*
have *abs* $(U*V*(V\hat{}X * \psi\ (U*(V+1))\ (2*X+1) - (V+1)\hat{}(2*X) * \psi\ (U\hat{}2*V)\ (X+1)))$
$\leq 2*int\ X*(V+1)\hat{}(2*X)*\psi\ (U\hat{}2*V)\ (X+1)$
using *lemma-4-4-cor assms* **by** *auto*
hence *ineq*: *abs* $(u*v*(v\hat{}X * real\text{-}of\text{-}int\ (\psi\ (U*(V+1))\ (2*X+1))$
$- (v+1)\hat{}(2*X) * real\text{-}of\text{-}int\ (\psi\ (U\hat{}2*V)\ (X+1))))$
$\leq 2*x*(v+1)\hat{}(2*X)*real\text{-}of\text{-}int\ (\psi\ (U\hat{}2*V)\ (X+1))$
using *eq1 eq2* **by** (*metis of-int-le-iff*)
have ϱ-*def*: $(v+1)\hat{}(2*X) = v\hat{}X*\varrho$ **using** *assms uvx-def* **by** *auto*
from *ineq* **have** *ineq2*: *abs* $(u*v*(v\hat{}X*real\text{-}of\text{-}int\ (\psi\ (U*(V+1))\ (2*X+1))$
$- (v\hat{}X*\varrho)*real\text{-}of\text{-}int\ (\psi\ (U\hat{}2*V)\ (X+1))))$
$\leq 2*x*(v\hat{}X*\varrho)*real\text{-}of\text{-}int\ (\psi\ (U\hat{}2*V)\ (X+1))$
using ϱ-*def* **by** *force*
have *U2VBe2*: $U\hat{}2*V \geq 2$
using *power2-eq-square*[*of U*] *mult-mono*[*of 2 U 1 U*] *mult-mono*[*of 2 U*U 1 V*] *assms* **by** *auto*
hence ψ-*pos*: $real\text{-}of\text{-}int\ (\psi\ (U\hat{}2*V)\ (X+1)) > 0$

using *assms lucas-monotone3*[*of* $U\hat{}2*V$ $X+1$] **by** *auto*
hence *fact*: *real-of-int* (ψ ($U\hat{}2*V$) ($X+1$))
$*$ (*real-of-int* (ψ ($U*(V+1)$) ($2*X+1$)) / (*real-of-int* (ψ ($U\hat{}2*V$) ($X+1$))))
$=$ *real-of-int* (ψ ($U*(V+1)$) ($2*X+1$))
by *force*
have *real-of-int* (ψ ($U\hat{}2*V$) ($X+1$))
$*$ ((*real-of-int* (ψ ($U*(V+1)$) ($2*X+1$)) / (*real-of-int* (ψ ($U\hat{}2*V$) ($X+1$))))
$-$ ϱ)
$=$ *real-of-int* (ψ ($U\hat{}2*V$) ($X+1$)) $*$ (*real-of-int* (ψ ($U*(V+1)$) ($2*X+1$)) /
(*real-of-int* (ψ ($U\hat{}2*V$) ($X+1$))))
$-$ *real-of-int* (ψ ($U\hat{}2*V$) ($X+1$))$*\varrho$
using *distrib-left*[*of real-of-int* (ψ ($U\hat{}2*V$) ($X+1$))
(*real-of-int* (ψ ($U*(V+1)$) ($2*X+1$)) / (*real-of-int* (ψ ($U\hat{}2*V$) ($X+1$))))
ϱ]
by *argo*
hence *real-of-int* (ψ ($U\hat{}2*V$) ($X+1$))
$*$ ((*real-of-int* (ψ ($U*(V+1)$) ($2*X+1$)) / (*real-of-int* (ψ ($U\hat{}2*V$) ($X+1$))))
$-$ ϱ)
$=$ *real-of-int* (ψ ($U*(V+1)$) ($2*X+1$)) $-$ *real-of-int* (ψ ($U\hat{}2*V$) ($X+1$))$*\varrho$
using *fact* **by** *auto*
hence *eq1*: $u*v*v\hat{}X*$*real-of-int* (ψ ($U\hat{}2*V$) ($X+1$))
$*$ ((*real-of-int* (ψ ($U*(V+1)$) ($2*X+1$)) / (*real-of-int* (ψ ($U\hat{}2*V$) ($X+1$))))
$-$ ϱ)
$=$ $u*v*(v\hat{}X*$*real-of-int* (ψ ($U*(V+1)$) ($2*X+1$)) $-$ $(v\hat{}X*\varrho)*$*real-of-int* (ψ
($U\hat{}2*V$) ($X+1$)))
by (*auto simp add*: *algebra-simps*)
have *ineqs-uvx*: $u > 0 \wedge v > 0 \wedge v\hat{}X > 0$ **using** *assms uvx-def* **by** *auto*
hence *abs* ($u*v*v\hat{}X*$*real-of-int* (ψ ($U\hat{}2*V$) ($X+1$))
$*$ ((*real-of-int* (ψ ($U*(V+1)$) ($2*X+1$)) / (*real-of-int* (ψ ($U\hat{}2*V$) ($X+1$))))
$-$ ϱ))
$=$ $u*v*v\hat{}X*$*real-of-int* (ψ ($U\hat{}2*V$) ($X+1$))
$*$ *abs* ((*real-of-int* (ψ ($U*(V+1)$) ($2*X+1$)) / (*real-of-int* (ψ ($U\hat{}2*V$) ($X+1$))))
$-$ ϱ)
using ψ*-pos* **by** (*auto simp add*: *abs-mult*)
hence $u*v*v\hat{}X*$*real-of-int* (ψ ($U\hat{}2*V$) ($X+1$))
$*$ *abs* ((*real-of-int* (ψ ($U*(V+1)$) ($2*X+1$)) / (*real-of-int* (ψ ($U\hat{}2*V$) ($X+1$))))
$-$ ϱ)
$\leq$ $2*x*(v\hat{}X*\varrho)*$*real-of-int* (ψ ($U\hat{}2*V$) ($X+1$))
using *eq1 ineq2* **by** *argo*
hence *abs* ((*real-of-int* (ψ ($U*(V+1)$) ($2*X+1$)) / (*real-of-int* (ψ ($U\hat{}2*V$)
($X+1$)))) $-$ ϱ)
$\leq$ $2*x*\varrho/(u*v)$
using *ineqs-uvx* ψ*-pos divide-right-mono*[*of* $u*v*v\hat{}X*$*real-of-int* (ψ ($U\hat{}2*V$)
($X+1$))
$*$ *abs* ((*real-of-int* (ψ ($U*(V+1)$) ($2*X+1$)) / (*real-of-int* (ψ ($U\hat{}2*V$) ($X+1$))))
$-$ ϱ)
$2*x*(v\hat{}X*\varrho)*$*real-of-int* (ψ ($U\hat{}2*V$) ($X+1$)) $u*v*v\hat{}X*$*real-of-int* (ψ
($U\hat{}2*V$) ($X+1$))] **by** *auto*
then show *?thesis* **using** *uvx-def* **by** *auto*

qed

This version condenses all inequalities using absolute values, and uses ϱ

lemma *lemma-4-4-cor-rho-abs*:
fixes U::*int* **and** V::*int* **and** X::*nat* **and** ϱ::*real*
assumes *abs* $U \geq 2*int\ X$ **and** $V \geq 1$ **and** $X \geq 1$
assumes $\varrho \equiv$ (*real-of-int* $(V+1)\hat{}(2*X))/$(*real-of-int* $V\hat{}X$)
shows *abs* $(\psi\ (U*(V+1))\ (2*X\ +1)/\ \psi\ (U\hat{}2*V)\ (X+1)\ -\ \varrho) \leq 2*int\ X*\varrho$ / $(abs\ U*V)$
proof −
have *even*: $\psi\ (abs\ U*(V+1))\ (2*X\ +1) = \psi\ (U*(V+1))\ (2*X\ +1)$
using *assms lucas-symmetry-A*[*of abs* $U\ *\ (V+1)\ 2*X+1$]
by (*smt* (*verit*, *best*) *dvd-triv-left even-plus-one-iff mult-minus-left zero-less-mult-iff zmult-eq-1-iff*)
have *abs* $(\psi\ (abs\ U*(V+1))\ (2*X\ +1)/\ \psi\ (U\hat{}2*V)\ (X\ +1)\ -\ \varrho) \leq 2*int\ X*\varrho$ / $(abs\ U*V)$
using *lemma-4-4-cor-rho assms* **by** *fastforce*
then show *?thesis* **using** *even* **by** *simp*
qed

end
theory *DFI-square-0*
imports *Pell-Equation*
begin

5.5 Square Criterion for Exponentiation

locale *bridge-variables*
begin

definition *D-f*:: *int* ⇒ *int* ⇒ *int* **where**
D-f A C = $(A\hat{}2\ -\ 4) * C\hat{}2\ +\ 4$

definition *E-f*::*int* ⇒ *int* ⇒ *int* ⇒ *int* **where**
E-f C D x = $C\hat{}2 * D * x$

definition *F-f*:: *int* ⇒ *int* ⇒ *int* **where**
F-f A E = $4 * (A\hat{}2\ -\ 4) * E\hat{}2\ +1$

definition *G-f*:: *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* **where**
G-f A C D E F = $1\ +\ C * D * F\ -\ 2 * (A\ +\ 2) * (A-2)\hat{}2 * E\hat{}2$

definition *H-f*:: *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* **where**
H-f B C F y = $C\ +\ B * F\ +\ (2*y\ -\ 1) * C * F$

definition *I-f*:: *int* ⇒ *int* ⇒ *int* **where**
I-f G H = $(G\hat{}2\ -\ 1) * H\hat{}2\ +\ 1$

definition *E-ACx*:: *int* ⇒ *int* ⇒ *int* ⇒ *int* **where**
E-ACx A C x = *E-f C* (*D-f A C*) *x*

definition *F-ACx*:: *int* ⇒ *int* ⇒ *int* ⇒ *int* **where**
F-ACx A C x = *F-f A* (*E-ACx A C x*)

definition *G-ACx*:: *int* ⇒ *int* ⇒ *int* ⇒ *int* **where**
G-ACx A C x = *G-f A C* (*D-f A C*) (*E-ACx A C x*) (*F-ACx A C x*)

definition *H-ABCxy*:: *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* **where**
H-ABCxy A B C x y = *H-f B C* (*F-ACx A C x*) *y*

definition *I-ABCxy*:: *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* **where**
I-ABCxy A B C x y = *I-f* (*G-ACx A C x*) (*H-ABCxy A B C x y*)

lemma *lemma-4-2-part-DF*:
fixes *A B*
defines *C* ≡ *ψ A* (*nat B*)
assumes *evA*: *even A A*≥*4 B*≥*3*
shows ∀ *n*. ∃ *x*≥*n*. *is-square* (*D-f A C*) ∧ *is-square* (*F-ACx A C x*)
proof
fix *n*

have *D-square*: *is-square* (*D-f A C*) **using** *C-def lucas-pell-part2*[*of C A*]
unfolding *D-f-def is-square-def* **by** *auto*

define *D* **where** *D* ≡ *D-f A C*
have *C-pos*: *C* > *0* **unfolding** *C-def* **using** *lucas-strict-monotonicity*[*of A nat*(*B*)−*1*] *assms* **by** *auto*
then have (*A* ^ *2* − *4*) ∗ *C* ^ *2*≥*0* **using** *assms* **by** *auto*
then have *D-pos*: *D* > *0* **unfolding** *D-def D-f-def* **by** *auto*
have *CD-pos*: *4* ∗ *C* ^ *2* ∗ *D* ≥ *1* **using** *C-pos D-pos*
by (*metis add.left-neutral add-diff-cancel-right*′ *linorder-not-less zero-less-mult-iff zero-less-numeral zero-less-power zle-diff1-eq*)
have ∃ *q*≥*2* + *nat*(*n* ∗ (*4* ∗ *C* ^ *2* ∗ *D*)). *ψ A q mod* (*4* ∗ *C* ^ *2* ∗ *D*) = *0*
using *lucas-modN C-pos CD-pos int-one-le-iff-zero-less* **by** *blast*
then obtain *q* **where** *prop-q*: *q*≥*2* + *nat*(*n* ∗ (*4* ∗ *C* ^ *2* ∗ *D*)) ∧ *ψ A q mod* (*4* ∗ *C* ^ *2* ∗ *D*) = *0* **by** *presburger*
then obtain *x* **where** *prop-x*: *ψ A q* = *x* ∗ (*4* ∗ *C* ^ *2* ∗ *D*)
by (*metis zmod-eq-0-iff mult.commute*)
define *E* **where** *E* = *E-f C D x*
define *F* **where** *F* = *F-f A E*
then have *4* ∗ *F* = (*A*^{2} − *4*) ∗ *4* ^ *2* ∗ (*C*^{2} ∗ *D* ∗ *x*)^2 + *4* **unfolding** *F-f-def E-def E-f-def* **by** *auto*
also have ... = (*A*^{2} − *4*) ∗ (*4* ∗ *C*^{2} ∗ *D* ∗ *x*)^2 + *4* **by** (*simp add*: *power-mult-distrib*)
finally have *4* ∗ *F* = (*A* ^ *2* − *4*) ∗ (*ψ A q*) ^ *2* +*4* **using** *prop-x* **by** (*simp add*:

mult.commute)
then have *is-square* (*4* ∗ *F*) **using** *lucas-pell-part2*[*of* ψ *A q A*] **unfolding** *is-square-def* **by** *auto*
then obtain *k* **where** *prop-k*: *4* ∗ *F* = *k*^*2* **unfolding** *is-square-def* **by** *auto*
then have *even* (*k*^*2*) **by** (*metis dvd-mult2 even-numeral*)
then have *even k* **by** *simp*
then obtain *l* **where** *k* = *2* ∗ *l* **by** *auto*
then have *F* = *l*^*2* **using** *prop-k* **by** *simp*
then have *F-square*: *is-square F* **unfolding** *is-square-def* **by** *auto*

have *q* ≥ *2* **using** *prop-q* **by** *auto*
then have ψ *A q* ≥ ψ *2 q* **using** *lucas-monotone-A* ‹*A*≥*4*› **by** *simp*
then have *q-minor*: *...* ≥ *int*(*q*) **using** *lucas-A-eq-2*[*of q*] **by** *simp*
then have *...* ≥ *int*(*nat*(*n* ∗ (*4* ∗ *C*^*2* ∗ *D*))) **using** *prop-q* **by** *auto*
then have *x-minor-1*: *...* ≥ *n* ∗ (*4* ∗ *C*^*2* ∗ *D*) **by** *linarith*
then have *x-minor*: *x* ≥ *n* **using** *prop-x*
proof (*cases n*≥*0*)
case *True*
then have *x* ∗ (*4* ∗ *C*^*2* ∗ *D*) ≥ *n* ∗ (*4* ∗ *C*^*2* ∗ *D*) **using** *x-minor-1 prop-x* **by** *auto*
then show *?thesis* **using** *CD-pos mult-right-le-imp-le* **by** *fastforce*
next
case *False*
have ψ *A q* ≥ *0* **using** *q-minor* **by** *simp*
then have *x* ≥ *0* **using** *prop-x C-pos D-pos*
by (*metis mult-less-cancel-right not-less not-numeral-le-zero power2-eq-square times-int-code*(*2*))
then show *?thesis* **using** ‹¬ *n*≥*0*› **by** *simp*
qed

have *F* = *F-ACx A C x* **using** *F-def E-def D-def* **unfolding** *F-ACx-def E-ACx-def* **by** *simp*
then show ∃ *x*≥*n*. *is-square* (*D-f A C*) ∧ *is-square* (*F-ACx A C x*) **using** *D-square F-square x-minor F-def E-def*
by *auto*
qed

definition *y-num-ABCx* :: *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* **where**
y-num-ABCx A B C x = ψ (*2*∗(*G-ACx A C x*)) (*nat B*) − *C* + (*C*−*B*) ∗ *F-ACx A C x*

definition *y-den-ABCx* :: *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* **where**
y-den-ABCx A B C x = *2* ∗ *C* ∗ *F-ACx A C x*

lemma *lemma-4-2-y-grows*:
fixes *A B*
defines *C* ≡ ψ *A* (*nat B*)
defines *y-num* ≡ *y-num-ABCx A B C* **and** *y-den* ≡ *y-den-ABCx A B C*
assumes *evA*: *even A A*≥*4 B*≥*3*

shows $\forall m.\ \exists n.\ \forall x.\ x{\geq}n \longrightarrow y\text{-}num\ x \geq m * y\text{-}den\ x \wedge y\text{-}den\ x > 0$
proof
have *easy-part*: $\forall x.\ y\text{-}den\ x > 0$
proof
fix x::*int*
have *A-min*: $A\hat{}2{-}4 \geq 0$ **using** *assms* **by** *auto*
have *C-min*: $C > 0$ **unfolding** *C-def* **using** *lucas-monotone1*[*of A nat B*] *assms* **by** *auto*
show *y-den x* > *0*
unfolding *y-den-def y-den-ABCx-def F-ACx-def F-f-def E-ACx-def E-f-def D-f-def*
using *A-min C-min* **by** *fastforce*
qed

fix *m*
define *n* **where** *n-def*: $n = C + 1 + 16{*}abs\ m{*}C{*}(A\hat{}2{-}4)$
have $\forall x.\ x{\geq}n \longrightarrow y\text{-}num\ x \geq m * y\text{-}den\ x \wedge y\text{-}den\ x > 0$
proof *safe*
fix *x* **assume** *x-def*: $x{\geq}n$
define *D* **where** $D = D\text{-}f\ A\ C$
define *E* **where** $E = E\text{-}ACx\ A\ C\ x$
define *F* **where** $F = F\text{-}ACx\ A\ C\ x$
define *G* **where** $G = G\text{-}ACx\ A\ C\ x$
have *C-pos*: $C > 0$ **unfolding** *C-def* **using** *lucas-monotone1*[*of A nat B*] *assms* **by** *auto*
have *A-min*: $A\hat{}2{-}4 \geq 1$ **using** ‹$A \geq 4$›
by (*smt* (*verit*) *one-less-numeral-iff power-one-right power-strict-increasing-iff semiring-norm*(*76*))
then have *D-min*: $D > 0$ **unfolding** *D-def D-f-def* **using** *C-pos*
by (*smt* (*verit*, *ccfv-SIG*) *int-distrib*(*2*) *one-le-power zmult-zless-mono2*)
have *n-min*: $n > 0$ **using** *n-def C-pos assms A-min*
by (*smt* (*verit*, *ccfv-SIG*) *int-distrib*(*3*) *int-distrib*(*4*) *zmult-zless-mono2*)
then have *E-min0*: $E \geq n$
unfolding *E-def E-ACx-def E-f-def* **using** *D-def D-min C-pos x-def*
by (*smt* (*verit*) *dvdI mult.commute mult-pos-pos zdvd-imp-le zero-less-power*)
then have *E-min1*: $E > 0$ **using** *n-min* **by** *auto*
have *n-min1*: $n \geq C{+}1$ **unfolding** *n-def* **using** *A-min C-pos* **by** *auto*
have *n-min2*: $n \geq 16{*}abs\ m * C * (A\hat{}2{-}4)$ **unfolding** *n-def* **using** *C-pos* **by** *auto*

define *g* **where** $g = 2{*}(A\hat{}2{-}4){*}(2{*}C{*}D{-}(A{-}2))$
have *C-min*: $C \geq A$ **using** *lucas-monotone1*[*of A nat B*] *C-def assms* **by** *auto*
then have $2{*}C{*}D - (A{-}2) \geq 1$ **using** *D-min*
by (*smt* (*verit*) *C-pos dvdI mult-pos-pos zdvd-imp-le*)
then have *g-min*: $g > 0$ **unfolding** *g-def* **using** *A-min* **by** *force*
have $G = 1 + C{*}D{*}(4{*}(A\hat{}2{-}4){*}E\hat{}2{+}1) - 2{*}(A + 2){*}(A - 2)\hat{}2{*}E\hat{}2$
unfolding *G-def G-ACx-def G-f-def F-ACx-def F-f-def D-def E-def* **by** *auto*
also have $\ldots = 1 + C{*}D + 2{*}(A\hat{}2{-}4){*}(2{*}C{*}D - (A{-}2)){*}E\hat{}2$
by (*auto simp add*: *algebra-simps power2-eq-square*)

also have ... = *1 + C∗D + g∗E^2* **unfolding** *g-def* **by** *auto*
also have ... ≥ *g∗E^2* **using** *C-pos D-min* **by** *auto*
finally have *G-min*: *G ≥ g∗E^2*.
then have *G-min2*: *G ≥ 1* **using** *g-min E-min1*
by (*smt* (*z3*) *mult-le-0-iff one-le-power*)

have *triv1*: *3 + (nat B − 3) = nat B* **using** *assms* **by** *auto*
have *C-min2*: *C ≥ B* **using** *lucas-monotone3*[*of A nat B*] *C-def assms* **by** *auto*

have *ψ-3*: *ψ k 3 = k^2 − 1* **for** *k*
by (*simp add*: *numeral-3-eq-3 power2-eq-square*)

have *fact1*: *y-num x = ψ (2∗G) (nat B) − C + (C−B)∗F*
unfolding *y-num-def y-num-ABCx-def G-def F-def* **by** *auto*
also have *fact2*: ... ≥ *ψ (2∗G) 3 − C + (C−B)∗F*
using *lucas-monotone2*[*of 2∗G 3 nat B − 3*] *triv1 G-min2* **by** *auto*
also have *fact3*: *ψ (2∗G) 3 − C + (C−B)∗F = (2∗G)^2 −1 − C + (C−B)∗F*
using *ψ-3*[*of 2∗G*] **by** *auto*
also have ... ≥ *4∗g^2∗(E^2)^2 − 1 − C + (C−B)∗F*
using *G-min* **apply** (*simp add*: *power2-eq-square algebra-simps*)
using *E-min1 g-min G-min2*
by (*smt* (*verit*) *mult.commute mult.left-commute mult-mono' mult-nonneg-nonneg*)
then have *step1*: *y-num x ≥ 4∗g^2∗(E^2)^2 − 1 − C + (C−B)∗F* **using** *fact1 fact2 fact3* **by** *auto*
have *fact4*: *4∗g^2∗(E^2)^2 − 1 − C + (C−B)∗F ≥ 4∗(E^2)^2 − 1 − C + (C−B)∗F*
using *g-min* **apply** *simp*
by (*meson E-min1 dvd-triv-right mult-pos-pos zdvd-imp-le zero-less-power*)
have *F > 0* **unfolding** *F-def F-ACx-def F-f-def* **using** *assms* **by** *auto*
then have *fact5*: *4∗(E^2)^2 − 1 − C + (C−B)∗F ≥ 4∗(E^2)^2 − 1 − C* **using** *C-min2* **by** *auto*
have *(E^2)^2 ≥ E^2* **using** *power2-eq-square*[*of E^2*]
by (*smt* (*z3*) *one-le-numeral power2-less-eq-zero-iff power-increasing power-one-right*)
then have *(E^2)^2 ≥ E* **using** *power2-eq-square E-min1*
by (*smt* (*verit*, *ccfv-SIG*) *power2-le-imp-le zero-le-power2*)
then have *(E^2)^2 ≥ n* **using** *E-min0* **by** *auto*
then have *(E^2)^2 ≥ C+1* **using** *n-min1* **by** *auto*
then have *4∗(E^2)^2 − 1 − C ≥ 3∗(E^2)^2* **by** *auto*
then have *step2*: *y-num x ≥ 3∗(E^2)^2* **using** *step1 fact4 fact5* **by** *auto*

have *fact6*: *4∗(A^2−4)∗E^2 ≥ 1* **using** *A-min E-min1*
by (*smt* (*verit*) *mult-pos-pos power2-less-eq-zero-iff*)
have *triv2*: *k > 0 ⟹ k ≤ l ⟹ abs m ∗ k ≤ abs m ∗ l* **for** *k l*::*int*
by (*simp add*: *mult.commute mult-le-cancel-right*)
have *fact7*: *m ∗ y-den x ≤ abs m ∗ y-den x* **using** *easy-part* **by** *auto*
have *y-den x = 2∗C∗(4∗(A^2−4)∗E^2 + 1)* **unfolding** *y-den-def y-den-ABCx-def F-ACx-def F-f-def E-def* **by** *simp*
then have *y-den x ≤ 2∗C∗(4∗(A^2−4)∗E^2 + 4∗(A^2−4)∗E^2)*
using *C-pos fact6* **by** (*smt* (*verit*, *ccfv-SIG*) *zmult-zless-mono2*)

then have *step3*: *y-den x* $\leq$ *16 * C * (A^2−4) * E^2*
by (*auto simp add*: *algebra-simps*)
then have *abs m * y-den x* $\leq$ *abs m * (16 * C * (A^2 − 4) * E^2)*
using *easy-part triv2*[*of y-den x* (*16 * C * (A^2 − 4) * E^2*)] **by** *auto*
then have *abs m * y-den x* $\leq$ (*16 * abs m * C * (A^2−4*)) * *E^2* **by** *auto*
then have *abs m * y-den x* $\leq$ *n*E^2* **using** *n-min2 E-min1*
by (*smt* (*verit*, *ccfv-SIG*) *mult-right-less-imp-less not-sum-power2-lt-zero*)
then have *m * y-den x* $\leq$ *n * E^2* **using** *fact7* **by** *auto*
then have *m*y-den x* $\leq$ *E* E^2* **using** *E-min0 E-min1*
by (*smt* (*verit*) *mult-right-less-imp-less zero-le-power*)
then have *m*y-den x* $\leq$*3*(E^2)^2* **using** *E-min1 power2-eq-square*[*of E^2*]
by (*smt* (*verit*) ‹$E^2 \leq (E^2)^2$› *mult-right-mono power-eq-0-iff power-strict-mono zero-power2*)
then have *m*y-den x* $\leq$ *y-num x* **using** *step2* **by** *auto*
then show *y-num x* $\geq$ *m*y-den x* **using** *easy-part* **by** *auto*
next
show $\bigwedge x.\ n \leq x \Longrightarrow 0 <$ *y-den x* **using** *easy-part* **by** *auto*
qed
then show $\exists\, n.\ \forall\, x{\geq}n.\ m *$ *y-den x* $\leq$ *y-num x* $\wedge\ 0 <$ *y-den x* **by** *auto*
qed

lemma *mod-mult*:
fixes *a*::*int* **and** *b*::*int* **and** *c*::*int* **and** *d*::*int*
assumes *a mod c = b mod c* $\wedge$ *a mod d = b mod d* $\wedge$ *coprime c d*
shows *a mod* (*c*d*) = *b mod* (*c*d*)
proof −
have *c dvd* (*b*−*a*) $\wedge$ *d dvd* (*b*−*a*) **using** *assms* **by** (*metis mod-eq-dvd-iff*)
then have (*c*d*) *dvd* (*b*−*a*) **using** *assms* **by** (*simp add*: *divides-mult*)
then show *?thesis* **using** *mod-eq-dvd-iff* **by** *metis*
qed

lemma *lemma-4-2-y-int*:
fixes *A B x*
defines *C* $\equiv$ ψ *A* (*nat B*)
defines *y-num* $\equiv$ *y-num-ABCx A B C* **and** *y-den* $\equiv$ *y-den-ABCx A B C*
assumes *evA*: *even A A*$\geq$*4 B*$\geq$*3*
shows *y-den x dvd y-num x*
proof −
define *D* **where** *D* $\equiv$ *D-f A C*
define *E* **where** *E* = *E-f C D x*
define *F* **where** *F* = *F-f A E*
define *G* **where** *G* = *G-f A C D E F*
define *H0* **where** *H0* = ψ (*2*G*) (*nat B*)

have *minG*: *G* $\geq$ *1+A*
proof −

have *min-A*: *A^2−4≥ 0* **using** *assms* **by** *auto*
then have *minD*: *D ≥ 1* **using** *D-def D-f-def* **by** *simp*
have *minF*: *F ≥ 1* **using** *F-def F-f-def min-A* **by** *auto*
have *a1*: *G = 1 + C∗D∗F − 2∗(A+2)∗(A−2)^2∗E^2* **using** *G-def G-f-def*
by *auto*
have *a2*: *... ≥ 1 + A∗D∗F − 2∗(A+2)∗(A−2)^2∗E^2*
using *assms lucas-monotone1*[*of A nat B*] *minD minF* **by** *auto*
have *a3*: *1 + A∗D∗F − 2∗(A+2)∗(A−2)^2∗E^2 ≥ 1 + A∗F − 2∗(A+2)∗(A−2)^2∗E^2*
using *minD minF evA*(*2*) **by** *fastforce*
have *a4*: *1 + A∗F − 2∗(A+2)∗(A−2)^2∗E^2 = 1 + A∗(4∗(A^2−4)∗E^2+1)*
− 2∗(A+2)∗(A−2)^2∗E^2
using *F-def F-f-def*[*of A E*] **by** *auto*
have *a5*: *1 + A∗(4∗(A^2−4)∗E^2+1) − 2∗(A+2)∗(A−2)^2∗E^2 = 1 + A*
+ 4∗A∗(A^2−4)∗E^2 − 2∗(A+2)∗(A−2)^2∗E^2
by (*auto simp add*: *algebra-simps*)
have *a6*: *1+A + 4∗A∗(A^2−4)∗E^2 − 2∗(A+2)∗(A−2)^2∗E^2 ≥ 1+A +*
2∗A∗(A^2−4)∗E^2 − 2∗(A+2)∗(A−2)^2∗E^2
using *assms min-A* **by** *auto*
have *a7*: *1+A + 2∗A∗(A^2−4)∗E^2 − 2∗(A+2)∗(A−2)^2∗E^2 ≥ 1+A +*
2∗(A−2)∗(A^2−4)∗E^2 − 2∗(A+2)∗(A−2)^2∗E^2
using *min-A* **by** (*smt* (*z3*) *mult-right-mono zero-le-power2*)
have *a8*: *1+A + 2∗(A−2)∗(A^2−4)∗E^2 − 2∗(A+2)∗(A−2)^2∗E^2 =*
1+A
by (*auto simp add*: *algebra-simps power2-eq-square*)
show *?thesis*
using *a1 a2 a3 a4 a5 a6 a7 a8* **by** *linarith*
qed

have *H0 mod* (*2∗G−2*)= *ψ* (*2∗G*) (*nat B*) *mod* (*2∗G−2*) **using** *H0-def* **by**
simp
also have *... = B mod* (*2∗G−2*) **using** *lucas-congruence2*[*of 2∗G nat B*] *assms*
minG
dvd-refl even-nat-iff even-numeral **by** *auto*
finally have *H0-1*: *H0 mod* (*2∗G−2*) = *B mod* (*2∗G−2*)**.**

have *H0 mod* (*2∗G−A*) = *ψ* (*2∗G*) (*nat B*) *mod* (*2∗G−A*) **using** *H0-def* **by**
simp
also have *... = C mod* (*2∗G−A*) **using** *C-def lucas-congruence*[*of 2∗G−A 2∗G*
A nat B] *minG*
by (*smt* (*z3*) *evA*(*2*) *mod-add-self2*)
finally have *H0-2*: *H0 mod* (*2∗G−A*) = *C mod* (*2∗G−A*)**.**

have *2∗G−A = 2∗(1 + C∗D∗F − 2∗(A+2)∗(A−2)^2∗E^2)−A* **using** *G-def*
G-f-def **by** *auto*
also have *... = 2∗(1+D∗C∗F) − 4∗(A^2−4)∗E^2∗(A−2) − A*
by (*auto simp add*: *algebra-simps power2-eq-square*)
also have *... = 2∗(1+D∗C∗F) − (F−1)∗(A−2) − A*
proof −
have *F−1 = 4∗(A^2−4)∗E^2* **using** *F-def F-f-def* **by** *auto*

```
  then show ?thesis by auto
 qed
 also have ... = F*(2*D*C - A + 2)
  by (auto simp add: algebra-simps)
 finally have G-moins: 2*G-A = F*(2*D*C - A + 2).

 have H0 mod F = C mod F
  using G-moins H0-2 by (metis dvd-triv-left mod-mod-cancel)
 also have ... = (C + B*F - C*F) mod F
  by (smt (verit) mod-mult-self3)
 finally have H0-3: H0 mod F = (C + B*F - C*F) mod F.

 have C-div-E: C dvd E using E-def E-f-def by simp
 then have C-div-E: 2*C dvd 2*E by auto
  have (2*G-2) mod (2*C) = 2*(C*D*F - 2*(A+2)*(A-2)^2*E^2) mod
(2*C)
   using G-def G-f-def[of A C D E F] by auto
 also have ... = (2*C*(D*F)+(-(A+2)*(A-2)^2*(2*E)*2*E)) mod (2*C)
  by (auto simp add: algebra-simps power2-eq-square)
 also have ... =-(A+2)*(A-2)^2*(2*E)*2*E mod (2*C)
  using mod-mult-self4[of 2*C D*F -(A+2)*(A-2)^2*(2*E)*2*E] by auto
 also have ... = 0 mod (2*C) using C-div-E by fastforce
 finally have (2*G-2) mod (2*C) = 0 mod (2*C).
 then have C-and-G: 2*C dvd (2*G-2) by auto

 have evCB: even (C-B) using lucas-parity2[of A nat B] C-def assms by auto
 have C-div-F: 2*C dvd (F-1)
 proof -
  have F-1 = 4* (A^2-4)*E*E unfolding F-def F-f-def using power2-eq-square[of
E] by simp
  then show ?thesis using C-div-E by auto
 qed
 have H0 mod (2*C) = B mod (2*C)
  using C-and-G H0-1 by (metis mod-mod-cancel)
 also have ... = (B + (B-C)*(F-1)) mod (2*C)
  using C-div-F by (metis dvdE dvd-mult mod-mult-self2)
 also have ... = (C + B*F - C*F) mod (2*C)
  by (auto simp add: algebra-simps)
 finally have H0-4: H0 mod (2*C) = (C + B*F - C*F) mod (2*C).

 have coprime F (2*C)
 proof -
  obtain k where k-def: F-1 = k*2*C using C-div-F by (metis dvdE mult.assoc
mult.commute)
  have 1 = F - (2*C)*k using k-def by (auto simp add: algebra-simps)
   then show ?thesis by (meson C-div-F coprime-def coprime-doff-one-right
dvd-trans)
 qed
```

then have *H0 mod* $(2*C*F) = (C + B*F - C*F)$ *mod* $(2*C*F)$
using *H0-4 H0-3 mod-mult*[*of H0 F C* + *B*F* − *C*F 2*C*] **by** (*auto simp add*: *algebra-simps*)
then have *H0-5*: $2*C*F$ *dvd* $(H0 - (C + B*F - C*F))$ **using** *mod-eq-dvd-iff* **by** *blast*

have *y-den-ok*: *y-den x* = *2*C*F*
unfolding *y-den-def y-den-ABCx-def F-ACx-def E-ACx-def F-def E-def D-def* **by** *simp*
have *y-num-ok*: *y-num x* = *H0* − (*C* + *B*F* − *C*F*)
unfolding *y-num-def y-num-ABCx-def G-ACx-def E-ACx-def F-ACx-def H0-def G-def D-def E-def F-def*
by (*simp add*: *algebra-simps*)
then show *?thesis* **using** *y-den-ok H0-5* **by** *simp*
qed

lemma *lemma-4-2*:
fixes *A B n*
defines *C* ≡ *ψ A* (*nat B*)
assumes *evA*: *even A A*≥*4 B*≥*3*
shows ∃ *x*≥*n*. ∃ *y*≥*n*. *is-square* (*D-f A C*) ∧ *is-square* (*F-ACx A C x*) ∧ *is-square* (*I-ABCxy A B C x y*)
proof −
define *y-num y-den* **where** *y-num* ≡ *y-num-ABCx A B C* **and** *y-den* ≡ *y-den-ABCx A B C*
obtain *m* **where** *y-prop*: ∀ *x*. *x*≥*m* ⟶ *y-num x* ≥ *n* ∗ *y-den x* ∧ *y-den x* > *0*
unfolding *y-num-def y-den-def* **using** *assms lemma-4-2-y-grows*[*of A B*] **by** *auto*
obtain *x* **where** *x-big-DF-square*: *x* ≥ *abs m*+*abs n* ∧ *is-square* (*D-f A C*) ∧ *is-square* (*F-ACx A C x*)
using *assms lemma-4-2-part-DF*[*of A B*] **by** *auto*
obtain *y* **where** *y-def*: *y-num x* = *y-den x* ∗ *y*
unfolding *y-num-def y-den-def* **using** *lemma-4-2-y-int*[*of A B x*] *assms* **by** *auto*
have *x-big'*: *x* ≥ *m* **using** *x-big-DF-square* **by** *linarith*
then have *y-big*: *y* ≥ *n* **using** *y-prop y-def* **by** *auto*
have *x-big*: *x* ≥ *n* **using** *x-big-DF-square* **by** *linarith*
define *F* **where** *F* = *F-ACx A C x*
define *G* **where** *G* = *G-ACx A C x*
define *H* **where** *H* = *H-ABCxy A B C x y*
define *I* **where** *I* = *I-ABCxy A B C x y*
have *H*∗*y-den x* = (*C* + *B*F* + (*2*y*−*1*)*C*F*)∗*y-den x*
unfolding *H-def H-ABCxy-def H-f-def F-def* **by** *simp*
also have ... = (*C* + *B*F* − *C*F*)∗*y-den x* + *2*C*F*∗*y-num x*
using *y-def* **by** (*auto simp add*: *algebra-simps*)
also have ... = (*C* + *B*F* − *C*F*)∗*2*C*F* + *2*C*F*∗(*ψ* (*2* ∗ *G*) (*nat B*) −

$C + (C - B) * F$)
unfolding *y-den-def y-den-ABCx-def y-num-def y-num-ABCx-def F-def G-def*
by *auto*
also have ... = $2*C*F*\psi\ (2*G)\ (nat\ B)$
by (*simp add*: *algebra-simps*)
also have ... = *y-den x* $*\ \psi\ (2*G)\ (nat\ B)$
unfolding *y-den-def y-den-ABCx-def F-def* **by** *simp*
finally have $H = \psi\ (2*G)\ (nat\ B)$ **using** *y-prop x-big'* **by** *auto*
then have $\exists\, m.\ H = \ \psi\ (2\ *\ G)\ m \vee H = \ -\ \psi\ (2\ *\ G)\ m$ **by** *auto*
then have *is-square I*
unfolding *I-def I-ABCxy-def I-f-def H-def G-def*
using *lucas-pell-corollary*[*of G-ACx A C x H-ABCxy A B C x y*] **by** *auto*
then have $x \geq n \wedge y \geq n \wedge$ *is-square* (*D-f A C*) $\wedge$ *is-square* (*F-ACx A C x*) $\wedge$ *is-square* (*I-ABCxy A B C x y*)
unfolding *I-def* **using** *x-big-DF-square y-big x-big* **by** *auto*
then show *?thesis* **by** *auto*
qed

lemma *lemma-4-2-cor*:
fixes *A B*
defines $C \equiv \psi\ A\ (nat\ B)$
assumes *evA*: *even A* $A{\geq}4$ $B{\geq}3$
shows $\forall\, n.\ \exists\, x{\geq}n.\ \exists\, y{\geq}n.$ *is-square* ((*D-f A C*) $*$ (*F-ACx A C x*) $*$ (*I-ABCxy A B C x y*))
proof
fix *n*
obtain *x y* **where** *prop-36*: $x{\geq}n \wedge y{\geq}n \wedge$ *is-square* (*D-f A C*) $\wedge$ *is-square* (*F-ACx A C x*) $\wedge$ *is-square* (*I-ABCxy A B C x y*)
using *lemma-4-2*[*of A B n*] *assms* **by** *auto*
obtain *d f i* **where** *d^2 = D-f A C* $\wedge$ *f^2 = F-ACx A C x* $\wedge$ *i^2 = I-ABCxy A B C x y*
using *prop-36 is-square-def* **by** *metis*
then have (*d*$*$*f*$*$*i*)*^2 = (*D-f A C*) $*$ (*F-ACx A C x*) $*$ (*I-ABCxy A B C x y*)
by (*auto simp add*: *power2-eq-square algebra-simps*)
then have *is-square* ((*D-f A C*) $*$ (*F-ACx A C x*) $*$ (*I-ABCxy A B C x y*))
unfolding *is-square-def* **by** *metis*
then show $\exists\, x{\geq}n.\ \exists\, y{\geq}n.$ *is-square* ((*D-f A C*) $*$ (*F-ACx A C x*) $*$ (*I-ABCxy A B C x y*))
using *prop-36* **by** *auto*
qed

end

end
theory *DFI-square-1*
imports *DFI-square-0 Lucas-Diophantine*

begin

fun *rec-forte-init012*::*nat* ⇒ *nat* **where**
rec-forte-init012 0 = *0* |
rec-forte-init012 (*Suc 0*) = *0* |
rec-forte-init012 (*Suc* (*Suc 0*)) = *0* |
rec-forte-init012 (*Suc* (*Suc* (*Suc n*))) = (∑ *i*≤*Suc* (*Suc n*). *rec-forte-init012 i*)

theorem *strong-induct-init012* [*consumes 0*, *case-names 0 1 2 sucsucsuc*]:
P 0 ⟹ *P* (*Suc 0*) ⟹ *P* (*Suc* (*Suc 0*)) ⟹ (⋀*n*. ((∀ *i*≤*Suc* (*Suc n*). *P i*) ⟹
P (*Suc* (*Suc* (*Suc n*)))))
⟹ *P* (*n*::*nat*)
using *rec-forte-init012.induct* **by** *auto*

lemma *sun-lemma2*:⋀*n k r*. *ψ A* (*k*∗*n*+*r*) =
(∑ *i*≤*n*. *int* (*n choose i*) ∗ (*ψ A* (*Suc k*) − *A*∗*ψ A k*) ^(*n*−*i*)∗(*ψ A k*) ^*i*∗*ψ A* (*r*+*i*))
proof −
have *case-n1*: ⋀*r*. *ψ A* (*k*+*r*)
= (∑ *i*≤*1*. *int* (*1 choose i*) ∗ (*ψ A* (*k*+*1*) − *A*∗*ψ A k*) ^(*1*−*i*)∗(*ψ A k*) ^*i*∗*ψ A* (*r*+*i*)) **for** *k*
proof (*induction k rule*: *strong-induct-init012*)
case *0*
then show *?case* **by** *auto*
next
case *1*
then show *?case* **by** *auto*
next
case *2*
then show *?case* **by** *auto*
next
case (*sucsucsuc k*)
note *t* = *this*
hence *IH-k*: *ψ A* (*k*+*s*)
= (∑ *i*≤*1*. *int* (*1 choose i*) ∗ (*ψ A* (*k*+*1*) − *A*∗*ψ A k*) ^(*1*−*i*)∗(*ψ A k*) ^*i*∗*ψ A* (*s*+*i*)) **for** *s* **by** *auto*
have *IH-Suck*: *ψ A* (*Suc k*+*s*)
= (∑ *i*≤*1*. *int* (*1 choose i*) ∗ (*ψ A* (*Suc k*+*1*) − *A*∗*ψ A* (*Suc k*)) ^(*1*−*i*)∗(*ψ A* (*Suc k*)) ^*i*∗*ψ A* (*s*+*i*))
for *s*
using *t Suc-n-not-le-n nat-le-linear* **by** *blast*
have *IH-SucSuck*: *ψ A* (*Suc* (*Suc k*)+*s*) = (∑ *i*≤*1*. *int* (*1 choose i*) ∗ (*ψ A* (*Suc* (*Suc k*)+*1*)
− *A*∗*ψ A* (*Suc* (*Suc k*))) ^(*1*−*i*)∗(*ψ A* (*Suc* (*Suc k*))) ^*i*∗*ψ A* (*s*+*i*)) **for** *s*
using *t Suc-n-not-le-n nat-le-linear* **by** *blast*
have *ψ A* (*Suc* (*Suc k*)+*r*) = (*ψ A* (*Suc* (*Suc k*)+*1*)
− *A*∗*ψ A* (*Suc* (*Suc k*)))∗*ψ A r* + *ψ A* (*Suc* (*Suc k*)) ∗ *ψ A* (*r*+*1*)
using *IH-SucSuck*[*of r*] **by** (*auto simp add*: *algebra-simps*)
hence *simp-psi-sucsuc*: *ψ A* (*Suc* (*Suc k*)+*r*)
= *ψ A* (*Suc* (*Suc k*)) ∗ *ψ A* (*r*+*1*) − *ψ A* (*Suc k*) ∗ *ψ A r* **by** *auto*
have *ψ A* (*Suc k* + *r*) = (*ψ A* (*Suc k* + *1*) − *A*∗*ψ A* (*Suc k*))∗*ψ A r* + *ψ A*

$(Suc\ k) * \psi\ A\ (r+1)$
using *IH-Suck*[*of r*] **by** (*auto simp add*: *algebra-simps*)
hence $\psi\ A\ (Suc\ k + r) = \psi\ A\ (Suc\ k) * \psi\ A\ (r+1) - \psi\ A\ k * \psi\ A\ r$ **by** *auto*
hence $\psi\ A\ (Suc\ (Suc\ (Suc\ k)) + r) = A*(\psi\ A\ (Suc\ (Suc\ k)) * \psi\ A\ (r+1) - \psi\ A\ (Suc\ k) * \psi\ A\ r)$
$- \psi\ A\ (Suc\ k) * \psi\ A\ (r+1) + \psi\ A\ k * \psi\ A\ r$
using *simp-psi-sucsuc* **by** *auto*
hence $\psi\ A\ (Suc\ (Suc\ (Suc\ k)) + r) = (A*\psi\ A\ (Suc\ (Suc\ k)) - \psi\ A\ (Suc\ k)) * \psi\ A\ (r+1)$
$- (A * \psi\ A\ (Suc\ k) - \psi\ A\ k)*\psi\ A\ r$ **by** (*auto simp add*: *algebra-simps*)
hence *simp-psi*: $\psi\ A\ (Suc\ (Suc\ (Suc\ k)) + r)$
$= \psi\ A\ (Suc\ (Suc\ (Suc\ k))) * \psi\ A\ (r+1) - \psi\ A\ (Suc\ (Suc\ k))*\psi\ A\ r$ **by** *auto*
have $(\sum i{\le}1.\ int\ (1\ choose\ i) * (\psi\ A\ (Suc\ (Suc\ (Suc\ k))+1)$
$- A*\psi\ A\ (Suc\ (Suc\ (Suc\ k)))) \hat{}(1-i)*(\psi\ A\ (Suc\ (Suc\ (Suc\ k)))) \hat{}i*\psi\ A\ (r+i))$
$= (\psi\ A\ (Suc\ (Suc\ (Suc\ (Suc\ k)))) - A*\psi\ A\ (Suc\ (Suc\ (Suc\ k))))*\psi\ A\ r$
$+ \psi\ A\ (Suc\ (Suc\ (Suc\ k))) *\psi\ A\ (r+1)$
by (*auto simp add*: *algebra-simps*)
hence $(\sum i{\le}1.\ int\ (1\ choose\ i) * (\psi\ A\ (Suc\ (Suc\ (Suc\ k))+1)$
$- A*\psi\ A\ (Suc\ (Suc\ (Suc\ k)))) \hat{}(1-i)*(\psi\ A\ (Suc\ (Suc\ (Suc\ k)))) \hat{}i*\psi\ A\ (r+i))$
$= - \psi\ A\ (Suc\ (Suc\ k)) * \psi\ A\ r + \psi\ A\ (Suc\ (Suc\ (Suc\ k))) * \psi\ A\ (r+1)$ **by** *simp*
then show *?case* **using** *simp-psi* **by** (*auto simp add*: *algebra-simps*)
qed
show $\bigwedge k\ r.\ \psi\ A\ (k*n+r)$
$= (\sum i{\le}n.\ int\ (n\ choose\ i) * (\psi\ A\ (Suc\ k) - A*\psi\ A\ k) \hat{}(n-i)*(\psi\ A\ k) \hat{}i*\psi\ A\ (r+i))$ **for** *n*
proof (*induction n rule*: ψ-*induct*)
case *0*
then show *?case* **by** *auto*
next
case *1*
then show *?case* **using** *case-n1* **by** *auto*
next
case (*sucsuc n*)
note $t = this$
have *eq1*: $\psi\ A\ (k*(n+2)+r)$
$= (\sum i{\le}n+1.\ int\ ((n+1)\ choose\ i) * (\psi\ A\ (Suc\ k) - A*\psi\ A\ k) \hat{}(n+1-i)*(\psi\ A\ k) \hat{}i*\psi\ A\ (k+r+i))$
using t[*of k k+r*] **by** (*auto simp add*: *algebra-simps*)
have *case-n1-to-ri*: $\psi\ A\ (k+r+i) = (\psi\ A\ (k+1) - A*\psi\ A\ k)*\psi\ A\ (r+i) + \psi\ A\ k * \psi\ A\ (r+i+1)$ **for** *i*
using *case-n1*[*of k r+i*] **by** (*auto simp add*: *algebra-simps*)
hence $(\psi\ A\ (k+1) - A*\psi\ A\ k) \hat{}(n+1-i)*(\psi\ A\ k) \hat{}i*\psi\ A\ (k+r+i)$
$= (\psi\ A\ (k+1) - A*\psi\ A\ k) \hat{}(n+1-i)*(\psi\ A\ k) \hat{}i*(\psi\ A\ (k+1) - A*\psi\ A\ k)*\psi\ A\ (r+i)$
$+ (\psi\ A\ (k+1) - A*\psi\ A\ k) \hat{}(n+1-i)*(\psi\ A\ k) \hat{}i*\psi\ A\ k*\psi\ A\ (r+i+1)$ **for** *i*

using *case-n1-to-ri*[*of i*] *distrib-left*[*of* (ψ *A* (*k*+*1*) − *A*∗ψ *A k*) ^(*n*+*1*−*i*)∗(ψ *A k*) ^*i*
(ψ *A* (*k*+*1*) − *A*∗ψ *A k*)∗ψ *A* (*r*+*i*) ψ *A k* ∗ ψ *A* (*r*+*i*+*1*)] **by** *auto*
hence (ψ *A* (*k*+*1*) − *A*∗ψ *A k*) ^(*n*+*1*−*i*)∗(ψ *A k*) ^*i*∗ψ *A* (*k*+*r*+*i*)
= (ψ *A* (*k*+*1*) − *A*∗ψ *A k*)∗(ψ *A* (*k*+*1*) − *A*∗ψ *A k*) ^(*n*+*1*−*i*)∗(ψ *A k*) ^*i*∗ψ *A* (*r*+*i*)
+ (ψ *A* (*k*+*1*) − *A*∗ψ *A k*) ^(*n*+*1*−*i*)∗ψ *A k*∗(ψ *A k*) ^*i*∗ψ *A* (*r*+*i*+*1*) **for** *i* **by** *auto*
hence (ψ *A* (*k*+*1*) − *A*∗ψ *A k*) ^(*n*+*1*−*i*)∗(ψ *A k*) ^*i*∗ψ *A* (*k*+*r*+*i*)
= (ψ *A* (*k*+*1*) − *A*∗ψ *A k*) ^(*Suc* (*n*+*1*−*i*))∗(ψ *A k*) ^*i*∗ψ *A* (*r*+*i*)
+ (ψ *A* (*k*+*1*) − *A*∗ψ *A k*) ^(*n*+*1*−*i*)∗(ψ *A k*) ^(*i*+*1*)∗ψ *A* (*r*+*i*+*1*) **for** *i*
using *power-Suc*[*of* ψ *A* (*k*+*1*) − *A*∗ψ *A k n*+*1*−*i*] *power-Suc*[*of* ψ *A k i*] **by** *auto*
hence (ψ *A* (*Suc k*) − *A*∗ψ *A k*) ^(*n*+*1*−*i*)∗(ψ *A k*) ^*i*∗ψ *A* (*k*+*r*+*i*)
= (ψ *A* (*Suc k*) − *A*∗ψ *A k*) ^(*Suc* (*n*+*1*−*i*))∗(ψ *A k*) ^*i*∗ψ *A* (*r*+*i*)
+ (ψ *A* (*Suc k*) − *A*∗ψ *A k*) ^(*n*+*1*−*i*)∗(ψ *A k*) ^(*i*+*1*) ∗ ψ *A* (*r*+*i*+*1*) **for** *i* **by** *auto*
hence *int* ((*n*+*1*) *choose i*) ∗ ((ψ *A* (*Suc k*) − *A*∗ψ *A k*) ^(*n*+*1*−*i*)∗(ψ *A k*) ^*i*∗ψ *A* (*k*+*r*+*i*))
= *int* ((*n*+*1*) *choose i*) ∗ ((ψ *A* (*Suc k*) − *A*∗ψ *A k*) ^(*Suc* (*n*+*1*−*i*))∗(ψ *A k*) ^*i*∗ψ *A* (*r*+*i*)
+ (ψ *A* (*Suc k*) − *A*∗ψ *A k*) ^(*n*+*1*−*i*)∗(ψ *A k*) ^(*i*+*1*) ∗ ψ *A* (*r*+*i*+*1*)) **for** *i* **by** *auto*
hence *int* ((*n*+*1*) *choose i*) ∗ ((ψ *A* (*Suc k*) − *A*∗ψ *A k*) ^(*n*+*1*−*i*)∗(ψ *A k*) ^*i*∗ψ *A* (*k*+*r*+*i*))
= *int* ((*n*+*1*) *choose i*) ∗ ((ψ *A* (*Suc k*) − *A*∗ψ *A k*) ^(*Suc* (*n*+*1*−*i*))∗(ψ *A k*) ^*i*∗ψ *A* (*r*+*i*))
+ *int* ((*n*+*1*) *choose i*) ∗ ((ψ *A* (*Suc k*) − *A*∗ψ *A k*) ^(*n*+*1*−*i*)∗(ψ *A k*) ^(*i*+*1*) ∗ ψ *A* (*r*+*i*+*1*)) **for** *i*
by (*auto simp add*: *algebra-simps*)
hence *int* ((*n*+*1*) *choose i*) ∗ (ψ *A* (*Suc k*) − *A*∗ψ *A k*) ^(*n*+*1*−*i*)∗(ψ *A k*) ^*i*∗ψ *A* (*k*+*r*+*i*)
= *int* ((*n*+*1*) *choose i*) ∗ (ψ *A* (*Suc k*) − *A*∗ψ *A k*) ^(*Suc* (*n*+*1*−*i*))∗(ψ *A k*) ^*i*∗ψ *A* (*r*+*i*)
+ *int* ((*n*+*1*) *choose i*) ∗ (ψ *A* (*Suc k*) − *A*∗ψ *A k*) ^(*n*+*1*−*i*)∗(ψ *A k*) ^(*i*+*1*) ∗ ψ *A* (*r*+*i*+*1*) **for** *i*
by (*auto simp add*: *algebra-simps*)
hence ψ *A* (*k*∗(*n*+*2*)+*r*) =
($\sum i \le n+1$. *int* ((*n*+*1*) *choose i*) ∗ (ψ *A* (*Suc k*) − *A*∗ψ *A k*) ^(*Suc* (*n*+*1*−*i*))∗(ψ *A k*) ^*i*∗ψ *A* (*r*+*i*)
+ *int* ((*n*+*1*) *choose i*) ∗ (ψ *A* (*Suc k*) − *A*∗ψ *A k*) ^(*n*+*1*−*i*)∗ψ *A k* ^(*i*+*1*) ∗ ψ *A* (*r*+*i*+*1*))
using *eq1* **by** *auto*
hence *eq2*: ψ *A* (*k*∗(*n*+*2*)+*r*) =
($\sum i \le n+1$. *int* ((*n*+*1*) *choose i*) ∗ (ψ *A* (*Suc k*) − *A*∗ψ *A k*) ^(*Suc* (*n*+*1*−*i*))∗(ψ *A k*) ^*i*∗ψ *A* (*r*+*i*))
+ ($\sum i \le n+1$. *int* ((*n*+*1*) *choose i*) ∗ (ψ *A* (*Suc k*) − *A*∗ψ *A k*) ^(*n*+*1*−*i*)∗(ψ *A k*) ^(*i*+*1*)∗ψ *A* (*r*+*i*+*1*))
using *sum.distrib*[*of* λi. *int* ((*n*+*1*) *choose i*)∗(ψ *A* (*Suc k*) − *A*∗ψ *A k*) ^(*Suc*

$(n+1-i))*(\psi\ A\ k)\hat{}i*\psi\ A\ (r+i)$
$\lambda i.\ int\ ((n+1)\ choose\ i)\ *(\psi\ A\ (Suc\ k)\ -\ A*\psi\ A\ k)\hat{}(n+1-i)*(\psi\ A\ k)\hat{}(i+1)*\psi\ A\ (r+i+1)\ \{..n+1\}]$
by *auto*
have $i \le n+1 \Longrightarrow int\ ((n+1)\ choose\ i) * (\psi\ A\ (Suc\ k) - A*\psi\ A\ k)\hat{}(n+1-i)*(\psi\ A\ k)\hat{}(i+1)*\psi\ A\ (r+i+1)$
$= int\ (n+1\ choose\ (i+1-1))*(\psi\ A\ (Suc\ k) - A*\psi\ A\ k)\ \hat{}(n+2-(i+1))*\psi\ A\ k\ \hat{}\ (i+1)*\psi\ A\ (r+(i+1))$ **for** i
by *auto*
hence $\psi\ A\ (k*(n+2)+r) =$
$(\sum i{\le}n+1.\ int\ ((n+1)\ choose\ i)\ *\ (\psi\ A\ (Suc\ k)\ -\ A*\psi\ A\ k)\hat{}(Suc\ (n+1-i))*(\psi\ A\ k)\hat{}i*\psi\ A\ (r+i))$
$+\ (\sum i{\le}n\ +\ 1.\ int\ (n+1\ choose\ (i+1-1))*(\psi\ A\ (Suc\ k) - A*\psi\ A\ k)\ \hat{}(n+2-(i+1))$
$*\ \psi\ A\ k\ \hat{}\ (i+1)*\psi\ A\ (r+(i+1)))$
using *eq2* **by** (*auto simp add*: *algebra-simps*)
hence $\psi\ A\ (k*(n+2)+r) =$
$(\sum i{=}0..n+1.\ int\ ((n+1)\ choose\ i)\ *\ (\psi\ A\ (Suc\ k)\ -\ A*\psi\ A\ k)\hat{}(Suc\ (n+1-i))*(\psi\ A\ k)\hat{}i*\psi\ A\ (r+i))$
$+\ (\sum i{=}0..n\ +\ 1.\ int\ (n+1\ choose\ (i+1-1))*(\psi\ A\ (Suc\ k) - A*\psi\ A\ k)\ \hat{}(n+2-(i+1))$
$*\ \psi\ A\ k\ \hat{}\ (i+1)*\psi\ A\ (r+(i+1)))$
using *atMost-atLeast0* **by** *presburger*
hence *eq3*: $\psi\ A\ (k*(n+2)+r) =$
$(\sum i{=}0..n+1.\ int\ ((n+1)\ choose\ i)\ *\ (\psi\ A\ (Suc\ k)\ -\ A*\psi\ A\ k)\hat{}(Suc\ (n+1-i))*(\psi\ A\ k)\hat{}i*\psi\ A\ (r+i))$
$+\ (\sum i{=}1..n+2.\ int\ ((n+1)\ choose\ (i-1))\ *\ (\psi\ A\ (Suc\ k)\ -\ A*\psi\ A\ k)\hat{}(n+2-i)*(\psi\ A\ k)\hat{}i*\psi\ A\ (r+i))$
using *translation-var-0-to-1*[*of* $\lambda i.\ int\ ((n+1)\ choose\ (i-1))*(\psi\ A\ (Suc\ k)\ -\ A*\psi\ A\ k)\hat{}(n+2-i)*(\psi\ A\ k)\hat{}i*\psi\ A\ (r+i)\ n+1$]
by *auto*
have *obvious*: $\{0\}\cup\{1..n+1\} = \{0..n+1\} \wedge \{0\}\cap\{1..n+1\} = \{\} \wedge finite\ \{0\} \wedge finite\ \{1..n+1\}$
$\wedge\ finite\ \{n+2\} \wedge \{1..n+1\}\cup\{n+2\} = \{1..n+2\} \wedge \{1..n+1\}\cap\{n+2\} = \{\}$ **by** *auto*
hence $\psi\ A\ (k*(n+2)+r) =$
$int\ ((n+1)\ choose\ 0) * (\psi\ A\ (Suc\ k)\ -\ A*\psi\ A\ k)\hat{}(Suc\ (n+1-0))*(\psi\ A\ k)\hat{}0*\psi\ A\ (r+0)$
$+\ (\sum i{=}1..n+1.\ int\ ((n+1)\ choose\ i)\ *\ (\psi\ A\ (Suc\ k)\ -\ A*\psi\ A\ k)\hat{}(Suc\ (n+1-i))*(\psi\ A\ k)\hat{}i*\psi\ A\ (r+i))$
$+\ (\sum i{=}1..n+1.\ int\ ((n+1)\ choose\ (i-1))\ *\ (\psi\ A\ (Suc\ k)\ -\ A*\psi\ A\ k)\hat{}(n+2-i)*(\psi\ A\ k)\hat{}i*\psi\ A\ (r+i))$
$+\ int\ ((n+1)\ choose\ ((n+2)-1)) * (\psi\ A\ (Suc\ k) - A*\psi\ A\ k)\hat{}(n+2-(n+2))*(\psi\ A\ k)\hat{}(n+2)*\psi\ A\ (r+(n+2))$
using *sum.union-disjoint*[*of* $\{0\}\ \{1..n+1\}\ \lambda i.\ int\ ((n+1)\ choose\ i)\ *\ (\psi\ A\ (Suc\ k)\ -\ A*\psi\ A\ k)\hat{}(Suc\ (n+1-i))*(\psi\ A\ k)\hat{}i*\psi\ A\ (r+i)$]
sum.union-disjoint[*of* $\{1..n+1\}\ \{n+2\}$
$\lambda i.\ int\ ((n+1)\ choose\ (i-1))\ *\ (\psi\ A\ (Suc\ k)\ -\ A*\psi\ A\ k)\hat{}(n+2-i)*(\psi\ A\ k)\hat{}i*\psi\ A\ (r+i)$]

Suc-1 add-Suc-right diff-le-self diff-self-eq-0 eq3 numeral-1-eq-Suc-0 numerals(1)
sum.atLeast-Suc-atMost sum.nat-ivl-Suc' **by** *auto*
hence $\psi\ A\ (k*(n+2)+r) =$
$int\ ((n+1)\ choose\ 0) * (\psi\ A\ (Suc\ k) - A*\psi\ A\ k)\hat{}(Suc\ (n+1-0))*(\psi\ A\ k)\hat{}0*\psi\ A\ (r+0)$
$+ (\sum i=1..n+1.\ int\ ((n+1)\ choose\ i) * (\psi\ A\ (Suc\ k) - A*\psi\ A\ k)\hat{}(Suc\ (n+1-i))*(\psi\ A\ k)\hat{}i*\psi\ A\ (r+i)$
$+ int\ ((n+1)\ choose\ (i-1)) * (\psi\ A\ (Suc\ k) - A*\psi\ A\ k)\hat{}(n+2-i)*(\psi\ A\ k)\hat{}i*\psi\ A\ (r+i))$
$+ int\ ((n+1)\ choose\ ((n+2)-1)) * (\psi\ A\ (Suc\ k) - A*\psi\ A\ k)\hat{}(n+2-(n+2))*(\psi\ A\ k)\hat{}(n+2)*\psi\ A\ (r+(n+2))$
using *sum.distrib*[*of* $\lambda i.\ int\ ((n+1)\ choose\ i) * (\psi\ A\ (Suc\ k) - A*\psi\ A\ k)\hat{}(Suc\ (n+1-i))*(\psi\ A\ k)\hat{}i*\psi\ A\ (r+i)$
$\lambda i.\ int\ ((n+1)\ choose\ (i-1)) * (\psi\ A\ (Suc\ k) - A*\psi\ A\ k)\hat{}(n+2-i)*(\psi\ A\ k)\hat{}i*\psi\ A\ (r+i)\ \{1..n+1\}$]
by *auto*
hence *eq4*: $\psi\ A\ (k*(n+2)+r) =$
$int\ ((n+2)\ choose\ 0) * (\psi\ A\ (Suc\ k) - A*\psi\ A\ k)\hat{}(n+2-0)*(\psi\ A\ k)\hat{}0*\psi\ A\ (r+0)$
$+ (\sum i=1..n+1.\ int\ ((n+1)\ choose\ i) * (\psi\ A\ (Suc\ k) - A*\psi\ A\ k)\hat{}(Suc\ (n+1-i))*(\psi\ A\ k)\hat{}i*\psi\ A\ (r+i)$
$+ int\ ((n+1)\ choose\ (i-1)) * (\psi\ A\ (Suc\ k) - A*\psi\ A\ k)\hat{}(n+2-i)*(\psi\ A\ k)\hat{}i*\psi\ A\ (r+i))$
$+ int\ ((n+2)\ choose\ (n+2)) * (\psi\ A\ (Suc\ k) - A*\psi\ A\ k)\hat{}(n+2-(n+2))*(\psi\ A\ k)\hat{}(n+2)*\psi\ A\ (r+(n+2))$
by (*auto simp add*: *algebra-simps*)
have $i\in\{1..n+1\} \Longrightarrow int\ ((n+1)\ choose\ i) * (\psi\ A\ (Suc\ k) - A*\psi\ A\ k)\hat{}(Suc\ (n+1-i))*(\psi\ A\ k)\hat{}i*\psi\ A\ (r+i)$
$+ int\ ((n+1)\ choose\ (i-1)) * (\psi\ A\ (Suc\ k) - A*\psi\ A\ k)\hat{}(n+2-i)*(\psi\ A\ k)\hat{}i*\psi\ A\ (r+i)$
$= int\ ((n+1)\ choose\ i) * (\psi\ A\ (Suc\ k) - A*\psi\ A\ k)\hat{}(n+2-i)*(\psi\ A\ k)\hat{}i*\psi\ A\ (r+i)$
$+ int\ ((n+1)\ choose\ (i-1)) * (\psi\ A\ (Suc\ k) - A*\psi\ A\ k)\hat{}(n+2-i)*(\psi\ A\ k)\hat{}i*\psi\ A\ (r+i)$ **for** i
by (*simp add*: *Suc-diff-le*)
hence $i\in\{1..n+1\} \Longrightarrow int\ ((n+1)\ choose\ i)*(\psi\ A\ (Suc\ k)-A*\psi\ A\ k)\hat{}(Suc\ (n+1-i))*(\psi\ A\ k)\hat{}i*\psi\ A\ (r+i)$
$+ int\ ((n+1)\ choose\ (i-1)) * (\psi\ A\ (Suc\ k) - A*\psi\ A\ k)\hat{}(n+2-i)*(\psi\ A\ k)\hat{}i*\psi\ A\ (r+i)$
$= (int\ ((n+1)\ choose\ i)$
$+ int\ ((n+1)\ choose\ (i-1))) * (\psi\ A\ (Suc\ k) - A*\psi\ A\ k)\hat{}(n+2-i)*(\psi\ A\ k)\hat{}i*\psi\ A\ (r+i)$ **for** i
by (*auto simp add*: *algebra-simps*)
hence $i\in\{1..n+1\} \Longrightarrow int\ ((n+1)\ choose\ i)*(\psi\ A\ (Suc\ k)-A*\psi\ A\ k)\hat{}(Suc\ (n+1-i))*(\psi\ A\ k)\hat{}i*\psi\ A\ (r+i)$
$+ int\ ((n+1)\ choose\ (i-1)) * (\psi\ A\ (Suc\ k) - A*\psi\ A\ k)\hat{}(n+2-i)*(\psi\ A\ k)\hat{}i*\psi\ A\ (r+i)$
$= int\ ((n+2)\ choose\ i)* (\psi\ A\ (Suc\ k) - A*\psi\ A\ k)\hat{}(n+2-i)*(\psi\ A\ k)\hat{}i*\psi$

```
A (r+i) for i
      using choose-reduce-nat[of n+2 i] by auto
    hence ψ A (k*(n+2)+r) = int ((n+2) choose 0) * (ψ A (Suc k) - A*ψ A
k) ^(n+2-0)*(ψ A k) ^0*ψ A (r+0)
      + (∑ i=1..n+1. int ((n+2) choose i) * (ψ A (Suc k) - A*ψ A k) ^(n+2-i)*(ψ
A k) ^i*ψ A (r+i))
       + int ((n+2) choose (n+2)) * (ψ A (Suc k) - A*ψ A k) ^(n+2-(n+2))*(ψ
A k) ^(n+2)*ψ A (r+(n+2))
    using eq4 by auto
    hence ψ A (k*(n+2)+r) = int ((n+2) choose 0) * (ψ A (Suc k) - A*ψ A
k) ^(n+2-0)*(ψ A k) ^0*ψ A (r+0)
      + (∑ i=1..n+2. int ((n+2) choose i) * (ψ A (Suc k) - A*ψ A k) ^(n+2-i)*(ψ
A k) ^i*ψ A (r+i))
      by auto
    hence ψ A (k*(n+2)+r) =
      (∑ i=0..n+2. int ((n+2) choose i) * (ψ A (Suc k) - A*ψ A k) ^(n+2-i)*(ψ
A k) ^i*ψ A (r+i))
      using obvious by (simp add: sum.atLeast-Suc-atMost)
    hence ψ A (k*(n+2)+r) =
       (∑ i≤n+2. int ((n+2) choose i) * (ψ A (Suc k) - A*ψ A k) ^(n+2-i)*(ψ
A k) ^i*ψ A (r+i))
      using atMost-atLeast0 by presburger
    then show ?case by simp
  qed
qed


lemma lucas-consec-coprime: coprime (ψ A k) (ψ A (Suc k))
proof (induction k)
  case 0
  then show ?case by simp
next
  case (Suc k)
  note IH=this
  have ∀ c. c dvd ψ A (Suc k) ∧ c dvd ψ A (Suc (Suc k)) ⟶ is-unit c
  proof
    fix c
    show c dvd ψ A (Suc k) ∧ c dvd ψ A (Suc (Suc k)) ⟶ is-unit c
    proof
      assume divhyp: c dvd ψ A (Suc k) ∧ c dvd ψ A (Suc (Suc k))
      then have c dvd A * ψ A (Suc k) - ψ A k by simp
      then have c dvd ψ A k using divhyp by algebra
      then show is-unit c using divhyp IH by fastforce
    qed
  qed
  then show ?case unfolding coprime-def by simp
qed
```

lemma *eq-mod-power*: **fixes** *a*::*int* **and** *b*::*int* **assumes** *a mod n* = *b mod n* **shows** *a^k mod n* = *b^k mod n*
proof (*induction k*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc k*)
note *IH*=*this*
then show *a ^ Suc k mod n* = *b ^ Suc k mod n* **using** *assms mod-mult-cong*[*of a n b a^k b^k*]
by *fastforce*
qed

lemma *euclids-lemma*: (*coprime* (*a*::*int*) *b*) ∧ *a dvd* (*b*∗*c*) ⟶ *a dvd c*
using *coprime-dvd-mult-right-iff* **by** *blast*

lemma *coprime-power*: **fixes** *a*::*int* **and** *b*::*int* **assumes** *coprime a b* **shows** *coprime* (*a^k*) *b*
proof (*induction k*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc k*)
note *IH*=*this*
then have *c dvd a ^ Suc k* ∧ *c dvd b* ⟹ *is-unit c* **for** *c*
proof −
assume *assm*: *c dvd a ^ Suc k* ∧ *c dvd b*
then have *coprime c a*
by (*meson assms coprime-def dvd-trans*)
then show *is-unit c*
using *euclids-lemma*[*of c a a^k*] *IH assm* **by** *fastforce*
qed
then show *coprime* (*a^*(*Suc k*)) *b*
by *fastforce*
qed

lemma *dvd-remove-psi*:
fixes *A*::*int* **and** *k*::*nat* **and** *m*::*nat*
assumes (*ψ A k*) *dvd* (*ψ A m*) **and** *A^2*−*4*≥*0* **and** *k*>*0*
shows (*int k*) *dvd* (*int m*)
proof −
define *r* **where** *r* = *m mod k*
then have ∃ *n*. *m* = *n* ∗ *k* + *r*
using *div-mod-decomp* **by** *blast*
then obtain *n* **where** *n-def*: *m* = *n* ∗ *k* + *r* **by** *auto*
have *termszero*:∀ *i*∈{*1*..*n*}. *int*(*n choose i*) ∗ (*ψ A* (*Suc k*)
− *A*∗*ψ A k*) ^(*n*−*i*) ∗ (*ψ A k*) ^*i* ∗ *ψ A* (*r*+*i*) *mod* (*ψ A k*) = *0*

by *simp*
have *yes*: *finite* $\{0\} \wedge$ *finite* $\{1..n\} \wedge \{0\} \cap \{1..n\} = \{\} \wedge \{0\} \cup \{1..n\} = \{0..n\}$ **by** *auto*
have *termszero2*: $(\sum i{\in}\{1..n\}.\ (int(n\ choose\ i) * (\psi\ A\ (Suc\ k) - A{*}\psi\ A\ k)\ \hat{}\ (n{-}i) * (\psi\ A\ k)\ \hat{}\ i$
$* \psi\ A\ (r{+}i))\ mod\ (\psi\ A\ k)) = 0$
using *termszero sum.neutral* **by** *fast*

have *eq1*: $\psi\ A\ m\ mod\ (\psi\ A\ k) = (\sum i{\leq}n.\ int(n\ choose\ i) * (\psi\ A\ (Suc\ k) - A{*}\psi\ A\ k)\ \hat{}\ (n{-}i)$
$* (\psi\ A\ k)\ \hat{}\ i * \psi\ A\ (r{+}i))\ mod\ (\psi\ A\ k)$
using *n-def sun-lemma2*[*of A k n r*] **by** (*simp add*: *mult.commute*)
also have *eq2*: ... $= (\sum i{\leq}n.\ (int(n\ choose\ i) * (\psi\ A\ (Suc\ k) - A{*}\psi\ A\ k)\ \hat{}\ (n{-}i)$
$* (\psi\ A\ k)\ \hat{}\ i$
$* \psi\ A\ (r{+}i))\ mod\ (\psi\ A\ k))\ mod\ (\psi\ A\ k)$
by (*simp add*: *mod-sum-eq*)
have *eq3*: $(\sum i{\leq}n.\ (int(n\ choose\ i) * (\psi\ A\ (Suc\ k) - A{*}\psi\ A\ k)\ \hat{}\ (n{-}i) * (\psi\ A\ k)\ \hat{}\ i * \psi\ A\ (r{+}i))\ mod\ (\psi\ A\ k))$
$= (\sum i{\leq}0.\ (int(n\ choose\ i) * (\psi\ A\ (Suc\ k) - A{*}\psi\ A\ k)\ \hat{}\ (n{-}i) * (\psi\ A\ k)\ \hat{}\ i * \psi\ A\ (r{+}i))\ mod\ (\psi\ A\ k))$
$+ (\sum i{\in}\{1..n\}.\ (int(n\ choose\ i) * (\psi\ A\ (Suc\ k) - A{*}\psi\ A\ k)\ \hat{}\ (n{-}i) * (\psi\ A\ k)\ \hat{}\ i * \psi\ A\ (r{+}i))\ mod\ (\psi\ A\ k))$
using *sum.union-disjoint*[*of* $\{0\}$ $\{1..n\}$
$(\lambda i.\ int\ (n\ choose\ i) * (\psi\ A\ (Suc\ k) - A * \psi\ A\ k)\ \hat{}\ (n - i) * \psi\ A\ k\ \hat{}\ i * \psi\ A\ (r + i)\ mod\ \psi\ A\ k)$]
yes
by (*metis atMost-0 atMost-atLeast0*)
moreover have *eq4*: ... $= (\sum i{\leq}0.\ (int(n\ choose\ i) * (\psi\ A\ (Suc\ k) - A{*}\psi\ A\ k)\ \hat{}\ (n{-}i)$
$* (\psi\ A\ k)\ \hat{}\ i * \psi\ A\ (r{+}i))\ mod\ (\psi\ A\ k))\ mod\ (\psi\ A\ k)$
using *termszero2* **by** *simp*
moreover have *eq5*: ... $= ((\psi\ A\ (Suc\ k) - A{*}\psi\ A\ k)\ \hat{}\ n * \psi\ A\ (r))\ mod\ (\psi\ A\ k)$
by *fastforce*
have *eq5'*: $(\sum i{\leq}n.\ (int(n\ choose\ i) * (\psi\ A\ (Suc\ k) - A{*}\psi\ A\ k)\ \hat{}\ (n{-}i) * (\psi\ A\ k)\ \hat{}\ i * \psi\ A\ (r{+}i))\ mod\ (\psi\ A\ k))$
$= ((\psi\ A\ (Suc\ k) - A{*}\psi\ A\ k)\ \hat{}\ n * \psi\ A\ (r))\ mod\ (\psi\ A\ k)$
using *eq3 eq4 eq5* **by** *metis*
have $(\psi\ A\ (Suc\ k) - A{*}\psi\ A\ k)\ mod\ (\psi\ A\ k) = (\psi\ A\ (Suc\ k))\ mod\ (\psi\ A\ k)$ **by** *algebra*
then have $(\psi\ A\ (Suc\ k) - A{*}\psi\ A\ k)\ \hat{}\ n\ mod\ (\psi\ A\ k) = (\psi\ A\ (Suc\ k))\ \hat{}\ n\ mod\ (\psi\ A\ k)$
using *eq-mod-power* **by** *fast*
then have *eq6*: $((\psi\ A\ (Suc\ k) - A{*}\psi\ A\ k)\ \hat{}\ n * \psi\ A\ (r))\ mod\ (\psi\ A\ k) = ((\psi\ A\ (Suc\ k))\ \hat{}\ n * \psi\ A\ (r))\ mod\ (\psi\ A\ k)$
using *mod-mult-cong* **by** *fast*
then have *eq7*: $0 = ((\psi\ A\ (Suc\ k))\ \hat{}\ n * \psi\ A\ (r))\ mod\ (\psi\ A\ k)$
using *assms eq1 eq2 eq5' eq6* **by** *simp*
then have *eq8*: $(\psi\ A\ k)\ dvd\ (\psi\ A\ (Suc\ k))\ \hat{}\ n * \psi\ A\ r$ **by** *fastforce*

have *coprime* $((\psi\ A\ (Suc\ k))\hat{}n)\ (\psi\ A\ k)$
using *lucas-consec-coprime*[*of A k*] *coprime-power coprime-commute* **by** *blast*
then have *eq9*: $\psi\ A\ k\ dvd\ \psi\ A\ r$
using *eq8 euclids-lemma coprime-commute* **by** *blast*

have $(abs\ A)\hat{}2 \geq 4$ **using** *assms* **by** *simp*
then have *ABe2*: $abs\ A \geq 2$
by (*smt* (*verit*, *ccfv-SIG*) *abs-1 abs-square-le-1 zero-power2*)
have $r<k$ **using** *r-def* ‹$k>0$› **by** *simp*
then have $r + (k - r - 1) = k-1 \wedge Suc\ (k-1) = k$ **by** *simp*
then have $\psi\ (abs\ A)\ r < \psi\ (abs\ A)\ k$
using *ABe2 lucas-monotone2*[*of abs A r k−r−1*] *lucas-strict-monotonicity*[*of abs A k−1*] **by** *simp*
then have $abs\ (\psi\ A\ r) < abs\ (\psi\ A\ k)$ **using** *lucas-symmetry-A-abs ABe2* **by** *fastforce*
then have *psir0*:$\psi\ A\ r = 0$ **using** *eq9*
by (*meson abs-dvd-iff dvd-abs-iff zdvd-not-zless zero-less-abs-iff*)
have *contr*: $r>0 \Longrightarrow abs(\psi\ A\ r) > 0$
using *ABe2 lucas-strict-monotonicity*[*of abs A r−1*] *lucas-symmetry-A-abs*[*of A r*] **by** *simp*
then have $r=0$ **using** *psir0* **by** *auto*
then have $int\ m = int\ k * int\ n$ **using** *n-def* **by** *simp*
then show *?thesis* **by** *simp*
qed

lemma *sun-lemma7*:
fixes *A*::*int* **and** *k*::*nat* **and** *m*::*nat*
assumes $A\hat{}2-4\geq 0$ **and** $(\psi\ A\ k)\hat{}2\ dvd\ \psi\ A\ m$ **and** $k>0$
shows $\psi\ A\ k\ dvd\ (int\ m)$
proof −
have $k\ dvd\ m$ **using** *assms dvd-remove-psi*[*of A k m*] **by** *auto*
then obtain *n* **where** *n-def*: $m = k * n$ **by** *auto*
then show *?thesis*
proof (*cases n=0*)
case *True*
then show *?thesis* **using** *n-def* **by** *simp*
next
case *False*
have $i\geq 2 \longrightarrow (\psi\ A\ k)\hat{}i = (\psi\ A\ k)\hat{}2 * (\psi\ A\ k)\hat{}(i-2)$ **for** *i*
by (*metis le-add-diff-inverse2 mult.commute power-add*)
then have $\forall i{\in}\{2..n\}.\ ((\psi\ A\ k)\hat{}i)\ mod\ (\psi\ A\ k)\hat{}2 = 0$
by (*meson atLeastAtMost-iff dvd-imp-mod-0 le-imp-power-dvd*)
then have *termszero*: $\forall i{\in}\{2..n\}.\ (int\ (n\ choose\ i) * (\psi\ A\ (Suc\ k) - A*\psi\ A\ k)\hat{}(n-i)$
$*(\psi\ A\ k)\hat{}i*\psi\ A\ i)\ mod\ (\psi\ A\ k)\hat{}2 = 0$
by *auto*
have *disj-sum*: *finite* $\{..(1::nat)\}\ \wedge$ *finite* $\{2..n\} \wedge \{..1\} \cap \{2..n\} = \{\}$
$\wedge\ \{0..1\}\cup\{2..n\} = \{0..n\}$
using ‹$n\neq 0$› **by** *auto*

have $0 = (\psi\ A\ m)\ mod\ (\psi\ A\ k)\ \hat{}\ 2$ **using** *assms* **by** *simp*
also have $... = (\sum i{\leq}n.\ int\ (n\ choose\ i) * (\psi\ A\ (Suc\ k) - A{*}\psi\ A\ k)\ \hat{}\ (n{-}i){*}(\psi\ A\ k)\ \hat{}\ i{*}\psi\ A\ i) mod\ (\psi\ A\ k)\ \hat{}\ 2$
using *n-def sun-lemma2*[*of A k n 0*] **by** *simp*
also have $... = (\sum i{\leq}n.\ int\ (n\ choose\ i) * (\psi\ A\ (Suc\ k) - A{*}\psi\ A\ k)\ \hat{}\ (n{-}i){*}(\psi\ A\ k)\ \hat{}\ i{*}\psi\ A\ i\ mod\ (\psi\ A\ k)\ \hat{}\ 2)\ mod\ (\psi\ A\ k)\ \hat{}\ 2$
using *mod-sum-eq*[*of* $(\lambda i.\ (int\ (n\ choose\ i) * (\psi\ A\ (Suc\ k) - A{*}\psi\ A\ k)\ \hat{}\ (n{-}i){*}(\psi\ A\ k)\ \hat{}\ i{*}\psi\ A\ i\ )\ )\ (\psi\ A\ k)\ \hat{}\ 2\ \{..n\}$] **by** *simp*
also have $... = (\ (\sum i{\leq}1.\ int\ (n\ choose\ i) * (\psi\ A\ (Suc\ k) - A{*}\psi\ A\ k)\ \hat{}\ (n{-}i){*}(\psi\ A\ k)\ \hat{}\ i{*}\psi\ A\ i\ mod\ (\psi\ A\ k)\ \hat{}\ 2\ )$
$+ (\sum i{\in}\{2..n\}.\ int\ (n\ choose\ i) * (\psi\ A\ (Suc\ k) - A{*}\psi\ A\ k)\ \hat{}\ (n{-}i){*}(\psi\ A\ k)\ \hat{}\ i{*}\psi\ A\ i\ mod\ (\psi\ A\ k)\ \hat{}\ 2\ )\ )mod\ (\psi\ A\ k)\ \hat{}\ 2$
using *disj-sum sum.union-disjoint*[*of* $\{..(1::nat)\}\ \{2..n\}$
$(\lambda i.\ (int\ (n\ choose\ i) * (\psi\ A\ (Suc\ k) - A{*}\psi\ A\ k)\ \hat{}\ (n{-}i){*}(\psi\ A\ k)\ \hat{}\ i{*}\psi\ A\ i\ mod\ (\psi\ A\ k)\ \hat{}\ 2)\ )$]
by (*metis* (*mono-tags*, *lifting*) *atMost-atLeast0*)
also have $... = (\sum i{\leq}1.\ int\ (n\ choose\ i) * (\psi\ A\ (Suc\ k) - A{*}\psi\ A\ k)\ \hat{}\ (n{-}i){*}(\psi\ A\ k)\ \hat{}\ i{*}\psi\ A\ i\ mod\ (\psi\ A\ k)\ \hat{}\ 2\ )\ mod\ (\psi\ A\ k)\ \hat{}\ 2$
using *termszero sum.neutral* **by** *simp*
finally have *mod-eq1*: $0 = int(n) * (\psi\ A\ (Suc\ k) - A{*}\psi\ A\ k)\ \hat{}\ (n{-}1){*}(\psi\ A\ k)\ mod\ (\psi\ A\ k)\ \hat{}\ 2$
by *force*
have $(abs\ A)\ \hat{}\ 2 \geq 4$ **using** *assms* **by** *simp*
then have *ABe2*: $abs\ A \geq 2$
by (*smt* (*verit*, *ccfv-SIG*) *abs-1 abs-square-le-1 zero-power2*)
have *psiAknn*: $abs(\psi\ A\ k) > 0$
using *ABe2 assms lucas-strict-monotonicity*[*of abs A k*−*1*] *lucas-symmetry-A-abs*[*of A k*] **by** *simp*
then have $\psi\ A\ k\ dvd\ int(n) * (\psi\ A\ (Suc\ k) - A{*}\psi\ A\ k)\ \hat{}\ (n{-}1)$
using *mod-eq1* **by** (*smt* (*z3*) *dvd-times-right-cancel-iff mod-0-imp-dvd power2-eq-square*)
then have *mod-eq2*:$0 = int(n) * (\psi\ A\ (Suc\ k) - A{*}\psi\ A\ k)\ \hat{}\ (n{-}1)\ mod\ (\psi\ A\ k)$ **by** *simp*
have $(\psi\ A\ (Suc\ k) - A{*}\psi\ A\ k)\ mod\ (\psi\ A\ k) = (\psi\ A\ (Suc\ k))\ mod\ (\psi\ A\ k)$ **by** *algebra*
then have $(\psi\ A\ (Suc\ k) - A{*}\psi\ A\ k)\ \hat{}\ (n{-}1)\ mod\ (\psi\ A\ k) = (\psi\ A\ (Suc\ k))\ \hat{}\ (n{-}1)\ mod\ (\psi\ A\ k)$
using *eq-mod-power* **by** *fast*
then have *mod-eq3*:$0 = int(n) * (\psi\ A\ (Suc\ k))\ \hat{}\ (n{-}1)\ mod\ (\psi\ A\ k)$
using *mod-eq2 mod-mult-cong* **by** *metis*
then have *eq4*:$(\psi\ A\ k)\ dvd\ int(n) * (\psi\ A\ (Suc\ k))\ \hat{}\ (n{-}1)$ **by** *fastforce*
have *coprime* $((\psi\ A\ (Suc\ k))\ \hat{}\ (n{-}1))\ (\psi\ A\ k)$
using *lucas-consec-coprime*[*of A k*] *coprime-power coprime-commute* **by** *blast*
then have $\psi\ A\ k\ dvd\ int\ n$
using *eq4 euclids-lemma coprime-commute mult.commute* **by** *metis*
then show $\psi\ A\ k\ dvd\ int\ m$ **using** *n-def* **by** *simp*
qed
qed

Introducing ψ and χ with both interger parameters. It is a broader definition

but induction is harder, that's why a lot of properties for these vesion are proved in the following lemmas

definition ψ-*int*::*int* $\Rightarrow$ *int* $\Rightarrow$ *int* **where**
ψ-*int A n* = $(-1)\hat{}$(*if n* $\geq$ *0 then 0 else 1*) $*$ ψ *A* (*nat* (*abs n*))
definition χ-*int*::*int* $\Rightarrow$ *int* $\Rightarrow$ *int* **where**
χ-*int A n* = χ *A* (*nat* (*abs n*))

lemma ψ-*int-eq*: ψ-*int A n* = (*if n* $\geq$ *0 then 1 else* -1) $*$ ψ *A* (*nat* (*abs n*))
unfolding ψ-*int-def* **by** *auto*

lemma ψ-*int-ind*:
fixes *A*::*int* **and** *n*::*int*
shows ψ-*int A* (*n*+2) = *A*$*$$\psi$-*int A* (*n*+1) $-$ ψ-*int A n*
proof $-$
consider (*2*) *n*=$-$*2* | (*1*) *n*=$-$*1* | (*pos*) *n* $\geq$ *0* |(*neg*) *n* $\leq$ $-$*3* **by** *linarith*
then show *?thesis*
proof *cases*
case *2*
note *t*=*this*
have ψ *A 2* = *A* $\wedge$ ψ *A 1* = *1* $\wedge$ ψ *A 0* = *0*
by (*smt* (*verit*, *ccfv-SIG*) ψ.*elims* ψ.*simps*(*1*) ψ.*simps*(*2*) *add-diff-cancel-left'* *diff-Suc-1*
mult-cancel-left2 nat-1-add-1 numeral-1-eq-Suc-0 numeral-One zero-neq-numeral)
hence ψ-*int A* ($-$*2*) = $-$*A* $\wedge$ ψ-*int A* ($-$*1*) = $-$*1* $\wedge$ ψ-*int A 0* = *0* **unfolding** ψ-*int-def* **by** *auto*
then show *?thesis* **using** *t* **by** *simp*
next
case *1*
note *t*=*this*
have ψ-*int A* ($-$*1*) = $-$*1* $\wedge$ ψ-*int A 0* = *0* $\wedge$ ψ-*int A 1* = *1* **unfolding** ψ-*int-def* **by** *auto*
then show *?thesis* **using** *t* **by** *simp*
next
case *pos*
have *triv-abs*: *n* $\geq$ *0* $\wedge$ *n*+*1* $\geq$ *0* $\wedge$ *n*+*2* $\geq$ *0* $\wedge$ *nat* (*abs* (*n*+*1*)) = *nat* (*abs n*) +*1*
$\wedge$ *nat* (*abs* (*n*+*2*)) = *nat* (*abs n*) +*2*
by (*simp add*: *nat-add-distrib pos*)
hence ψ-*int A n* = ψ *A* (*nat* (*abs n*)) $\wedge$ ψ-*int A* (*n*+*1*) = ψ *A* (*nat* (*abs n*) +*1*)
$\wedge$ ψ-*int A* (*n*+*2*) = ψ *A* (*nat* (*abs n*) +*2*)
unfolding ψ-*int-def* **by** (*simp add*: *triv-abs mult-numeral-1*)
then show *?thesis* **by** *auto*
next
case *neg*
hence *triv-abs2*: *n* $<$ *0* $\wedge$ *n*+*1* $<$ *0* $\wedge$ *n*+*2* $<$ *0* $\wedge$ *nat* (*abs* (*n*+*1*)) = *nat* (*abs* (*n*+*2*)) +*1*
$\wedge$ *nat* (*abs n*) = *nat* (*abs* (*n*+*2*)) +*2*
by *auto*
hence ψ-*int A n* = $-\psi$ *A* (*nat* (*abs* (*n*+*2*))+*2*) $\wedge$ ψ-*int A* (*n*+*1*) = $-\psi$ *A* (*nat*

```
(abs (n+2)) +1)
      ∧ ψ-int A (n+2) = −ψ A (nat (abs (n+2)))
    unfolding ψ-int-def by auto
  then show ?thesis by auto
qed
qed

lemma χ-int-ind:
  fixes A::int and n::int
  shows χ-int A (n+2) = A∗χ-int A (n+1) − χ-int A n
proof −
  consider (2)  n=−2 | (1) n=−1 | (pos) n ≥ 0 |(neg) n ≤ −3 by linarith
  then show ?thesis
  proof cases
    case 2
    note t=this
    have χ A 2 = A∗A−2 ∧ χ A 1 = A ∧ χ A 0 = 2
      by (metis One-nat-def Suc-1 χ.simps(1) χ.simps(2) χ.simps(3))
   hence χ-int A (−2) = A∗A−2 ∧ χ-int A (−1) = A ∧ χ-int A 0 = 2 unfolding
χ-int-def by auto
    then show ?thesis using t by simp
  next
    case 1
    note t=this
   have χ-int A (−1) = A ∧ χ-int A 0 = 2 ∧ χ-int A 1 = A unfolding χ-int-def
by auto
    then show ?thesis using t by simp
next
  case pos
  have triv-abs: n ≥ 0 ∧ n+1 ≥ 0 ∧ n+2 ≥ 0 ∧ nat (abs (n+1)) = nat (abs n)
+1
      ∧ nat (abs (n+2)) = nat (abs n) +2
    by (simp add: nat-add-distrib pos)
  hence χ-int A n = χ A (nat (abs n)) ∧ χ-int A (n+1) = χ A (nat (abs n) +1)
      ∧ χ-int A (n+2) = χ A (nat (abs n) +2)
    unfolding χ-int-def by (simp add: triv-abs mult-numeral-1)
  then show ?thesis by auto
next
  case neg
  hence triv-abs2: n < 0 ∧ n+1 < 0 ∧ n+2 < 0 ∧ nat (abs (n+1)) = nat (abs
(n+2)) +1
      ∧ nat (abs n) = nat (abs (n+2)) +2
    by auto
  hence χ-int A n = χ A (nat (abs (n+2))+2) ∧ χ-int A (n+1) = χ A (nat (abs
(n+2)) +1)
      ∧ χ-int A (n+2) = χ A (nat (abs (n+2)))
    unfolding χ-int-def by auto
  then show ?thesis by auto
qed
```

qed

lemma *ψ-int-odd*:
fixes *A*::*int* **and** *n*::*int*
shows *ψ-int A* (−*n*) = −*ψ-int A n*
unfolding *ψ-int-def* **by** *auto*

lemma *χ-int-even*:
fixes *A*::*int* **and** *n*::*int*
shows *χ-int A* (−*n*) = *χ-int A n*
unfolding *χ-int-def* **by** *auto*

lemma *technical-lemma1*:
fixes *k*::*int* **and** *r*::*int* **and** *A*::*int*
shows *ψ-int A* (*k*+*r*) = *ψ-int A r* ∗ *χ-int A k* + *ψ-int A* (*k*−*r*)
proof −
have *case-pos*: *ψ-int A* (*int l* +*s*) = *ψ-int A s* ∗ *χ-int A* (*int l*) + *ψ-int A* (*int l*−*s*)
for *l*::*nat* **and** *s*::*int*
proof (*induction l rule*: *ψ-induct*)
case *0*
have *χ-int A* (*int 0*) = *2* ∧ *ψ-int A* (*int 0* − *s*) = − *ψ-int A s*
unfolding *χ-int-def ψ-int-def* **by** *auto*
then show *?case* **by** *auto*
next
case *1*
have *χ-int A* (*int 1*) = *A* ∧ *ψ-int A* (*int 1* − *s*) = −*ψ-int A* (*s* − *1*)
unfolding *χ-int-def* **using** *ψ-int-odd*[*of A s* − *1*] **by** *auto*
then show *?case* **using** *ψ-int-ind*[*of A s* − *1*] **by** *auto*
next
case (*sucsuc l*)
note *t* = *this*
have *ψ-int A* (*int* (*Suc* (*Suc l*)) + *s*) = *ψ-int A* (*int l* + *s* +*2*)
by (*auto simp add*: *algebra-simps*)
hence *ψ-int A* (*int* (*Suc* (*Suc l*))+*s*) = *A*∗*ψ-int A* (*int* (*Suc l*)+*s*) − *ψ-int A* (*int l*+*s*)
using *ψ-int-ind*[*of A int l*+*s*] **by** (*auto simp add*: *algebra-simps*)
hence *ψ-int A* (*int* (*Suc* (*Suc l*))+*s*) = *ψ-int A s*∗(*A*∗*χ-int A* (*int l* + *1*) − *χ-int A* (*int l*))
+ *A*∗*ψ-int A* (*int l* − *s* + *1*) − *ψ-int A* (*int l* − *s*)
using *t* **by** (*auto simp add*: *algebra-simps*)
hence *ψ-int A* (*int* (*Suc* (*Suc l*))+*s*) = *ψ-int A s*∗*χ-int A* (*int l* + *2*) + *ψ-int A* (*int l* − *s* +*2*)
using *ψ-int-ind*[*of A int l* − *s*] *χ-int-ind*[*of A int l*] **by** *auto*
then show *?case* **by** (*auto simp add*: *algebra-simps*)
qed
then show *?thesis*
proof (*cases k*)
case (*nonneg k*)

```
    then show ?thesis using case-pos[of k r] by auto
  next
    case (neg n)
    hence intnatk: int (nat (abs k)) = − k by auto
    hence ψ-int A (−k+r) = ψ-int A r ∗ χ-int A (−k) + ψ-int A (−k−r)
      using case-pos[of nat (abs k) r] by argo
    hence −ψ-int A (k−r) = ψ-int A r ∗ χ-int A k − ψ-int A (k+r)
      using ψ-int-odd[of A k−r] ψ-int-odd[of A k+r] χ-int-even[of A k] by auto
    then show ?thesis by auto
  qed
qed
```

It is now much easier to state the following lemma

```
lemma technical-lemma2:
  fixes r::int and A::int and n::int and q::int and k::int
  assumes n ≠ 0 and χ-int A n = 2∗k
  shows ψ-int A (2∗n+r) mod (χ-int A n) = (−ψ-int A r) mod (χ-int A n)
and ψ-int A (4∗n∗q+r) mod k = ψ-int A r mod k
proof −
  have ψ-int A (2∗n+s) = ψ-int A (n+s) ∗ χ-int A n + ψ-int A (−s) for s
    using technical-lemma1[of A n n+s] by auto
  hence ψ-int A (2∗n+s) mod (χ-int A n) = ψ-int A (−s) mod (χ-int A n) for s
    by (auto simp add: algebra-simps)
  hence first-fact: ψ-int A (2∗n+s) mod (χ-int A n) = (−ψ-int A s) mod (χ-int
A n) for s
    using ψ-int-odd[of A s] by auto
  thus ψ-int A (2∗n+r) mod (χ-int A n) = (−ψ-int A r) mod (χ-int A n) by
auto
  have dvd-mod: a dvd b ⟹ l mod b = m mod b ⟹ l mod a = m mod a for
a::int and b and l and m
    by (metis mod-mod-cancel)
  have cor-modk: ψ-int A (2∗n+s) mod k = (−ψ-int A s) mod k for s
    using assms first-fact dvd-mod[of k χ-int A n ψ-int A (2∗n+s) −ψ-int A s]
by auto
  have cor-modk2: ψ-int A (−2∗n+s) mod k = (−ψ-int A s) mod k for s
    using cor-modk[of −2∗n∗s] using mod-minus-cong by (smt (z3) cor-modk)

  have q-identity: ⋀r. ψ-int A (4∗n∗int s+r) mod k = ψ-int A r mod k
     ∧ ψ-int A (−4∗n∗int s+r) mod k = ψ-int A r mod k for s::nat
  proof (induction s)
    case 0
    then show ?case by auto
  next
    case (Suc s)
    note t = this
    have ψ-int A (4∗n∗int (Suc s)+r) = ψ-int A (4∗n∗int s + (4∗n+r))
     ∧ ψ-int A (−4∗n∗int (Suc s)+r) = ψ-int A (−4∗n∗int s + (−4∗n+r))
      by (auto simp add: algebra-simps)
    hence ψ-int A (4∗n∗int (Suc s)+r) mod k = ψ-int A (4∗n+r) mod k
```

```
    ∧ ψ-int A (−4∗n∗int (Suc s)+r) mod k = ψ-int A (−4∗n+r) mod k
    using t[of 4∗n+r] t[of −4∗n+r] by auto
  hence ψ-int A (4∗n∗int (Suc s)+r) mod k = (−ψ-int A (2∗n+r)) mod k
    ∧ ψ-int A (−4∗n∗int (Suc s)+r) mod k = (−ψ-int A (−2∗n+r)) mod k
    using cor-modk[of 2∗n+r] cor-modk2[of −2∗n+r] by auto
  thus ψ-int A (4∗n∗int (Suc s)+r) mod k = ψ-int A r mod k
    ∧ ψ-int A (−4∗n∗int (Suc s)+r) mod k = ψ-int A r mod k
   using cor-modk[of r] cor-modk2[of r] mod-minus-cong by (metis add.inverse-inverse)
 qed
 thus ψ-int A (4∗n∗q+r) mod k = ψ-int A r mod k
 proof (cases q)
  case (nonneg l)
  then show ?thesis using q-identity[of nat q r] by auto
 next
  case (neg l)
  then show ?thesis using q-identity[of nat (−q) r] by (auto simp add: algebra-simps)
 qed
qed


lemma lucas-solves-pell:
 fixes A :: int
 shows (A ^2−4)∗(ψ-int A m) ^2 + 4 = (χ-int A m) ^2
 unfolding ψ-int-def χ-int-def using lucas-pell-part3 by auto

lemma pell-yields-lucas:
 fixes A Y :: int
 shows (∃ k. (A ^2−4)∗Y^2 + 4 = k^2) = (∃ m. Y = ψ-int A m)


proof (rule iffI)
 assume ∃ k. (A^2 − 4) ∗ Y^2 + 4 = k^2
 then obtain m :: nat where m: Y = ψ A m ∨ Y = − ψ A m
   using lucas-pell-nat(1) by auto
 show ∃ m :: int. Y = ψ-int A m
  apply (rule disjE[OF m])
  subgoal
   apply (rule exI[of - int m]) unfolding ψ-int-def by auto
  subgoal
   apply (rule exI[of - − int m]) unfolding ψ-int-def by auto
  done
next
 assume ∃ m. Y = ψ-int A m
 then obtain m where m: Y = ψ-int A m by auto
 show ∃ k. (A^2 − 4) ∗ Y^2 + 4 = k^2
  unfolding lucas-pell-nat
  apply (cases 0 ≤ m)
  subgoal
   apply (rule exI[of - nat m])
```

```
    using m unfolding ψ-int-def
    by auto
  subgoal
    apply (rule exI[of - nat (− m)])
    using m unfolding ψ-int-def
    by auto
  done
qed

corollary technical-lemma2-part2:
fixes r::int and A::int and n::int and q::int and k::int
  assumes n ≠ 0 and χ-int A n = 2*k
  shows ψ-int A (4*n*q+r) mod k = ψ-int A r mod k
  using technical-lemma2 assms by auto

corollary technical-cor3:
  fixes r::int and A::int and n::int and k::int
  assumes n ≠ 0 and χ-int A n = 2*k
  shows ψ-int A (2*n+r) mod k = (−ψ-int A r) mod k
proof −
  have (ψ-int A (2*n+r) + ψ-int A r) mod χ-int A n = 0 mod χ-int A n
    using technical-lemma2[of n A k r] assms mod-add-cong[of ψ-int A (2*n+r)
χ-int A n
      −ψ-int A r ψ-int A r ψ-int A r] by (auto simp add: algebra-simps)
  hence χ-int A n dvd (ψ-int A (2*n+r) + ψ-int A r) by auto
  hence k dvd (ψ-int A (2*n+r) + ψ-int A r) using assms by auto
  hence (ψ-int A (2*n+r) + ψ-int A r) mod k = 0 mod k by auto
  thus ?thesis using mod-diff-cong[of ψ-int A (2*n+r) + ψ-int A r k 0 ψ-int A
r ψ-int A r]
    by auto
qed

end
theory DFI-square-2
  imports DFI-square-1 HOL.NthRoot
begin

lemma sun-lemma10-rec:
  fixes A::int and n::int and t::int and k::int
  assumes A > 2 and n > 3 and χ-int A n = 2*k
  shows (s mod (4*n) = t mod (4*n) ∨ (s+t) mod (4*n) = (2*n) mod (4*n))
      ⟹ (ψ-int A s mod k = ψ-int A t mod k)
proof −
  have rec1: (s+t) mod (4*n) = (2*n) mod (4*n) ⟹ ψ-int A s mod k = ψ-int
A t mod k
  proof −
    assume s-plus-t: (s+t) mod (4*n) = (2*n) mod (4*n)
    hence s-plus-t-2: (s+t−2*n) mod (4*n) = 0 mod (4*n)
      using mod-diff-cong[of s+t 4*n 2*n 2*n 2*n] by auto
```

hence *s-plus-t-3*: $(4*n)$ *dvd* $(s+t-2*n)$ **by** *auto*
obtain q **where** *q-def*: $(s+t-2*n) = (4*n)*q$ **using** *s-plus-t-3*
by *blast*
have ψ-*int A s mod k*= ψ-*int A* $(4*n*q + 2*n-t)$ *mod k* **using** *q-def* **by** (*smt* (*verit*))
hence ψ-*int A s mod k* = ψ-*int A* $(2*n-t)$ *mod k*
using *technical-lemma2*[*of n A k q*] *assms* **by** (*smt* (*z3*))
hence ψ-*int A s mod k* = $(-\ \psi$-*int A* $(-t))$ *mod k*
using *assms technical-cor3*[*of n A k* $-t$] **by** *auto*
thus ψ-*int A s mod k* = ψ-*int A t mod k* **using** ψ-*int-odd* **by** *auto*
qed
have *rec2*: *s mod* $(4*n)$ = *t mod* $(4*n) \Longrightarrow$ ψ-*int A s mod k* = ψ-*int A t mod k*
proof −
assume *s-eq-t*: *s mod* $(4*n)$ = *t mod* $(4*n)$
obtain q **where** *q-def*: $s = 4*n*q + t$ **using** *s-eq-t* **by** (*smt* (*verit*, *ccfv-SIG*) *mod-eqE*)
thus ψ-*int A s mod k* = ψ-*int A t mod k* **using** *q-def technical-lemma2*[*of n A k q*] *assms* **by** *auto*
qed
show (*s mod* $(4*n)$ = *t mod* $(4*n) \vee$ $(s+t)$ *mod* $(4*n)$ = $(2*n)$ *mod* $(4*n))$
$\Longrightarrow$ (ψ-*int A s mod k* = ψ-*int A t mod k*) **using** *rec1 rec2* **by** *blast*
qed

Some results about Lucas sequences seen as real numbers

lemma *expr-of-*ψ*-and-*χ:
fixes *A*::*int* **and** *n*::*nat* **and** α::*real*
assumes $A > 2$ **and** $\alpha\hat{}2 = A\hat{}2-4$ **and** $\alpha > 0$
defines $\beta p \equiv (A+\alpha)\ /\ 2$ **and** $\beta m \equiv (A-\alpha)\ /\ 2$
shows *real-of-int* $(\psi\ A\ n) = (\beta p\hat{}n-\beta m\hat{}n)\ /\ \alpha \wedge$
real-of-int $(\chi\ A\ n) = \beta p\hat{}n+\beta m\hat{}n$
proof (*induction n rule*: ψ-*induct*)
case *0*
then show *?case* **by** *auto*
next
case *1*
have $\beta p\hat{}1-\beta m\hat{}1 = \beta p - \beta m$ **by** (*auto simp add*: *algebra-simps*)
hence $\beta p\hat{}1-\beta m\hat{}1 = \alpha$ **using** β*p-def* β*m-def*
by (*metis add.commute add-right-cancel diff-add-cancel real-average-minus-first*)
hence $(\beta p\hat{}1-\beta m\hat{}1)\ /\ \alpha = 1 \wedge$ *real-of-int* $(\psi\ A\ 1) = 1$ **using** *assms* **by** *auto*
hence *part-*ψ: *real-of-int* $(\psi\ A\ 1) = (\beta p\hat{}1-\beta m\hat{}1)\ /\ \alpha$ **by** *metis*
have $\beta p\hat{}1+\beta m\hat{}1$ = *real-of-int A* **unfolding** β*p-def* β*m-def*
by (*smt* (*z3*) *field-sum-of-halves power-one-right*)
then show *?case* **using** *part-*ψ **by** *auto*
next
case (*sucsuc n*)
note t = *this*
have β*p-eq*: $A*\beta p-1 = \beta p\hat{}2$
proof −
have $\beta p\hat{}2 = (A+\alpha)*(A+\alpha)/4$ **unfolding** β*p-def* **using** *power2-eq-square*[*of*

$(A+\alpha)$ / 2] **by** *auto*
hence $\beta p\hat{}2 = (A\hat{}2+2*A*\alpha+\alpha\hat{}2)/4$ **by** (*auto simp add*: *power2-eq-square algebra-simps*)
hence $\beta p\hat{}2 = (A\hat{}2 + A*\alpha - 2)/2$ **using** *assms*(2) **by** (*auto simp add*: *algebra-simps*)
hence $\beta p\hat{}2 = A*(A+\alpha)/2 - 1$ **using** *power2-eq-square*[*of A*]
by (*auto simp add*: *algebra-simps diff-divide-distrib*)
thus *?thesis* **using** βp-*def* **by** *auto*
qed
have βm-*eq*: $A*\beta m-1 = \beta m\hat{}2$
proof −
have $\beta m\hat{}2 = (A-\alpha)*(A-\alpha)/4$ **unfolding** βm-*def* **using** *power2-eq-square*[*of* $(A-\alpha)$ / 2] **by** *auto*
hence $\beta m\hat{}2 = (A\hat{}2-2*A*\alpha+\alpha\hat{}2)/4$ **by** (*auto simp add*: *power2-eq-square algebra-simps*)
hence $\beta m\hat{}2 = (A\hat{}2 - A*\alpha - 2)/2$ **using** *assms*(2) **by** (*auto simp add*: *algebra-simps*)
hence $\beta m\hat{}2 = (A*(A-\alpha) - 2)/2$ **by** (*auto simp add*: *algebra-simps power2-eq-square*)
hence $\beta m\hat{}2 = (A*(A-\alpha))/2 - 1$ **by** (*auto simp add*: *diff-divide-distrib*)
hence $\beta m\hat{}2 = A*(A-\alpha)/2 - 1$ **by** *auto*
thus *?thesis* **using** βm-*def* **by** *auto*
qed
have *real-of-int* $(\psi\ A\ (Suc\ (Suc\ n))) = A * (\beta p\hat{}(Suc\ n)-\beta m\hat{}(Suc\ n))$ / α − $(\beta p\hat{}n-\beta m\hat{}n)$ / α
using t **by** *auto*
hence *real-of-int* $(\psi\ A\ (Suc\ (Suc\ n))) = (A * (\beta p*\beta p\hat{}n-\beta m*\beta m\hat{}n))$ / α − $(\beta p\hat{}n-\beta m\hat{}n)$ / α
by *simp*
hence *real-of-int* $(\psi\ A\ (Suc\ (Suc\ n))) = (A * (\beta p*\beta p\hat{}n-\beta m*\beta m\hat{}n) - (\beta p\hat{}n-\beta m\hat{}n))$ / α
using *assms* **by** (*smt* (*verit*, *ccfv-SIG*) *add-divide-distrib*)
hence *real-of-int* $(\psi\ A\ (Suc\ (Suc\ n))) = ((A*\beta p-1)*\beta p\hat{}n - (A*\beta m-1)*\beta m\hat{}n)$ / α
by (*auto simp add*: *algebra-simps*)
hence *real-of-int* $(\psi\ A\ (Suc\ (Suc\ n))) = (\beta p\hat{}2*\beta p\hat{}n - \beta m\hat{}2*\beta m\hat{}n)/\alpha$ **using** βp-*eq* βm-*eq* **by** *auto*
hence *form-*ψ: *real-of-int* $(\psi\ A\ (Suc\ (Suc\ n))) = (\beta p\hat{}(Suc\ (Suc\ n))-\beta m\hat{}(Suc\ (Suc\ n)))$ / α
using *power-add*[*of* βp 2 *n*] *power-add*[*of* βm 2 *n*] **by** (*metis add-2-eq-Suc*)

have *real-of-int* $(\chi\ A\ (Suc\ (Suc\ n))) = A * (\beta p\hat{}(Suc\ n)+\beta m\hat{}(Suc\ n)) - (\beta p\hat{}n+\beta m\hat{}n)$
using t **by** *auto*
hence *real-of-int* $(\chi\ A\ (Suc\ (Suc\ n))) = (A * (\beta p*\beta p\hat{}n+\beta m*\beta m\hat{}n)) - (\beta p\hat{}n+\beta m\hat{}n)$
by *simp*
hence *real-of-int* $(\chi\ A\ (Suc\ (Suc\ n))) = (A*\beta p-1)*\beta p\hat{}n + (A*\beta m-1)*\beta m\hat{}n$
by (*auto simp add*: *algebra-simps*)
hence *real-of-int* $(\chi\ A\ (Suc\ (Suc\ n))) = \beta p\hat{}2*\beta p\hat{}n + \beta m\hat{}2*\beta m\hat{}n$ **using** βp-*eq* βm-*eq* **by** *auto*

hence *form-χ*: *real-of-int* (χ *A* (*Suc* (*Suc n*))) = βp^(*Suc* (*Suc n*))+βm^(*Suc* (*Suc n*))
using *power-add*[*of* βp *2 n*] *power-add*[*of* βm *2 n*] **by** (*metis add-2-eq-Suc*)
then show *?case* **using** *form-ψ form-χ* **by** *auto*
qed

lemma *χ-is-Bigger-sqrt5ψ*: *A* > *2* ⟹ χ *A n* > *sqrt 5*∗ψ *A n*
proof (*cases n*)
case *0*
then show *?thesis* **by** *auto*
next
case (*Suc k*)
note *t* = *this*
assume *A-def*: *A* > *2*
hence *A* ≥ *3* **by** *simp*
hence *A*∗*A* ≥ *9* **using** *mult-mono*[*of 3 A 3 A*] *A-def* **by** *auto*
hence *DiscB4*: *A*^*2*−*4* ≥ *5* **using** *power2-eq-square*[*of A*] **by** *auto*
define α **where** α = *sqrt*(*A*^*2*−*4*)
have *α-def2*: α^*2* = *A*^*2* − *4*
unfolding *α-def* **using** *DiscB4 of-int-0-le-iff real-sqrt-pow2* **by** *fastforce*
hence *αB2*: α ≥ *sqrt 5*
using *DiscB4* **apply** *simp* **by** (*metis DiscB4 α-def of-int-le-iff of-int-numeral real-sqrt-le-iff*)
hence *α-pos*: α > *0*
by (*meson not-numeral-le-zero order.trans real-sqrt-le-0-iff verit-comp-simplify1*(*3*))
define βp **where** βp = (*A*+α)/*2*
define βm **where** βm = (*A*−α)/*2*
have α^*2* < *A*^*2* **using** *α-def2* **by** *auto*
hence α < *A* **using** *real-sqrt-less-iff*[*of* α^*2 A*^*2*] *A-def αB2* **by** *auto*
hence βm > *0* **using** *βm-def* **by** *simp*
hence *βm-pos*: βm^*n* > *0* **by** *auto*
have χ *A n* = βp^*n*+βm^*n*
unfolding *βp-def βm-def* **using** *expr-of-ψ-and-χ*[*of A α n*] *α-def2 A-def α-pos* **by** *simp*
hence χ *A n* > βp^*n* − βm^*n*
using *βm-pos diff-mono*[*of* βp^*n*+βm^*n* βp^*n*+βm^*n 2*∗βm^*n 0*] **by** *linarith*
hence χ *A n* > α ∗ ((βp^*n* − βm^*n*)/ α) **using** *αB2* **by** *auto*
hence χ *A n* > α ∗ ψ *A n*
unfolding *βp-def βm-def* **using** *expr-of-ψ-and-χ*[*of A α n*] *α-def2 A-def α-pos* **by** *simp*
hence χ *A n* > *sqrt 5*∗ ψ *A n*
using *lucas-strict-monotonicity*[*of A k*] *t A-def αB2 mult-left-mono*[*of sqrt 5 α ψ A n*]
by (*smt* (*z3*) *mult-of-int-commute of-int-0-less-iff*)
then show *?thesis* **by** *auto*
qed

lemma *χ-is-Bigger-2ψ*: *A* > *2* ⟹ χ *A n* > *2*∗ψ *A n*
proof −

assume *A-def*: $A > 2$
have *ineq*: $sqrt\ 5 > 2$
by (*metis numeral-less-iff order-refl real-sqrt-four real-sqrt-less-iff semiring-norm*(*79*))
have $\chi\ A\ n > sqrt\ 5 * \psi\ A\ n$ **using** χ*-is-Bigger-sqrt5*ψ *A-def* **by** *auto*
thus *?thesis* **using** *ineq mult-right-mono*[*of 2 sqrt 5* ψ *A n*] *A-def*
using *lucas-monotone2*[*of A 0 n*] **by** *auto*
qed

lemma ψ*-ineq-opti*:
fixes *A*::*int* **and** *n*::*nat*
assumes $A > 2$
shows $5* \psi\ A\ n < 2*\psi\ A\ (n+1)$
proof −
have $A \geq 3$ **using** *assms* **by** *simp*
hence $A*A \geq 9$ **using** *mult-mono*[*of 3 A 3 A*] *assms* **by** *auto*
hence *DiscB4*: $A\hat{}2-4 > 4$ **using** *power2-eq-square*[*of A*] **by** *auto*
define α **where** $\alpha = sqrt(A\hat{}2-4)$
have α*-def2*: $\alpha\hat{}2 = A\hat{}2 - 4$
unfolding α*-def* **using** *DiscB4* **using** *of-int-0-le-iff real-sqrt-pow2* **by** *fastforce*
hence $\alpha B2$: $\alpha > 2$
using *DiscB4* **by** (*metis* α*-def of-int-less-iff of-int-numeral real-sqrt-four real-sqrt-less-iff*)
define βp **where** $\beta p = (A+\alpha)/2$
define βm **where** $\beta m = (A-\alpha)/2$
have $\alpha\hat{}2 < A\hat{}2$ **using** α*-def2* **by** *auto*
hence $\alpha < A$ **using** *real-sqrt-less-iff*[*of* $\alpha\hat{}2\ A\hat{}2$] *assms* $\alpha B2$ **by** *auto*
hence $\beta m > 0$ **using** βm*-def* **by** *simp*
hence βm*-pos*: $\beta m\hat{}n > 0$ **by** *auto*
have $\beta pB2$: $\beta p > 5/2$ **unfolding** βp*-def* **using** *assms* $\alpha B2$ **by** *auto*
have $\psi\ A\ (n+1) = (\beta p\hat{}(n+1) - \beta m\hat{}(n+1))/\ \alpha$
using *expr-of-*ψ*-and-*χ[*of A* α *n+1*] *assms* α*-def2* $\alpha B2$ βp*-def* βm*-def* **by** *fastforce*
hence $\psi\ A\ (n+1) = ((\beta p*\beta p\hat{}n - \beta p*\beta m\hat{}n) + (\beta p*\beta m\hat{}n - \beta m*\beta m\hat{}n))/\ \alpha$
using *power-Suc*[*of* βp *n*] *power-Suc*[*of* βm *n*] **by** *simp*
hence $\psi\ A\ (n+1) = (\beta p*\beta p\hat{}n - \beta p*\beta m\hat{}n)/\ \alpha + (\beta p*\beta m\hat{}n - \beta m*\beta m\hat{}n)/\ \alpha$
using *add-divide-distrib*[*of* $(\beta p*\beta p\hat{}n - \beta p*\beta m\hat{}n)$ $(\beta p*\beta m\hat{}n - \beta m*\beta m\hat{}n)$ α] *assms* **by** *simp*
hence $\psi\ A\ (n+1) = \beta p*(\beta p\hat{}n-\beta m\hat{}n)/\ \alpha + \beta m\hat{}n*(\beta p-\beta m)/\ \alpha$ **by** (*auto simp add*: *algebra-simps*)
hence $\psi\ A\ (n+1) = \beta p*\psi\ A\ n + \beta m\hat{}n*(\beta p-\beta m)/\ \alpha$
using βp*-def* βm*-def expr-of-*ψ*-and-*χ[*of A* α *n*] α*-def2* $\alpha B2$ *assms* **by** *fastforce*
hence $\psi\ A\ (n+1) = \beta p*\psi\ A\ n + \beta m\hat{}n$
using βp*-def* βm*-def diff-divide-distrib*[*of* $A+\alpha$ $A-\alpha$ *2*] $\alpha B2$ **apply** *simp* **by** (*simp add*: βm*-def*)
hence $\psi\ A\ (n+1) > \beta p * \psi\ A\ n$ **using** βm*-pos* **by** *auto*
thus *?thesis* **using** $\beta pB2$ **using** *lucas-monotone2*[*of A 0 n*] *assms mult-right-mono*[*of 5/2* βp ψ *A n*] **by** *auto*
qed

lemma *ψ-doubles*:
fixes *A*::*int* **and** *n*::*nat*
assumes *A* > *2*
shows *2*∗*ψ A n* < *ψ A* (*n*+*1*)
proof −
have *A* ≥ *3* **using** *assms* **by** *simp*
hence *A*∗*A* ≥ *9* **using** *mult-mono*[*of 3 A 3 A*] *assms* **by** *auto*
hence *DiscB4*: *A*^*2*−*4* > *4* **using** *power2-eq-square*[*of A*] **by** *auto*
define *α* **where** *α* = *sqrt*(*A*^*2*−*4*)
have *α-def2*: *α*^*2* = *A*^*2* − *4*
unfolding *α-def* **using** *DiscB4* **using** *of-int-0-le-iff real-sqrt-pow2* **by** *fastforce*
hence *αB2*: *α* > *2*
using *DiscB4* **by** (*metis α-def of-int-less-iff of-int-numeral real-sqrt-four real-sqrt-less-iff*)
define *βp* **where** *βp* = (*A*+*α*)/*2*
define *βm* **where** *βm* = (*A*−*α*)/*2*
have *α*^*2* < *A*^*2* **using** *α-def2* **by** *auto*
hence *α* < *A* **using** *real-sqrt-less-iff*[*of α*^*2 A*^*2*] *assms αB2* **by** *auto*
hence *βm* > *0* **using** *βm-def* **by** *simp*
hence *βm-pos*: *βm*^*n* > *0* **by** *auto*
have *βpB2*: *βp* > *2* **unfolding** *βp-def* **using** *assms αB2* **by** *auto*
have *ψ A* (*n*+*1*) = (*βp*^(*n*+*1*) − *βm*^(*n*+*1*))/ *α*
using *expr-of-ψ-and-χ*[*of A α n+1*] *assms α-def2 αB2 βp-def βm-def* **by** *fastforce*
hence *ψ A* (*n*+*1*) = ((*βp*∗*βp*^*n* − *βp*∗*βm*^*n*) + (*βp*∗*βm*^*n* − *βm*∗*βm*^*n*))/ *α*
using *power-Suc*[*of βp n*] *power-Suc*[*of βm n*] **by** *simp*
hence *ψ A* (*n*+*1*) = (*βp*∗*βp*^*n* − *βp*∗*βm*^*n*)/ *α* + (*βp*∗*βm*^*n* − *βm*∗*βm*^*n*)/ *α*
using *add-divide-distrib*[*of* (*βp*∗*βp*^*n* − *βp*∗*βm*^*n*) (*βp*∗*βm*^*n* − *βm*∗*βm*^*n*) *α*] *assms* **by** *simp*
hence *ψ A* (*n*+*1*) = *βp*∗(*βp*^*n*−*βm*^*n*)/ *α* + *βm*^*n*∗(*βp*−*βm*)/ *α* **by** (*auto simp add*: *algebra-simps*)
hence *ψ A* (*n*+*1*) = *βp*∗*ψ A n* + *βm*^*n*∗(*βp*−*βm*)/ *α*
using *βp-def βm-def expr-of-ψ-and-χ*[*of A α n*] *α-def2 αB2 assms* **by** *fastforce*
hence *ψ A* (*n*+*1*) = *βp*∗*ψ A n* + *βm*^*n*
using *βp-def βm-def diff-divide-distrib*[*of A*+*α A*−*α 2*] *αB2* **apply** *simp* **by** (*simp add*: *βm-def*)
hence *ψ A* (*n*+*1*) > *βp* ∗ *ψ A n* **using** *βm-pos* **by** *auto*
thus *?thesis* **using** *βpB2 lucas-monotone2*[*of A 0 n*] *assms mult-right-mono*[*of 2 βp ψ A n*] **by** *auto*
qed

lemma *distinct-residus*:
fixes *A*::*int* **and** *n*::*int* **and** *k*::*int* **and** *i*::*int* **and** *j*::*int*
assumes *A* > *2* **and** *n* > *3* **and** *χ-int A n* = *2*∗*k* **and** *i*∈{−*n*..*n*} **and** *j*∈{−*n*..*n*} **and** *i*≠*j*
shows *ψ-int A i mod k* ≠ *ψ-int A j mod k*
proof −
have *χ-maj*: *χ A m* > *2* ∗ *ψ A m* **for** *m*::*nat*
using *assms χ-is-Bigger-2ψ*[*of A m*] **by** *simp*

have *non-null*: $l\in\{1..n\} \longrightarrow \psi\text{-}int\ A\ l\ mod\ k \neq 0 \wedge \psi\text{-}int\ A\ (-l)\ mod\ k \neq 0$ **for** l
proof
assume *lLen*: $l\in\{1..n\}$
hence *notz*: $\psi\ A\ (nat\ l) > 0$ **using** *assms lucas-strict-monotonicity*[*of A nat l*−*1*] **by** *force*
hence *nonzero*: $\psi\text{-}int\ A\ l > 0 \wedge \psi\text{-}int\ A\ (-l) < 0$ **unfolding** ψ*-int-def* **using** *lLen* **by** *force*
have $\psi\ A\ (nat\ l) \leq \psi\ A\ (nat\ n)$
using *assms lLen lucas-monotone2*[*of A nat l nat n − nat l*] **by** *force*
hence $\psi\ A\ (nat\ l) < k$
using χ*-maj*[*of nat n*] *assms* **unfolding** χ*-int-def* **by** *auto*
hence $\psi\ A\ (nat\ l)\ mod\ k \neq 0$ **using** *notz* **by** *simp*
thus $\psi\text{-}int\ A\ l\ mod\ k \neq 0 \wedge \psi\text{-}int\ A\ (-l)\ mod\ k \neq 0$
unfolding ψ*-int-def* **using** *lLen zmod-zminus1-not-zero* **by** *simp*
qed

have *prepart2*:$l\in\{1..n\} \wedge m\in\{1..n\} \wedge l<m \Longrightarrow \psi\text{-}int\ A\ l\ mod\ k \neq \psi\text{-}int\ A\ m\ mod\ k$ **for** $l\ m$
proof −
assume *assm*: $l\in\{1..n\} \wedge m\in\{1..n\} \wedge l<m$
then have $\psi\ A\ (nat\ l) < \psi\ A\ (nat\ m)$
using ψ*-int-def assms lucas-monotone2*[*of A nat l nat m − nat l − 1*]
lucas-strict-monotonicity[*of A nat m −1*] **by** *force*
then have *Le*: $\psi\text{-}int\ A\ l < \psi\text{-}int\ A\ m$ **using** *assm* ψ*-int-def* **by** *simp*
have ψ*lBe0*: $\psi\text{-}int\ A\ l \geq 0$
using *assms* ψ*-int-def assm lucas-strict-monotonicity*[*of A nat l − 1*] **by** *simp*
have $\psi\ A\ (nat\ m) \leq \psi\ A\ (nat\ n)$
using *lucas-monotone2*[*of A nat m nat n − nat m*] *assms assm* **by** *force*
then have $\psi\ A\ (nat\ m) < k$
using ‹*n*>*3*› ‹$\chi\text{-}int\ A\ n = 2{*}k$› χ*-maj*[*of nat n*] χ*-int-def*[*of A n*] **by** *simp*
then have $\psi\text{-}int\ A\ m < k$ **using** ψ*-int-def assm* **by** *simp*
then have $0 < \psi\text{-}int\ A\ m - \psi\text{-}int\ A\ l \wedge \psi\text{-}int\ A\ m - \psi\text{-}int\ A\ l < k$ **using** *Le* ψ*lBe0* **by** *auto*
then have $(\psi\text{-}int\ A\ m - \psi\text{-}int\ A\ l)\ mod\ k \neq 0$ **by** *simp*
then show $\psi\text{-}int\ A\ l\ mod\ k \neq \psi\text{-}int\ A\ m\ mod\ k$ **by** (*metis dvd-imp-mod-0 mod-eq-dvd-iff*)
qed
then have *part2*: $l\neq m \Longrightarrow l\in\{1..n\} \wedge m\in\{1..n\} \Longrightarrow \psi\text{-}int\ A\ l\ mod\ k \neq \psi\text{-}int\ A\ m\ mod\ k$ **for** $l\ m$
proof (*cases l*<*m*)
case *True*
then show $l\in\{1..n\} \wedge m\in\{1..n\} \Longrightarrow \psi\text{-}int\ A\ l\ mod\ k \neq \psi\text{-}int\ A\ m\ mod\ k$
using *prepart2*[*of l m*] **by** *simp*
next
case *False*
note *t*=*this*
then show $l\neq m \Longrightarrow l\in\{1..n\} \wedge m\in\{1..n\} \Longrightarrow \psi\text{-}int\ A\ l\ mod\ k \neq \psi\text{-}int\ A\ m$

mod k
proof −
assume *l≠m*
then have *m<l* **using** *t* **by** *simp*
then show *l∈{1..n} ∧ m∈{1..n} ⟹ ψ-int A l mod k ≠ ψ-int A m mod k*
using *prepart2*[*of m l*]
by *simp*
qed
qed

have *small-result*: *0 < a ∧ a < 2∗k ∧ a ≠ k ⟹ ¬ k dvd a* **for** *a*
proof (*rule ccontr*)
assume *assm*: *0 < a ∧ a < 2∗k ∧ a ≠ k*
have *k-pos*: *k > 0* **using** *χ-maj*[*of nat n*] *assms χ-int-def*[*of A n*] *lucas-strict-monotonicity*[*of A nat n−1*]
by *auto*
assume *hypoth*: *¬ ¬ k dvd a*
then obtain *x* **where** *x-def*: *k∗x = a* **by** *force*
consider (*neg*) *x ≤ 0* | (*1*) *x=1* | (*pos*) *x ≥ 2* **by** *linarith*
then show *False*
proof (*cases*)
case *neg*
then show *?thesis* **using** *assm* **by** (*smt* (*verit*, *del-insts*) *x-def zero-less-mult-pos*)
next
case *1*
then show *?thesis* **using** *assm x-def* **by** *simp*
next
case *pos*
then show *?thesis* **using** *assm k-pos mult-left-mono*[*of 2 x k*] *x-def* **by** *auto*
qed
qed

have *rel-ψ-χ-lin*:*2∗ψ A x = A∗ψ A* (*x+1*) *− χ A* (*x+1*) **for** *x*
proof (*induction x rule*: *ψ-induct*)
case *0*
then show *?case* **by** *auto*
next
case *1*
then show *?case* **by** *auto*
next
case (*sucsuc n*)
note *t = this*
have *A∗ψ A* (*n+2+1*) *− χ A* (*n+2+1*) *= A∗*(*A∗ψ A* (*n+1+1*) *− ψ A* (*n+1*)) *− *(*A∗χ A* (*n+1+1*) *− χ A* (*n+1*))
by *auto*
hence *A∗ψ A* (*n+2+1*) *− χ A* (*n+2+1*) *= A∗*(*A∗ψ A* (*n+1+1*) *− χ A* (*n+1+1*)) *− *(*A∗ψ A* (*n+1*) *− χ A* (*n+1*))
by (*auto simp add*: *algebra-simps*)
then show *?case* **using** *t* **by** *auto*

qed

have *prepart3*: $l \in \{1..n-1\} \Longrightarrow (\psi\text{-}int\ A\ l + \psi\text{-}int\ A\ n)\ mod\ k \neq 0\ mod\ k$ **for** l
proof −
have ψ*-frac*: $\psi\ A\ m \leq \psi\ A\ (m+1) * 2/5$ **for** m **using** *assms* ψ*-ineq-opti*[*of A m*] **by** *auto*
assume *assm*: $l \in \{1..n-1\}$
have ψ*-pos*: $0 < \psi\text{-}int\ A\ l \wedge 0 < \psi\text{-}int\ A\ n \wedge 0 < \psi\ A\ (nat\ n)$
using *assms assm* ψ*-int-def*[*of A l*] ψ*-int-def*[*of A n*] *lucas-strict-monotonicity*[*of A nat l−1*]
lucas-strict-monotonicity[*of A nat n−1*] **by** *auto*
have *lesser-2k*: $\psi\text{-}int\ A\ l + \psi\text{-}int\ A\ n < 2*k$
using ψ*-int-def*[*of A l*] ψ*-int-def*[*of A n*] *assm assms lucas-monotone4*[*of A nat l nat n*]
χ*-maj*[*of nat n*] χ*-int-def*[*of A n*] **apply** *simp* **by** *linarith*
have *pos*: $\psi\text{-}int\ A\ l + \psi\text{-}int\ A\ n > 0$ **using** ψ*-pos* **by** *auto*
consider (*small*) $l \leq n-3$ | (*n-m2*) $l = n-2$ | (*n-m1*) $l = n-1$ **using** *assm* **by** *force*
then show *?thesis*
proof *cases*
case *small*
hence *ineq1*: $\psi\text{-}int\ A\ l + \psi\text{-}int\ A\ n \leq \psi\ A\ (nat\ n - 3) + \psi\ A\ (nat\ n)$
using *assm* ψ*-int-def*[*of A l*] ψ*-int-def*[*of A n*] *lucas-monotone4*[*of A nat l nat n−3*] *assms*
by *auto*
have *nat-eq*: $Suc\ (nat\ n-3) = nat\ n-2 \wedge Suc\ (nat\ n-2) = nat\ n-1 \wedge Suc\ (nat\ n-1) = nat\ n$
using *assms* **by** *auto*
hence $\psi\ A\ (nat\ n-3) \leq \psi\ A\ (nat\ n) * 8/125$
using *assms* ψ*-frac*[*of nat n−3*] ψ*-frac*[*of nat n − 2*] ψ*-frac*[*of nat n − 1*] **by** *auto*
hence *ineq2*: $\psi\text{-}int\ A\ l + \psi\text{-}int\ A\ n \leq \psi\ A\ (nat\ n)*(133/125)$ **using** *ineq1* **by** *auto*
have *ineq3*: $133/125 < sqrt\ (5)\ /\ 2$
proof −
have *obv*: $125*sqrt\ 5 > 0 \wedge (266::int) > 0$ **by** *simp*
have *obv2*: $(133/125 < sqrt\ (5)\ /\ 2) = (266 < 125*sqrt\ 5)$ **by** *auto*
have $266\hat{}2 < (125*sqrt\ 5)\hat{}2$ **by** *auto*
hence $266 < 125*sqrt\ 5$ **using** *obv* **by** (*smt* (*z3*) *pos2 power-mono-iff*)
then show *?thesis* **using** *obv2* **by** *auto*
qed
hence $\psi\ A\ (nat\ n)*(133/125) < \psi\ A\ (nat\ n)*(sqrt\ 5\ /\ 2)$
using ψ*-pos mult-strict-left-mono*[*of* $\psi\ A\ (nat\ n)$ *133/125 sqrt 5 / 2*] **by** *auto*
hence $\psi\text{-}int\ A\ l + \psi\text{-}int\ A\ n < (\psi\ A\ (nat\ n) * sqrt\ 5)\ /\ 2$ **using** *ineq2* **by** *auto*
hence $\psi\text{-}int\ A\ l + \psi\text{-}int\ A\ n < k$
using *assms* χ*-int-def*[*of A n*] χ*-is-Bigger-sqrt5*ψ[*of A nat n*]
by (*simp add*: *mult-of-int-commute*)

then show *?thesis* **using** *small-result*[*of* ψ-*int A l* + ψ-*int A n*] *pos lesser-2k*
by *auto*
next
case *n-m2*
have ψ-*int A* (*n*−2) + ψ-*int A n* ≠ *k*
proof (*rule ccontr*)
have *dev-nat*: *nat* (*n* − 2) = *nat n* − 2 ∧ *nat n* = *Suc* (*Suc* (*nat n* − 2))
using *assms* **by** *auto*
assume *hypoth*: ¬(ψ-*int A* (*n*−2) + ψ-*int A n* ≠ *k*)
hence 2∗(ψ *A* (*nat* (*n* − 2)) + ψ *A* (*nat n*)) = χ *A* (*nat n*)
using *assms* ψ-*int-def*[*of A n*−2] ψ-*int-def*[*of A n*] χ-*int-def*[*of A n*] **by**
auto
hence 2∗(ψ *A* (*nat n* − 2) + ψ *A* (*Suc* (*Suc* (*nat n* − 2)))) = χ *A* (*nat n*)
using *dev-nat* **by** *simp*
hence *A*∗(2∗ψ *A* (*Suc* (*nat n* − 2))) = χ *A* (*nat n*) **by** *auto*
hence *A*∗(*A*∗ψ *A* (*Suc* (*nat n* − 2) + 1) − χ *A* (*Suc* (*nat n* − 2) + 1))
= χ *A* (*nat n*)
using *rel-*ψ*-*χ*-lin*[*of Suc* (*nat n* − 2)] **by** *auto*
hence *A*∗(*A*∗ψ *A* (*nat n*) − χ *A* (*nat n*)) = χ *A* (*nat n*)
using *assms* **by** (*metis Suc-eq-plus1 dev-nat*)
hence *A* ^ 2∗ψ *A* (*nat n*) = (*A*+1) ∗ χ *A* (*nat n*)
using *power2-eq-square*[*of A*] **by** (*auto simp add*: *algebra-simps*)
hence (*A* ^ 2∗ψ *A* (*nat n*)) ^ 2 = (*A*+1) ^ 2∗((*A* ^ 2−4)∗ψ *A* (*nat n*) ^ 2 + 4)
using *power-mult-distrib*[*of A*+1 χ *A* (*nat n*) 2] *lucas-pell-part3*[*of A nat*
n] **by** *auto*
hence ((*A*+1) ^ 2∗(*A* ^ 2−4)−*A* ^ 2∗*A* ^ 2)∗ψ *A* (*nat n*) ^ 2 + 4∗(*A*+1) ^ 2 =
0
by (*auto simp add*: *algebra-simps*)
hence ((*A* ^ 2+2∗*A*+1)∗(*A* ^ 2−4)−*A* ^ 2∗*A* ^ 2) ∗ψ *A* (*nat n*) ^ 2 + 4∗(*A*+1) ^ 2
= 0
using *power2-sum*[*of A* 1] **apply** *simp* **by** (*smt* (*verit*, *best*))
hence (2∗*A*∗*A*∗*A*−3∗*A*∗*A*−8∗*A*−4)∗ψ *A* (*nat n*) ^ 2 + 4∗(*A*+1) ^ 2 = 0
using *power2-eq-square*[*of A*] **by** (*auto simp add*: *algebra-simps*)
hence *equation*: ((2∗*A*+1)∗(*A*+1)∗(*A*−3)−1)∗ψ *A* (*nat n*) ^ 2 + 4∗(*A*+1) ^ 2
= 0
by (*auto simp add*: *algebra-simps*)
have 4∗(*A*+1) ^ 2 ≥ 64
using *assms power2-eq-square*[*of A*+1] *mult-mono*[*of* 4 *A*+1 4 *A*+1] **by**
auto
hence ((2∗*A*+1)∗(*A*+1)∗(*A*−3)−1)∗ψ *A* (*nat n*) ^ 2 ≤ −64 **using** *equation*
by *auto*
hence ((2∗*A*+1)∗(*A*+1)∗(*A*−3)−1) < 0
by (*smt* (*verit*, *ccfv-SIG*) ψ-*pos assms*(1) *int-distrib*(2) *pos-zmult-eq-1-iff*
zero-less-power zmult-zless-mono2)
hence *ineq2*: (2∗*A*+1)∗(*A*+1)∗(*A*−3) ≤ 0 **by** *auto*
have (2∗*A*+1)∗(*A*+1) > 0 **using** *assms* **by** *auto*
hence *A*=3 **using** *ineq2 assms* **by** (*smt* (*verit*) *mult-pos-pos*)
hence 8 ^ 2 = ψ 3 (*nat n*) ^ 2 **using** *equation* **by** *auto*
hence *eq*: 8 = ψ 3 (*nat n*) **using** *lucas-strict-monotonicity*[*of* 3 *nat n*−1]

assms **apply** *simp*
by (*smt* (*verit*, *ccfv-SIG*) ‹$8^2 = (\psi\ 3\ (nat\ n))^2$› *power2-eq-imp-eq*)
have $\psi\ 3\ 3 = \psi\ 3\ (Suc\ (Suc\ (Suc\ 0)))$ **using** *numeral-3-eq-3* **by** *presburger*
hence *eq2*: $\psi\ 3\ 3 = \psi\ 3\ (nat\ n)$ **using** *eq* **by** *auto*
hence $\psi\ 3\ (Suc\ 3) > \psi\ 3\ (nat\ n) \wedge nat\ n \geq Suc\ 3$ **using** *lucas-strict-monotonicity*[*of 3 3*] *assms* **by** *auto*
then show *False* **using** *lucas-monotone4*[*of 3 Suc 3 nat n*] *assms* **by** *auto*
qed
hence ψ-*int A l* + ψ-*int A n* $\neq$ *k* **using** *n-m2* **by** *auto*
then show *?thesis* **using** *pos lesser-2k small-result*[*of* ψ-*int A l* + ψ-*int A n*] **by** *auto*
next
case *n-m1*
have ψ-*int A* (*n*−*1*) + ψ-*int A n* $\neq$ *k*
proof (*rule ccontr*)
have *dev-nat*: *nat* (*n*−*1*) = *nat n* − *1* ∧ *nat n* − *1* +*1* = *nat n* **using** *assms* **by** *auto*
assume *hypoth*: ¬(ψ-*int A* (*n*−*1*) + ψ-*int A n* $\neq$ *k*)
hence *2*∗ψ *A* (*nat n* − *1*) + *2*∗ψ *A* (*nat n*) = χ *A* (*nat n*)
using *assms* ψ-*int-def*[*of A n*−*1*] ψ-*int-def*[*of A n*] χ-*int-def*[*of A n*] *dev-nat* **by** *auto*
hence *A*∗ψ *A* (*nat n*) − χ *A* (*nat n*) + *2*∗ψ *A* (*nat n*) = χ *A* (*nat n*)
using *rel-*ψ*-*χ*-lin*[*of nat n* − *1*] *dev-nat* **by** *auto*
hence (*A*+*2*)∗ψ *A* (*nat n*) = *2*∗χ *A* (*nat n*) **by** (*auto simp add*: *algebra-simps*)
hence (*A*+*2*) ^*2*∗ψ *A* (*nat n*) ^*2* = *4*∗((*A* ^*2*−*4*)∗ψ *A* (*nat n*) ^*2* +*4*)
using *lucas-pell-part3*[*of A nat n*] *power-mult-distrib*[*of A*+*2* ψ *A* (*nat n*) *2*]
by (*auto simp add*: *algebra-simps*)
hence (*4*∗(*A* ^*2*−*4*) − (*A*+*2*) ^*2*)∗ψ *A* (*nat n*) ^*2* + *16* = *0* **by** (*auto simp add*: *algebra-simps*)
hence (*3*∗*A*∗*A*−*20* − *4*∗*A*)∗ψ *A* (*nat n*) ^*2* +*16* = *0*
using *power2-eq-square*[*of A*] *power2-sum*[*of A 2*] **by** (*auto simp add*: *algebra-simps*)
hence *equation*: (*3*∗*A*−*10*)∗(*A*+*2*)∗ψ *A* (*nat n*) ^*2* + *16* = *0* **by** (*auto simp add*: *algebra-simps*)
have (*A*+*2*)∗ψ *A* (*nat n*) ^*2* > *0* **using** ψ-*pos assms* **by** *auto*
hence *3*∗*A*−*10* < *0*
using *equation* **by** (*smt* (*verit*, *ccfv-SIG*) *int-distrib*(*4*) *zmult-zless-mono2*)
hence *A*=*3* **using** *assms* **by** *auto*
hence *5*∗ψ *3* (*nat n*) ^*2* = *16* **using** *equation* **by** *auto*
hence *0 mod 5* = (*16*::*int*) *mod 5* **by** *presburger*
then show *False* **by** *simp*
qed
hence ψ-*int A l* + ψ-*int A n* $\neq$ *k* **using** *n-m1* **by** *auto*
then show *?thesis* **using** *pos lesser-2k small-result*[*of* ψ-*int A l* + ψ-*int A n*] **by** *auto*
qed
qed

have *part3*: $l\in\{1..n\} \wedge m\in\{-n..-1\} \Longrightarrow \psi\text{-}int\ A\ l\ mod\ k \neq \psi\text{-}int\ A\ m\ mod\ k$ **for** $l\ m$
proof −
assume *assm*: $l\in\{1..n\} \wedge m\in\{-n..-1\}$
define h **where** $h = -m$
consider (*lesser-n*) $l<n \wedge h<n$ | (*less-eq*) $l<n \wedge h=n$ | (*eq-less*) $l=n \wedge h<n$
| (*eq-n*) $l=n \wedge h=n$
using *assm h-def* **apply** *simp* **by** *linarith*
thus *?thesis*
proof *cases*
case *lesser-n*
have *lh-pos*: $l > 0 \wedge h > 0$ **using** *assm h-def* **by** *auto*
have $\psi\text{-}int\ A\ l - \psi\text{-}int\ A\ m = \psi\text{-}int\ A\ l + \psi\text{-}int\ A\ h$ **using** ψ*-int-odd h-def* **by** *simp*
hence *eq1*: $\psi\text{-}int\ A\ l - \psi\text{-}int\ A\ m = \psi\ A\ (nat\ l) + \psi\ A\ (nat\ h)$ **using** *lh-pos* ψ*-int-def* **by** *auto*
hence *diff-pos*: $\psi\text{-}int\ A\ l - \psi\text{-}int\ A\ m > 0$
using *assms lucas-strict-monotonicity*[*of A nat l* − *1*] *lucas-strict-monotonicity*[*of A nat h* − *1*] *lh-pos* **by** *auto*
have $nat\ l \leq nat\ n - 1 \wedge nat\ h \leq nat\ n - 1$ **using** *lesser-n h-def* **by** *auto*
have *obv*: $nat\ n - 1 + 1 = nat\ n \wedge nat\ n - 1 > 2$ **using** *assms* **by** *auto*
have $\psi\ A\ (nat\ l) + \psi\ A\ (nat\ h) \leq 2*\psi\ A\ (nat\ n-1)$
using *lesser-n lucas-monotone4*[*of A nat l nat n*−*1*] *lucas-monotone4*[*of A nat h nat n*−*1*]
assms **by** *linarith*
hence $\psi\text{-}int\ A\ l - \psi\text{-}int\ A\ m < \psi\ A\ (nat\ n)$ **using** *eq1* ψ*-doubles*[*of A nat n*−*1*] *assms obv*
by *auto*
hence $\psi\text{-}int\ A\ l - \psi\text{-}int\ A\ m < k$ **using** χ*-maj*[*of nat n*] χ*-int-def*[*of A n*] *assms*
by *simp*
hence $(\psi\text{-}int\ A\ l - \psi\text{-}int\ A\ m)\ mod\ k \neq 0\ mod\ k$ **using** *diff-pos* **by** *auto*
then show *?thesis* **using** *mod-diff-cong* **by** *fastforce*
next
case *less-eq*
hence $(\psi\text{-}int\ A\ h + \psi\text{-}int\ A\ l)\ mod\ k \neq 0\ mod\ k$
using *prepart3*[*of l*] *assm* **by** (*simp add*: *add.commute*)
hence $(\psi\text{-}int\ A\ l - \psi\text{-}int\ A\ m)\ mod\ k \neq 0\ mod\ k$ **using** *h-def* ψ*-int-odd* **by** *auto*
then show *?thesis* **using** *mod-diff-cong* **by** *fastforce*
next
case *eq-less*
hence $(\psi\text{-}int\ A\ h + \psi\text{-}int\ A\ l)\ mod\ k \neq 0\ mod\ k$
using *prepart3*[*of h*] *assm h-def* **by** (*simp add*: *add.commute*)
hence $(\psi\text{-}int\ A\ l - \psi\text{-}int\ A\ m)\ mod\ k \neq 0\ mod\ k$ **using** *h-def* ψ*-int-odd* **by** *auto*
then show *?thesis* **using** *mod-diff-cong* **by** *fastforce*
next
case *eq-n*

have $\psi B2$: $\psi\ A\ (nat\ n) > 2$ **using** *lucas-monotone3*[*of A nat n*] *assms* **by** *auto*
have *fund*: $(20-A\hat{}2)*(\psi\ A\ (nat\ n))\hat{}2 \neq 4$
proof (*cases* $20-A\hat{}2 = 0$)
case *True*
then show *?thesis* **by** *simp*
next
case *False*
have *abs* $((20 - A\hat{}2)*(\psi\ A\ (nat\ n))\hat{}2) = abs\ (20 - A\hat{}2) * (\psi\ A\ (nat\ n) * \psi\ A\ (nat\ n))$
using *abs-mult*[*of* $20-A\hat{}2\ \psi\ A\ (nat\ n)\hat{}2$] $\psi B2$ *power2-eq-square*[*of* $\psi\ A\ (nat\ n)$]
abs-mult[*of* $\psi\ A\ (nat\ n)\ \psi\ A\ (nat\ n)$] **by** *auto*
hence *abs* $((20 - A\hat{}2)*(\psi\ A\ (nat\ n))\hat{}2) \geq \psi\ A\ (nat\ n) * \psi\ A\ (nat\ n)$
using *False mult-right-mono*[*of 1 abs* $(20 - A\hat{}2)\ \psi\ A\ (nat\ n) * \psi\ A\ (nat\ n)$] **by** *auto*
hence *abs* $((20 - A\hat{}2)*(\psi\ A\ (nat\ n))\hat{}2) > 4$
using $\psi B2$ *mult-strict-mono*[*of 2* $\psi\ A\ (nat\ n)\ 2\ \psi\ A\ (nat\ n)$]
by (*auto simp add*: *mult-strict-mono*)
then show *?thesis* **by** *auto*
qed
have $(20-A\hat{}2)*(\psi\ A\ (nat\ n))\hat{}2 = 4\hat{}2*(\psi\ A\ (nat\ n))\hat{}2 - (A\hat{}2-4)*(\psi\ A\ (nat\ n))\hat{}2$
by (*auto simp add*: *algebra-simps*)
hence $(20-A\hat{}2)*(\psi\ A\ (nat\ n))\hat{}2 = (4*(\psi\ A\ (nat\ n)))\hat{}2 - \chi\ A\ (nat\ n)\hat{}2 + 4$
using *lucas-pell-part3*[*of A nat n*] **by** (*auto simp add*: *algebra-simps*)
hence $(4*(\psi\ A\ (nat\ n)))\hat{}2 - \chi\ A\ (nat\ n)\hat{}2 \neq 0$ **using** *fund* **by** *auto*
hence $4*\psi\ A\ (nat\ n) \neq \chi\ A\ (nat\ n)$ **by** *algebra*
hence *not-k*: $\psi\text{-}int\ A\ l + \psi\text{-}int\ A\ h \neq k$ **using** *assms eq-n* ψ*-int-def*[*of A n*]
using χ*-int-def add-cancel-right-right left-minus-one-mult-self*
mult.commute mult.left-commute power-add **by** *fastforce*
have *pos*: $\psi\text{-}int\ A\ l + \psi\text{-}int\ A\ h > 0$ **using** *eq-n* ψ*-int-def*[*of A n*] $\psi B2$ **by** *auto*
have *lesser-*χ: $\psi\text{-}int\ A\ l + \psi\text{-}int\ A\ h < 2*k$
using *assms eq-n* χ*-maj*[*of nat n*] ψ*-int-def*[*of A n*] χ*-int-def*[*of A n*] **by** *auto*
have *not-dvd*: $\neg\ (k\ dvd\ (\psi\text{-}int\ A\ l + \psi\text{-}int\ A\ h))$
using *small-result*[*of* $\psi\text{-}int\ A\ l + \psi\text{-}int\ A\ h$] *pos not-k lesser-*$\chi$ **by** *auto*
hence $(\psi\text{-}int\ A\ l - \psi\text{-}int\ A\ m)\ mod\ k \neq 0\ mod\ k$ **using** ψ*-int-odd*[*of A m*] *h-def* **by** *auto*
then show *?thesis* **using** *mod-diff-cong* **by** *fastforce*
qed
qed

consider (*null-pos*) $i=0 \wedge j>0$ | (*null-neg*) $i=0 \wedge j<0$ |
(*pos-null*) $i>0 \wedge j=0$ | (*pos-pos*) $i>0 \wedge j>0 \wedge i\neq j$ | (*pos-neg*) $i>0 \wedge j<0$ |
(*neg-null*) $i<0 \wedge j=0$ | (*neg-pos*) $i<0 \wedge j>0$ | (*neg-neg*) $i<0 \wedge j<0 \wedge$

```
i≠j
  using ‹j ∈ {− n..n}› ‹i ∈ {− n..n}› by (metis assms(6) linorder-neqE-linordered-idom)
 then show ?thesis
 proof cases
  case null-pos
  then show ?thesis using  ψ-int-def non-null[of j] ‹j ∈ {− n..n}› by force
  next
   case null-neg
   then show ?thesis using ψ-int-def non-null[of −j] ‹j ∈ {− n..n}› by force
  next
   case pos-null
   then show ?thesis using ψ-int-def non-null[of i] ‹i ∈ {− n..n}› by force
  next
   case pos-pos
    then show ?thesis using ψ-int-def part2[of i j] ‹i ∈ {− n..n}› ‹j ∈ {−
n..n}› by force
  next
   case pos-neg
   then show ?thesis using part3[of i j] assms by auto
  next
   case neg-null
   then show ?thesis using ψ-int-def non-null[of −i] ‹i ∈ {− n..n}› by force
  next
   case neg-pos
   then show ?thesis using part3[of j i] assms by auto
  next
   case neg-neg
   then show ?thesis using ψ-int-def part2[of −i −j] ‹i ∈ {− n..n}› ‹j ∈ {−
n..n}›
     apply simp by (metis equation-minus-iff mod-minus-cong)
 qed
qed


lemma case-lesser-than-4n:
 fixes A::int and n::int and s::int  and t::int and k::int
 assumes A > 2 and n > 3 and χ-int A n = 2*k and 0≤s ∧ s<4*n ∧ 0≤t ∧
t<4*n
 shows (ψ-int A s mod k = ψ-int A t mod k)
    ⟹ (s mod (4*n) = t mod (4*n) ∨ (s+t) mod (4*n) = (2*n) mod (4*n))
proof −
 assume ψ-eq: ψ-int A s mod k = ψ-int A t mod k
 consider (11) 0≤ s ∧ s≤n ∧ 0≤t   ∧ t≤n   | (21) n≤s ∧ s≤3*n ∧ 0≤t   ∧
t≤n   | (41) 3*n≤s ∧ s≤4*n ∧ 0≤t   ∧ t≤n   |
          (12) 0≤ s ∧ s≤n ∧ n≤t   ∧ t≤3*n | (22) n≤s ∧ s≤3*n ∧ n≤t   ∧
t≤3*n | (42) 3*n≤s ∧ s≤4*n ∧ n≤t ∧ t≤3*n |
          (14) 0≤ s ∧ s≤n ∧ 3*n≤t ∧ t≤4*n | (24) n≤s ∧ s≤3*n ∧ 3*n≤t ∧
t≤4*n | (44) 3*n≤s ∧ s≤4*n ∧ 3*n≤t ∧ t≤4*n
   using assms by (smt (verit) 11)
```

thus *?thesis*
proof (*cases*)
case *11*
have *s=t* **using** *distinct-residus*[*of A n k s t*] *assms ψ-eq 11* **by** *auto*
then show *?thesis* **by** *simp*
next
case *21*
have *ineq*: $-n \leq 2*n-s \wedge 2*n-s \leq n$ **using** *21* **by** *auto*
have *ψ-int A* $(2*n-s)$ *mod k* = *ψ-int A t mod k*
using *assms ψ-eq technical-lemma2*[*of n A k* $-s$] *ψ-int-odd*[*of A* $-s$]
by (*smt* (*z3*) *sun-lemma10-rec*)
hence $t = 2*n-s$ **using** *distinct-residus*[*of A n k* $2*n-s$ *t*] *21 ineq assms* **by** *auto*
then show *?thesis* **by** *simp*
next
case *41*
have *ineq*: $-n \leq -4*n+s \wedge -4*n+s \leq n$ **using** *41* **by** *auto*
have *ψ-int A* $(-4*n+s)$ *mod k* = *ψ-int A t mod k*
using *technical-lemma2-part2*[*of n A k* -1 *s*] *ψ-eq assms* **by** *auto*
hence $t = -4*n+s$ **using** *distinct-residus*[*of A n k* $-4*n+s$ *t*] *41 ineq assms* **by** *auto*
then show *?thesis* **by** *simp*
next
case *12*
have *ineq*: $-n \leq 2*n-t \wedge 2*n-t \leq n$ **using** *12* **by** *auto*
have *ψ-int A* $(2*n-t)$ *mod k* = *ψ-int A s mod k*
using *assms ψ-eq technical-lemma2*[*of n A k* $-t$] *ψ-int-odd*[*of A* $-t$]
by (*smt* (*z3*) *sun-lemma10-rec*)
hence $s = 2*n-t$ **using** *distinct-residus*[*of A n k* $2*n-t$ *s*] *12 ineq assms* **by** *auto*
then show *?thesis* **by** *simp*
next
case *22*
have *ineq*: $-n \leq 2*n-s \wedge 2*n-s \leq n \wedge -n \leq 2*n-t \wedge 2*n-t \leq n$ **using** *22* **by** *auto*
have *ψ-int A* $(2*n-s)$ *mod k* = *ψ-int A* $(2*n-t)$ *mod k*
using *assms ψ-eq technical-lemma2*[*of n A k* $-t$] *technical-lemma2*[*of n A k* $-s$]
ψ-int-odd[*of A* $-s$] *ψ-int-odd*[*of A* $-s$] **by** (*smt* (*z3*) *sun-lemma10-rec*)
hence $2*n-s = 2*n-t$ **using** *distinct-residus*[*of A n k* $2*n-t$ $2*n-s$] *ineq assms* **by** *auto*
then show *?thesis* **by** *simp*
next
case *42*
have *ineq*: $-n \leq s-4*n \wedge s-4*n \leq n \wedge -n \leq 2*n-t \wedge 2*n-t \leq n$ **using** *42* **by** *auto*
have *ψ-int A* $(s-4*n)$ *mod k* = *ψ-int A* $(2*n-t)$ *mod k*
using *assms ψ-eq technical-lemma2*[*of n A k* $-t$] *technical-lemma2-part2*[*of n A k* -1 *s*]

```
    ψ-int-odd[of A −t] by (smt (z3) sun-lemma10-rec)
  hence s−4∗n = 2∗n − t using distinct-residus[of A n k s−4∗n 2∗n−t] ineq
assms by auto
  then show ?thesis by simp
next
  case 14
  have ineq: −n ≤ −4∗n+t ∧ −4∗n+t ≤n using 14 by auto
  have ψ-int A (−4∗n+t) mod k = ψ-int A s mod k
    using technical-lemma2-part2[of n A k −1 t] ψ-eq assms by auto
  hence s = −4∗n+t using distinct-residus[of A n k −4∗n+t s] 14 ineq assms
by auto
  then show ?thesis by simp
next
  case 24
  have ineq: −n ≤ t−4∗n ∧ t−4∗n ≤ n ∧ −n ≤ 2∗n−s ∧ 2∗n−s ≤ n using 24
by auto
  have ψ-int A (t−4∗n) mod k = ψ-int A (2∗n−s) mod k
    using assms ψ-eq technical-lemma2[of n A k −s] technical-lemma2-part2[of n
A k −1 t]
      ψ-int-odd[of A −s] by (smt (z3) sun-lemma10-rec)
  hence t−4∗n = 2∗n − s using distinct-residus[of A n k t−4∗n 2∗n−s] ineq
assms by auto
  then show ?thesis by simp
next
  case 44
  have ineq: −n ≤ s−4∗n ∧ s−4∗n ≤ n ∧ −n ≤ t−4∗n ∧ t−4∗n ≤ n using 44
by auto
  have ψ-int A (s−4∗n) mod k = ψ-int A (t−4∗n) mod k
   using assms ψ-eq technical-lemma2-part2[of n A k −1 s] technical-lemma2-part2[of
n A k −1 t]
    by (smt (z3) sun-lemma10-rec)
  then show ?thesis using distinct-residus[of A n k t−4∗n s−4∗n] ineq assms
by auto
qed
qed

lemma mod-pos:
  fixes k::int and n::int
  assumes n > 0
  shows 0 ≤ k mod n ∧ k mod n < n
  by (simp add: assms)

lemma lesser-4n-to-all:
  fixes A::int and n::int and s::int and t::int and k::int
  assumes A > 2 and n > 3 and χ-int A n = 2∗k
  shows (ψ-int A s mod k = ψ-int A t mod k)
     ⟹ (s mod (4∗n) = t mod (4∗n) ∨ (s+t) mod (4∗n) = (2∗n) mod (4∗n))
proof −
  assume hyp-true: ψ-int A s mod k = ψ-int A t mod k
```

have *dvd-mod*: *a dvd b* $\Longrightarrow$ *l mod b = m mod b* $\Longrightarrow$ *l mod a = m mod a* **for** *a*::*int* **and** *b* **and** *l* **and** *m*
by (*metis mod-mod-cancel*)
define *s0* **where** *s0 = s mod (4*n)*
define *t0* **where** *t0 = t mod (4*n)*
have *hyp-s*: *s mod (4*n) = s0 mod (4*n)* $\wedge$ *0* $\leq$ *s0* $\wedge$ *s0 < 4*n* **unfolding** *s0-def* **using** *assms* **by** *auto*
have *hyp-t*: *t mod (4*n) = t0 mod (4*n)* $\wedge$ *0* $\leq$ *t0* $\wedge$ *t0 < 4*n* **unfolding** *t0-def* **using** *assms* **by** *auto*
have *(s+t) mod (4*n) = (s0+t0) mod (4*n)* **using** *hyp-s hyp-t mod-add-cong*[*of s 4*n s0 t t0*]
by *auto*
hence *impl*: *(s0 mod (4*n) = t0 mod (4*n)* $\vee$ *(s0+t0) mod (4*n) = (2*n) mod (4*n))*
$\Longrightarrow$ *(s mod (4*n) = t mod (4*n)* $\vee$ *(s+t) mod (4*n) = (2*n) mod (4*n))*
using *hyp-s hyp-t* **by** *auto*
obtain *qs* **where** *qs-def*: *s = 4*n*qs + s0* **using** *s0-def* **by** (*metis mult-div-mod-eq*)
obtain *qt* **where** *qt-def*: *t = 4*n*qt + t0* **using** *t0-def* **by** (*metis mult-div-mod-eq*)
have *ψ-int A s mod k = ψ-int A s0 mod k* $\wedge$ *ψ-int A t mod k = ψ-int A t0 mod k*
using *technical-lemma2-part2*[*of n A k qs s0*] *qs-def assms*
technical-lemma2-part2[*of n A k qt t0*] *qt-def* **by** *auto*
hence *ψ-int A s0 mod k = ψ-int A t0 mod k* **using** *hyp-true* **by** *auto*
then have *s0 mod (4*n) = t0 mod (4*n)* $\vee$ *(s0+t0) mod (4*n) = (2*n) mod (4*n)*
using *case-lesser-than-4n*[*of A n k s0 t0*] *assms hyp-s hyp-t* **by** *auto*
thus *s mod (4*n) = t mod (4*n)* $\vee$ *(s+t) mod (4*n) = (2*n) mod (4*n)* **using** *impl* **by** *auto*
qed

lemma *sun-lemma10-dir*:
fixes *A*::*int* **and** *n*::*int* **and** *s*::*int* **and** *t*::*int* **and** *k*::*int*
assumes *A > 2* **and** *n > 3* **and** *χ-int A n = 2*k*
shows *(ψ-int A s mod k = ψ-int A t mod k)*
$\Longrightarrow$ *(s mod (4*n) = t mod (4*n)* $\vee$ *(s+t) mod (4*n) = (2*n) mod (4*n))*
using *assms lesser-4n-to-all*[*of A n k s t*] **by** *simp*

lemma (**in** *bridge-variables*) *sun-lemma24*:
fixes *A*::*int* **and** *B*::*int* **and** *C*::*int* **and** *x*::*int* **and** *y*::*int*
assumes *abs A* $\geq$ *2*
shows *is-square (D-f A C * F-ACx A C x * I-ABCxy A B C x y) = (is-square (D-f A C)*
$\wedge$ *is-square (F-ACx A C x)* $\wedge$ *is-square (I-ABCxy A B C x y))*
proof −
have *converse*: *(is-square (D-f A C)* $\wedge$ *is-square (F-ACx A C x)* $\wedge$ *is-square (I-ABCxy A B C x y))*
$\Longrightarrow$ *is-square (D-f A C * F-ACx A C x * I-ABCxy A B C x y)*
proof −
assume *(is-square (D-f A C)* $\wedge$ *is-square (F-ACx A C x)* $\wedge$ *is-square (I-ABCxy*

```
A B C x y))
  then obtain d f i where D-f A C = d^2 ∧ F-ACx A C x = f^2 ∧ I-ABCxy
A B C x y = i^2
    by (auto simp add: is-square-def)
  hence D-f A C * F-ACx A C x * I-ABCxy A B C x y = (d*f*i)^2 by (auto
simp add: algebra-simps)
  thus ?thesis using is-square-def by auto
 qed
 have direct: is-square (D-f A C * F-ACx A C x * I-ABCxy A B C x y)
    ⟹ ((is-square (D-f A C) ∧ is-square (F-ACx A C x) ∧ is-square (I-ABCxy
A B C x y)))
 proof −
  assume hyp: is-square (D-f A C * F-ACx A C x * I-ABCxy A B C x y)
   have square-decomp: is-square (a*b) ∧ coprime a b ∧ a ≥ 0 ∧ b ≥ 0 ⟹
is-square a ∧ is-square b
    for a b :: int
  proof −
   assume hypo: is-square (a*b) ∧ coprime a b ∧ a ≥ 0 ∧ b ≥ 0
   then obtain k where k-def: a*b = k^2 using is-square-def by auto
   define c where c = gcd a k
   define d where d = gcd b k
  have eq1: c*c dvd a*b ∧ d*d dvd a*b using c-def d-def k-def power2-eq-square[of
k]
     by (auto simp add: mult-dvd-mono)
    have coprime c b ∧ coprime d a using c-def d-def hypo coprime-divisors[of c
a b b]
       coprime-divisors[of d b a a] by (simp add: coprime-commute)
   hence cop: coprime (c*c) b ∧ coprime (d*d) a by auto
  hence eq2: c*c dvd a ∧ d*d dvd b using eq1 by (meson coprime-dvd-mult-left-iff
euclids-lemma)
   have k dvd a*b using k-def by auto
   hence k dvd c*d using c-def d-def apply simp
     by (metis (no-types, lifting) division-decomp dvd-triv-left dvd-triv-right
gcd-greatest-iff mult-dvd-mono)
   hence a*b dvd (c*d*c*d) using k-def power2-eq-square[of k] by auto
    hence a dvd (d*d)*(c*c) ∧ b dvd (c*c)*(d*d) by (metis dvd-mult-right
mult.assoc mult.commute)
   hence a dvd (c*c) ∧ b dvd (d*d)
    using cop euclids-lemma[of a d*d c*c] coprime-commute[of a d*d]
     euclids-lemma[of b c*c d*d] coprime-commute[of b c*c] by auto
   hence a=c*c ∧ b=d*d
    using eq2 hypo zdvd-antisym-nonneg zero-le-square
    by simp
  thus ?thesis using is-square-def[of a] is-square-def[of b] power2-eq-square[of
c]
       power2-eq-square[of d] by metis
 qed
 have copFD: coprime (F-ACx A C x) (D-f A C)
 proof −
```

have *F-ACx A C x mod (D-f A C) = (4∗(A^2−4)∗(C^2∗D-f A C∗x)^2+1) mod (D-f A C)*
unfolding *F-ACx-def F-f-def E-ACx-def E-f-def* **by** *simp*
hence *F-ACx A C x mod (D-f A C) = (4∗(A^2−4)∗(C^2∗x)^2∗D-f A C∗D-f A C+1) mod (D-f A C)*
using *power-mult-distrib[of C^2∗x D-f A C 2] power2-eq-square[of D-f A C]*
by (*auto simp add*: *algebra-simps*)
hence *F-ACx A C x mod (D-f A C) = 1 mod (D-f A C)* **by** *auto*
thus *?thesis* **by** (*metis coprimeI coprime-mod-left-iff mod-by-0*)
qed
have *copID*: *coprime (I-ABCxy A B C x y) (D-f A C)*
proof −
have *G-ACx A C x mod (D-f A C)*
= (1 + (C∗F-ACx A C x)∗ D-f A C − 2∗(A+2)∗(A−2)^2∗(C^2∗D-f A C∗x)^2) mod (D-f A C)
unfolding *G-ACx-def G-f-def E-ACx-def E-f-def* **by** (*auto simp add*: *algebra-simps*)
hence *G-ACx A C x mod (D-f A C)*
= (1 + (C∗F-ACx A C x)∗ D-f A C − 2∗(A+2)∗(A−2)^2∗(C^2∗x)^2∗D-f A C∗ D-f A C) mod (D-f A C)
using *power2-eq-square[of D-f A C] power-mult-distrib[of D-f A C]*
by (*auto simp add*: *algebra-simps*)
hence *G-ACx A C x mod (D-f A C) = 1 mod (D-f A C)* **by** (*smt* (*z3*) *mod-mult-self3*)
hence (*G-ACx A C x^2 − 1*) *mod (D-f A C) = 0 mod (D-f A C)*
by (*metis cancel-comm-monoid-add-class.diff-cancel mod-diff-left-eq power-mod power-one*)
hence ((*G-ACx A C x^2 − 1*)∗*H-ABCxy A B C x y*) *mod (D-f A C) = 0 mod (D-f A C)* **by** *auto*
hence *I-ABCxy A B C x y mod (D-f A C) = 1 mod (D-f A C)* **unfolding** *I-ABCxy-def I-f-def*
using *mod-add-cong[of (G-ACx A C x^2 − 1)∗H-ABCxy A B C x y D-f A C 0 1 1]* **apply** *simp*
by (*metis* (*no-types*) ‹((*G-ACx A C x*)^2 − 1) *mod D-f A C = 0 mod D-f A C*› *mod-add-left-eq mod-mult-left-eq mod-mult-self2-is-0 mod-mult-self4 mult-numeral-1*)
thus *?thesis* **by** (*metis coprimeI coprime-mod-left-iff mod-by-0*)
qed
have *2∗G-ACx A C x mod (F-ACx A C x)*
= (2∗C∗D-f A C ∗ F-ACx A C x + (2−4∗(A+2)∗(A−2)^2∗E-ACx A C x^2)) mod (F-ACx A C x)
unfolding *G-ACx-def G-f-def* **by** (*auto simp add*: *algebra-simps*)
hence *2∗G-ACx A C x mod (F-ACx A C x) = (2−4∗(A+2)∗(A−2)^2∗E-ACx A C x^2) mod (F-ACx A C x)*
by *auto*
hence *2∗G-ACx A C x mod (F-ACx A C x) = (2−4∗(A^2−4)∗E-ACx A C x^2 ∗ (A−2)) mod (F-ACx A C x)*
by (*auto simp add*: *algebra-simps power2-eq-square*)
hence *2∗G-ACx A C x mod (F-ACx A C x) = (2−(F-ACx A C x−1) ∗ (A−2)) mod (F-ACx A C x)*

unfolding *F-ACx-def F-f-def* **by** *auto*
hence *2∗G-ACx A C x mod (F-ACx A C x) = (A − (A−2)∗F-ACx A C x) mod (F-ACx A C x)*
by (*auto simp add*: *algebra-simps*)
hence *A-eq-2G-modF*: *2∗G-ACx A C x mod (F-ACx A C x) = A mod (F-ACx A C x)*
by (*metis add.commute diff-add-cancel mod-mult-self3*)
have *copIF*: *coprime (I-ABCxy A B C x y) (F-ACx A C x)*
proof −
have *H-ABCxy A B C x y mod (F-ACx A C x) = C mod (F-ACx A C x)*
unfolding *H-ABCxy-def H-f-def* **by** *auto*
hence *H-ABCxy A B C x y^2 mod (F-ACx A C x) = C^2 mod (F-ACx A C x)* **by** (*metis power-mod*)
hence *H-eqC*: *((A^2−4)∗H-ABCxy A B C x y^2 +4) mod (F-ACx A C x) = D-f A C mod (F-ACx A C x)*
using *mod-mult-cong*[*of A^2−4 F-ACx A C x A^2−4 H-ABCxy A B C x y^2 C^2*]
unfolding *D-f-def* **by** (*metis* (*no-types*) *add.commute mod-add-right-eq*)
have *((2∗G-ACx A C x)∗(2∗G-ACx A C x)−4) mod (F-ACx A C x) = (A∗A−4) mod (F-ACx A C x)*
using *A-eq-2G-modF* **by** (*metis* (*no-types*) *mod-diff-left-eq mod-mult-left-eq mod-mult-right-eq*)
hence *A-eq2G*: *(((2∗G-ACx A C x)∗(2∗G-ACx A C x)−4)∗H-ABCxy A B C x y^2 +4) mod (F-ACx A C x)*
= ((A∗A−4)∗H-ABCxy A B C x y^2 +4) mod (F-ACx A C x)
using *mod-mult-cong*[*of ((2∗G-ACx A C x)∗(2∗G-ACx A C x)−4) F-ACx A C x A∗A−4*
H-ABCxy A B C x y^2 H-ABCxy A B C x y^2]
by (*smt* (*verit*, *ccfv-SIG*) *add.commute mod-add-right-eq*)
have *4∗I-ABCxy A B C x y mod (F-ACx A C x)*
= 4∗((G-ACx A C x^2−1)∗H-ABCxy A B C x y^2 +1) mod (F-ACx A C x)
unfolding *I-ABCxy-def I-f-def* **by** *auto*
hence *4∗I-ABCxy A B C x y mod (F-ACx A C x)*
= (((2∗G-ACx A C x)^2−4)∗H-ABCxy A B C x y^2 +4) mod (F-ACx A C x)
using *power-mult-distrib*[*of 2 G-ACx A C x 2*] **by** (*auto simp add*: *algebra-simps*)
hence *4∗I-ABCxy A B C x y mod (F-ACx A C x)*
= (((2∗G-ACx A C x)∗(2∗G-ACx A C x)−4)∗H-ABCxy A B C x y^2 +4) mod (F-ACx A C x)
by (*auto simp add*: *power2-eq-square*)
hence *4∗I-ABCxy A B C x y mod (F-ACx A C x) = ((A∗A−4)∗H-ABCxy A B C x y^2 +4) mod (F-ACx A C x)*
using *A-eq2G* **by** *auto*
hence *4∗I-ABCxy A B C x y mod (F-ACx A C x) = ((A^2−4)∗H-ABCxy A B C x y^2 +4) mod (F-ACx A C x)*
using *power2-eq-square*[*of A*] **by** *auto*
hence *4∗I-ABCxy A B C x y mod (F-ACx A C x) = D-f A C mod (F-ACx*

A C x) **using** *H-eqC* **by** *auto*
hence *gcd* (*4***I-ABCxy A B C x y*) (*F-ACx A C x*) = *gcd* (*D-f A C*) (*F-ACx A C x*)
by (*metis gcd-red-int*)
hence *coprime* (*4***I-ABCxy A B C x y*) (*F-ACx A C x*)
using *copFD* **by** (*metis coprime-iff-gcd-eq-1 gcd.commute*)
then show *?thesis* **by** *simp*
qed
have *A-pos*: *A^2−4* ≥ *0*
using *assms power2-eq-square*[*of abs A*] *mult-mono*[*of 2 abs A 2 abs A*] **by** *auto*
hence *D-pos*: *D-f A C* ≥ *1* **unfolding** *D-f-def* **by** *auto*
have *F-pos*: *F-ACx A C x* ≥ *1* **unfolding** *F-ACx-def F-f-def* **using** *A-pos* **by** *auto*
obtain *k* **where** *D-f A C* ∗ *F-ACx A C x* ∗ *I-ABCxy A B C x y* = *k^2* **using** *hyp is-square-def* **by** *auto*
hence *I-pos*: *I-ABCxy A B C x y* ≥ *0* **using** *D-pos F-pos*
by (*smt* (*verit*, *ccfv-SIG*) *mult-pos-neg mult-pos-pos zero-le-power2*)
have *copDF-I*: *coprime* (*D-f A C* ∗ *F-ACx A C x*) (*I-ABCxy A B C x y*)
using *copID copIF coprime-commute* **by** *auto*
hence *is-square* (*I-ABCxy A B C x y*) ∧ *is-square* (*D-f A C* ∗ *F-ACx A C x*)
using *I-pos D-pos F-pos square-decomp*[*of D-f A C* ∗ *F-ACx A C x I-ABCxy A B C x y*] *hyp*
by *auto*
then show *?thesis*
using *square-decomp*[*of D-f A C F-ACx A C x*] *D-pos F-pos copFD coprime-commute* **by** *auto*
qed
show *?thesis* **using** *direct converse* **by** *argo*
qed

end
theory *DFI-square-3*
imports *DFI-square-2*
begin

A few lemmas before the proof

lemma *lucas-pell-corollary-int*:
fixes *A*::*int* **and** *X*::*int*
shows (∃ *k*. (*A^2−4*)∗*X^2* + *4* = *k^2*) ⟹(∃ *m*. *X* = *ψ-int A m*)
proof −
assume *assumption*: (∃ *k*. (*A^2−4*)∗*X^2* + *4* = *k^2*)
then obtain *m0* **where** *m0-def*: *X* = *ψ A m0* ∨ *X* = −*ψ A m0* **using** *lucas-pell-part1* **by** *auto*
then obtain *e* **where** *e-def*: *X* = *e*∗*ψ A m0* ∧ (*e*=*1* ∨ *e*=−*1*) **by** *force*
define *m* **where** *m* ≡ *e*∗ *int m0*
hence *m-def*: *X* = *ψ-int A m*
using *ψ-int-def*[*of A m*] *m0-def e-def mult-minus-left power-one-right* **by** *fastforce*

then show *?thesis* **by** *auto*
qed

lemma *lucas-modN-int*:
fixes *A*::*int* **and** *B*::*int* **and** *n*::*int*
assumes *A mod n* = *B mod n*
shows (*ψ-int A m*) *mod n* = (*ψ-int B m*) *mod n*
proof (*cases m*)
case (*nonneg k*)
then show *?thesis*
using *lucas-congruence*[*of n A B k*] *ψ-int-def*[*of A m*] *ψ-int-def*[*of A m*] *ψ-int-def*[*of B m*] *assms*
by *auto*
next
case (*neg k*)
then show *?thesis*
using *lucas-congruence*[*of n A B Suc k*] *ψ-int-def*[*of A m*] *ψ-int-def*[*of A m*] *ψ-int-def*[*of B m*] *assms*
apply *simp* **by** (*metis Suc-as-int mod-minus-eq*)
qed

lemma *div-mod*: (*n*::*int*) *dvd m* ⟹ *k mod m* = *l mod m* ⟹ *k mod n* = *l mod n*
proof −
assume *assm1*: *n dvd m*
assume *assm2*: *k mod m* = *l mod m*
hence (*k*−*l*) *mod m* = *0 mod m* **using** *mod-diff-cong*[*of k m l l l*] **by** *auto*
hence *m dvd* (*k*−*l*) **by** *auto*
hence *n dvd* (*k*−*l*) **using** *assm1* **by** *auto*
hence (*k*−*l*) *mod n* = *0 mod n* **by** *auto*
thus *?thesis* **using** *mod-add-cong*[*of k*−*l n 0 l l*] **by** *auto*
qed

lemma *ψ-int-minusA*: *ψ-int* (−*X*) *n* = (−*1*) ^(*nat* (*abs n*) +*1*)∗*ψ-int X n* **for** *n X*
proof (*cases n*)
case (*nonneg k*)
note *t*=*this*
hence *k*=*nat* (*abs n*) **by** *auto*
then show *?thesis* **using** *lucas-symmetry-A2*[*of* −*X k*] *ψ-int-def*[*of* −*X n*] *ψ-int-def*[*of X n*] *t* **by** *auto*
next
case (*neg k*)
note *t*=*this*
hence *Suc k* = *nat* (*abs n*) **by** *auto*
then show *?thesis* **using** *lucas-symmetry-A2*[*of* −*X Suc k*] *ψ-int-def*[*of* −*X n*] *ψ-int-def*[*of X n*] *t*
apply *simp* **by** (*smt* (*z3*) *mult.assoc mult.commute*)
qed

lemma *eq-ψ-int*: *abs X > 1 ⟹ abs (ψ-int X n) = ψ (abs X) (nat (abs n))* **for** *X n*
proof −
assume *assm*: *abs X > 1*
then show *?thesis*
proof (*cases X*)
case (*nonneg Y*)
hence *abs (ψ-int X n) = abs (ψ-int (abs X) n)* **by** *auto*
hence *abs (ψ-int X n) = abs ((−1)^(if 0 ≤ n then 0 else 1) ∗ ψ (abs X) (nat (abs n)))*
using *ψ-int-def*[*of abs X n*] **by** *auto*
hence *abs (ψ-int X n) = abs ((−1)^(if 0 ≤ n then 0 else 1))∗abs (ψ (abs X) (nat (abs n)))*
using *abs-mult* **by** *auto*
hence *abs (ψ-int X n) = abs (ψ (abs X) (nat (abs n)))* **by** *auto*
then show *?thesis* **using** *lucas-monotone3*[*of abs X nat (abs n)*] *assm* **by** *auto*
next
case (*neg Y*)
hence *−X = abs X* **by** *auto*
hence *abs (ψ-int X n) = abs ((−1)^(nat (abs n)+1)∗ψ-int (abs X) n)*
using *ψ-int-minusA*[*of −X n*] **by** (*smt* (*verit*))
hence *abs (ψ-int X n) = abs ((−1)^(nat (abs n)+1))∗abs (ψ-int (abs X) n)*
using *abs-mult* **by** *metis*
hence *abs (ψ-int X n) = abs (ψ-int (abs X) n)* **by** *auto*
hence *abs (ψ-int X n) = abs ((−1)^(if 0 ≤ n then 0 else 1) ∗ ψ (abs X) (nat (abs n)))*
using *ψ-int-def*[*of abs X n*] **by** *auto*
hence *abs (ψ-int X n) = abs ((−1)^(if 0 ≤ n then 0 else 1))∗abs (ψ (abs X) (nat (abs n)))*
using *abs-mult* **by** *auto*
hence *abs (ψ-int X n) = abs (ψ (abs X) (nat (abs n)))* **by** *auto*
then show *?thesis* **using** *lucas-monotone3*[*of abs X nat (abs n)*] *assm* **by** *auto*
qed
qed

Lemma 10 in Sun

lemma *sun-lemma10-dir-int*:
fixes *A*::*int* **and** *n*::*int* **and** *s*::*int* **and** *t*::*int* **and** *k*::*int*
assumes *abs A > 2* **and** *n > 3* **and** *χ-int (abs A) n = 2∗k*
shows (*ψ-int A s mod k = ψ-int A t mod k*)
⟹ (s mod (2∗n) = t mod (2∗n) ∨ (s+t) mod (2∗n) = 0 mod (2∗n))
proof −
assume *hyp*: (*ψ-int A s mod k = ψ-int A t mod k*)
then show *?thesis*
proof (*cases A*)
case (*nonneg B*)

then show *?thesis* **using** *sun-lemma10-dir*[*of B n k s t*] *assms hyp div-mod*[*of 2∗n 4∗n s t*]
div-mod[*of 2∗n 4∗n s+t 2∗n*] *gr0I mod-eq-0-iff-dvd*
mod-self of-nat-0-less-iff zmod-int **by** *auto*
next
case (*neg B*)
note *t* = *this*
define *C* **where** *C* = *abs A*
hence *C-def2*: *C* = −*A* **using** *t* **by** *auto*
have (−1) ^(*nat* (*abs s*) +1)∗*ψ-int C s mod k* = (−1) ^(*nat* (*abs t*)+1)∗*ψ-int C t mod k*
using *hyp ψ-int-minusA*[*of* −*A s*] *ψ-int-minusA*[*of* −*A t*] *C-def2* **by** *auto*
hence *eq*: (−1) ^(*nat* (*abs s*)+1)∗*ψ-int C s*∗(−1) ^(*nat* (*abs s*)+1) *mod k*
= (−1) ^(*nat* (*abs t*)+1)∗*ψ-int C t*∗(−1) ^(*nat* (*abs s*)+1) *mod k*
using *mod-mult-cong*[*of* (−1) ^(*nat* (*abs s*) +1)∗*ψ-int C s k* (−1) ^(*nat* (*abs t*) +1)∗*ψ-int C t*
(−1) ^(*nat* (*abs s*)+1) (−1) ^(*nat* (*abs s*)+1)] **by** *blast*
hence *ψ-int C s mod k* = *ψ-int C t mod k* ∨ *ψ-int C s mod k* = (−*ψ-int C t*) *mod k*
by (*smt* (*z3*) *left-minus-one-mult-self minus-one-mult-self mod-minus-minus mult.left-commute*
mult-cancel-left2 mult-minus-right square-eq-1-iff)
hence *ψ-int C s mod k* = *ψ-int C t mod k* ∨ *ψ-int C s mod k* = *ψ-int C* (−*t*) *mod k*
using *ψ-int-odd*[*of C t*] **by** *auto*
hence *s mod* (4∗*n*) = *t mod* (4∗*n*) ∨ (*s*+*t*) *mod* (4∗*n*) = 2∗*n mod* (4∗*n*)
∨ *s mod* (4∗*n*) = (−*t*) *mod* (4∗*n*) ∨ (*s*−*t*) *mod* (4∗*n*) = 2∗*n mod* (4∗*n*)
using *sun-lemma10-dir*[*of C n k s t*] *sun-lemma10-dir*[*of C n k s* −*t*] *hyp assms*
by (*metis C-def add.inverse-inverse diff-minus-eq-add*)
hence *s mod* (2∗*n*) = *t mod* (2∗*n*) ∨ (*s*+*t*) *mod* (2∗*n*) = *0 mod* (2∗*n*)
∨ *s mod* (2∗*n*) = (−*t*) *mod* (2∗*n*) ∨ (*s*−*t*) *mod* (2∗*n*) = *0 mod* (2∗*n*)
using *div-mod*[*of* 2∗*n* 4∗*n s t*] *div-mod*[*of* 2∗*n* 4∗*n s*+*t* 2∗*n*]
div-mod[*of* 2∗*n* 4∗*n s* −*t*] *div-mod*[*of* 2∗*n* 4∗*n s*−*t* 2∗*n*] **by** *auto*
then show *?thesis* **using** *mod-add-cong*[*of s* 2∗*n* −*t t t*] *mod-add-cong*[*of s*−*t* 2∗*n* 0 *t t*]
by *auto*
qed
qed

Theorem in Sun

theorem (**in** *bridge-variables*) *sun-theorem*:
fixes *A*::*int* **and** *B*::*int* **and** *C*::*int* **and** *x*::*int* **and** *y*::*int*
assumes *abs B* > *1* **and** 2∗*abs B* < *abs A* − *2* **and** (*A*−2) *dvd* (*C*−*B*) **and** *x* ≠ *0*
and *is-square* (*D-f A C* ∗ *F-ACx A C x* ∗ *I-ABCxy A B C x y*)
shows *C* = *ψ-int A B*
proof −
have *A-B2*: *abs A* > *2* **using** *assms* **by** *auto*

have *A^2 = (abs A)^2* **by** *auto*
hence *A-pos*: *A^2−4 > 0*
using *assms power2-eq-square*[*of abs A*] *mult-strict-mono*[*of 2 abs A 2 abs A*] **by** *auto*
have *DFI-sq*: (*is-square* (*D-f A C*) ∧ *is-square* (*F-ACx A C x*) ∧ *is-square* (*I-ABCxy A B C x y*))
using *sun-lemma24*[*of A C x B y*] *assms* **by** *auto*
then obtain *s* **where** *s-def*: *C* = *ψ-int A s*
using *lucas-pell-corollary-int*[*of A C*] *D-f-def*[*of A C*] *is-square-def*[*of D-f A C*] **by** *auto*
obtain *k0* **where** *k-def*: *k0^2 = F-ACx A C x* **using** *DFI-sq is-square-def* **by** *metis*
hence *(2∗k0)^2 = (A^2−4)∗(4∗E-ACx A C x)^2 + 4*
unfolding *F-ACx-def F-f-def* **by** (*auto simp add*: *algebra-simps*)
then obtain *m* **where** *m-def*: *4∗E-ACx A C x = ψ-int A m*
using *lucas-pell-corollary-int*[*of A 4∗E-ACx A C x*] **by** *metis*
obtain *k1* **where** *k1-def*: *k1^2 = I-ABCxy A B C x y* **using** *DFI-sq is-square-def* **by** *metis*
hence *(2∗k1)^2 = ((2∗G-ACx A C x)^2−4)∗H-ABCxy A B C x y^2+4*
unfolding *I-ABCxy-def I-f-def* **using** *power-mult-distrib*[*of 2 k1 2*] **by** (*auto simp add*: *algebra-simps*)
then obtain *t* **where** *t-def*: *H-ABCxy A B C x y = ψ-int* (*2∗G-ACx A C x*) *t*
using *lucas-pell-corollary-int*[*of 2∗G-ACx A C x H-ABCxy A B C x y*] **by** *metis*

have *sB1*: *abs s > 1*
proof −
consider (*0*) *s=0* | (*1*) *s=1* | (*m1*) *s=−1* | (*ok*) *abs s > 1* **by** *linarith*
then show *?thesis*
proof (*cases*)
case *0*
hence *C= 0* **using** *s-def ψ-int-def*[*of A s*] **by** *auto*
hence (*A−2*) *dvd B* **using** *assms* **by** *auto*
hence *abs B ≥ abs* (*A−2*) **using** *assms* **using** *dvd-imp-le-int* **by** *force*
hence *abs B ≥ abs A − 2* **by** *auto*
hence *abs A − 2 > 2∗abs A − 4* **using** *assms* **by** *auto*
then show *?thesis* **using** *assms* **by** *auto*
next
case *1*
hence *C= 1* **using** *s-def ψ-int-def*[*of A s*] **by** *auto*
hence (*A−2*) *dvd* (*B−1*) **using** *assms* **by** *presburger*
hence *abs* (*B−1*) ≥ *abs* (*A−2*) **using** *assms* **using** *dvd-imp-le-int* **by** *fastforce*
hence *abs B + 1 ≥ abs A − 2* **by** *auto*
hence *abs A > 2∗abs A − 4* **using** *assms* **by** *auto*
then show *?thesis* **using** *assms* **by** *auto*
next
case *m1*
hence *C= −1* **using** *s-def ψ-int-def*[*of A s*] **by** *auto*
hence (*A−2*) *dvd* (*−1−B*) **using** *assms* **by** *presburger*

hence *abs (−1−B) ≥ abs (A−2)* **using** *assms* **using** *dvd-imp-le-int* **by** *fastforce*
hence *abs B + 1 ≥ abs A − 2* **by** *auto*
hence *abs A > 2∗abs A − 4* **using** *assms* **by** *auto*
then show *?thesis* **using** *assms* **by** *auto*
qed
qed

hence *C-Bigger-A*: *abs C ≥ abs A*
proof −
have *abs C = abs (ψ-int A s)* **using** *s-def* **by** *auto*
hence *abs C = ψ (abs A) (nat (abs s))* **using** *eq-ψ-int[of A s] A-B2* **by** *auto*
hence *abs C ≥ ψ (abs A) (Suc (Suc 0))*
using *sB1 lucas-monotone4[of abs A Suc (Suc 0) nat (abs s)] A-B2* **by** *auto*
then show *?thesis* **by** *auto*
qed

hence *C-Bigger-B*: *abs C > abs B* **using** *assms* **by** *auto*
hence *C-nonzero*: *C ≠ 0* **by** *auto*
hence *C2Be1*: *C^2 ≥ 1* **by** (*smt* (*verit*, *best*) *power2-less-eq-zero-iff*)
have *DBe4*: *D-f A C ≥ 4* **unfolding** *D-f-def* **using** *A-pos* **by** *force*
have *DBeA2*: *D-f A C ≥ A^2* **unfolding** *D-f-def* **using** *A-pos mult-left-mono[of 1 C^2 A^2−4] C2Be1*
by *presburger*
have *E-non0*: *E-ACx A C x ≠ 0* **unfolding** *E-ACx-def E-f-def* **using** *DBe4 assms C-nonzero* **by** *auto*
define *n* **where** *n = nat (abs m)*
hence *ψ-A-n-eq-4E*: *ψ (abs A) n = abs (4∗E-ACx A C x)* **using** *m-def eq-ψ-int[of A m] assms* **by** *auto*
hence *eq-ψn*: *ψ (abs A) n = 4∗C^2∗D-f A C ∗ abs x* **unfolding** *E-ACx-def E-f-def* **using** *C2Be1 DBe4*
by (*auto simp add*: *abs-mult*)
hence *ψ (abs A) n ≥ D-f A C* **using** *C2Be1 assms DBe4* **apply** *simp* **by** (*smt* (*z3*) *mult-le-0-iff*)
hence *ψ (abs A) n > A∗A − 1* **using** *DBeA2 power2-eq-square[of A]* **by** *auto*
hence *ψ (abs A) n > ψ A (Suc (Suc (Suc 0)))* **by** *auto*
hence *n > Suc (Suc (Suc 0))* **using** *lucas-monotone4[of abs A n Suc (Suc (Suc 0))] assms*
apply *simp* **using** *le-eq-less-or-eq nat-le-linear* **by** *blast*
hence *nB3*: *3 < int n* **by** *auto*
have *χ (abs A) n^2 mod 4 = ((A^2−4)∗ψ (abs A) n^2 + 4) mod 4*
using *lucas-pell-part3[of abs A n]* **by** *auto*
hence *χ (abs A) n^2 mod 4 = (A^2−4)∗(C^2∗D-f A C∗abs x∗4) ^2 mod 4*
using *eq-ψn* **by** (*auto simp add*: *algebra-simps*)
hence *χ (abs A) n^2 mod 4 = (A^2−4)∗(C^2∗D-f A C∗abs x) ^2∗4∗4 mod 4*
using *power-mult-distrib[of C^2∗D-f A C∗abs x 4 2] power2-eq-square[of 4]* **by** *auto*
hence *χ (abs A) n^2 mod 4 = 0 mod 4* **by** *simp*
hence *4 dvd χ (abs A) n^2* **by** *presburger*

hence *2^2 dvd χ (abs A) n^2* **by** *auto*
hence *even-χ*: *2 dvd χ (abs A) n*
by (*smt* (*z3*) *bits-one-mod-two-eq-one even-push-bit-iff exp-dvdE mod2-eq-if one-power2 power-mod*
zero-neq-numeral)
then obtain *k* **where** *k-def*: *χ (abs A) n = 2∗k* **by** *auto*
hence *k-def2*: *χ-int (abs A) (int n) = 2∗k* **using** *χ-int-def*[*of abs A int n*] **by** *auto*

have *FBe1*: *F-ACx A C x ≥ 1*
unfolding *F-ACx-def F-f-def* **using** *A-pos mult-mono*[*of 0 A^2−4 0 E-ACx A C x^2*] **by** *auto*
have *H-eqC-modF*: *H-ABCxy A B C x y mod (F-ACx A C x) = C mod (F-ACx A C x)*
unfolding *H-ABCxy-def H-f-def* **by** *auto*
have *2∗G-ACx A C x mod (F-ACx A C x) =*
(2∗C∗D-f A C ∗ F-ACx A C x + (2−4∗(A+2)∗(A−2)^2∗E-ACx A C x^2)) mod (F-ACx A C x)
unfolding *G-ACx-def G-f-def* **by** (*auto simp add*: *algebra-simps*)
hence *2∗G-ACx A C x mod (F-ACx A C x) = (2−4∗(A+2)∗(A−2)^2∗E-ACx A C x^2) mod (F-ACx A C x)*
by *auto*
hence *2∗G-ACx A C x mod (F-ACx A C x) = (2−4∗(A^2−4)∗E-ACx A C x^2 ∗ (A−2)) mod (F-ACx A C x)*
by (*auto simp add*: *algebra-simps power2-eq-square*)
hence *2∗G-ACx A C x mod (F-ACx A C x) = (2−(F-ACx A C x−1) ∗ (A−2)) mod (F-ACx A C x)*
unfolding *F-ACx-def F-f-def* **by** *auto*
hence *2∗G-ACx A C x mod (F-ACx A C x) = (A − (A−2)∗F-ACx A C x) mod (F-ACx A C x)*
by (*auto simp add*: *algebra-simps*)
hence *A-eq-2G-modF*: *2∗G-ACx A C x mod (F-ACx A C x) = A mod (F-ACx A C x)*
by (*metis add.commute diff-add-cancel mod-mult-self3*)

hence *ψ-int A t mod (F-ACx A C x) = ψ-int (2∗G-ACx A C x) t mod (F-ACx A C x)*
using *lucas-modN-int*[*of A F-ACx A C x 2∗G-ACx A C x t*] *assms A-eq-2G-modF FBe1* **by** *auto*
hence *eq-modF*: *ψ-int A t mod (F-ACx A C x) = ψ-int A s mod (F-ACx A C x)*
using *t-def s-def H-eqC-modF* **by** *auto*
have *4∗F-ACx A C x = (A^2−4)∗(4∗E-ACx A C x)^2 +4*
unfolding *F-ACx-def F-f-def power2-eq-square*[*of 4*] *power-mult-distrib*[*of 4 E-ACx A C x 2*] **by** *auto*
hence *4∗F-ACx A C x = ((abs A)^2−4)∗(abs (4∗E-ACx A C x))^2 +4* **by** *auto*
hence *χ (abs A) n^2 = 4∗F-ACx A C x* **using** *lucas-pell-part3*[*of abs A n*] *ψ-A-n-eq-4E* **by** *auto*

hence *4∗k∗k dvd 4∗F-ACx A C x* **using** *k-def power2-eq-square*[*of 2∗k*] **by** *auto*
hence *k dvd F-ACx A C x* **by** *auto*
hence *ψ-int A t mod k* = *ψ-int A s mod k* **using** *eq-modF* **by** (*metis mod-mod-cancel*)
hence *s mod* (*2∗int n*) = *t mod* (*2∗int n*) ∨ (*s*+*t*) *mod* (*2∗int n*) = *0 mod* (*2∗int n*) ∧ (*2∗int n dvd 4∗int n*)
using *k-def2 assms nB3 A-B2 sun-lemma10-dir-int*[*of A n k s t*] **by** *auto*
hence *t-pm-s-mod-2n*: *t mod* (*2∗int n*) = *s mod* (*2∗int n*) ∨ *t mod* (*2∗int n*) = (− *s*) *mod* (*2∗int n*)
using *mod-diff-cong*[*of s* + *t 2∗int n 0 s s*] **by** *auto*

have *C2-dvd-4E*: *C*^*2 dvd abs* (*4∗E-ACx A C x*) **unfolding** *E-ACx-def E-f-def* **by** *auto*
hence *ψ-int A s*^*2 dvd ψ* (*abs A*) *n* **using** *s-def ψ-A-n-eq-4E* **by** *auto*
hence *ψ* (*abs A*) (*nat* (*abs s*))^*2 dvd ψ* (*abs A*) *n* **using** *eq-ψ-int*[*of A s*] *A-B2*
by (*metis abs-of-nat int-one-le-iff-zero-less int-ops*(*1*) *not-numeral-le-zero power2-abs verit-comp-simplify1*(*3*) *verit-la-disequality verit-la-generic zero-less-abs-iff*)
hence *ψ* (*abs A*) (*nat* (*abs s*)) *dvd n* **using** *sun-lemma7*[*of abs A nat* (*abs s*) *n*] *A-pos sB1*
by *auto*
hence *ψ-int A s dvd n* **using** *eq-ψ-int*[*of A s*] *A-B2* **by** (*smt* (*verit*) *nat-abs-dvd-iff*)
hence *C dvd n* **using** *s-def* **by** *auto*
hence *2∗C dvd 2∗int n* **by** *auto*
hence *t-pm-s-mod-2C*: *t mod* (*2∗C*) = *s mod* (*2∗C*) ∨ *t mod* (*2∗C*) = (− *s*) *mod* (*2∗C*)
using *t-pm-s-mod-2n div-mod*[*of 2∗C 2∗int n t s*] *div-mod*[*of 2∗C 2∗int n t* − *s*] **by** *auto*
have *F-ACx A C x* = *4∗*(*A*^*2−4*)∗*((*C∗D-f A C* ∗ *x*)∗*C*)^*2+1*
unfolding *F-ACx-def F-f-def E-ACx-def E-f-def* **using** *power2-eq-square*[*of C*]
by (*auto simp add*: *algebra-simps*)
hence *F-ACx A C x* = ((*A*^*2−4*)∗(*C∗D-f A C* ∗ *x*)^*2∗2∗C*)∗(*2∗C*)+*1*
using *power-mult-distrib*[*of C∗D-f A C∗x C 2*] *power2-eq-square*[*of C*]
by (*auto simp add*: *algebra-simps*)
hence *F-eq-1-mod-2C*: *F-ACx A C x mod* (*2∗C*) = *1 mod* (*2∗C*) **by** (*metis mod-mult-self3*)
have *2∗C dvd 4∗E-ACx A C x* **unfolding** *E-ACx-def E-f-def* **using** *power2-eq-square*[*of C*] **by** *auto*
hence *TwoC-dvd-4E*: *2∗C dvd* ((*A*+*2*)∗(*A*−*2*)^*2∗E-ACx A C x*)∗(*4∗E-ACx A C x*) **by** *auto*
have *2∗G-ACx A C x* =
2 + (*D-f A C* ∗ *F-ACx A C x*)∗(*2∗C*) − ((*A*+*2*)∗(*A*−*2*)^*2∗E-ACx A C x*)∗(*4∗E-ACx A C x*)
unfolding *G-ACx-def G-f-def* **using** *power2-eq-square*[*of E-ACx A C x*]
by (*auto simp add*: *algebra-simps*)
hence *2∗G-ACx A C x mod* (*2∗C*) = (*2* − ((*A*+*2*)∗(*A*−*2*)^*2∗E-ACx A C x*)∗(*4∗E-ACx A C x*)) *mod* (*2∗C*)
by (*metis mod-diff-left-eq mod-mult-self1*)
hence *TwoG-eq-2-mod-2C*: *2∗G-ACx A C x mod* (*2∗C*) = *2 mod* (*2∗C*)
using *TwoC-dvd-4E* **by** (*metis minus-mod-self2 mod-mod-cancel*)
have *F-ACx A C x mod 2* = *1 mod 2* **unfolding** *F-ACx-def F-f-def*

by (*metis even-mult-iff even-numeral mod-mod-cancel mod-mult-self4*)
hence $((2*y-1)*F\text{-}ACx\ A\ C\ x\ +\ 1)\ mod\ 2\ =\ 0\ mod\ 2$ **by** *presburger*
hence $2\ dvd\ ((2*y-1)*F\text{-}ACx\ A\ C\ x\ +\ 1)$ **by** *presburger*
hence *fact1*: $(2*C)\ dvd\ ((2*y-1)*F\text{-}ACx\ A\ C\ x\ +\ 1)*C$ **by** *auto*
have $H\text{-}ABCxy\ A\ B\ C\ x\ y\ =\ ((2*y-1)*F\text{-}ACx\ A\ C\ x\ +1)*C\ +\ B*F\text{-}ACx\ A\ C\ x$
unfolding *H-ABCxy-def H-f-def* **by** (*auto simp add*: *algebra-simps*)
hence $H\text{-}ABCxy\ A\ B\ C\ x\ y\ mod\ (2*C)=\ B*F\text{-}ACx\ A\ C\ x\ mod\ (2*C)$ **using** *fact1* **by** *fastforce*
hence *H-eq-B-mod-2C*: $H\text{-}ABCxy\ A\ B\ C\ x\ y\ mod\ (2*C)=\ B\ mod\ (2*C)$
using $F\text{-}eq\text{-}1\text{-}mod\text{-}2C\ mod\text{-}mult\text{-}cong[of\ B\ 2*C\ B\ F\text{-}ACx\ A\ C\ x\ 1]$ **by** *auto*
have $\psi\text{-}int\ (2*G\text{-}ACx\ A\ C\ x)\ t\ mod\ (2*C)\ =\ \psi\text{-}int\ 2\ t\ mod\ (2*C)$
using $TwoG\text{-}eq\text{-}2\text{-}mod\text{-}2C\ lucas\text{-}modN\text{-}int[of\ 2*G\text{-}ACx\ A\ C\ x\ 2*C\ 2\ t]\ assms\ C\text{-}nonzero$ **by** *auto*
hence $\psi\text{-}int\ (2*G\text{-}ACx\ A\ C\ x)\ t\ mod\ (2*C)\ =\ (-\ 1)\ \hat{}\ (if\ 0\ \leq\ t\ then\ 0\ else\ 1)\ *\ |t|\ mod\ (2*C)$
using $\psi\text{-}int\text{-}def[of\ 2\ t]\ lucas\text{-}A\text{-}eq\text{-}2[of\ nat\ (abs\ t)]$
by (*simp add*: ‹$\psi\ 2\ (nat\ |t|)\ =\ int\ (nat\ |t|)$› ‹$\psi\text{-}int\ (2\ *\ G\text{-}ACx\ A\ C\ x)\ t\ mod\ (2\ *\ C)\ =\ \psi\text{-}int\ 2\ t\ mod\ (2\ *\ C)$›
‹$\psi\text{-}int\ 2\ t\ =\ (-\ 1)\ \hat{}\ (if\ 0\ \leq\ t\ then\ 0\ else\ 1)\ *\ \psi\ 2\ (nat\ |t|)$› *nat-0-le*)
hence $\psi\text{-}2G\text{-}eq\text{-}t\text{-}mod\text{-}2C$: $\psi\text{-}int\ (2*G\text{-}ACx\ A\ C\ x)\ t\ mod\ (2*C)\ =\ t\ mod\ (2*C)$ **by** *auto*
hence *B-pm-s-mod-2C*: $B\ mod\ (2*C)\ =\ s\ mod\ (2*C)\ \vee\ B\ mod\ (2*C)\ =\ (-s)\ mod\ (2*C)$
using *H-eq-B-mod-2C t-def t-pm-s-mod-2C* **by** *auto*
hence $B\ mod\ (2*C)\ =\ s\ mod\ (2*C)$
proof −
consider (*plus*) $B\ mod\ (2*C)\ =\ s\ mod\ (2*C)$ | (*minus*) $B\ mod\ (2*C)\ =\ (-s)\ mod\ (2*C)$
using *B-pm-s-mod-2C* **by** *auto*
then show *?thesis*
proof (*cases*)
case *plus*
then show *?thesis* **by** *auto*
next
case *minus*
hence $(B+s)\ mod\ (2*C)\ =\ 0\ mod\ (2*C)$ **using** $mod\text{-}add\text{-}cong[of\ B\ 2*C\ -s\ s\ s]$ **by** *auto*
then obtain *z* **where** $B+s\ =\ 2*C*z$ **by** *auto*
hence *z-def*: $B\ =\ 2*C*z\ -\ s$ **by** *auto*
consider (*0*) $z\ =\ 0$ | (*n0*) $abs\ z\ >\ 0$ **by** *linarith*
then show *?thesis*
proof (*cases*)
case *0*
hence *B-eq-ms*: $B\ =\ -\ s$ **using** *z-def* **by** *auto*
have $(C\ -\ B)\ mod\ (A-2)\ =\ 0\ mod\ (A-2)$ **using** *assms* **by** *auto*
hence $B\ mod\ (A-2)\ =\ C\ mod\ (A-2)$ **using** $mod\text{-}add\text{-}cong[of\ C-B\ A-2\ 0\ B\ B]$ **by** *auto*
hence $B\ mod\ (A-2)\ =\ \psi\text{-}int\ A\ s\ mod\ (A-2)\ \wedge\ A\ mod\ (A-2)\ =\ 2\ mod$

```
(A−2)
        using s-def by (smt (z3) minus-mod-self2)
      hence fact3: B mod (A−2) = ψ-int 2 s mod (A−2)
        using lucas-modN-int[of A A−2 2 s] by auto
      hence B mod (A−2) = s mod (A−2)
      proof −
        have ψ2: ψ 2 q = int q for q
        proof (induction q rule: ψ-induct)
          case 0
          then show ?case by auto
        next
          case 1
          then show ?case by auto
        next
          case (sucsuc n)
          then show ?case by auto
        qed
        have (−1) ^(if 0 ≤ s then 0 else 1)∗int (nat (abs s)) = s by auto
         then show ?thesis using ψ-int-def[of 2 s] ψ2[of nat (abs s)] fact3 by
presburger
      qed
      hence (−s) mod (A−2) = s mod (A−2) using B-eq-ms by auto
     hence 2∗(−s) mod (A−2) = 0 mod (A−2) using mod-diff-cong[of −s A−2
s s s] by auto
      hence (A−2) dvd 2∗(−s) by presburger
      hence (A−2) dvd 2∗B using B-eq-ms by simp
       hence abs (A−2) < abs (2∗B) using assms A-B2 dvd-imp-le-int[of 2∗B
A−2] by linarith
      hence abs A − 2 < 2∗abs B by auto
      hence abs A − 1 ≤ 2∗abs B by auto
      hence abs A − 1 < abs A − 2 using assms by auto
      then show ?thesis by auto
    next
      case n0
      have abs B ≥ abs (2∗C∗z) − abs s using z-def by auto
      hence abs B ≥ 2∗abs C ∗ abs z − abs s by auto
      hence abs B ≥ 2∗abs C − abs s using n0 mult-left-mono[of 1 abs z 2∗abs
C] by linarith
     hence abs B ≥ abs C + ψ (abs A) (nat (abs s)) − abs s using eq-ψ-int[of
A s] s-def A-B2
        by auto
      hence abs B ≥ abs C using lucas-monotone3[of A nat (abs s)] A-B2
        by (smt (verit, ccfv-SIG) C-Bigger-A assms(2) le-add1 lucas-monotone3
nat-le-iff of-nat-1 of-nat-add)
      hence abs B ≥ abs A using C-Bigger-A by auto
      then show ?thesis using assms by auto
    qed
  qed
 qed
```

hence $(B - s)$ *mod* $(2*C) = 0$ *mod* $(2*C)$ **using** *mod-diff-cong*[*of* B $2*C$ s s s] **by** *auto*
then obtain z **where** *z-def*: $B- s = 2*C*z$ **by** *auto*
have $B=s$
proof −
consider (0) $z=0$ | $(n0)$ *abs* $z \geq 1$ **by** *linarith*
then show *?thesis*
proof (*cases*)
case *0*
then show *?thesis* **using** *z-def* **by** *auto*
next
case *n0*
have *abs* $B \geq$ *abs* $(2*C*z) -$ *abs* s **using** *z-def* **by** *auto*
hence *abs* $B \geq 2*$*abs* $C *$ *abs* $z -$ *abs* s **by** *auto*
hence *abs* $B \geq 2*$*abs* $C -$ *abs* s **using** *n0 mult-left-mono*[*of 1 abs z 2*∗*abs C*] **by** *linarith*
hence *abs* $B \geq$ *abs* $C + \psi$ (*abs* A) (*nat* (*abs* s)) $-$ *abs* s **using** *eq-ψ-int*[*of A s*] *s-def A-B2*
by *auto*
hence *abs* $B \geq$ *abs* C **using** *lucas-monotone3*[*of A nat* (*abs* s)] *A-B2*
by (*smt* (*verit*, *ccfv-SIG*) *C-Bigger-A assms*(*2*) *le-add1 lucas-monotone3 nat-le-iff of-nat-1 of-nat-add*)
hence *abs* $B \geq$ *abs* A **using** *C-Bigger-A* **by** *auto*
then show *?thesis* **using** *assms* **by** *auto*
qed
qed
then show *?thesis* **using** *s-def* **by** *auto*
qed

end
theory *Bridge-Theorem-Imp*
imports *HOL.Binomial*
../MPoly-Utils/Poly-Extract
../Lucas-Sequences/DFI-square-0
../Lucas-Sequences/Lucas-Diophantine
../Lucas-Sequences/Lemma-4-4
begin

6 The Bridge Theorem

6.1 Constructing polynomials

context *bridge-variables*
begin

definition L :: *int* ⇒ *int* ⇒ *int*
where $L\ l\ Y = l * Y$
definition U :: *int* ⇒ *int* ⇒ *int* ⇒ *int*

where $U\ l\ X\ Y = 2 * X * L\ l\ Y$
definition $V :: int \Rightarrow int \Rightarrow int \Rightarrow int$
where $V\ w\ g\ Y = 4 * w * g * Y$
definition $A :: int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
where $A\ l\ w\ g\ Y\ X = U\ l\ X\ Y * (V\ w\ g\ Y + 1)$
definition $B :: int \Rightarrow int$
where $B\ X = 2 * X + 1$
definition $C :: int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
where $C\ l\ w\ h\ g\ Y\ X = B\ X + (A\ l\ w\ g\ Y\ X - 2) * h$
definition $D :: int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
where $D\ l\ w\ h\ g\ Y\ X = ((A\ l\ w\ g\ Y\ X)^2 - 4) * (C\ l\ w\ h\ g\ Y\ X)^2 + 4$
definition $E :: int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
where $E\ l\ w\ h\ x\ g\ Y\ X = (C\ l\ w\ h\ g\ Y\ X)^2 * D\ l\ w\ h\ g\ Y\ X * x$
definition $F :: int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
where $F\ l\ w\ h\ x\ g\ Y\ X = 4 * ((A\ l\ w\ g\ Y\ X)^2 - 4) * (E\ l\ w\ h\ x\ g\ Y\ X)^2 + 1$
definition $G :: int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
where $G\ l\ w\ h\ x\ g\ Y\ X = 1 + C\ l\ w\ h\ g\ Y\ X * D\ l\ w\ h\ g\ Y\ X * F\ l\ w\ h\ x\ g\ Y\ X -$
$2 * (A\ l\ w\ g\ Y\ X + 2) * (A\ l\ w\ g\ Y\ X - 2)^2 * (E\ l\ w\ h\ x\ g\ Y\ X)^2$
definition $H :: int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
where $H\ l\ w\ h\ x\ y\ g\ Y\ X = C\ l\ w\ h\ g\ Y\ X + B\ X * F\ l\ w\ h\ x\ g\ Y\ X + (2 * y - 1) *$
$C\ l\ w\ h\ g\ Y\ X * F\ l\ w\ h\ x\ g\ Y\ X$
definition $I :: int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
where $I\ l\ w\ h\ x\ y\ g\ Y\ X = ((G\ l\ w\ h\ x\ g\ Y\ X)^2 - 1) * (H\ l\ w\ h\ x\ y\ g\ Y\ X)^2 + 1$
definition $W :: int \Rightarrow int \Rightarrow int$
where $W\ w\ b = b * w$
definition $K :: int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
where $K\ k\ l\ w\ g\ Y\ X = X + 1 + k * ((U\ l\ X\ Y)^2 * V\ w\ g\ Y - 2)$
definition $J :: int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
where $J\ k\ l\ w\ g\ Y\ X = K\ k\ l\ w\ g\ Y\ X$

definition $S :: int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$ **where**
$S\ l\ w\ g\ Y\ X = 2 * A\ l\ w\ g\ Y\ X - 5$

definition $T :: int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$ **where**
$T\ l\ w\ h\ g\ Y\ X\ b = 3 * (W\ w\ b) * (C\ l\ w\ h\ g\ Y\ X) - 2 * ((W\ w\ b)^2 - 1)$

poly-extract L
poly-extract U
poly-extract V
poly-extract A
poly-extract B
poly-extract C
poly-extract D
poly-extract E
poly-extract F

poly-extract *G*
poly-extract *H*
poly-extract *I*
poly-extract *W*
poly-extract *K*
poly-extract *S*
poly-extract *T*

definition *d2a* **where** *d2a l w h x y g Y X = is-square*(*D l w h g Y X* ∗ *F l w h x g Y X*
∗ *I l w h x y g Y X*)
definition *d2b* **where** *d2b k l w x g Y X = is-square*((*U l X Y^4*∗*V w g Y^2*−*4*)∗*K k l w g Y X^2+4*)
definition *d2c* **where**
d2c l w h b g Y X ≡ *S l w g Y X dvd T l w h g Y X b*
definition *d2d* **where** *d2d b w X*=(*b*∗*w*=*2^nat*(*B X*))
definition *d2e* **where** *d2e k l w h g Y X*=((*4*∗*g*∗(*C l w h g Y X*)−*4*∗*g*∗(*L l Y*)∗
(*K k l w g Y X*)) *^2*<(*K k l w g Y X*) *^2*)
definition *d2f* **where** *d2f k l w h g Y X*=((*2*∗(*C l w h g Y X*)−*2*∗(*L l Y*)∗
(*K k l w g Y X*)) *^2*<(*K k l w g Y X*) *^2*)

definition *statement1* **where**
statement1 b Y X ≡ *is-power2 b*
∧ *Y dvd int* (*2* ∗ *nat X choose nat X*)

definition *statement2* **where**
statement2 b Y X g = (∃ *h k l w x y*::*int*.(*l*∗*x*≠*0*)∧(*d2a l w h x y g Y X*)∧
(*d2b k l w x g Y X*)∧(*d2c l w h b g Y X*)∧(*d2f k l w h g Y X*))
definition *statement2a* **where**
statement2a b Y X g = (∃ *h k l w x y*::*int*.(*d2a l w h x y g Y X*)∧
(*d2b k l w x g Y X*)∧(*d2c l w h b g Y X*)∧(*d2e k l w h g Y X*)
∧ (*h*≥*b*)∧(*k*≥*b*)∧(*l*≥*b*)∧(*w*≥*b*)∧(*x*≥*b*)∧(*y*≥*b*))

end

lemma *min-power*:
fixes *a*::*int* **and** *n*::*nat*
assumes *a* ≥ *3*
shows (*a*−*1*) *^*(*n*+*2*) ≥ *3* + *int n* + (*a*−*2*) *^2*
proof (*induction n*)
case *0*
have (*a*−*1*) *^2* = *1* − *2*∗*a* + *a^2* **by** (*auto simp add*: *algebra-simps power2-eq-square*)
also have ... = *2*∗*a* − *3* + (*a^2* − *4*∗*a* + *4*) **by** *auto*
also have ... = *2*∗*a* − *3* + (*a*−*2*) *^2* **by** (*auto simp add*: *algebra-simps power2-eq-square*)
also have ... ≥ *3* + (*a*−*2*) *^2* **using** *assms* **by** *auto*
finally show *?case* **using** *plus-nat.add-0* **by** *presburger*
next

case (*Suc n*)
note *t* = *this*
have (*a*−*1*) ^(*Suc n* + *2*) = (*a*−*1*)∗(*a*−*1*) ^(*n*+*2*) **by** *auto*
then have (*a*−*1*) ^(*Suc n* + *2*) ≥ *2*∗(*a*−*1*) ^(*n*+*2*) **using** *assms* **by** *auto*
then have (*a*−*1*) ^(*Suc n* + *2*) ≥ (*a*−*1*) ^(*n*+*2*) + (*a*−*1*) ^(*n*+*2*) **by** *auto*
then have (*a*−*1*) ^(*Suc n* + *2*) ≥ *1* + (*a*−*1*) ^(*n*+*2*) **using** *assms*
by (*smt* (*verit*) *one-le-power power-one*)
then show *?case* **using** *t* **by** *auto*
qed

lemma *change-sum*:
fixes *f*::*int* ⇒ *int* **and** *n*::*nat*
shows (∑ *i*≤*n*. (*f* (*int i*))) = *sum* (λ*i*. *f i*) (*set*[*0*..*int n*])
proof (*induction n*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc n*)
note *t* = *this*
have (∑ *i*≤*Suc n*. *f* (*int i*)) = (∑ *i*≤*n*. *f* (*int i*)) + *f* (*int* (*Suc n*)) **by** *auto*
hence (∑ *i*≤*Suc n*. *f* (*int i*)) = *sum* (λ*i*. *f i*) (*set*[*0*..*int n*]) + *f* (*int* (*Suc n*))
using *t* **by** *auto*
hence *first-eq*: (∑ *i*≤*Suc n*. *f* (*int i*)) = *sum* (λ*i*. *f i*) (*set*[*0*..*int n*])
+ *sum* (λ*i*. *f i*) (*set*[*int* (*Suc n*)..*int* (*Suc n*)])
by *auto*
have (*set*[*0*..*int n*]) ∩ (*set*[*int* (*Suc n*)..*int* (*Suc n*)]) = {} ∧
(*set*[*0*..*int n*]) ∪ (*set*[*int* (*Suc n*)..*int* (*Suc n*)]) = (*set*[*0*..*int* (*Suc n*)])
by *auto*
hence (∑ *i*≤*Suc n*. *f* (*int i*)) = *sum* (λ*i*. *f i*) (*set*[*0*..*int* (*Suc n*)])
using *first-eq* **by** (*metis List.finite-set sum.union-disjoint*)
then show *?case* **by** *simp*
qed

lemma *chang-var1*:
fixes *f*::*int* ⇒ *int* **and** *n*::*nat*
shows *sum* (λ*i*. *f* (*i*+*1*)) (*set*[*0*..*int n*]) = *sum* (λ*i*. *f i*) (*set*[*1*..*int* (*Suc n*)])
proof (*induction n*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc n*)
note *t* = *this*
have (*set*[*0*..*int n*])∩{*int* (*Suc n*)} = {} ∧ (*set*[*0*..*int n*])∪{*int* (*Suc n*)} =
(*set*[*0*..*int* (*Suc n*)])
∧ *finite* {*int* (*Suc n*)} ∧ *finite* (*set*[*0*..*int n*]) **by** *auto*
hence *eq1*: *sum* (λ*i*. *f* (*i*+*1*)) (*set*[*0*..*int* (*Suc n*)]) = *sum* (λ*i*. *f* (*i*+*1*)) (*set*[*0*..*int*
n])
+ *f* (*int* (*Suc n*) +*1*)
using *sum.union-disjoint* **by** (*smt* (*z3*) *Int-Un-eq*(*3*) *Un-insert-right disjoint-insert*(*2*)

sum.insert-if)
have (*set*[*1..int* (*Suc n*)])∩{*int* (*Suc* (*Suc n*))} = {} ∧ (*set*[*1..int* (*Suc n*)])∪{*int* (*Suc* (*Suc n*))}
= (*set*[*1..int* (*Suc* (*Suc n*))]) ∧ *finite* {*int* (*Suc* (*Suc n*))} ∧ *finite* (*set*[*1..int* (*Suc n*)])
by *auto*
hence *sum* (λ*i*. *f i*) (*set*[*1..int* (*Suc* (*Suc n*))]) = *sum* (λ*i*. *f i*) (*set*[*1..int* (*Suc n*)])
+ *f* (*int* (*Suc n*) + *1*)
using *sum.union-disjoint*[*of* (*set*[*1..int* (*Suc n*)]) {*int* (*Suc* (*Suc n*))} *f*] **by** *auto*
then show *?case* **using** *t eq1* **by** *auto*
qed

lemma *chang-var*:
fixes *f*::*int* ⇒ *int* **and** *n*::*nat* **and** *m*::*nat*
shows *sum* (λ*i*. *f i*) (*set*[*int n..int* (*n*+*m*)]) = *sum* (λ*i*. *f* (*int* (*n*+*m*) − *i*)) (*set*[*0..int m*])
proof (*induction m*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc m*)
note *t* = *this*
have (*set*[*int n..int* (*n*+*m*)])∩(*set*[*int* (*n*+*Suc m*)*..int* (*n*+*Suc m*)]) = {}
∧ (*set*[*int n..int* (*n*+*m*)])∪(*set*[*int* (*n*+*Suc m*)*..int* (*n*+*Suc m*)]) = (*set*[*int n..int* (*n*+*Suc m*)])
∧ *finite* (*set*[*int n..int* (*n*+*m*)]) ∧ *finite* (*set*[*int* (*n*+*Suc m*)*..int* (*n*+*Suc m*)])
by *auto*
hence *sum* (λ*i*. *f i*) (*set*[*int n..int* (*n*+*Suc m*)]) = *sum* (λ*i*. *f i*) (*set*[*int n..int* (*n*+*m*)])
+ *f* (*int* (*n* + *Suc m*))
using *sum.union-disjoint*[*of* (*set*[*int n..int* (*n*+*m*)]) (*set*[*int* (*n* + *Suc m*)*..int* (*n*+*Suc m*)]) *f*]
by *auto*
hence *eq1*: *sum* (λ*i*. *f i*) (*set*[*int n..int* (*n*+*Suc m*)]) = *sum* (λ*i*. *f* (*int* (*n*+*m*) − *i*)) (*set*[*0..int m*])
+ *f* (*int* (*n* + *Suc m*))
using *t* **by** *auto*
have *int* (*n*+ *Suc m*) − (*i*+*1*) = *int* (*n*+*m*) − *i* **for** *i* **by** *auto*
hence *sum* (λ*i*. *f* (*int* (*n*+*m*) − *i*)) (*set*[*0..int m*]) = *sum* (λ*i*. *f* (*int* (*n*+*Suc m*) − (*i*+*1*))) (*set*[*0..int m*])
by *presburger*
hence *eq2*: *sum* (λ*i*. *f i*) (*set*[*int n..int* (*n*+*Suc m*)]) =
sum (λ*i*. *f* (*int* (*n*+*Suc m*) − (*i*+*1*))) (*set*[*0..int m*]) + *f* (*int* (*n* + *Suc m*))
using *eq1* **by** *presburger*
define *g*::*int*⇒*int* **where** *g i* = *f* (*int* (*n* + *Suc m*) − *i*) **for** *i*
hence *sum* (λ*i*. *f i*) (*set*[*int n..int* (*n*+*Suc m*)]) = *sum* (λ*i*. *g* (*i*+*1*)) (*set*[*0..int m*]) + *g 0*

using *eq2* **by** *auto*
hence *eq3*: *sum* (λ*i*. *f i*) (*set*[*int n*..*int* (*n*+*Suc m*)]) = *sum* (λ*i*. *g i*) (*set*[*1*..*int* (*Suc m*)]) + *g 0*
using *chang-var1*[*of g m*] **by** *auto*
have (*set*[*1*..*int* (*Suc m*)])∪{*0*} = (*set*[*0*..*int* (*Suc m*)]) ∧ (*set*[*1*..*int* (*Suc m*)])∩{*0*} = {}
∧ *finite* (*set*[*1*..*int* (*Suc m*)]) ∧ *finite* {*0*} **by** *auto*
hence *sum* (λ*i*. *f i*) (*set*[*int n*..*int* (*n*+*Suc m*)]) = *sum* (λ*i*. *g i*) (*set*[*0*..*int* (*Suc m*)])
using *eq3 sum.union-disjoint*[*of* {*0*} (*set*[*int n*..*int* (*n*+*Suc m*)]) *g*]
by (*smt* (*verit*) *Int-empty-right finite-insert insert-disjoint*(*2*) *sum.insert sum-Un*)
then show *?case* **using** *g-def* **by** *auto*
qed

6.2 Proof of implication (*1*)⟹(*3*)

context *bridge-variables*
begin

lemma *theorem-II-1-3*:
assumes *b-def1*:(*b*::*int*)≥*0* **and** *Y-def*:(*Y*::*int*)≥*b*∧*Y*≥*2^8* **and** *X-def*:(*X*::*int*)≥*3*∗*b*
and *g-def*:(*g*::*int*)≥*1*
shows (*statement1 b Y X*) ⟹ (*statement2a b Y X g*)
proof −
assume *state1*:*statement1 b Y X*
hence *b-def*: *b* ≥ *1* **unfolding** *statement1-def is-power2-def*
using *b-def1* **by** *force*

hence *X-pos*: *X* > *0* **using** *assms* **by** *auto*

define *w* **where** *w-def*:*w*=(*2^nat*(*B X*))*div b*
have *is-power2-b*:*is-power2 b* **using** *state1* **and** *statement1-def* **by** *auto*
have *is-power2B-b-bsq*:*2^nat* (*B X*) > *b^2*
proof −
have *obvious-natX*: *nat* (*2*∗*X*+*1*) = *1*+*2*∗*nat X* **using** *X-pos* **by** *auto*
have *pow-2-useful*: (*2*::*int*) ^(*2*∗*n*+*1*) = *2*∗(*2^n*) ^*2* **for** *n*
proof (*induction n*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc n*)
then show *?case* **by** (*auto simp add*: *algebra-simps*)
qed
have *2^n* ≥ *int n* **for** *n*
proof (*induction n*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc n*)

```
      note t = this
      have (2::int) ^(Suc n) - 2^n = 2^n by auto
      then have (2::int) ^(Suc n) - 2^n ≥ 1 by auto
      then have (2::int) ^(Suc n) ≥ 2^n + 1 by auto
      then have (2::int) ^(Suc n) ≥ int n + 1 using t by linarith
      then show ?case by auto
    qed
    hence pow-2-bigger: ((2::int) ^n) ^2≥ (int n) ^2 for n by auto
    have (2::int) ^nat(B X) = 2^nat(2*X+1)
      by (simp add: B-def)
    then have (2::int) ^nat (B X) = 2^(1+2*nat X)
      using obvious-natX by simp
    hence (2::int) ^nat (B X) = 2*(2^nat X) ^2
      using pow-2-useful by auto
    hence (2::int) ^nat(B X) > (2^nat X) ^2 by auto
      hence 2^nat(B X) > (int (nat X)) ^2 using pow-2-bigger[of nat X] by
linarith
      hence 2^nat (B X) > X^2 using X-pos by auto
      thus ?thesis using X-def b-def
        by (smt (verit, best) power2-less-imp-less)
  qed
  have is-power2B-b-b:2^nat (B X) > b
  proof −
    show ?thesis using b-def and is-power2B-b-bsq
      by (smt (verit, ccfv-threshold) self-le-power zero-less-numeral)
  qed
```

Proof of d2d and $w \geq b$

```
  have ∃k. b = 2^k using is-power2-b unfolding is-power2-def
    by (simp add: b-def1)

  then obtain k where k-def: b=2^k by auto
  then have k < nat(B X) using is-power2B-b-b by auto
  hence w = 2^(nat(B X) - k) using k-def w-def
    by (simp add: ‹k < nat (B X)› less-imp-le-nat power-diff-power-eq)
  hence sat-4-2d:d2d b w X using k-def unfolding d2d-def
    by (metis Nat.add-diff-assoc ‹k < nat (B X)› add-diff-cancel-left' less-imp-le-nat
power-add)
  have wBeb:w≥b
  proof −
   have w*b>b^2 using sat-4-2d is-power2B-b-bsq d2d-def by (metis mult.commute)
    hence w*b>b*b by (simp add: power2-eq-square)
    thus ?thesis using b-def by auto
  qed

  have b ≥ 1 using is-power2-b using b-def by auto
  hence VBe1: V w g Y ≥ 1 unfolding V-def using assms wBeb mult-mono[of
1 w 1 g*Y]
    mult-mono[of 1 g 1 Y] by auto
```

Introduction of ϱ

define *ϱ-int* **where** *pI-def*:ϱ-*int* = ((2∗*nat X*) *choose nat X*) + *nat*((*V w g Y*)∗*sum*
(λ*i*. *int* (2∗*nat X choose nat i*) ∗ (*V w g Y*) ^*nat* (*i*−*X*−1)) (*set*[*X*+1..2∗*X*]))

define *l* **where** *l-def*:*l*=*int* ϱ-*int div Y*
have ϱ-*int* ≥ *nat*((*V w g Y*)∗*sum*
(λ*i*. *int* (2∗*nat X choose nat i*) ∗ (*V w g Y*) ^*nat* (*i*−*X*−1)) (*set*[*X*+1..2∗*X*]))
using *pI-def* **by** *auto*
then have ϱ-*int-min1*: *int* ϱ-*int* ≥ (*V w g Y*)∗*sum*
(λ*i*. *int* (2∗*nat X choose nat i*) ∗ (*V w g Y*) ^*nat* (*i*−*X*−1)) (*set*[*X*+1..2∗*X*])
by *auto*
have *V-pos*: *V w g Y* >0 **unfolding** *V-def* **using** *assms wBeb b-def* **by** *force*
hence *V-pos2*: (*V w g Y*) ^(*nat i*) ≥1 **for** *i* **by** *auto*
have *binom-pos*: *int* (2∗*nat X choose nat i*) ≥ 0 **for** *i* **by** *auto*
have *binom-strict-pos*: ⋀*i*. *i*∈*set*[*X*+1..2∗*X*] ⟹ *int* (2∗*nat X choose nat i*)
> 0
proof −
fix *i*::*int*
assume *i-def*: *i*∈*set*[*X*+1..2∗*X*]
show *int* (2∗*nat X choose nat i*) > 0 **using** *zero-less-binomial*[*of nat i* 2∗*nat*
X] *i-def*
by *auto*
qed
have *min-binom1*: *int* (2∗*nat X choose nat i*) ∗ (*V w g Y*) ^*nat* (*i*−*X*−1)
≥ *int* (2∗*nat X choose nat i*) **for** *i*
using *V-pos2*[*of i*−*X*−1] *binom-pos*[*of i*] **by** (*simp add*: *mult-le-cancel-left1*)
hence *terms-pos*: ⋀*i*. *i*∈*set*[*X*+1..2∗*X*] ⟹ *int* (2∗*nat X choose nat i*) ∗ (*V*
w g Y) ^*nat* (*i*−*X*−1) > 0
using *binom-strict-pos* **by** (*smt* (*z3*))
define *enof*::*int*⇒*int* **where** ⋀*i*. *enof i* = *int* (2∗*nat X choose nat i*) ∗ (*V w*
g Y) ^*nat* (*i*−*X*−1)
have *set*[*X*+1..2∗*X*] ≠ {} ∧ *finite* (*set*[*X*+1..2∗*X*]) **using** *X-pos* **by** *auto*
hence *sum-pos*: *sum* (λ*i*. *int* (2∗*nat X choose nat i*) ∗ (*V w g Y*) ^*nat* (*i*−*X*−1))
(*set*[*X*+1..2∗*X*]) > 0
using *terms-pos Groups-Big.ordered-comm-monoid-add-class.sum-pos*[*of set*[*X*+1..2∗*X*]
enof]
enof-def **by** *auto*
have *lBeb*:*l*≥*b*
proof −
have *sum* (λ*i*. *int* (2∗*nat X choose nat i*) ∗ (*V w g Y*) ^*nat* (*i*−*X*−1))
(*set*[*X*+1..2∗*X*]) ≥ 1
using *sum-pos* **by** *auto*
hence *V w g Y* ∗ (∑ *i*∈*set* [*X* + 1..2 ∗ *X*]. *int* (2∗*nat X choose nat i*) ∗
V w g Y ^ *nat* (*i* − *X* − 1)) ≥ *V w g Y* **using** *V-pos* **by** *auto*
hence *int* ϱ-*int*≥*V w g Y* **using** ϱ-*int-min1* **by** *auto*
hence *l*≥(*V w g Y*)*div Y* **using** *l-def* **using** *Y-def zdiv-mono1* **by** *auto*
hence *l*≥4∗*g*∗*w* **using** *V-def*

by (*smt* (*verit*, *ccfv-threshold*) *Y-def b-def int-distrib*(*1*)
mult.commute nonzero-mult-div-cancel-right)
hence *l*≥*w* **using** *g-def* **by** (*smt* (*verit*) *b-def mult-le-cancel-right1 wBeb*)
thus *?thesis* **using** *wBeb* **by** *auto*
qed

Introduction of h

define *h* **where** *h* = (*ψ* (*A l w g Y X*) (*nat*(*B X*)) − *B X*) *div* (*A l w g Y X* − *2*)
have *UBe2*:*U l X Y*≥*2* **using** *U-def lBeb Y-def X-def b-def*
by (*smt* (*verit*, *ccfv-SIG*) *L-def b-def mult-le-cancel-left1*)
have *A-Be-2Vp1*:*A l w g Y X* ≥ *2* ∗ (*V w g Y* + *1*)
proof −
have *V w g Y* + *1* ≥*0* **using** *VBe1* **by** *simp*
hence *maj1*: *U l X Y* ∗ (*V w g Y* + *1*) ≥ *2* ∗ (*V w g Y* + *1*)
using *UBe2 Rings.ordered-semiring-class.mult-right-mono*[*of 2 U l X Y V w g Y* + *1*]
by *simp*
have *A l w g Y X* = *U l X Y* ∗ (*V w g Y* + *1*) **using** *A-def* **by** *auto*
thus *?thesis* **using** *maj1* **by** *simp*
qed
have *ABe4*:*A l w g Y X* ≥ *4*
proof −
show *?thesis* **using** *VBe1 A-Be-2Vp1* **by** *auto*
qed
have *VBw*:*V w g Y* > *w*
proof −
have *V w g Y*=*4*∗*g*∗*Y*∗*w* **using** *V-def* **by** *auto*
thus *?thesis* **using** *Y-def g-def wBeb b-def*
by (*smt* (*verit*, *best*) *mult-cancel-right1 mult-le-cancel-right1*)
qed
have *BBe3*: *nat*(*B X*) ≥ *3* **unfolding** *B-def* **using** *assms b-def* **by** *auto*
have *h-def2*:*ψ* (*A l w g Y X*) (*nat*(*B X*)) = *B X* + (*A l w g Y X* − *2*)∗*h*
proof −
have *ψ* (*A l w g Y X*) (*nat*(*B X*)) *mod* (*A l w g Y X* − *2*) = *B X mod* (*A l w g Y X* − *2*)
using *BBe3 lucas-congruence2*[*of A l w g Y X nat* (*B X*)] *ABe4* **by** *simp*
hence (*ψ* (*A l w g Y X*) (*nat*(*B X*)) − *B X*) *mod* (*A l w g Y X* − *2*) = *0*
using *ABe4 mod-diff-eq*
[*of* (*ψ* (*A l w g Y X*) (*nat*(*B X*))) *A l w g Y X* − *2 B X*]
by *simp*
thus *?thesis* **using** *h-def* **by** *auto*
qed
have *hBeb*:*h*≥*b*
proof −
have *ψ* (*A l w g Y X*) (*nat* (*B X*)) ≥ *B X* + (*A l w g Y X* − *2*)^*2*
proof −
have *BBe3*: *B X* ≥ *3* **using** *X-pos B-def* **by** (*smt* (*verit*, *ccfv-threshold*))
have *some-trivial-facts*: *nat* (*B X*) − *3* + *2* = *nat* (*B X*) − *1* ∧ *3* + *int*

(nat (B X) − 3) = B X
using *BBe3* **by** *auto*
have *Suc (Suc (nat (B X) − 2)) = nat (B X) ∧ nat (B X) − 2 + 1 = nat (B X) − 1*
using *BBe3* **by** *auto*
hence *ψ (A l w g Y X) (nat (B X)) ≥ (A l w g Y X − 1) ^(nat (B X) − 1)*
using *lucas-exp-growth-gt[of A l w g Y X nat (B X)−2] ABe4* **by** *auto*
hence *ψ (A l w g Y X) (nat (B X)) ≥ B X + (A l w g Y X−2) ^2*
using *min-power[of A l w g Y X nat (B X) − 3] BBe3 ABe4 some-trivial-facts* **by** *fastforce*
then show *?thesis* **by** *simp*
qed
hence *B X + (A l w g Y X − 2)∗h ≥ B X + (A l w g Y X − 2) ^2*
using *h-def2* **by** *auto*
hence *(A l w g Y X − 2)∗h ≥ (A l w g Y X − 2) ^2* **by** *auto*
hence *h≥A l w g Y X − 2* **using** *ABe4 power2-eq-square[of A l w g Y X − 2]* **by** *auto*
hence *h≥2∗V w g Y* **using** *A-Be-2Vp1* **by** *auto*
hence *h≥w* **using** *VBw VBe1* **by** *auto*
thus *?thesis* **using** *wBeb* **by** *auto*
qed

Introduction of x y and d2a

define *A-g* **where** *A-g = A l w g Y X*
define *B-g* **where** *B-g = B X*
define *C-g* **where** *C-g = ψ A-g (nat B-g)*
have *C-well-def*: *C-g = C l w h g Y X* **unfolding** *C-g-def C-def A-g-def B-g-def*

using *h-def2* **by** *simp*
then have *D-well-def*: *D l w h g Y X = D-f A-g C-g* **unfolding** *D-def D-f-def A-g-def* **by** *simp*
then have *E-well-def*: *E l w h x g Y X = E-ACx A-g C-g x* **for** *x*
unfolding *E-def E-ACx-def E-f-def A-g-def* **using** *C-well-def* **by** *simp*
have *F-well-def*: *F l w h x g Y X = F-ACx A-g C-g x* **for** *x*
unfolding *F-def F-ACx-def F-f-def* **using** *C-well-def E-well-def[of x] D-well-def A-g-def*
by *simp*
have *G-well-def*: *G l w h x g Y X = G-ACx A-g C-g x* **for** *x*
unfolding *G-def G-ACx-def G-f-def* **using** *D-well-def E-well-def[of x] F-well-def[of x]*
C-well-def A-g-def **by** *simp*
have *H-well-def*: *H l w h x y g Y X = H-ABCxy A-g B-g C-g x y* **for** *x y*
unfolding *H-def H-ABCxy-def H-f-def B-g-def* **using** *C-well-def F-well-def[of x]* **by** *simp*
have *I-well-def*: *I l w h x y g Y X = I-ABCxy A-g B-g C-g x y* **for** *x y*
unfolding *I-def I-ABCxy-def I-f-def* **using** *G-well-def[of x] H-well-def[of x y]* **by** *auto*
have *conditions-req*: *A-g ≥ 4 ∧ even A-g ∧ B-g ≥ 3*

unfolding *A-g-def B-g-def B-def* **using** *ABe4 X-pos A-def U-def* **by** *auto*
have $\exists x{\geq}b.\ \exists y{\geq}b.$ *d2a l w h x y g Y X*
unfolding *d2a-def* **using** *lemma-4-2-cor*[*of A-g B-g*]
by (*simp add*: *C-g-def D-well-def F-well-def I-well-def lemma-4-2-cor*[*of A-g B-g*]
conditions-req is-square-def)
then obtain *x y* **where** *sat-4-2a*:$x \geq b \wedge y \geq b \wedge$ *d2a l w h x y g Y X* **by** *auto*

Introduction of k

have *X-plus-1Be2*: *nat X + 1* $\geq$ *2* **using** *X-pos* **by** *auto*
hence *intnatX*: *int* (*nat X +1*) = *X + 1* **by** *auto*
have *V-pos*: *V w g Y > 0* **unfolding** *V-def* **using** *assms wBeb b-def* **by** *force*
hence *U2VBe3*: *U l X Y ^2 * V w g Y* $\geq$ *3* **using** *UBe2 power2-eq-square*
by (*smt* (*verit*, *ccfv-SIG*) *VBw b-def mult-cancel-right2 mult-less-cancel-right2 wBeb*)
hence ψ (*U l X Y ^2*V w g Y*) (*nat X + 1*) *mod* (*U l X Y ^2*V w g Y−2*)
= *int* (*nat X+1*) *mod* (*U l X Y ^2*V w g Y−2*)
using *lucas-congruence2*[*of U l X Y ^2 * V w g Y nat X + 1*] *X-plus-1Be2* **by** *auto*
hence ψ (*U l X Y ^2*V w g Y*) (*nat X + 1*) *mod* (*U l X Y ^2*V w g Y−2*)
= (*X+1*) *mod* (*U l X Y ^2*V w g Y−2*)
using *intnatX* **by** *presburger*
then obtain *k* **where** *k-def*: ψ ((*U l X Y*) *^2*V w g Y*) (*nat X + 1*)=*X+1*+((*U l X Y*) *^2*V w g Y−2*)*k*
by (*metis mod-eqE*)
have *kBeb*:*k*$\geq$*b*
proof −
have *weird-eq*: *nat X − 2 + 2 = nat X* $\wedge$ *3+int* (*nat X − 2*) = *X +1* **using** *assms b-def* **by** *auto*
have *Suc* (*Suc* (*nat X −1*)) = *nat X +1* $\wedge$ *nat X − 1 + 1 = nat X* **using** *X-pos* **by** *auto*
hence ψ ((*U l X Y*) *^2*V w g Y*) (*nat X +1*)$\geq$((*U l X Y*) *^2*V w g Y−1*) *^nat X*
using *lucas-exp-growth-gt*[*of* (*U l X Y*) *^2*V w g Y nat X−1*] *U2VBe3* **by** *auto*
hence ψ ((*U l X Y*) *^2*V w g Y*) (*nat X +1*)$\geq$*1+X+*((*U l X Y*) *^2*V w g Y−2*) *^2*
using *min-power*[*of* (*U l X Y*) *^2*V w g Y nat X − 2*] *weird-eq U2VBe3*
by (*smt* (*verit*, *del-insts*))
hence *X+1+*((*U l X Y*) *^2*V w g Y−2*)*k*$\geq$*1+X+*((*U l X Y*) *^2*V w g Y−2*) *^2* **using** *k-def* **by** *auto*
hence ((*U l X Y*) *^2*V w g Y−2*)*k*$\geq$((*U l X Y*) *^2*V w g Y−2*) *^2* **by** *auto*
hence ((*U l X Y*) *^2*V w g Y−2*)*k*$\geq$((*U l X Y*) *^2*V w g Y−2*)*((*U l X Y*) *^2*V w g Y−2*)
by (*simp add*: *power2-eq-square*)
hence *ineq1*: *k*$\geq$((*U l X Y*) *^2*V w g Y−2*) **using** *UBe2 VBe1*
by (*smt* (*verit*, *best*) *mult-le-cancel-left1 mult-le-cancel-left-pos one-power2*

power2-diff)
have *U2Be4*: *U l X Y ^2≥ 4* **using** *UBe2 power2-eq-square*[*of U l X Y*]
by (*smt* (*verit*, *best*) *dbl-simps*(*3*) *dbl-simps*(*5*) *numeral-eq-one-iff numeral-times-numeral one-power2 power2-le-imp-le power2-sum semiring-norm*(*11*))
hence (*U l X Y*)^*2*V w g Y−2 >= *4*V w g Y − 2* **using** *V-pos* **by** *auto*
hence *ineq2*: *k ≥ 4*V w g Y − 2* **using** *ineq1* **by** *auto*
have *VBeb*:*V w g Y≥b* **using** *wBeb g-def Y-def V-def VBe1*
by (*smt* (*verit*) *mult-le-cancel-right1*)
hence *k ≥ 4*b − 2* **using** *ineq2* **by** *auto*
hence *k ≥ b + 1* **using** *assms b-def* **by** *auto*
thus *?thesis* **by** *auto*
qed

Proof of d2b

have *sat-4-d2b*: *d2b k l w x g Y X* **unfolding** *d2b-def is-square-def*
proof −
have *0*: *U l X Y ^ 4 * (V w g Y)*^{2} *= (U l X Y ^ 2 * (V w g Y))^2* **by** *algebra*
have *X + 1 + k * ((U l X Y)*^{2} ** V w g Y − 2) = ψ ((U l X Y)*^{2} ** V w g Y) (nat X+1)*
using *k-def* **by** *simp*
thus ∃ *m*. *(U l X Y ^ 4 * (V w g Y)^2 − 4) * (K k l w g Y X)*^{2} *+ 4 = m*^{2}
using *lucas-pell-part2*[*of K k l w g Y X U l X Y ^2*V w g Y*] *K-def 0* **by** *auto*
qed

Proof of d2c

have *sat-4-d2c*: *d2c l w h b g Y X*
proof −
have *A-pos*:*0 < A l w g Y X* **unfolding** *A-def* **using** *UBe2 VBe1* **by** *simp*
have *B-pos*: *B X ≥ 0* **using** *assms* **unfolding** *B-def* **by** *force*
have *hmm*: *((2::int) ^ (nat (B X))) ^ 2 = (2::int) ^ (nat (B X) * 2)*
using *Semiring-Normalization.comm-semiring-1-class.semiring-normalization-rules*(*31*) [*of 2 nat*(*B X*) *2*] **by** *fast*
hence *W2*: *(W w b)*^{2} *= (2::int) ^ (2 * nat (B X))* **using** *sat-4-2d* **unfolding** *d2d-def W-def*
by *force*
have *C-psi*: *ψ (A l w g Y X) (nat (B X)) = C l w h g Y X* **using** *C-g-def C-well-def A-g-def*
B-g-def **by** *fast*
have *3 * 2 ^ nat (B X) * ψ (A l w g Y X) (nat (B X)) mod (2 * A l w g Y X − 5)*
*= 2 * (2 ^ (2 * nat (B X)) − 1) mod (2 * A l w g Y X − 5)*
using *sat-4-2d* **unfolding** *W-def d2d-def*
using *lucas-diophantine-dir*[*of nat (B X) A l w g Y X*] **by** *force*
hence *3 * W w b * C l w h g Y X mod (2 * A l w g Y X − 5) = 2 * ((W w b)*^{2} *− 1)*
*mod (2 * A l w g Y X − 5)* **using** *sat-4-2d C-psi W2* **unfolding** *W-def d2d-def* **by** *auto*

then have $(3 * W\ w\ b * C\ l\ w\ h\ g\ Y\ X - 2 * ((W\ w\ b)^2 - 1))\ mod\ (2 * A\ l\ w\ g\ Y\ X - 5) = 0$
using *mod-diff-cong* **by** *fastforce*
hence *goal*: $2 * A\ l\ w\ g\ Y\ X - 5\ dvd\ 3 * W\ w\ b * C\ l\ w\ h\ g\ Y\ X - 2 * ((W\ w\ b)^2 - 1)$ **by** *force*
show *d2c l w h b g Y X*
using *goal* **unfolding** *S-def*[*symmetric*] *T-def*[*symmetric*] *d2c-def* **by** *auto*
qed

have $V\ w\ g\ Y \leq V\ w\ g\ Y * (\sum i{\in}set\ [X + 1..2 * X].\ int\ (2 * nat\ X\ choose\ nat\ i) * V\ w\ g\ Y\ \hat{}\ nat\ (i - X - 1))$ **using** *sum-pos V-pos* **by** *auto*
hence *ϱ-int-min2*: $int\ \varrho\text{-}int \geq V\ w\ g\ Y$ **using** *ϱ-int-min1* **by** *auto*
have *l-def2*: $int\ \varrho\text{-}int = l{*}Y$
proof −
have *eq1*: $int\ \varrho\text{-}int = int\ ((2{*}nat\ X)\ choose\ nat\ X) + int(nat((V\ w\ g\ Y){*}sum\ (\lambda i.\ int\ (2{*}nat\ X\ choose\ nat\ i) * (V\ w\ g\ Y)\ \hat{}nat\ (i{-}X{-}1))\ (set[X{+}1..2{*}X])))$
using *pI-def* **by** *auto*
hence *int-sum*: $int\ \varrho\text{-}int = int\ ((2{*}nat\ X)\ choose\ nat\ X) + V\ w\ g\ Y * sum\ (\lambda i.\ int\ (2{*}nat\ X\ choose\ nat\ i) * (V\ w\ g\ Y)\ \hat{}nat\ (i{-}X{-}1))\ (set[X{+}1..2{*}X])$
using *V-pos sum-pos* **by** *auto*
have *div-binom*: $int\ ((2{*}nat\ X)\ choose\ nat\ X)\ mod\ Y = 0\ mod\ Y$
using *state1 statement1-def*[*of b Y X*]
by *presburger*
have $V\ w\ g\ Y\ mod\ Y = 0\ mod\ Y$ **unfolding** *V-def* **by** *auto*
hence $int\ \varrho\text{-}int\ mod\ Y = 0\ mod\ Y$ **using** *int-sum div-binom* **by** *auto*
then show *?thesis* **using** *l-def* **by** *auto*
qed

have $(U\ l\ X\ Y)\ \hat{}2{*}V\ w\ g\ Y \geq U\ l\ X\ Y\ \hat{}2$ **using** *V-pos mult-le-cancel-left1* **by** *fastforce*
hence $(U\ l\ X\ Y)\ \hat{}2{*}V\ w\ g\ Y \geq U\ l\ X\ Y$ **using** *power2-eq-square*[*of U l X Y*] *mult-le-cancel-left1*[*of U l X Y U l X Y*] **by** (*auto simp add*: *UBe2*)
hence $U\ l\ X\ Y\ \hat{}2 * V\ w\ g\ Y \geq 2 * X * L\ l\ Y$ **using** *U-def* **by** *auto*
hence $U\ l\ X\ Y\ \hat{}2 * V\ w\ g\ Y \geq 2 * X * int\ \varrho\text{-}int$ **using** *L-def l-def2* **by** *auto*
hence $U\ l\ X\ Y\ \hat{}2 * V\ w\ g\ Y \geq 2 * X * V\ w\ g\ Y$ **using** *ϱ-int-min2 X-pos*
by (*smt* (*z3*) *mult-left-mono*)
hence *U2VBe2X*: $U\ l\ X\ Y\ \hat{}2 * V\ w\ g\ Y \geq 2{*}X$ **using** *V-pos X-pos mult-left-mono*[*of 1 2*X V w g Y*]
using ‹$(U\ l\ X\ Y)^2 \leq (U\ l\ X\ Y)^2 * V\ w\ g\ Y$› **by** *auto*
hence *U2VBe2*: $U\ l\ X\ Y\ \hat{}2 * V\ w\ g\ Y > 1$ **using** *X-pos* **by** *auto*

define *ϱ-frac* **where** $\varrho\text{-}frac = sum\ (\lambda i.\ int\ (2{*}nat\ X\ choose\ nat\ i) * (V\ w\ g\ Y)\ \hat{}(nat\ i))\ (set[0..X{-}1])$
have *decomp-of-p*: $(V\ w\ g\ Y)\ \hat{}nat\ X{*}int\ \varrho\text{-}int + \varrho\text{-}frac = (V\ w\ g\ Y + 1)\ \hat{}(2{*}nat\ X)$

proof −
have *binom-eq1*: $(V\ w\ g\ Y{+}1)\hat{}(2{*}nat\ X) = (\sum i{\leq}2{*}nat\ X.\ int\ (2{*}nat\ X\ choose\ i) * (V\ w\ g\ Y)\hat{}i)$
using *binomial-ring*[*of* $V\ w\ g\ Y\ 1\ 2{*}nat\ X$] **by** *auto*
define *new-function*::*int* ⇒ *int* **where**
$new\text{-}function\ i = int\ (2{*}nat\ X\ choose\ nat\ i) * (V\ w\ g\ Y)\hat{}(nat\ i)$ **for** *i*
hence $(V\ w\ g\ Y{+}1)\hat{}(2{*}nat\ X) = (\sum i{\leq}2{*}nat\ X.\ new\text{-}function\ (int\ i))$ **using** *binom-eq1* **by** *auto*
hence *binom-eq2*: $(V\ w\ g\ Y{+}1)\hat{}(2{*}nat\ X) = sum\ (\lambda i.\ new\text{-}function\ i)\ (set[0..2{*}X])$
using *X-pos change-sum*[*of new-function* $2{*}nat\ X$] **by** *auto*
have $(set[0..2{*}X]) = (set[0..X-1])\cup(set[X..2{*}X]) \wedge (set[0..X-1])\cap(set[X..2{*}X]) = \{\}$
$\wedge$ *finite* $(set[0..X-1]) \wedge$ *finite* $(set[X..2{*}X])$ **by** *auto*
hence *binom-eq3*: $(V\ w\ g\ Y{+}1)\hat{}(2{*}nat\ X) = \varrho\text{-}frac + sum\ (\lambda i.\ new\text{-}function\ i)\ (set[X..2{*}X])$
using *binom-eq2* *ϱ-frac-def sum.union-disjoint*[*of* $(set[0..X-1])\ (set[X..2{*}X])$ *new-function*] **by** (*metis* (*mono-tags*, *lifting*) ‹*new-function* ≡ $\lambda i.\ int\ (2 * nat\ X\ choose\ nat\ i) * V\ w\ g\ Y\ \hat{}\ nat\ i$› *sum.cong*)
have $(set[X..2{*}X]) = \{X\}\cup(set[X{+}1..2{*}X]) \wedge \{X\}\cap(set[X{+}1..2{*}X]) = \{\}$
$\wedge$ *finite* $\{X\}$
$\wedge$ *finite* $(set[X{+}1..2{*}X])$ **using** *assms* **by** *auto*
hence $sum\ (\lambda i.\ new\text{-}function\ i)\ (set[X..2{*}X]) = new\text{-}function\ X + sum\ (\lambda i.\ new\text{-}function\ i)\ (set[X{+}1..2{*}X])$
using *sum.union-disjoint*[*of* $\{X\}\ (set[X{+}1..2{*}X])$ *new-function*] **by** *auto*
hence *binom-eq4*: $(V\ w\ g\ Y{+}1)\hat{}(2{*}nat\ X) = \varrho\text{-}frac + int\ (2{*}nat\ X\ choose\ nat\ X){*}(V\ w\ g\ Y)\hat{}nat\ X + sum\ (\lambda i.\ new\text{-}function\ i)\ (set[X{+}1..2{*}X])$
using *binom-eq3 new-function-def* **by** *auto*
define *other-func*::*int*⇒*int* **where**
$other\text{-}func\ i = int\ (2{*}nat\ X\ choose\ nat\ i) * (V\ w\ g\ Y)\hat{}(nat\ (i-X-1))$ **for** *i*
have $\bigwedge i.\ i{\in}(set[X{+}1..2{*}X]) \Longrightarrow (V\ w\ g\ Y)\hat{}(nat\ i) = (V\ w\ g\ Y)\hat{}(nat\ X + 1){*}(V\ w\ g\ Y)\hat{}nat\ (i-X-1)$
proof −
fix *i*
assume $i{\in}(set[X{+}1..2{*}X])$
hence $nat\ X\ {+}1\ {+}\ nat\ (i-X-1) = nat\ i$ **by** *auto*
thus $(V\ w\ g\ Y)\hat{}(nat\ i) = (V\ w\ g\ Y)\hat{}(nat\ X + 1){*}(V\ w\ g\ Y)\hat{}nat\ (i-X-1)$
using *power-add*[*of* $V\ w\ g\ Y\ nat\ X\ {+}1\ nat\ (i-X-1)$] **by** *auto*
qed
hence $\bigwedge i.\ i{\in}(set[X{+}1..2{*}X]) \Longrightarrow new\text{-}function\ i = (V\ w\ g\ Y)\hat{}(nat\ X + 1){*}$ $(int\ (2{*}nat\ X\ choose\ nat\ i){*}(V\ w\ g\ Y)\hat{}nat\ (i-X-1))$ **using** *new-function-def* **by** *auto*
hence $sum\ (\lambda i.\ new\text{-}function\ i)\ (set[X{+}1..2{*}X]) =$
$sum\ (\lambda i.\ (V\ w\ g\ Y)\hat{}(nat\ X\ {+}1){*}\ (int\ (2{*}nat\ X\ choose\ nat\ i){*}(V\ w\ g\ Y)\hat{}nat\ (i-X-1)))$
$(set[X{+}1..2{*}X])$ **by** *auto*

hence *sum* (λ*i*. *new-function i*) (*set*[*X*+1..2∗*X*]) =
sum (λ*i*. (*V w g Y*) ^(*nat X*+1)∗*other-func i*) (*set*[*X*+1..2∗*X*])
using *other-func-def* **by** *auto*
hence *sum* (λ*i*. *new-function i*) (*set*[*X*+1..2∗*X*]) = (*V w g Y*) ^(*nat X* +1)∗
sum (λ*i*. (*int* (2∗*nat X choose nat i*)∗(*V w g Y*) ^*nat* (*i*−*X*−1))) (*set*[*X*+1..2∗*X*])
using *sum-distrib-left*[*of* (*V w g Y*) ^(*nat X* +1) *other-func* (*set*[*X*+1..2∗*X*])]
other-func-def
by *auto*
hence *sum* (λ*i*. *new-function i*) (*set*[*X*+1..2∗*X*]) = (*V w g Y*) ^*nat X*∗(*V w*
g Y∗
sum (λ*i*. (*int* (2∗*nat X choose nat i*)∗(*V w g Y*) ^*nat* (*i*−*X*−1))) (*set*[*X*+1..2∗*X*]))
using *power-Suc*[*of V w g Y nat X*] **by** *auto*
hence (*V w g Y*+1) ^(2∗*nat X*) = ϱ-*frac* + *int* (2∗*nat X choose nat X*)∗(*V*
w g Y) ^*nat X*
+ (*V w g Y*) ^*nat X*∗(*V w g Y*∗ *sum* (λ*i*. (*int* (2∗*nat X choose nat i*)∗(*V*
w g Y) ^*nat* (*i*−*X*−1)))
(*set*[*X*+1..2∗*X*]))
using *binom-eq4* **by** *auto*
hence *binom-eq5*: (*V w g Y*+1) ^(2∗*nat X*) = ϱ-*frac* + (*V w g Y*) ^*nat X* ∗
(*int* (2∗*nat X choose nat X*)
+ (*V w g Y*∗ *sum* (λ*i*. (*int* (2∗*nat X choose nat i*)∗(*V w g Y*) ^*nat* (*i*−*X*−1)))
(*set*[*X*+1..2∗*X*])))
by (*auto simp add*: *algebra-simps*)
have *int* ϱ-*int* = *int* (2∗*nat X choose nat X*) +
(*V w g Y*∗ *sum* (λ*i*. (*int* (2∗*nat X choose nat i*)∗(*V w g Y*) ^*nat* (*i*−*X*−1)))
(*set*[*X*+1..2∗*X*]))
using *pI-def VBw* ‹*V w g Y* ≤ *V w g Y* ∗ (∑*i*∈*set* [*X* + 1..2 ∗ *X*]. *int*
(2 ∗ *nat X choose*
nat i) ∗ *V w g Y* ^ *nat* (*i* − *X* − 1))› *b-def wBeb* **by** *linarith*
thus *?thesis* **using** *binom-eq5* **by** *auto*
qed

have ϱ-*frac-pos*: ϱ-*frac* ≥ 0
proof −
define *q*::*int*⇒*int* **where** *q i* = *int* (2∗*nat X choose nat i*) ∗ (*V w g Y*) ^(*nat*
i) **for** *i*
have *int* (2∗*nat X choose nat i*) ≥ 0 ∧ (*V w g Y*) ^(*nat i*) ≥ 0 **for** *i*
using *V-pos* **by** *auto*
hence *q-pos*: *q i* ≥ 0 **for** *i*
using *q-def* **by** *auto*
have ϱ-*frac* = *sum* (λ*i*. *q i*) (*set*[0..*X*−1]) **unfolding** ϱ-*frac-def q-def* **by**
auto
thus *?thesis* **using** *q-pos* **by** (*simp add*: *sum-nonneg*)
qed

have ϱ-*frac-L-inv8g*: 8∗*g*∗ϱ-*frac* < (*V w g Y*) ^*nat X*
proof −
define *f1*::*int* ⇒ *int* **where** *f1 i* = *int* (2∗*nat X choose nat i*) ∗(*V w g*
Y) ^(*nat i*) **for** *i*

define *f2*::*int* ⇒ *int* **where** *f2 i* = *int* (2∗*nat X choose nat i*) ∗(*V w g Y*) ^(*nat* (*X*−1)) **for** *i*
define *f3*::*int* ⇒ *int* **where** *f3 i* = *int* (2∗*nat X choose nat i*) **for** *i*
have *i*∈(*set*[0..*X*−1]) ⟹ (*V w g Y*) ^(*nat i*) ≤ (*V w g Y*) ^(*nat* (*X*−1)) **for** *i*
proof −
fix *i*
assume *i-def*: *i*∈(*set*[0..*X*−1])
have *nat i* ≤ *nat* (*X*−1) **using** *i-def* **by** *auto*
thus (*V w g Y*) ^(*nat i*) ≤ (*V w g Y*) ^(*nat* (*X*−1)) **by** (*simp add*: *VBe1 power-increasing*)
qed
hence *i*∈(*set*[0..*X*−1]) ⟹ *f1 i* ≤ *f2 i* **for** *i* **using** *f1-def f2-def mult-left-mono*
by (*metis binom-pos*)
hence *sum* (λ*i*. *f1 i*) (*set*[0..*X*−1]) ≤ *sum* (λ*i*. *f2 i*) (*set*[0..*X*−1])
using *sum-mono*[*of* (*set*[0..*X*−1]) *f1 f2*] **by** *auto*
hence ϱ*-frac* ≤ *sum* (λ*i*. *int* (2∗*nat X choose nat i*) ∗(*V w g Y*) ^(*nat* (*X*−1))) (*set*[0..*X*−1])
using ϱ*-frac-def f1-def f2-def* **by** *auto*
hence *ineq-pfrac-1*: ϱ*-frac* ≤ *sum* (λ*i*. *int* (2∗*nat X choose nat i*))
(*set*[0..*X*−1]) ∗(*V w g Y*) ^(*nat* (*X*−1))
using *sum-distrib-right*[*of f3* (*set*[0..*X*−1]) (*V w g Y*) ^(*nat* (*X*−1))] *f3-def* **by** *auto*
have *ineq-binom*: *k*<*n* ⟹ (∑ *i*≤*k*. *n choose i*) < 2^*n* **for** *k*::*nat* **and** *n*::*nat*
proof −
define *f* **where** *f i* = *n choose i* **for** *i*
assume *kn-def*: *k* < *n*
have *eq-1*: (∑ *i*≤*n*. *n choose i*) = 2^*n* **by** (*simp add*: *choose-row-sum*)
have *n choose* (*k*+1) > *0* **using** *kn-def* **by** *auto*
hence *ineq-strict1*: (∑ *i*≤*k*. *n choose i*) < (∑ *i*≤*k*+1. *n choose i*) **by** *auto*
have {..*k*+1} ⊆ {..*n*} ∧ (∀ *i*. *n choose i* ≥ *0*) ∧ *finite* {..*n*} **using** *kn-def* **by** *auto*
hence (∑ *i*≤*k*+1. *n choose i*) ≤ (∑ *i*≤*n*. *n choose i*)
using *sum-mono2*[*of* {..*n*} {..*k*+1} *f*] **by** (*auto simp add*: *f-def*)
thus *?thesis* **using** *eq-1 ineq-strict1* **by** *presburger*
qed
define *intchoose2X*::*int* ⇒ *int* **where** *intchoose2X i* = *int* (2∗*nat X choose nat i*) **for** *i*
hence *ineq-pfrac-2*: ϱ*-frac* ≤ *sum* (λ*i*. *intchoose2X i*) (*set*[0..*X*−1]) ∗(*V w g Y*) ^(*nat* (*X*−1))
using *ineq-pfrac-1* **by** *auto*
have *eq1*: *sum* (λ*i*. *intchoose2X i*) (*set*[0..*X*−1]) ≤ (∑ *i*≤*nat*(*X*−1). *int* (2∗*nat X choose i*))
using *change-sum*[*of intchoose2X nat*(*X*−1)] *X-pos intchoose2X-def* **by** *auto*
have *eq2*: (∑ *i*≤*nat*(*X*−1). *int* (2∗*nat X choose i*)) = *int* (∑ *i*≤*nat*(*X*−1). 2∗*nat X choose i*)
by *auto*

have $(\sum i{\leq}nat(X-1).\ 2{*}nat\ X\ choose\ i) < 2\hat{}(2{*}nat\ X)$ **using** *ineq-binom*[*of* $nat(X-1)$ $2{*}nat\ X$]
X-pos **by** *auto*
hence *int* $(\sum i{\leq}nat(X-1).\ 2{*}nat\ X\ choose\ i) < 2\hat{}(2{*}nat\ X)$
by (*metis of-nat-less-iff of-nat-numeral of-nat-power*)
hence *eq3*: *sum* $(\lambda i.\ intchoose2X\ i\ )\ (set[0..X-1]) < 2\hat{}(2{*}nat\ X)$
using *eq1 eq2* **by** *presburger*
have $(V\ w\ g\ Y)\hat{}(nat\ (X-1)) > 0$ **using** *V-pos* **by** *auto*
hence $\varrho\text{-}frac < (V\ w\ g\ Y)\hat{}(nat\ (X-1)) * 2\hat{}(2{*}nat\ X)$
using *eq3 ineq-pfrac-2 mult.commute* **by** (*smt* (*verit*, *ccfv-SIG*) *mult-strict-right-mono*)
hence $V\ w\ g\ Y * \varrho\text{-}frac < (V\ w\ g\ Y)\hat{}(nat\ X) * 2\hat{}(2{*}nat\ X)$
using *mult-strict-left-mono*[*of* $\varrho\text{-}frac\ (V\ w\ g\ Y)\hat{}(nat\ (X-1)) * 2\hat{}(2{*}nat\ X)\ V\ w\ g\ Y$]
V-pos power-Suc[*of* $V\ w\ g\ Y\ nat\ (X-1)$] *X-pos*
using *ab-semigroup-mult-class.mult-ac*(*1*) *mult.commute nat-diff-distrib* **by** *fastforce*
hence *ineq-pfrac-3*: $4{*}w{*}g{*}Y{*}\varrho\text{-}frac < (V\ w\ g\ Y)\hat{}nat\ X * 2\hat{}(2{*}nat\ X)$ **using** *V-def* **by** *auto*
have $4{*}w{*}g{*}b{*}\varrho\text{-}frac \leq 4{*}w{*}g{*}Y{*}\varrho\text{-}frac$
using *ϱ-frac-pos assms wBeb* **by** (*smt* (*z3*) *mult-less-cancel-left2 mult-mono*)
hence $4{*}g{*}(b{*}w){*}\varrho\text{-}frac \leq 4{*}w{*}g{*}Y{*}\varrho\text{-}frac$ **by** (*auto simp add*: *algebra-simps*)
hence *ineq-p-frac-4*: $4{*}g{*}2\hat{}(nat\ (B\ X)){*}\ \varrho\text{-}frac \leq 4{*}w{*}g{*}Y{*}\varrho\text{-}frac$
using *sat-4-2d d2d-def*[*of b w X*] **by** *force*
have $nat\ (B\ X) = Suc\ (2{*}nat\ X)$ **using** *B-def*[*of X*] *X-pos* **by** *auto*
hence $8{*}g{*}\ \varrho\text{-}frac * 2\hat{}(2{*}nat\ X) \leq 4{*}w{*}g{*}Y{*}\varrho\text{-}frac$
using *power-Suc*[*of 2 2*∗*nat X*] *ineq-p-frac-4* **by** *auto*
hence $8{*}g{*}\varrho\text{-}frac{*}2\hat{}(2{*}nat\ X) < (V\ w\ g\ Y)\hat{}nat\ X * 2\hat{}(2{*}nat\ X)$ **using** *ineq-pfrac-3* **by** *presburger*
thus *?thesis* **by** *auto*
qed

have *UV-pos*: $U\ l\ X\ Y * V\ w\ g\ Y \geq 0$ **using** *UBe2 V-pos* **by** *auto*
have $C\ l\ w\ h\ g\ Y\ X = \psi\ (A\ l\ w\ g\ Y\ X)\ (nat\ (B\ X))$
unfolding *C-def* **using** *h-def2* **by** *auto*
hence *C-def2*: $C\ l\ w\ h\ g\ Y\ X = \psi\ (U\ l\ X\ Y * (V\ w\ g\ Y + 1))\ (2{*}nat\ X + 1)$
unfolding *A-def B-def* **using** *X-pos* **by** (*smt* (*z3*) *mult-2 nat-add-distrib nat-numeral*
one-eq-numeral-iff)
have $2{*}X \leq U\ l\ X\ Y$ **unfolding** *U-def L-def* **using** *X-pos assms*(*2*) *lBeb assms*(*1*) *b-def*
by (*smt* (*verit*, *ccfv-threshold*) *mult-le-cancel-left1*)
hence $2{*}int(nat\ X) \leq U\ l\ X\ Y \wedge nat\ X \geq 1$ **using** *X-pos* **by** *auto*
hence $abs\ (U\ l\ X\ Y * V\ w\ g\ Y * ((V\ w\ g\ Y)\hat{}nat\ X * C\ l\ w\ h\ g\ Y\ X - (V\ w\ g\ Y + 1)\hat{}(2{*}nat\ X) *$
$\psi\ ((U\ l\ X\ Y)\hat{}2{*}V\ w\ g\ Y)\ (nat\ X + 1)))$
$\leq 2{*}X{*}(V\ w\ g\ Y + 1)\hat{}(2{*}nat\ X) * \psi\ ((U\ l\ X\ Y)\hat{}2{*}V\ w\ g\ Y)\ (nat\ X + 1)$
using *lemma-4-4-cor*[*of nat X U l X Y V w g Y*] *assms C-def2 VBe1* **by** *auto*
hence $U\ l\ X\ Y * V\ w\ g\ Y * abs\ (((V\ w\ g\ Y)\hat{}nat\ X * C\ l\ w\ h\ g\ Y\ X - (V\ w\ g\ Y + 1)\hat{}(2{*}nat\ X) *$

$\psi\ ((U\ l\ X\ Y)\ \hat{}2{*}V\ w\ g\ Y)\ (nat\ X\ {+}1))) \leq 2{*}X{*}(V\ w\ g\ Y\ {+}\ 1)\ \hat{}(2{*}nat\ X)$
$*\ \psi\ ((U\ l\ X\ Y)\ \hat{}2{*}V\ w\ g\ Y)$
$(nat\ X\ {+}1)$
using *UV-pos abs-mult*[*of* $U\ l\ X\ Y * V\ w\ g\ Y\ ((V\ w\ g\ Y)\ \hat{}nat\ X * C\ l\ w\ h$
$g\ Y\ X$
$-\ (V\ w\ g\ Y\ {+}\ 1)\ \hat{}(2{*}nat\ X) * \psi\ ((U\ l\ X\ Y)\ \hat{}2{*}V\ w\ g\ Y)\ (nat\ X\ {+}1))$]
by *auto*

hence $V\ w\ g\ Y * U\ l\ X\ Y * abs\ (((V\ w\ g\ Y)\ \hat{}nat\ X * C\ l\ w\ h\ g\ Y\ X - (V\ w$
$g\ Y\ {+}\ 1)\ \hat{}(2{*}nat\ X) *$
$\psi\ ((U\ l\ X\ Y)\ \hat{}2{*}V\ w\ g\ Y)\ (nat\ X\ {+}1))) \leq 2{*}X{*}(V\ w\ g\ Y\ {+}\ 1)\ \hat{}(2{*}nat\ X)$
$*\ \psi\ ((U\ l\ X\ Y)\ \hat{}2{*}V\ w\ g\ Y)$
$(nat\ X\ {+}1)$
by (*auto simp add*: *algebra-simps*)

hence $(2{*}X){*}L\ l\ Y{*}V\ w\ g\ Y{*}abs\ (((V\ w\ g\ Y)\ \hat{}nat\ X * C\ l\ w\ h\ g\ Y\ X - (V$
$w\ g\ Y\ {+}\ 1)\ \hat{}(2{*}nat\ X) *$
$\psi\ ((U\ l\ X\ Y)\ \hat{}2{*}V\ w\ g\ Y)\ (nat\ X\ {+}1))) \leq (2{*}X){*}(V\ w\ g\ Y\ {+}\ 1)\ \hat{}(2{*}nat$
$X) * \psi\ ((U\ l\ X\ Y)\ \hat{}2{*}V\ w\ g\ Y)$
$(nat\ X\ {+}1)$
unfolding *U-def* **by** (*auto simp add*: *algebra-simps*)

hence $L\ l\ Y{*}V\ w\ g\ Y{*}abs\ (((V\ w\ g\ Y)\ \hat{}nat\ X * C\ l\ w\ h\ g\ Y\ X - (V\ w\ g\ Y$
$+\ 1)\ \hat{}(2{*}nat\ X) *$
$\psi\ ((U\ l\ X\ Y)\ \hat{}2{*}V\ w\ g\ Y)\ (nat\ X\ {+}1))) \leq (V\ w\ g\ Y\ {+}\ 1)\ \hat{}(2{*}nat\ X) * \psi$
$((U\ l\ X\ Y)\ \hat{}2{*}V\ w\ g\ Y)$
$(nat\ X\ {+}1)$
using *X-pos* **by** *auto*

hence *ineq-abs-1*: $int\ \varrho\text{-}int * V\ w\ g\ Y{*}abs\ (((V\ w\ g\ Y)\ \hat{}nat\ X * C\ l\ w\ h\ g\ Y$
X
$-\ (V\ w\ g\ Y\ {+}\ 1)\ \hat{}(2{*}nat\ X) * \psi\ ((U\ l\ X\ Y)\ \hat{}2{*}V\ w\ g\ Y)\ (nat\ X\ {+}1))) \leq$
$(V\ w\ g\ Y\ {+}\ 1)\ \hat{}(2{*}nat\ X)$
$* \psi\ ((U\ l\ X\ Y)\ \hat{}2{*}V\ w\ g\ Y)\ (nat\ X\ {+}1)$
unfolding *L-def* **using** *l-def2* **by** *auto*

have *psi-pos*: $\psi\ ((U\ l\ X\ Y)\ \hat{}2{*}V\ w\ g\ Y)\ (nat\ X\ {+}1) > 0$
using *lucas-strict-monotonicity*[*of* $(U\ l\ X\ Y)\ \hat{}2{*}V\ w\ g\ Y\ nat\ X$] *U2VBe2* **by**
auto

have *ϱ-int-pos*: $int\ \varrho\text{-}int > 0$ **using** *l-def2 lBeb assms b-def* **by** *auto*

have *ϱ-Le-2pint*: $(V\ w\ g\ Y\ {+}\ 1)\ \hat{}(2{*}nat\ X) \leq int\ \varrho\text{-}int{*}2{*}V\ w\ g\ Y\ \hat{}nat\ X$

proof −

have $\varrho\text{-}frac \leq 8{*}g{*}\varrho\text{-}frac$ **using** *assms ϱ-frac-pos*

by (*metis mult-mono mult-numeral-1 mult-right-mono numeral-One one-le-numeral*
zero-le-numeral)

hence *maj-ϱ-frac1*: $\varrho\text{-}frac \leq V\ w\ g\ Y\ \hat{}nat\ X$ **using** *ϱ-frac-L-inv8g* **by** *auto*

have $int\ \varrho\text{-}int > 0 \wedge V\ w\ g\ Y\ \hat{}nat\ X \geq 0$ **using** *ϱ-int-pos V-pos* **by** *auto*

hence $\varrho\text{-}frac \leq int\ \varrho\text{-}int * V\ w\ g\ Y\ \hat{}\ nat\ X$

by (*smt* (*verit*, *ccfv-threshold*) *maj-ϱ-frac1 mult-le-cancel-right1*)

hence $\varrho\text{-}frac \leq V\ w\ g\ Y\ \hat{}nat\ X * int\ \varrho\text{-}int$ **by** (*auto simp add* :*algebra-simps*)

hence $(V\ w\ g\ Y\ {+}\ 1)\ \hat{}(2{*}nat\ X) \leq 2{*}V\ w\ g\ Y\ \hat{}nat\ X{*}int\ \varrho\text{-}int$ **using**
decomp-of-p

add-mono[*of* $V\ w\ g\ Y\ \hat{}nat\ X{*}int\ \varrho\text{-}int\ V\ w\ g\ Y\ \hat{}nat\ X{*}int\ \varrho\text{-}int\ \varrho\text{-}frac$
$V\ w\ g\ Y\ \hat{}nat\ X{*}int\ \varrho\text{-}int$] **by** *auto*

thus *?thesis* **by** (*auto simp add*: *algebra-simps*)
qed
hence *int ϱ-int * V w g Y*abs (((V w g Y) ^nat X * C l w h g Y X − (V w g Y + 1) ^(2*nat X) *
ψ ((U l X Y) ^2*V w g Y) (nat X +1))) ≤ int ϱ-int * 2 * V w g Y ^nat X
* ψ ((U l X Y) ^2*V w g Y) (nat X +1)*
using *ineq-abs-1 psi-pos mult-right-mono*[*of* (V w g Y + 1) ^(2*nat X) *int ϱ-int * 2
* V w g Y ^nat X ψ ((U l X Y) ^2*V w g Y) (nat X +1)*] **by** *linarith*
hence *ineq-abs-2*: *V w g Y*abs (((V w g Y) ^nat X * C l w h g Y X − (V w g Y + 1) ^(2*nat X) *
ψ ((U l X Y) ^2*V w g Y) (nat X +1))) ≤ 2 * V w g Y ^nat X * ψ ((U l X Y) ^2*V w g Y) (nat X +1)*
using *ϱ-int-pos* **by** *auto*
have *VBe32g*: *V w g Y ≥ 32*g*
proof −
have *V w g Y = 4*g*w*Y* **unfolding** *V-def* **by** *simp*
hence *V w g Y ≥ 4*g*(b*w)* **using** *assms wBeb b-def* **by** *auto*
hence *V w g Y ≥ 4*g*2^nat (B X)* **using** *sat-4-2d d2d-def*[*of b w X*] **by** *auto*
hence *V w g Y ≥ 4*g*2^(nat(2*X+1))* **unfolding** *B-def* **by** *auto*
hence *V w g Y ≥ 4*g*2^3* **using** *X-pos power-increasing*[*of 3 nat (2*X+1) 2*]
by (*smt* (*z3*) *BBe3 B-def g-def zmult-zless-mono2*)
thus *?thesis* **by** *auto*
qed
have *U l X Y * abs (((V w g Y) ^nat X * C l w h g Y X − (V w g Y + 1) ^(2*nat X) *
ψ ((U l X Y) ^2*V w g Y) (nat X +1))) ≥ 0*
using *UBe2* **by** *auto*
hence *32*g * abs ((V w g Y) ^nat X * C l w h g Y X − (V w g Y + 1) ^(2*nat X) *
ψ ((U l X Y) ^2*V w g Y) (nat X +1)) ≤ 2*V w g Y ^nat X * ψ ((U l X Y) ^2*V w g Y) (nat X +1)*
using *ineq-abs-2 VBe32g mult-right-mono*[*of 32*g V w g Y abs* ((V w g Y) ^nat X *
C l w h g Y X − (V w g Y + 1) ^(2*nat X) * ψ ((U l X Y) ^2*V w g Y) (nat X +1))*] **by** *auto*
hence *ineq-abs-3*: *16*g*abs ((V w g Y) ^nat X * C l w h g Y X − (V w g Y + 1) ^(2*nat X) *
ψ ((U l X Y) ^2*V w g Y) (nat X +1)) ≤ V w g Y ^nat X * ψ ((U l X Y) ^2*V w g Y) (nat X +1)*
by *auto*

have *V-po-X-pos*: (V w g Y) ^nat X > 0 **using** *V-pos* **by** *auto*
hence (V w g Y) ^nat X * abs (C l w h g Y X − L l Y * ψ ((U l X Y) ^2*V w g Y) (nat X +1))
= abs ((V w g Y) ^nat X * C l w h g Y X − (V w g Y) ^nat X * L l Y * ψ ((U l X Y) ^2*V w g Y)

$(nat\ X\ +1))$
using *abs-mult*[*of* $(V\ w\ g\ Y)\hat{}nat\ X\ C\ l\ w\ h\ g\ Y\ X - L\ l\ Y * \psi\ ((U\ l\ X\ Y)\hat{}2 * V\ w\ g\ Y)\ (nat\ X\ +1)$]
by (*auto simp add*: *algebra-simps*)
hence $(V\ w\ g\ Y)\hat{}nat\ X * abs\ (C\ l\ w\ h\ g\ Y\ X - L\ l\ Y * \psi\ ((U\ l\ X\ Y)\hat{}2 * V\ w\ g\ Y)\ (nat\ X\ +1)) \leq$
$abs\ ((V\ w\ g\ Y)\hat{}nat\ X * C\ l\ w\ h\ g\ Y\ X - (V\ w\ g\ Y\ +1)\hat{}(2 * nat\ X) * \psi\ ((U\ l\ X\ Y)\hat{}2 * V\ w\ g\ Y)$
$(nat\ X\ +1)) + abs\ ((V\ w\ g\ Y\ +1)\hat{}(2 * nat\ X) * \psi\ ((U\ l\ X\ Y)\hat{}2 * V\ w\ g\ Y)\ (nat\ X\ +1) -$
$(V\ w\ g\ Y)\hat{}nat\ X * L\ l\ Y * \psi\ ((U\ l\ X\ Y)\hat{}2 * V\ w\ g\ Y)\ (nat\ X\ +1))$
using *abs-triangle-ineq*[*of* $(V\ w\ g\ Y)\hat{}nat\ X * C\ l\ w\ h\ g\ Y\ X - (V\ w\ g\ Y\ +1)\hat{}(2 * nat\ X)$
$* \psi\ ((U\ l\ X\ Y)\hat{}2 * V\ w\ g\ Y)\ (nat\ X\ +1)\ (V\ w\ g\ Y\ +1)\hat{}(2 * nat\ X) * \psi\ ((U\ l\ X\ Y)\hat{}2 * V\ w\ g\ Y)$
$(nat\ X\ +1) - (V\ w\ g\ Y)\hat{}nat\ X * L\ l\ Y * \psi\ ((U\ l\ X\ Y)\hat{}2 * V\ w\ g\ Y)\ (nat\ X\ +1)$] **by** *auto*
hence $16 * g * (V\ w\ g\ Y)\hat{}nat\ X * abs\ (C\ l\ w\ h\ g\ Y\ X - L\ l\ Y * \psi\ ((U\ l\ X\ Y)\hat{}2 * V\ w\ g\ Y)\ (nat\ X\ +1)) \leq$
$16 * g * (abs\ ((V\ w\ g\ Y)\hat{}nat\ X * C\ l\ w\ h\ g\ Y\ X - (V\ w\ g\ Y\ +1)\hat{}(2 * nat\ X) * \psi\ ((U\ l\ X\ Y)\hat{}2 * V\ w\ g\ Y)$
$(nat\ X\ +1)) + abs\ ((V\ w\ g\ Y\ +1)\hat{}(2 * nat\ X) * \psi\ ((U\ l\ X\ Y)\hat{}2 * V\ w\ g\ Y)\ (nat\ X\ +1) -$
$(V\ w\ g\ Y)\hat{}nat\ X * L\ l\ Y * \psi\ ((U\ l\ X\ Y)\hat{}2 * V\ w\ g\ Y)\ (nat\ X\ +1)))$
using *mult-left-mono*[*of* $(V\ w\ g\ Y)\hat{}nat\ X * abs\ (C\ l\ w\ h\ g\ Y\ X - L\ l\ Y * \psi\ ((U\ l\ X\ Y)\hat{}2 * V\ w\ g\ Y)$
$(nat\ X\ +1))\ abs\ ((V\ w\ g\ Y)\hat{}nat\ X * C\ l\ w\ h\ g\ Y\ X - (V\ w\ g\ Y\ +1)\hat{}(2 * nat\ X)$
$* \psi\ ((U\ l\ X\ Y)\hat{}2 * V\ w\ g\ Y)\ (nat\ X\ +1)) + abs\ ((V\ w\ g\ Y\ +1)\hat{}(2 * nat\ X) * \psi\ ((U\ l\ X\ Y)\hat{}2 * V\ w\ g\ Y)$
$(nat\ X\ +1) - (V\ w\ g\ Y)\hat{}nat\ X * L\ l\ Y * \psi\ ((U\ l\ X\ Y)\hat{}2 * V\ w\ g\ Y)\ (nat\ X\ +1))\ 16 * g]\ assms$
by *auto*
hence *ineq-abs-4*: $16 * g * (V\ w\ g\ Y)\hat{}nat\ X * abs\ (C\ l\ w\ h\ g\ Y\ X - L\ l\ Y * \psi\ ((U\ l\ X\ Y)\hat{}2 * V\ w\ g\ Y)$
$(nat\ X\ +1)) \leq 16 * g * abs\ ((V\ w\ g\ Y)\hat{}nat\ X * C\ l\ w\ h\ g\ Y\ X - (V\ w\ g\ Y\ +1)\hat{}(2 * nat\ X)$
$* \psi\ ((U\ l\ X\ Y)\hat{}2 * V\ w\ g\ Y)\ (nat\ X\ +1)) + 16 * g * abs\ ((V\ w\ g\ Y\ +1)\hat{}(2 * nat\ X) * \psi\ ((U\ l\ X\ Y)\hat{}2 * V\ w\ g\ Y)$
$(nat\ X\ +1) - (V\ w\ g\ Y)\hat{}nat\ X * L\ l\ Y * \psi\ ((U\ l\ X\ Y)\hat{}2 * V\ w\ g\ Y)\ (nat\ X\ +1))$
by (*auto simp add*: *algebra-simps*)
have $abs\ ((V\ w\ g\ Y\ +1)\hat{}(2 * nat\ X) * \psi\ ((U\ l\ X\ Y)\hat{}2 * V\ w\ g\ Y)\ (nat\ X\ +1)$
$-$
$(V\ w\ g\ Y)\hat{}nat\ X * L\ l\ Y * \psi\ ((U\ l\ X\ Y)\hat{}2 * V\ w\ g\ Y)\ (nat\ X\ +1)) = abs\ (((V\ w\ g\ Y\ +1)\hat{}(2 * nat\ X)$
$- (V\ w\ g\ Y)\hat{}nat\ X * int\ \varrho\text{-}int) * \psi\ ((U\ l\ X\ Y)\hat{}2 * V\ w\ g\ Y)\ (nat\ X\ +1))$
using *l-def2 L-def*[*of l Y*]
by (*auto simp add*: *algebra-simps*)

hence *abs* ((*V w g Y* +*1*) ^(*2*∗*nat X*)∗*ψ* ((*U l X Y*) ^*2*∗*V w g Y*) (*nat X* +*1*) − (*V w g Y*) ^*nat X* ∗ *L l Y* ∗ *ψ* ((*U l X Y*) ^*2*∗*V w g Y*) (*nat X* +*1*)) = *abs* ((*V w g Y* +*1*) ^(*2*∗*nat X*) − (*V w g Y*) ^*nat X* ∗ *int ϱ-int*) ∗ *ψ* ((*U l X Y*) ^*2*∗*V w g Y*) (*nat X* +*1*)
using *abs-mult*[*of* (*V w g Y* +*1*) ^(*2*∗*nat X*) − (*V w g Y*) ^*nat X* ∗ *int ϱ-int* *ψ* ((*U l X Y*) ^*2*∗*V w g Y*) (*nat X* +*1*)] *psi-pos* **by** *auto*

hence *abs* ((*V w g Y* +*1*) ^(*2*∗*nat X*)∗*ψ* ((*U l X Y*) ^*2*∗*V w g Y*) (*nat X* +*1*) − (*V w g Y*) ^*nat X* ∗ *L l Y* ∗ *ψ* ((*U l X Y*) ^*2*∗*V w g Y*) (*nat X* +*1*)) ≤ *abs ϱ-frac* ∗ *ψ* ((*U l X Y*) ^*2*∗*V w g Y*) (*nat X* +*1*)
using *decomp-of-p* **by** (*smt* (*verit*, *del-insts*))

hence *16*∗*g*∗*abs* ((*V w g Y* +*1*) ^(*2*∗*nat X*)∗*ψ* ((*U l X Y*) ^*2*∗*V w g Y*) (*nat X* +*1*) − (*V w g Y*) ^*nat X* ∗ *L l Y* ∗ *ψ* ((*U l X Y*) ^*2*∗*V w g Y*) (*nat X* +*1*)) ≤ *16*∗*g*∗(*ϱ-frac*∗*ψ* ((*U l X Y*) ^*2*∗*V w g Y*) (*nat X* +*1*))
using *assms mult-left-mono*[*of abs* ((*V w g Y* +*1*) ^(*2*∗*nat X*)∗*ψ* ((*U l X Y*) ^*2*∗*V w g Y*) (*nat X* +*1*) − (*V w g Y*) ^*nat X* ∗ *L l Y* ∗ *ψ* ((*U l X Y*) ^*2*∗*V w g Y*) (*nat X* +*1*)) *ϱ-frac*∗*ψ* ((*U l X Y*) ^*2* ∗*V w g Y*) (*nat X* +*1*) *16*∗*g*] *ϱ-frac-pos* **by** *auto*

hence *16*∗*g*∗*abs* ((*V w g Y* +*1*) ^(*2*∗*nat X*)∗*ψ* ((*U l X Y*) ^*2*∗*V w g Y*) (*nat X* +*1*) − (*V w g Y*) ^*nat X* ∗ *L l Y* ∗ *ψ* ((*U l X Y*) ^*2*∗*V w g Y*) (*nat X* +*1*)) ≤ *8*∗*g*∗*ϱ-frac*∗(*2*∗*ψ* ((*U l X Y*) ^*2*∗*V w g Y*) (*nat X* +*1*))
by (*auto simp add*: *algebra-simps*)

hence *16*∗*g*∗*abs* ((*V w g Y* +*1*) ^(*2*∗*nat X*)∗*ψ* ((*U l X Y*) ^*2*∗*V w g Y*) (*nat X* +*1*) − (*V w g Y*) ^*nat X* ∗ *L l Y* ∗ *ψ* ((*U l X Y*) ^*2*∗*V w g Y*) (*nat X* +*1*)) ≤ (*V w g Y*) ^*nat X* ∗ (*2*∗ *ψ* ((*U l X Y*) ^*2*∗*V w g Y*) (*nat X* +*1*))
using *ϱ-frac-L-inv8g mult-right-mono*[*of 8*∗*g*∗*ϱ-frac* (*V w g Y*) ^*nat X* *2*∗*ψ* ((*U l X Y*) ^*2*∗*V w g Y*) (*nat X* +*1*)] *psi-pos* **by** *linarith*

hence *16*∗*g*∗*abs* ((*V w g Y*) ^*nat X* ∗ *C l w h g Y X* − (*V w g Y* +*1*) ^(*2*∗*nat X*)∗*ψ* ((*U l X Y*) ^*2*∗*V w g Y*) (*nat X* +*1*)) + *16*∗*g*∗*abs* ((*V w g Y* +*1*) ^(*2*∗*nat X*)∗*ψ* ((*U l X Y*) ^*2*∗*V w g Y*) (*nat X* +*1*) − (*V w g Y*) ^*nat X* ∗ *L l Y* ∗ *ψ* ((*U l X Y*) ^*2*∗*V w g Y*) (*nat X* +*1*)) ≤ *V w g Y* ^*nat X* ∗ *ψ* ((*U l X Y*) ^*2*∗*V w g Y*) (*nat X* +*1*) + (*V w g Y*) ^*nat X* ∗ (*2*∗ *ψ* ((*U l X Y*) ^*2*∗*V w g Y*) (*nat X* +*1*))
using *ineq-abs-3 add-mono*[*of 16*∗*g*∗*abs* ((*V w g Y*) ^*nat X* ∗ *C l w h g Y X* − (*V w g Y* +*1*) ^(*2*∗*nat X*) ∗*ψ* ((*U l X Y*) ^*2*∗*V w g Y*) (*nat X* +*1*)) *V w g Y* ^*nat X* ∗ *ψ* ((*U l X Y*) ^*2*∗*V w g Y*) (*nat X* +*1*) *16*∗*g*∗*abs* ((*V w g Y* +*1*) ^(*2*∗*nat X*)∗*ψ* ((*U l X Y*) ^*2*∗*V w g Y*) (*nat X* +*1*) − (*V w g Y*) ^*nat X* ∗ *L l Y* ∗ *ψ* ((*U l X Y*) ^*2*∗*V w g Y*) (*nat X* +*1*))

$(V\ w\ g\ Y)\ \hat{}nat\ X * (2{*}\psi\ ((U\ l\ X\ Y)\ \hat{}2{*}V\ w\ g\ Y)\ (nat\ X + 1))]$ **by** *blast*

hence $16{*}g{*}(V\ w\ g\ Y)\ \hat{}nat\ X * abs\ (C\ l\ w\ h\ g\ Y\ X - L\ l\ Y * \psi\ ((U\ l\ X\ Y)\ \hat{}2{*}V\ w\ g\ Y)\ (nat\ X + 1))$
$\leq V\ w\ g\ Y\ \hat{}nat\ X * \psi\ ((U\ l\ X\ Y)\ \hat{}2{*}V\ w\ g\ Y)\ (nat\ X + 1) + (V\ w\ g\ Y)\ \hat{}nat\ X$
$* (2 * \psi\ ((U\ l\ X\ Y)\ \hat{}2{*}V\ w\ g\ Y)\ (nat\ X + 1))$ **using** *ineq-abs-4* **by** *auto*

hence $(V\ w\ g\ Y)\ \hat{}nat\ X * (16{*}g{*}abs\ (C\ l\ w\ h\ g\ Y\ X - L\ l\ Y * \psi\ ((U\ l\ X\ Y)\ \hat{}2{*}V\ w\ g\ Y)\ (nat\ X + 1)))$
$\leq V\ w\ g\ Y\ \hat{}nat\ X * (3{*}\psi\ ((U\ l\ X\ Y)\ \hat{}2{*}V\ w\ g\ Y)\ (nat\ X + 1))$ **by** (*auto simp add*: *algebra-simps*)

hence $16{*}g{*}abs\ (C\ l\ w\ h\ g\ Y\ X - L\ l\ Y * \psi\ ((U\ l\ X\ Y)\ \hat{}2{*}V\ w\ g\ Y)\ (nat\ X + 1)) \leq$
$3{*}\psi\ ((U\ l\ X\ Y)\ \hat{}2{*}V\ w\ g\ Y)\ (nat\ X + 1)$ **using** *V-po-X-pos* **by** *auto*

hence $16{*}g{*}abs\ (C\ l\ w\ h\ g\ Y\ X - L\ l\ Y * \psi\ ((U\ l\ X\ Y)\ \hat{}2{*}V\ w\ g\ Y)\ (nat\ X + 1))$
$< 4{*}\psi\ ((U\ l\ X\ Y)\ \hat{}2{*}V\ w\ g\ Y)\ (nat\ X + 1)$ **using** *psi-pos* **by** *auto*

hence $4{*}g{*}abs\ (C\ l\ w\ h\ g\ Y\ X - L\ l\ Y * \psi\ ((U\ l\ X\ Y)\ \hat{}2{*}V\ w\ g\ Y)\ (nat\ X + 1))$
$< \psi\ ((U\ l\ X\ Y)\ \hat{}2{*}V\ w\ g\ Y)\ (nat\ X + 1)$ **by** *auto*

hence *ineq-abs-5*: $abs\ (4{*}g{*}(C\ l\ w\ h\ g\ Y\ X - L\ l\ Y * \psi\ ((U\ l\ X\ Y)\ \hat{}2{*}V\ w\ g\ Y)\ (nat\ X + 1)))$
$< \psi\ ((U\ l\ X\ Y)\ \hat{}2{*}V\ w\ g\ Y)\ (nat\ X + 1)$ **using** *assms* **by** (*auto simp add*: *abs-mult*)

have *K-is-psi*: $K\ k\ l\ w\ g\ Y\ X = \psi\ ((U\ l\ X\ Y)\ \hat{}2{*}V\ w\ g\ Y)\ (nat\ X + 1)$ **using** *k-def K-def* **by** *auto*

hence $abs\ (4{*}g{*}(C\ l\ w\ h\ g\ Y\ X - L\ l\ Y * K\ k\ l\ w\ g\ Y\ X))$
$< K\ k\ l\ w\ g\ Y\ X$ **using** *ineq-abs-5* **by** *auto*

hence $abs\ (4{*}g{*}C\ l\ w\ h\ g\ Y\ X - 4{*}g{*}L\ l\ Y * K\ k\ l\ w\ g\ Y\ X) < K\ k\ l\ w\ g\ Y\ X$
by (*auto simp add*: *algebra-simps*)

hence $abs\ (4{*}g{*}C\ l\ w\ h\ g\ Y\ X - 4{*}g{*}L\ l\ Y * K\ k\ l\ w\ g\ Y\ X){*}abs\ (4{*}g{*}C\ l\ w\ h\ g\ Y\ X$
$- 4{*}g{*}L\ l\ Y * K\ k\ l\ w\ g\ Y\ X) < K\ k\ l\ w\ g\ Y\ X{*}K\ k\ l\ w\ g\ Y\ X$ **using** *mult-strict-mono*[*of*
$abs\ (4{*}g{*}C\ l\ w\ h\ g\ Y\ X - 4{*}g{*}L\ l\ Y * K\ k\ l\ w\ g\ Y\ X)\ K\ k\ l\ w\ g\ Y\ X$
$abs\ (4{*}g{*}C\ l\ w\ h\ g\ Y\ X - 4{*}g{*}L\ l\ Y * K\ k\ l\ w\ g\ Y\ X)\ K\ k\ l\ w\ g\ Y\ X$]
psi-pos K-is-psi **by** *auto*

hence $abs\ (4{*}g{*}C\ l\ w\ h\ g\ Y\ X - 4{*}g{*}L\ l\ Y * K\ k\ l\ w\ g\ Y\ X)\ \hat{}2 < K\ k\ l\ w\ g\ Y\ X\ \hat{}2$
by (*auto simp add*: *power2-eq-square*)

hence *sat-4-2e*: *d2e k l w h g Y X* **unfolding** *d2e-def* **by** *auto*

show *statement2a b Y X g* **unfolding** *statement2a-def* **using** *hBeb kBeb lBeb wBeb sat-4-2a sat-4-2d*
sat-4-d2b sat-4-d2c sat-4-2e **by** *force*

qed

end

end
theory *Bridge-Theorem-Rev*
imports *../Lucas-Sequences/DFI-square-3*
Bridge-Theorem-Imp
HOL−Computational-Algebra.Primes
begin

lemma *div-pow'*:
fixes *a*::*real* **and** *n*::*nat* **and** *p*::*nat*
assumes *n≥p* **and** *a≠0*
shows *a^n / a^p = a^(n−p)*
proof −
have *a^p * a^(n−p) = a^n* **using** *assms power-add*[*of a p n−p*] **by** *fastforce*
hence *a^p * a^(n−p) / a^p = a^n / a^p* **using** *assms divide-cancel-right* **by** *fastforce*
thus *?thesis* **using** *assms* **by** *fastforce*
qed

lemma *inv-decr*:
fixes *a*::*real* **and** *b*::*real*
assumes *a ≥ b* **and** *b > 0*
shows *1/a ≤ 1/b*
by (*simp add*: *assms*(*1*) *assms*(*2*) *frac-le*)

lemma *div-pow*:
fixes *a*::*real* **and** *n*::*nat* **and** *m*::*nat*
assumes *m<n* **and** *a≠0*
shows *a^n/a^m = a*a^(n−m−1)*
proof −
have *Suc* (*n−m−1*) = *n−m* ∧ (*n−m*)+*m* = *n* **using** *assms* **by** (*auto simp add*: *algebra-simps*)
hence *a*a^(n−m−1)*a^m = a^n* **using** *power-Suc*[*of a n−m−1*] *power-add*[*of a n−m m*] **by** *auto*
thus *?thesis* **by** (*metis assms*(*2*) *nonzero-mult-div-cancel-right power-eq-0-iff*)
qed

lemma *power-majoration*:
fixes *a*::*real* **and** *n*::*nat*
assumes *0 < a* **and** *a ≤ 1*
shows (*1+a*) ^*n ≤ 1 + (2^n−1)*a*
proof (*induction n*)
case *0*
then show *?case* **using** *assms* **by** *auto*
next
case (*Suc n*)
note *t = this*
have (*1+a*) ^(*Suc n*) = (*1+a*)*(*1+a*) ^*n* **using** *power-Suc* **by** *auto*

hence $(1+a)\hat{}(Suc\ n) \leq (1+a)*(1+(2\hat{}n-1)*a)$ **using** *t assms* **by** *auto*
hence $(1+a)\hat{}(Suc\ n) \leq 1+a+(1+a)*((2\hat{}n-1)*a)$ **by** (*auto simp add*: *algebra-simps*)
hence $(1+a)\hat{}(Suc\ n) \leq 1+a+2*(2\hat{}n-1)*a$ **using** *assms mult-right-mono*[*of* $1+a$ 2 $(2\hat{}n-1)*a$]
by (*smt* (*verit*, *ccfv-SIG*) *one-le-power ring-class.ring-distribs*(2) *t*)
hence $(1+a)\hat{}(Suc\ n) \leq 1+(2*2\hat{}n-1)*a$ **by** (*auto simp add*: *algebra-simps*)
then show *?case* **using** *power-Suc*[*of 2 n*] **by** *auto*
qed

lemma *div-reg*:
fixes a::*int* **and** b::*int* **and** c::*int* **and** d::*int*
assumes $a \leq b$ **and** $c \geq d$ **and** $d>0$ **and** $a \geq 0$
shows $a/c \leq b/d$
proof −
have $a * d \leq b * c$ **using** *assms mult-mono* **by** *fastforce*
hence *real-of-int a* $*$ *real-of-int d* $\leq$ *real-of-int b* $*$ *real-of-int c*
by (*metis of-int-le-iff of-int-mult*)
thus *?thesis* **using** *assms*
by (*meson frac-le of-int-0-le-iff of-int-0-less-iff of-int-le-iff order.trans*)
qed

lemma *lucas-modN-int*:
fixes α::*int* **and** m::*int*
shows ψ*-int* α *m mod* $(\alpha - 2) = m$ *mod* $(\alpha - 2)$
proof −
have ψ*-int* α *m mod* $(\alpha - 2) = (-1)\hat{}(if\ m \geq 0\ then\ 0\ else\ 1) * \psi\ \alpha\ (nat\ (abs\ m))$ *mod* $(\alpha - 2)$
unfolding ψ*-int-def* **by** *simp*
hence ψ*-int* α *m mod* $(\alpha - 2) = (-1)\hat{}(if\ m \geq 0\ then\ 0\ else\ 1) * int\ (nat\ (abs\ m))$ *mod* $(\alpha - 2)$
using *lucas-congruence2*[*of* α *nat* (*abs m*)] *mod-mult-cong*[*of* $(-1)\hat{}(if\ m \geq 0\ then\ 0\ else\ 1)$ $\alpha-2$
$(-1)\hat{}(if\ m \geq 0\ then\ 0\ else\ 1)$ $\psi\ \alpha\ (nat\ (abs\ m))$ *int* (*nat* (*abs m*))] **by** *presburger*
then show *?thesis* **by** *auto*
qed

6.3 Proof of implication $(2) \Longrightarrow (1)$

lemma (**in** *bridge-variables*) *theorem-II-2-1*:
assumes *b-def*:(b::*int*)≥ 0 **and** *Y-def*:(Y::*int*)$\geq b \wedge Y \geq 2\hat{}8$ **and** *X-def*:(X::*int*)$\geq 3*b$
and *g-def*:(g::*int*)≥ 1
shows (*statement2 b Y X g*)$\Longrightarrow$(*statement1 b Y X*)
proof −
assume *state2*: *statement2 b Y X g*
then obtain *h k l w x y* **where** *state2-def*: $l*x \neq 0 \wedge$ *d2a l w h x y g Y X* $\wedge$ *d2b k l w x g Y X* $\wedge$
d2c l w h b g Y X $\wedge$ *d2f k l w h g Y X* **unfolding** *statement2-def* **by** *auto*
have *sat-a*: *d2a l w h x y g Y X* **using** *state2-def* **by** *auto*
have *sat-b*: *d2b k l w x g Y X* **using** *state2-def* **by** *auto*

have *sat-c*: *d2c l w h b g Y X* **using** *state2-def* **by** *auto*
have *sat-f*: *d2f k l w h g Y X* **using** *state2-def* **by** *auto*
have *lx-nonzero*: $l \neq 0 \wedge x \neq 0$ **using** *state2-def* **by** *auto*

have *W-nonzero*: *W w b* $\neq$ *0*
proof −
have *W w b* = *0* $\Longrightarrow$ *False*
proof −
assume *hyp*: *W w b* = *0*
hence (*2*∗*A l w g Y X*−*5*) *dvd 2*
using *sat-c* **unfolding** *d2c-def S-def T-def* **by** *auto*
hence *abs* (*2*∗*A l w g Y X* − *5*) ≤ *2*
using *dvd-imp-le-int* **by** *presburger*
hence *2*∗*A l w g Y X* ≤ *7* ∧ *2*∗*A l w g Y X* ≥ *3* **by** *auto*
hence *A-is-2-or-3*: *A l w g Y X* ≤ *3* ∧ *A l w g Y X* ≥ *2* **by** *auto*
have *abs* (*A l w g Y X*) = *abs* (*U l X Y*) ∗ *abs* (*V w g Y* + *1*)
unfolding *A-def U-def* **using** *abs-mult* **by** *auto*
hence *eq*: *abs* (*A l w g Y X*) = *2*∗*X*∗*Y*∗*abs l* ∗ *abs* (*V w g Y* +*1*)
unfolding *U-def L-def* **using** *abs-mult*[*of 2*∗*X l*∗*Y*] *abs-mult*[*of 2 X*] *abs-mult*[*of l Y*] *assms*
by *auto*
hence *X-nonzero*: *X* ≠ *0* **using** *A-is-2-or-3* **by** *auto*
hence *several-ineq*: *X* ≥ *1* ∧ *Y* ≥ *256* ∧ *abs l* ≥ *1* ∧ *abs* (*V w g Y* +*1*) ≥ *0*
using *assms lx-nonzero* **by** *auto*
hence *2*∗*X*∗*Y* ≥ *2*∗*256* ∧ *abs l* ≥ *1* ∧ *abs* (*V w g Y* +*1*) ≥ *0*
using *mult-mono*[*of 1 X 256 Y*] *mult-left-mono*[*of 256 X*∗*Y 2*] **by** *force*
hence (*2*∗*X*∗*Y*)∗*abs l*∗*abs* (*V w g Y* +*1*) ≥ (*2*∗*256*)∗*abs* (*V w g Y* +*1*)
using *mult-mono*[*of 2*∗*256 2*∗*X*∗*Y 1 abs l*]
mult-right-mono[*of 2*∗*256 2*∗*X*∗*Y*∗*abs l abs* (*V w g Y* +*1*)] **by** *linarith*
hence *ineq*: *2*∗*X*∗*Y*∗*abs l* ∗ *abs* (*V w g Y* +*1*) ≥ *2*∗*256*∗*abs* (*V w g Y* +*1*)
by (*auto simp add*: *mult-mono*)
hence *abs* (*A l w g Y X*) ≥ *2*∗*256*∗*abs* (*V w g Y* +*1*) **using** *ineq eq* **by** *presburger*
hence *3* ≥ *2*∗*256*∗*abs* (*V w g Y* +*1*) ∧ *abs* (*V w g Y* +*1*) ≥ *0* **using** *A-is-2-or-3* **by** *auto*
hence *abs* (*V w g Y* +*1*) = *0*
by (*smt* (*verit*, *best*) *comm-semiring-class.distrib distrib-left distrib-right mult-cancel-left2*
mult-le-0-iff mult-mono ring-class.ring-distribs(*1*) *ring-class.ring-distribs*(*2*))
hence *V w g Y* +*1* = *0* **by** *auto*
hence *A l w g Y X* = *0* **unfolding** *A-def* **by** *auto*
then show *False* **using** *A-is-2-or-3* **by** *auto*
qed
then show *?thesis* **by** *auto*
qed

hence *bBe1*: *b* ≥ *1* **using** *assms W-nonzero W-def*[*of w b*] **by** *auto*
hence *XBe3*: *X* ≥ *3* **using** *X-def* **by** *auto*
have *absL-Be1*: *abs* (*L l Y*) ≥ *1* **unfolding** *L-def* **using** *lx-nonzero assms*

by (*smt* (*z3*) *bBe1 no-zero-divisors*)
have *V-mult-4*: *4 dvd V w g Y* **unfolding** *V-def* **by** *auto*
hence *V w g Y +1* ≠ *0* **by** *presburger*
hence *abs* (*V w g Y +1*) ≥ *1* **by** *auto*
hence *abs* (*A l w g Y X*) ≥ *abs* (*U l X Y*)
unfolding *A-def* **using** *abs-mult*[*of U l X Y V w g Y +1*]
mult-left-mono[*of 1 abs* (*V w g Y +1*) *abs* (*U l X Y*)] **by** *auto*
hence *abs* (*A l w g Y X*) ≥ *2* ∗*X* ∗ (*abs l* ∗ *Y*)
unfolding *U-def L-def* **using** *abs-mult*[*of 2*∗*X l*∗*Y*] *abs-mult*[*of 2 X*] *abs-mult*[*of l Y*] *XBe3 assms*(*2*)
by *auto*
hence *abs* (*A l w g Y X*) ≥ *2*∗*X*∗*Y*∗*abs l* ∧ *abs l* ≥ *1* ∧ *2*∗*X*∗*Y* ≥ *0*
using *lx-nonzero XBe3 assms*(*2*) *mult-mono*[*of 3 X 256 Y*]
by (*smt* (*z3*) *b-def mult.assoc mult.commute mult-nonneg-nonneg*)
hence *abs* (*A l w g Y X*) ≥ *2*∗*X*∗*Y* **using** *mult-left-mono*[*of 1 abs l 2*∗*X*∗*Y*] **by** *auto*
hence *abs* (*A l w g Y X*) ≥ *2*∗*X*∗*Y* ∧ *Y* ≥ *4* ∧ *2*∗*X* ≥ *0* **using** *mult-mono*[*of 0 2 3 X*] *XBe3 assms*(*2*) **by** *simp*
hence *abs* (*A l w g Y X*) ≥ *8*∗*X* **using** *mult-left-mono*[*of 4 Y 2*∗*X*] **by** *auto*
hence *abs* (*A l w g Y X*) ≥ *4*∗*X* + *4*∗*X* **by** *auto*
hence *abs* (*A l w g Y X*) > *4*∗*X* + *4* **using** *mult-strict-left-mono*[*of 1 X 4*] *XBe3* **by** *auto*
hence *abs-A-B-2Bp2*: *abs* (*A l w g Y X*) > *2*∗*B X* + *2* **unfolding** *B-def* **by** *auto*
hence *abs-A-Be16*: *abs* (*A l w g Y X*) ≥ *16*
unfolding *B-def* **using** *XBe3 assms*(*2*) *mult-mono*[*of 2 X 4 Y*] **by** *auto*
have *abs-Am2-B-2B*: *abs* (*A l w g Y X*) − *2* > *2*∗*B X* **using** *abs-A-B-2Bp2* **by** *auto*
have *B-B1*: *abs* (*B X*) > *1* **unfolding** *B-def* **using** *XBe3* **by** *auto*
have *Am2-dvd-CmB*: (*A l w g Y X*−*2*) *dvd* (*C l w h g Y X*−*B X*) **unfolding** *C-def* **by** *auto*
have *D-eq-D*: *D-f* (*A l w g Y X*) (*C l w h g Y X*) = *D l w h g Y X* **unfolding** *D-f-def D-def* **by** *auto*
hence *E-eq-E*: *E-ACx* (*A l w g Y X*) (*C l w h g Y X*) *x* = *E l w h x g Y X*
unfolding *E-ACx-def E-f-def E-def* **by** *auto*
hence *F-eq-F*: *F-ACx* (*A l w g Y X*) (*C l w h g Y X*) *x* = *F l w h x g Y X*
unfolding *F-ACx-def F-f-def F-def* **by** *auto*
hence *G-eq-G*: *G-ACx* (*A l w g Y X*) (*C l w h g Y X*) *x* = *G l w h x g Y X*
unfolding *G-ACx-def G-f-def G-def* **using** *D-eq-D E-eq-E* **by** *auto*
hence *H-eq-H*: *H-ABCxy* (*A l w g Y X*) (*B X*) (*C l w h g Y X*) *x y* = *H l w h x y g Y X*
unfolding *H-ABCxy-def H-f-def H-def* **using** *F-eq-F* **by** *auto*
hence *I-eq-I*: *I-ABCxy* (*A l w g Y X*) (*B X*) (*C l w h g Y X*) *x y* = *I l w h x y g Y X*
unfolding *I-ABCxy-def I-f-def I-def* **using** *G-eq-G* **by** *auto*
hence *DFI-square*: *is-square* (*D-f* (*A l w g Y X*) (*C l w h g Y X*) ∗ *F-ACx* (*A l w g Y X*) (*C l w h g Y X*) *x*
∗ *I-ABCxy* (*A l w g Y X*) (*B X*) (*C l w h g Y X*) *x y*)
using *sat-a is-square-def D-eq-D F-eq-F* **unfolding** *d2a-def* **by** *simp*

hence *C-is-ψBA*: *C l w h g Y X* = *ψ-int* (*A l w g Y X*) (*B X*)
using *sun-theorem*[*of B X A l w g Y X C l w h g Y X x y*] *B-B1 abs-Am2-B-2B*

Am2-dvd-CmB lx-nonzero **by** (*smt* (*z3*) *B-def XBe3*)

have *w-nonzero*: *w* ≠ *0* **using** *W-nonzero W-def*[*of w b*] **by** *auto*
have *B-Be7*: *B X* ≥ *7* **unfolding** *B-def* **using** *XBe3* **by** *auto*
hence *nat* (*B X*) ≥ *Suc* (*Suc* (*Suc 0*)) **by** *auto*
then obtain *Bm3* **where** *B-3Suc*: *Suc* (*Suc* (*Suc Bm3*)) = *nat* (*B X*)
by (*metis Suc-leD Suc-n-not-le-n rec-forte-init012.cases*)
hence *SucSucBm3*: *Bm3* + *2* = *nat* (*B X*) − *1* **by** *auto*
have *abs* (*V w g Y*) = *4*∗ *abs w* ∗ *g* ∗ *Y* **unfolding** *V-def*
using *abs-mult*[*of 4*∗*w*∗*g Y*] *abs-mult*[*of 4*∗*w g*] *abs-mult*[*of 4 w*] *assms*(*2*) *assms*(*4*) **by** *auto*
hence *abs* (*V w g Y*) = *4* ∗ *abs w* ∗ *g*∗*Y* ∧ *abs w* ≥ *1* ∧ *g* ≥ *1* ∧ *Y* ≥ *2*
using *assms*(*4*) *assms*(*2*) *w-nonzero* **by** *auto*
hence *absV-Be8*: *abs* (*V w g Y*) ≥ *8* **using** *mult-left-mono*[*of 2 Y 4*∗*abs w* ∗*g*]
by (*smt* (*z3*) *dvd-imp-le-int dvd-triv-left mult-le-0-iff*)
have *abs* (*U l X Y*) = *2*∗*X*∗*abs l*∗*Y* ∧ *X* ≥ *1* ∧ *Y* ≥ *1* **and** *abs-lBe1*: *abs l* ≥ *1*
unfolding *U-def L-def* **using** *abs-mult*[*of 2*∗*X l*∗*Y*] *abs-mult*[*of 2 X*] *abs-mult*[*of l Y*]
XBe3 assms(*2*) *lx-nonzero* **by** *auto*
have *abs* (*U l X Y*) = *2*∗*X*∗*abs l*∗*Y* ∧ *X* ≥ *1* ∧ *Y* ≥ *1* ∧ *abs l* ≥ *1*
unfolding *U-def L-def* **using** *abs-mult*[*of 2*∗*X l*∗*Y*] *abs-mult*[*of 2 X*] *abs-mult*[*of l Y*] *XBe3*
assms(*2*) *lx-nonzero* **by** *auto*
hence *absUBe2*: *abs* (*U l X Y*) ≥ *2*
using *mult-mono*[*of 2 2*∗*X*∗*abs l 1 Y*] *mult-mono*[*of 2 2*∗*X 1 abs l*] *mult-left-mono*[*of 1 X 2*]
by *auto*
hence *other-ineq-U2*: *abs* (*U l X Y*^*2*) ≥ *2*
using *power2-eq-square*[*of U l X Y*] *mult-mono*[*of 1 abs* (*U l X Y*) *2 abs* (*U l X Y*)] **by** *auto*
have *abs* (*V w g Y* +*1*) ≤ *abs* (*V w g Y*) +*1* **by** *auto*
hence *abs* (*V w g Y* +*1*) ≤ *2*∗*abs* (*V w g Y*)−*1* **using** *absV-Be8* **by** *auto*
hence *abs* (*V w g Y* +*1*) ≤ *abs* (*U l X Y*^*2*)∗*abs* (*V w g Y*)−*1*
using *other-ineq-U2 mult-right-mono*[*of 2 abs* (*U l X Y*^*2*) *abs* (*V w g Y*)] **by** *auto*
hence *ineq-Vp1*: *abs* (*V w g Y* +*1*) ≤ *abs* (*U l X Y*^*2*∗*V w g Y*) − *1* **using** *abs-mult* **by** *auto*
have *abs* (*C l w h g Y X*) = *ψ* (*abs* (*A l w g Y X*)) (*nat* (*abs* (*B X*)))
using *eq-ψ-int*[*of A l w g Y X B X*] *C-is-ψBA abs-A-Be16* **by** *force*
hence *abs* (*C l w h g Y X*) = *ψ* (*abs* (*A l w g Y X*)) (*nat* (*B X*)) **using** *B-def*[*of X*] *XBe3* **by** *auto*
hence *abs* (*C l w h g Y X*) ≤ (*abs* (*A l w g Y X*))^(*nat* (*B X*)−*1*)
using *lucas-exp-growth-lt*[*of abs* (*A l w g Y X*) *Bm3*] *B-3Suc SucSucBm3 abs-A-Be16* **by** *fastforce*
hence *abs* (*C l w h g Y X*) ≤ (*abs* (*U l X Y*) ∗ *abs*((*V w g Y* +*1*))) ^(*2*∗*nat*

X)
unfolding *B-def A-def* **using** *XBe3 abs-mult*[*of U l X Y V w g Y +1*]
by (*smt* (*z3*) *mult-2 nat-1 nat-add-distrib nat-diff-distrib numeral-1-eq-Suc-0 numerals*(*1*))
hence *abs* (*C l w h g Y X*) ≤ (*abs* (*U l X Y*))^(*2*∗*nat X*)∗(*abs* (*V w g Y +1*))^(*2*∗*nat X*)
using *power-mult-distrib*[*of abs* (*U l X Y*) *abs* (*V w g Y +1*) *2*∗*nat X*] **by** *auto*
hence *maj-C1*:*abs* (*C l w h g Y X*) ≤ (*abs* (*U l X Y*))^(*2*∗*nat X*)∗(*abs* (*U l X Y*^*2*∗*V w g Y*) − *1*)^(*2*∗*nat X*)
using *ineq-Vp1*
by (*smt* (*verit*, *ccfv-SIG*) *int-distrib*(*1*) *power-less-imp-less-base zero-le-even-power*′ *zmult-zless-mono2*)

have *is-square* ((*U l X Y*^*4*∗*V w g Y*^*2* − *4*)∗(*K k l w g Y X*)^*2*+*4*)
using *sat-b is-square-def* **unfolding** *d2b-def* **by** *auto*
hence *is-square* (((*U l X Y*^*2*∗*U l X Y*^*2*)∗(*V w g Y*∗*V w g Y*) − *4*)∗(*K k l w g Y X*)^*2*+*4*)
using *power-add*[*of U l X Y 2 2*] *power2-eq-square*[*of V w g Y*] **by** *auto*
hence *is-square* (((*U l X Y*^*2*∗*V w g Y*)^*2*−*4*)∗(*K k l w g Y X*)^*2*+*4*)
using *power2-eq-square*[*of U l X Y*^*2*∗*V w g Y*] **by** (*auto simp add*: *algebra-simps*)
then obtain *R* **where** *R-def*: *K k l w g Y X* = *ψ-int* (*U l X Y*^*2*∗*V w g Y*) *R*
using *lucas-pell-corollary-int*[*of U l X Y*^*2*∗*V w g Y K k l w g Y X*] *is-square-def* **by** *auto*
have *abs-U2V-B2*: *abs* (*U l X Y*^*2*∗*V w g Y*) > *2*
using *abs-mult*[*of U l X Y*^*2 V w g Y*] *other-ineq-U2 absV-Be8*
mult-strict-mono[*of 1 abs* (*U l X Y*^*2*) *2 abs* (*V w g Y*)] **by** *force*
have (*X+1*) *mod* (*U l X Y*^*2*∗*V w g Y* − *2*) = *K k l w g Y X mod* (*U l X Y*^*2*∗*V w g Y* − *2*)
unfolding *K-def* **by** *auto*
hence (*X+1*) *mod* (*U l X Y*^*2*∗*V w g Y* − *2*) = *ψ-int* (*U l X Y*^*2*∗*V w g Y*) *R mod* (*U l X Y*^*2*∗*V w g Y* − *2*)
using *R-def* **by** *auto*
hence (*X+1*) *mod* (*U l X Y*^*2*∗*V w g Y* − *2*) = *R mod* (*U l X Y*^*2*∗*V w g Y* − *2*)
using *lucas-modN-int*[*of U l X Y*^*2*∗*V w g Y R*] **by** *auto*
hence (*X+1*−*R*) *mod* (*U l X Y*^*2*∗*V w g Y*−*2*) = *0 mod* (*U l X Y*^*2*∗*V w g Y*−*2*)
using *mod-diff-cong*[*of X+1 U l X Y*^*2*∗*V w g Y*−*2 R R R*] **by** *auto*
hence *U l X Y*^*2*∗*V w g Y*−*2 dvd* (*R*−(*X+1*)) **by** *algebra*
then obtain *r* **where** *R* −(*X+1*) =*r*∗(*U l X Y*^*2*∗*V w g Y*−*2*) **by** *force*
hence *r-def*: *R* = *X+1*+*r*∗(*U l X Y*^*2*∗*V w g Y*−*2*) **by** *auto*

have *r-0*: *r* ≠ *0* ⟹ *False*
proof −
assume *hyp*: *r* ≠ *0*
hence *abs R* ≥ *abs* (*r*∗(*U l X Y*^*2*∗*V w g Y* −*2*)) − *abs* (*X+1*) **using** *r-def* **by** *auto*
hence *abs R* ≥ *abs r* ∗ *abs* (*U l X Y*^*2*∗*V w g Y* − *2*) − *X* − *1* **using** *abs-mult*

XBe3 **by** *auto*
hence *abs R ≥ abs (U l X Y^2∗V w g Y − 2) − X − 1*
using *hyp mult-right-mono*[*of 1 abs r abs (U l X Y^2∗V w g Y − 2)*] **by** *force*
hence *abs R ≥ abs (U l X Y^2∗V w g Y) − X − 3* **by** *auto*
hence *abs R ≥ abs (U l X Y∗U l X Y∗V w g Y) − X − 3*
using *power2-eq-square*[*of U l X Y*] **by** *auto*
hence *abs R ≥ abs (U l X Y)∗(abs (U l X Y)∗abs (V w g Y)) − X − 3* **by** (*auto simp add*: *abs-mult*)
hence *abs R ≥ abs (U l X Y) − X − 3*
using *absV-Be8 absUBe2 mult-mono*[*of 1 abs (U l X Y) 1 abs (V w g Y)*]
mult-left-mono[*of 1 abs (U l X Y) ∗ abs (V w g Y) abs (U l X Y)*] **by** *auto*
hence *abs R ≥ 2∗X∗ (abs l ∗Y) − X − 3 ∧ Y ≥ 4 ∧ abs l ≥ 1 ∧ 2∗X ≥ 0*
unfolding *U-def L-def* **using** *assms*(*2*) *lx-nonzero XBe3* **by** *auto*
hence *abs R ≥ 8∗X − X − 3*
using *mult-mono*[*of 1 abs l 4 Y*] *mult-left-mono*[*of 4 abs l ∗ Y 2∗X*] **by** *linarith*
hence *abs R > 4∗X + 4 − X − 3* **using** *XBe3* **by** *auto*
hence *ineq-absR*: *abs R > 3∗X +1* **by** *auto*
hence *ineq-absRm1*: *abs R − 1 > 3∗X* **by** *auto*
have *aRBe2*: *nat (abs R) ≥ 2* **using** *XBe3 ineq-absR* **by** *simp*
have *abs (K k l w g Y X) = ψ (abs (U l X Y^2∗V w g Y)) (nat (abs R))*
using *R-def eq-ψ-int abs-U2V-B2* **by** *force*
hence *0*: *abs (K k l w g Y X) ≥ (abs (U l X Y^2∗V w g Y) − 1) ^ (nat (abs R) − 1)*
using *abs-U2V-B2 aRBe2 lucas-exp-growth-gt*[*of abs (U l X Y^2∗V w g Y) nat (abs R) − 2*] *Suc-eq-plus1*
by (*smt* (*verit*, *ccfv-SIG*) *One-nat-def add-2-eq-Suc′ add-diff-cancel-left′ le-add-diff-inverse2 plus-1-eq-Suc*)
have *nat (abs R) − 1 ≥ 3 ∗ nat X* **using** *XBe3 ineq-absRm1* **by** *auto*
hence *min-K*: *abs (K k l w g Y X) ≥ (abs (U l X Y^2∗V w g Y) − 1) ^ (3 ∗ nat X)*
using *0 abs-U2V-B2 power-increasing*[*of 3 ∗ nat X nat (abs R) − 1 abs (U l X Y^2∗V w g Y) − 1*]
by *linarith*

have *abs (V w g Y) ≥ 8* **using** *absV-Be8* **by** *blast*
hence *abs ((U l X Y)^2 ∗ V w g Y) ≥ 8 ∗ (U l X Y)^2*
by (*metis abs-mult-pos mult.commute mult-right-mono power2-eq-square zero-le-square*)
hence *abs ((U l X Y)^2 ∗ V w g Y) − 1 ≥ 2 ∗ (abs (U l X Y))^2*
using *abs-U2V-B2* **by** *auto*
hence *2∗ (abs (U l X Y))^2 / (abs (U l X Y^2∗V w g Y) − 1) ≤ 1*
using *abs-U2V-B2* **by** (*smt* (*verit*) *divide-le-eq-1-pos of-int-le-1-iff of-int-le-iff*)
hence *min-U2V*: *(abs (U l X Y))^2 / (abs (U l X Y^2∗V w g Y) − 1) ≤ 1/2* **by** *linarith*
have *B0*: *abs (C l w h g Y X) ≥ 0 ∧ (abs (U l X Y^2∗V w g Y) − 1)^(3∗nat X) > 0*
using *abs-U2V-B2* **by** *auto*

hence *1*: *abs* (*C l w h g Y X*) / *abs* (*K k l w g Y X*) ≤ (*abs* (*U l X Y*))^(*2*∗*nat X*) ∗
(*abs* (*U l X Y*^*2*∗*V w g Y*) − *1*)^(*2*∗*nat X*) / (*abs* (*U l X Y*^*2*∗*V w g Y*) − *1*)^(*3*∗*nat X*)
using *min-K maj-C1 div-reg* **by** *presburger*

have (*abs* (*U l X Y*))^(*2*∗*nat X*) ∗ (*abs* (*U l X Y*^*2*∗*V w g Y*) − *1*)^(*2*∗*nat X*) ∗ (*abs* (*U l X Y*^*2*∗*V w g Y*) − *1*)^(*nat X*)
= (*abs* (*U l X Y*))^(*2*∗*nat X*) ∗ (*abs* (*U l X Y*^*2*∗*V w g Y*) − *1*)^(*3*∗*nat X*)
using *XBe3* **by** (*simp add*:*algebra-simps power2-eq-square power3-eq-cube power-mult*)
hence *2*: (*abs* (*U l X Y*))^(*2*∗*nat X*) ∗ (*abs* (*U l X Y*^*2*∗*V w g Y*) − *1*)^(*2*∗*nat X*) / (*abs* (*U l X Y*^*2*∗*V w g Y*) − *1*)^(*3*∗*nat X*)
= (*abs* (*U l X Y*))^(*2*∗*nat X*) / (*abs* (*U l X Y*^*2*∗*V w g Y*) − *1*)^(*nat X*)
using *of-int-mult frac-eq-eq abs-U2V-B2 B0 XBe3*
by (*smt* (*z3*) *of-int-pos one-less-power zero-less-nat-eq*)
have (*abs* (*U l X Y*))^(*2*∗*nat X*) / (*abs* (*U l X Y*^*2*∗*V w g Y*) − *1*)^(*nat X*)
= ((*abs* (*U l X Y*))^*2* / (*abs* (*U l X Y*^*2*∗*V w g Y*) − *1*))^(*nat X*)
using *B0 XBe3 abs-U2V-B2 of-int-power*
by (*simp add*: *power-divide power-mult*)
hence *3*: *abs* (*C l w h g Y X*) / *abs* (*K k l w g Y X*)
≤ ((*abs* (*U l X Y*))^*2* / (*abs* (*U l X Y*^*2*∗*V w g Y*) − *1*))^(*nat X*)
using *1 2* **by** *simp*
have *4*: ... ≤ (*1* / *2*)^(*nat X*)
using *min-U2V XBe3 Power.linordered-semidom-class.power-mono*
[*of* (*abs* (*U l X Y*))^*2* / (*abs* (*U l X Y*^*2*∗*V w g Y*) − *1*) *1/2 nat X*]
abs-U2V-B2
by (*smt* (*verit*, *ccfv-SIG*) *of-int-0-le-iff zero-le-divide-iff zero-le-power2*)
have *5*: (*1/2*)^(*Suc n*) ≤ (*1/2*::*real*) **for** *n*::*nat*
proof (*induction n*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc n*)
note *HR*=*this*
have (*1/2*::*real*)^(*Suc n*) ≥ (*1/2*::*real*)^(*Suc* (*Suc n*)) **by** *simp*
then show *?case* **using** *HR order-trans* **by** *simp*
qed

hence *abs* (*C l w h g Y X*) / *abs* (*K k l w g Y X*) ≤ *1/2*
using *3 4 XBe3 5*[*of nat X*−*1*] *order-trans*
by (*smt* (*verit*, *del-insts*) *Suc-pred′ zero-less-nat-eq*)
hence *maj-CovK*: *abs* ((*C l w h g Y X*) / (*K k l w g Y X*)) ≤ *1/2* **by** *simp*

have *abs* (*2* ∗ *C l w h g Y X* − *2* ∗ *L l Y* ∗ *K k l w g Y X*) < *abs* (*K k l w g Y X*)
using *sat-f* **unfolding** *d2f-def*
using *abs-le-square-iff linorder-not-less* **by** *blast*
hence *abs* (*2* ∗ *C l w h g Y X* − *2* ∗ *L l Y* ∗ *K k l w g Y X*) / *abs* (*K k l w g*

Y X) < *1*
using *of-int-add* **by** (*smt* (*verit*, *ccfv-SIG*) *divide-less-eq-1 of-int-less-iff*)
hence *abs 2* ∗ *abs* (*C l w h g Y X* − *L l Y* ∗ *K k l w g Y X*) / *abs* (*K k l w g Y X*) < *1*
using *abs-mult*[*of 2*::*int C l w h g Y X* − *L l Y* ∗ *K k l w g Y X*]
by (*metis Groups.mult-ac*(*1*) *int-distrib*(*4*))
hence *abs* (*C l w h g Y X* − *L l Y* ∗ *K k l w g Y X*) / *abs* (*K k l w g Y X*) < *1/2*
by *linarith*
hence *6*: *abs* ((*C l w h g Y X* − *L l Y* ∗ *K k l w g Y X*) / *K k l w g Y X*) < *1/2*
using *of-int-abs of-int-diff Fields.field-abs-sgn-class.abs-divide* **by** *simp*
have *real-of-int* (*C l w h g Y X* − *L l Y* ∗ *K k l w g Y X*) / *real-of-int* (*K k l w g Y X*)
= *real-of-int* (*C l w h g Y X*) / *real-of-int* (*K k l w g Y X*) − *real-of-int* (*L l Y*) ∗
real-of-int (*K k l w g Y X*) / *real-of-int* (*K k l w g Y X*)
using *Fields.division-ring-class.diff-divide-distrib of-int-diff of-int-mult* **by** *metis*
hence *abs* (*C l w h g Y X* / *K k l w g Y X* − *L l Y*) < *1/2*
using *min-K B0 6* **by** *force*
hence *maj-LmCovK*: *abs* (*L l Y* − *C l w h g Y X* / *K k l w g Y X*) < *1/2*
by *linarith*

have *abs* (*L l Y*) ≤ *abs* (*L l Y* − *C l w h g Y X* / (*K k l w g Y X*)) +
abs ((*C l w h g Y X*) / (*K k l w g Y X*))
using *Groups.ordered-ab-group-add-abs-class.abs-triangle-ineq* **by** *linarith*
hence *abs* (*L l Y*) < *1* **using** *maj-LmCovK maj-CovK* **by** *linarith*
hence *l* ∗ *Y* = *0* **unfolding** *L-def* **by** *simp*
then show *?thesis* **using** *assms abs-lBe1* **by** *simp*
qed
hence *R* = *X*+*1* **using** *r-def* **by** *auto*
hence *K-is-ψU2V*: *K k l w g Y X* = *ψ-int* (*U l X Y*^*2∗*V w g Y*) (*X*+*1*) **using** *R-def* **by** *auto*
have *abs* (*U l X Y*^*2∗*V w g Y*) = *abs* (*U l X Y*)∗ (*abs* (*U l X Y*) ∗ *abs* (*V w g Y*))
using *abs-mult*[*of U l X Y*^*2 V w g Y*] *power2-eq-square*[*of abs* (*U l x Y*)]
apply *simp*
using *power2-eq-square* **by** *blast*
hence *abs* (*U l X Y*^*2∗*V w g Y*) > *abs* (*U l X Y*) ∗ *2*
using *mult-strict-left-mono*[*of 2 abs* (*U l X Y*) ∗ *abs* (*V w g Y*) *abs* (*U l X Y*)]
absUBe2 absV-Be8 mult-strict-mono[*of 1 abs* (*U l X Y*) *2 abs* (*V w g Y*)] **by** *linarith*
hence *abs* (*U l X Y*^*2∗*V w g Y*) > *abs* (*2∗X∗*(*l∗Y*))∗*2* **unfolding** *U-def L-def* **by** *auto*
hence *abs* (*U l X Y*^*2∗*V w g Y*) > *4∗X∗*(*abs l* ∗ *Y*)
using *abs-mult*[*of 2∗X l∗Y*] *abs-mult*[*of 2 X*] *abs-mult*[*of l Y*] *assms*(*2*) *XBe3* **by** *linarith*
hence *U2V-B4X*: *abs* (*U l X Y*^*2∗*V w g Y*) > *4∗X* **using** *mult-strict-left-mono*[*of*

*1 abs l * Y 4*X]*
mult-strict-mono[of 1 abs l 1 Y] assms(2) lx-nonzero XBe3 mult-left-mono[of 3 X 4]
apply *simp*
by (*smt* (*z3*) ‹⟦*1 < |l| * Y*; *0 < 4 * X*⟧ ⟹ *4 * X * 1 < 4 * X * (|l| * Y)*› *pos-zmult-eq-1-iff zmult-zless-mono2*)
hence *abs* (*K k l w g Y X*) = *ψ* (*abs* (*U l X Y^2*V w g Y*)) (*nat* (*X+1*))
using *K-is-ψU2V XBe3 eq-ψ-int[of U l X Y^2*V w g Y X+1]* **by** *auto*
hence *K-nonzero*: *K k l w g Y X* ≠ *0*
using *lucas-monotone3[of abs* (*U l X Y^2*V w g Y*) *nat* (*X+1*)] *U2V-B4X XBe3* **by** *auto*
have *abs* (*real-of-int* (*C l w h g Y X*) /(*K k l w g Y X*))
= *abs* (*ψ-int* (*A l w g Y X*) (*2*X+1*) / *ψ-int* (*U l X Y^2*V w g Y*) (*X+1*))
using *K-is-ψU2V C-is-ψBA B-def[of X]* **by** *auto*
hence *abs* (*real-of-int* (*C l w h g Y X*) /(*K k l w g Y X*))
= *abs* (*ψ-int* (*A l w g Y X*) (*2*X+1*)) / *abs* (*ψ-int* (*U l X Y^2*V w g Y*) (*X+1*)) **by** *auto*
hence *eq-CoverK*: *abs* (*real-of-int* (*C l w h g Y X*) /(*K k l w g Y X*))
= *ψ* (*abs* (*A l w g Y X*)) (*nat* (*2*X+1*))/ *ψ* (*abs* (*U l X Y^2*V w g Y*)) (*nat* (*X+1*))
using *eq-ψ-int XBe3 abs-U2V-B2 abs-A-Be16* **by** *auto*
define *ϱ* **where** *ϱ* = (*abs* (*V w g Y*)+1) ^(*2*nat X*)/(*abs* (*V w g Y*)) ^(*nat X*)
have *nat2X12X*: *nat* (*2*X+1*) = *2*nat X +1* ∧ *nat* (*X+1*) = *nat X +1*
using *XBe3* **by** (*auto simp add*: *algebra-simps*)
hence *eq2-CoverK*: *abs* (*real-of-int* (*C l w h g Y X*) /(*K k l w g Y X*))
= *ψ* (*abs* (*A l w g Y X*)) (*2*nat X+1*)/ *ψ* (*abs* (*U l X Y^2*V w g Y*)) (*nat X+1*)
using *eq-CoverK* **by** *auto*

have *e^2 < f^2* ⟹ *abs e < abs f* **for** *e f*::*int*
by (*smt* (*verit*, *del-insts*) *abs-if-raw abs-le-square-iff*)
hence *abs* (*2*C l w h g Y X − 2*L l Y * K k l w g Y X*) < *abs* (*K k l w g Y X*)
using *sat-f* **unfolding** *d2f-def* **by** *blast*
hence *ineq1-CmLK*: *abs* (*2*real-of-int* (*C l w h g Y X*) − *2*real-of-int* (*L l Y*) * *K k l w g Y X*)
< *abs* (*real-of-int* (*K k l w g Y X*))
by (*metis* (*no-types*) *of-int-abs of-int-diff of-int-less-iff of-int-mult of-int-numeral*)
have *2*real-of-int* (*K k l w g Y X*) * (*real-of-int* (*C l w h g Y X*) / (*K k l w g Y X*) − *real-of-int* (*L l Y*))
= *2*real-of-int* (*C l w h g Y X*) − *2*real-of-int* (*L l Y*) * *K k l w g Y X*
using *K-nonzero distrib-left[of 2*real-of-int* (*K k l w g Y X*)
real-of-int (*C l w h g Y X*) / (*K k l w g Y X*) *real-of-int* (*L l Y*)]
by (*auto simp add*: *algebra-simps*)
hence *abs* (*2*real-of-int* (*K k l w g Y X*) * (*real-of-int* (*C l w h g Y X*) / (*K k l w g Y X*)
− *real-of-int* (*L l Y*))) < *abs* (*real-of-int* (*K k l w g Y X*))
using *ineq1-CmLK* **by** *presburger*
hence *abs* (*2*real-of-int* (*K k l w g Y X*) * (*real-of-int* (*C l w h g Y X*) / (*K k

l w g Y X)
  − *real-of-int* (*L l Y*))) < *abs* (*2*real-of-int* (*K k l w g Y X*))/*2* **by** *argo*
 hence *abs* (*2*real-of-int* (*K k l w g Y X*)) ∗ *abs* (*real-of-int* (*C l w h g Y X*) / (*K k l w g Y X*)
  − *real-of-int* (*L l Y*)) < *abs* (*2*real-of-int* (*K k l w g Y X*))∗ (*1/2*)
  by (*auto simp add*: *abs-mult*)
 hence *ineq2-CmLK*: *abs* (*real-of-int* (*C l w h g Y X*) / (*K k l w g Y X*) − *real-of-int* (*L l Y*)) < *1/2*
  using *K-nonzero*
  using *abs-ge-zero mult-less-cancel-left* **by** *blast*
 have *e∗f* ≥ *0* ⟹ *abs* (*e−f*) < *i* ⟹ *abs e* < *abs f* +*i* **for** *e f i*::*real* **by** *auto*
 hence *ineq3-CmLK*: *abs* (*real-of-int* (*C l w h g Y X*) / (*K k l w g Y X*)) < *abs* (*real-of-int* (*L l Y*)) + *1/2*
  using *ineq2-CmLK* **by** *argo*

 have *ψAB-is-pos*: *ψ* (*abs* (*A l w g Y X*)) (*2∗nat X+1*) ≥ (*0*::*real*)
  using *lucas-monotone3*[*of abs* (*A l w g Y X*) *2∗nat X* +*1*]*XBe3 abs-A-Be16 U2V-B4X* **by** *auto*
 have *ψU2V-is-pos*: *ψ* (*abs* (*U l X Y^2∗V w g Y*)) (*nat X+1*) > (*0*::*real*)
  using *lucas-monotone3*[*of abs* (*U l X Y^2∗V w g Y*) *nat X* +*1*] *XBe3 abs-A-Be16 U2V-B4X* **by** *auto*
 have *many-easy-ineq*: *2∗nat X* − *1* + *1* = *2∗nat X*
  ∧ *Suc* (*Suc* (*2∗nat X* − *1*)) = *2∗nat X* +*1* ∧ *nat X* − *2* + *2* = *nat X*
  ∧ *Suc* (*Suc* (*Suc* (*nat X* − *2*))) = *nat X* +*1* ∧ *1* < *abs* (*A l w g Y X*)
  ∧ *1* < *abs* ((*U l X Y*)^2 ∗ *V w g Y*)
  using *XBe3 abs-A-Be16 U2V-B4X* **by** (*auto simp add*: *algebra-simps*)
 hence *ψ* (*abs* (*A l w g Y X*)) (*2∗nat X+1*) ≥ (*abs* (*A l w g Y X*) − *1*) ^(*2∗nat X*)
  ∧ *ψ* (*abs* (*U l X Y^2∗V w g Y*)) (*nat X+1*) ≤ (*abs* (*U l X Y^2∗V w g Y*)) ^(*nat X*)
  using *lucas-exp-growth-gt*[*of abs* (*A l w g Y X*) *2∗nat X* − *1*]
   lucas-exp-growth-lt[*of abs* (*U l X Y^2∗V w g Y*) *nat X−2*]
  by (*simp add*: *lucas-exp-growth-gt*[*of abs* (*A l w g Y X*) *2∗nat X* − *1*]
    lucas-exp-growth-lt[*of abs* (*U l X Y^2∗V w g Y*) *nat X−2*])
 hence *real-ineq-ψ*: *real-of-int* (*ψ* (*abs* (*A l w g Y X*)) (*2∗nat X+1*)) ≥ (*abs* (*A l w g Y X*) − *1*) ^(*2∗nat X*)
  ∧ *real-of-int* (*ψ* (*abs* (*U l X Y^2∗V w g Y*)) (*nat X+1*)) ≤ (*abs* (*U l X Y^2∗V w g Y*)) ^(*nat X*)
  by *presburger*

 have *abs* (*A l w g Y X*) − *1* = *real-of-int* (*abs* (*U l X Y*))∗*abs* (*V w g Y* +*1*) − *abs* (*U l X Y*)∗(*1*/(*abs* (*U l X Y*)))
  unfolding *A-def* **using** *abs-mult absUBe2* **by** *auto*
 hence *eqA*: *abs* (*A l w g Y X*) − *1* = *abs* (*U l X Y*)∗(*abs* (*V w g Y* +*1*) − *1*/*abs* (*U l X Y*))
  using *distrib-left*[*of abs* (*U l X Y*) *abs* (*V w g Y+1*) *1*/(*abs* (*U l X Y*))]
  by (*smt* (*verit*, *ccfv-SIG*) *ring-class.ring-distribs*(*1*))
 have *real-of-int* (*ψ* (*abs* (*A l w g Y X*)) (*2∗nat X+1*)) ≥ (*real-of-int* (*abs* (*A l w g Y X*) − *1*)) ^(*2∗nat X*)

by (*metis real-ineq-ψ of-int-power-eq-of-int-cancel-iff*)
hence *ψ* (*abs* (*A l w g Y X*)) (*2∗nat X+1*) ≥ (*abs* (*U l X Y*)∗(*abs* (*V w g Y +1*) − *1/abs* (*U l X Y*)))^(*2∗nat X*)
using *eqA* **by** *metis*
hence *other-ineq-ψAB*: *ψ* (*abs* (*A l w g Y X*)) (*2∗nat X+1*) ≥ (*abs* (*U l X Y*))^(*2∗nat X*)∗(*abs* (*V w g Y +1*) − *1/abs* (*U l X Y*))^(*2∗nat X*)
using *power-mult-distrib*[*of abs* (*U l X Y*) *abs* (*V w g Y +1*) − *1/abs* (*U l X Y*) *2∗nat X*]
by *auto*
have *1/abs* (*U l X Y*) ≤ *1* **using** *absUBe2 divide-le-eq-1-pos* **by** *auto*
hence *abs* (*V w g Y +1*) − *1/abs* (*U l X Y*) ≥ *abs* (*V w g Y*) − *2* **by** *linarith*
hence *several-pow-pos*: (*abs* (*V w g Y +1*) − *1/abs* (*U l X Y*))^(*2∗nat X*) ≥ (*abs* (*V w g Y*) − *2*)^(*2∗nat X*)
∧ (*abs* (*U l X Y*))^(*2∗nat X*) > *0* ∧ (*abs* (*V w g Y*) − *2*)^(*2∗nat X*) ≥ *0*
using *power-mono*[*of abs* (*V w g Y*) − *2 abs* (*V w g Y +1*) − *1/abs* (*U l X Y*) *2∗nat X*] *absV-Be8*
by (*smt* (*z3*) *absUBe2 of-int-1-le-iff of-int-power zero-less-power*)
hence *other-ineq-ψAB-2*: *real-of-int* (*ψ* (*abs* (*A l w g Y X*)) (*2∗nat X+1*))
≥ *real-of-int* ((*abs* (*U l X Y*))^(*2∗nat X*))∗*real-of-int*((*abs* (*V w g Y*) − *2*)^(*2∗nat X*))
using *other-ineq-ψAB mult-left-mono*[*of real-of-int* (*abs* (*V w g Y*) − *2*)^(*2∗nat X*)
(*abs* (*V w g Y +1*) − *1/abs* (*U l X Y*))^(*2∗nat X*) *real-of-int* (*abs* (*U l X Y*))^(*2∗nat X*)]
by (*smt* (*z3*) *of-int-power zero-le-even-power′*)

have *real-of-int* (*ψ* (*abs* (*U l X Y^2∗V w g Y*)) (*nat X+1*)) ≤ (*abs* (*U l X Y^2*)∗*abs* (*V w g Y*))^(*nat X*)
using *abs-mult*[*of U l X Y^2 V w g Y*] *real-ineq-ψ* **by** *auto*
hence *real-of-int* (*ψ* (*abs* (*U l X Y^2∗V w g Y*)) (*nat X +1*))
≤ (*abs* (*U l X Y^2*))^(*nat X*)∗*abs* (*V w g Y*)^(*nat X*)
using *power-mult-distrib*[*of abs* (*U l X Y^2*) *abs* (*V w g Y*) *nat X*] **by** *metis*
hence *real-of-int* (*ψ* (*abs* (*U l X Y^2∗V w g Y*)) (*nat X +1*))
≤ (*abs* (*U l X Y*)^2)^(*nat X*)∗*abs* (*V w g Y*)^(*nat X*)
by *auto*
hence *real-of-int* (*ψ* (*abs* (*U l X Y^2∗V w g Y*)) (*nat X +1*))
≤ (*abs* (*U l X Y*))^(*2∗nat X*)∗*abs* (*V w g Y*)^(*nat X*)
using *power-mult*[*of abs* (*U l X Y*) *2 nat X*] **by** *auto*
hence *ineq-ψU2V*: *real-of-int* (*ψ* (*abs* (*U l X Y^2∗V w g Y*)) (*nat X +1*))
≤ *real-of-int* ((*abs* (*U l X Y*))^(*2∗nat X*))∗*real-of-int* ((*abs* (*V w g Y*))^(*nat X*)) **by** *auto*
hence *ψ* (*abs* (*A l w g Y X*)) (*2∗nat X+1*) / *ψ* (*abs* (*U l X Y^2∗V w g Y*)) (*nat X+1*)
≥ *real-of-int* ((*abs* (*U l X Y*))^(*2∗nat X*))∗*real-of-int*((*abs* (*V w g Y*) − *2*)^(*2∗nat X*))
/ (*real-of-int* ((*abs* (*U l X Y*))^(*2∗nat X*))∗*real-of-int* ((*abs* (*V w g Y*))^(*nat X*)))
using *ψAB-is-pos ψU2V-is-pos other-ineq-ψAB-2 frac-le*

```
    [of ψ (abs (A l w g Y X)) (2*nat X+1)
       real-of-int ((abs (U l X Y))^(2*nat X))*real-of-int((abs (V w g Y) −
2)^(2*nat X))
      ψ (abs (U l X Y^2*V w g Y)) (nat X+1)
      real-of-int ((abs (U l X Y))^(2*nat X))*real-of-int ((abs (V w g Y))^(nat
X))]
   by auto
 hence abs (real-of-int (C l w h g Y X) /(K k l w g Y X))
    ≥ real-of-int ((abs (U l X Y))^(2*nat X))*real-of-int((abs (V w g Y) −
2)^(2*nat X))
   / (real-of-int ((abs (U l X Y))^(2*nat X))*real-of-int ((abs (V w g Y))^(nat
X)))
   using eq2-CoverK by auto
 hence abs (real-of-int (C l w h g Y X) /(K k l w g Y X))
    ≥ real-of-int((abs (V w g Y) − 2)^(2*nat X)) / real-of-int ((abs (V w g
Y))^(nat X))
   using several-pow-pos by auto
 hence abs (real-of-int (C l w h g Y X) /(K k l w g Y X))
    ≥ real-of-int(((abs (V w g Y) − 2)^2)^(nat X)) / real-of-int ((abs (V w g
Y))^(nat X))
   using power-mult[of abs (V w g Y) − 2 2 nat X] by auto
 hence abs (real-of-int (C l w h g Y X) /(K k l w g Y X))
    ≥ (real-of-int((abs (V w g Y) − 2)^2))^(nat X) / (real-of-int (abs (V w g
Y)))^(nat X) by auto
 hence ineq-CoverK-1: abs (real-of-int (C l w h g Y X) /(K k l w g Y X))
  ≥ ((real-of-int((abs (V w g Y) − 2)^2)) / (real-of-int (abs (V w g Y))))^(nat
X)
   by (metis power-divide)

 have (abs (V w g Y)−2)^2 ≥ abs (V w g Y)*abs (V w g Y) − 4*abs (V w g
Y)
   using power2-diff[of abs (V w g Y) 2] power2-eq-square[of V w g Y] by auto
 hence 2*(abs (V w g Y) − 2)^2 ≥ abs (V w g Y)*abs (V w g Y) + abs (V w
g Y)*abs(V w g Y) − 8*abs (V w g Y)
   by auto
 hence 2*(abs (V w g Y)−2)^2 ≥ abs (V w g Y)*abs (V w g Y) − abs (V w g
Y)
   using absV-Be8 mult-right-mono[of 7 abs (V w g Y) abs (V w g Y)] by force
 hence 2*(abs (V w g Y)−2)^2 ≥ abs (V w g Y)*(abs (V w g Y) − 1)
   by (auto simp add: algebra-simps)
 hence ineq-absVm2: 2*real-of-int ((abs (V w g Y)−2)^2)
  ≥ real-of-int (abs (V w g Y))*real-of-int (abs (V w g Y)−1)
   by (metis of-int-le-iff of-int-mult of-int-numeral)
 have (2*real-of-int ((abs (V w g Y)−2)^2))/(2*abs (V w g Y)) = (real-of-int((abs
(V w g Y) − 2)^2))
   / (real-of-int (abs (V w g Y))) ∧ (real-of-int (abs (V w g Y))*real-of-int (abs
(V w g Y)−1))
   /(2*abs (V w g Y)) = (abs (V w g Y) − 1)/2 using absV-Be8 by auto
 hence (real-of-int((abs (V w g Y) − 2)^2)) / (real-of-int (abs (V w g Y))) ≥
```

(abs (V w g Y) − 1)/2
using *ineq-absVm2 divide-right-mono[of abs (V w g Y)*(abs (V w g Y) − 1)*
2(abs (V w g Y)−2) ^2*
*2*abs (V w g Y)]*
absV-Be8 divide-right-mono of-int-0 of-int-power-le-of-int-cancel-iff zero-power2
by *auto*
hence *((real-of-int((abs (V w g Y) − 2) ^2)) / (real-of-int (abs (V w g Y)))) ^(nat X)*
≥ ((abs (V w g Y) − 1)/2) ^(nat X)
using *power-mono[of (abs (V w g Y) − 1)/2*
(real-of-int((abs (V w g Y) − 2) ^2)) / (real-of-int (abs (V w g Y))) nat X] absV-Be8
by *auto*
hence *abs (real-of-int (C l w h g Y X) /(K k l w g Y X)) ≥ ((abs (V w g Y) − 1)/2) ^(nat X)*
using *ineq-CoverK-1* **by** *auto*
hence *ineq-L-Vm1: abs (real-of-int (L l Y)) + 1/2 ≥ ((abs (V w g Y) − 1)/2) ^(nat X)*
using *ineq3-CmLK* **by** *auto*
have *(abs (V w g Y) − 1)/2 ≥ 2* **using** *absV-Be8* **by** *auto*
hence *((abs (V w g Y) − 1)/2) ^(nat X) ≥ 2* **using** *power-mono[of 2 (abs (V w g Y) − 1)/2 nat X]*
power-increasing-iff[of 2 1 nat X] XBe3 **by** *(smt (verit) self-le-power zero-less-nat-eq)*
hence *((abs (V w g Y) − 1)/2) ^(nat X) > 1/2*((abs (V w g Y) − 1)/2) ^(nat X)+1/2* **by** *auto*
hence *abs (real-of-int (L l Y)) > 1/2*((abs (V w g Y) − 1)/2) ^(nat X)*
using *ineq-L-Vm1* **by** *linarith*
hence *2*abs (real-of-int (L l Y)) > ((abs (V w g Y)−1)/2) ^(nat X)* **by** *auto*
hence *ineq-L-Vm1-2: 2*abs (L l Y) > ((abs (V w g Y)−1)/2) ^(nat X)* **by** *auto*
have *real-of-int X*(abs (V w g Y)−1) ≥ (abs (V w g Y)−1)*
using *XBe3 absV-Be8 mult-right-mono[of 1 real-of-int X abs (V w g Y)−1] of-int-1-le-iff*
by *auto*
hence *X*(abs (V w g Y)−1) > (abs (V w g Y)−1)/2* **using** *absV-Be8* **by** *auto*

hence *2*abs (L l Y)*real-of-int (X*(abs (V w g Y) − 1))*
> ((abs (V w g Y) − 1)/2) ^(nat X) ((abs (V w g Y) − 1)/2)* **using** *XBe3 absV-Be8 mult-strict-mono*
*[of ((abs (V w g Y) − 1)/2) ^(nat X) 2*abs (L l Y) (abs (V w g Y)−1)/2 X*(abs (V w g Y) − 1)]*
ineq-L-Vm1-2 absL-Be1 **by** *auto*
hence *2*abs (L l Y)*real-of-int (X*(abs (V w g Y) − 1)) > ((abs (V w g Y) − 1)/2) ^(nat X + 1)*
using *power-add[of (abs (V w g Y) − 1)/2 nat X 1]* **by** *(metis power-one-right)*
hence *2*abs (L l Y)*X*(abs (V w g Y) − 1) > ((abs (V w g Y) − 1)/2) ^(nat X + 1)* **by** *auto*
hence *2*abs (L l Y)*X*abs (V w g Y + 1) > ((abs (V w g Y) − 1)/2) ^(nat X + 1)* **using** *XBe3*

mult-left-mono[*of abs* (*V w g Y*) − *1 abs* (*V w g Y* + *1*) *2*∗*abs* (*L l Y*)∗*X*]
by (*smt* (*z3*) *nat-0-iff nat-abs-mult-distrib nat-mult-distrib of-int-less-iff*)
hence *abs* (*2*∗*L l Y* ∗ *X*∗(*V w g Y* + *1*)) > ((*abs* (*V w g Y*) − *1*)/*2*) ^(*nat X* + *1*)
using *XBe3* **by** (*auto simp add*: *abs-mult*)
hence *absA-B-Vm1*: *abs* (*A l w g Y X*) > ((*abs* (*V w g Y*) − *1*)/*2*) ^(*nat X* + *1*)
unfolding *A-def U-def* **by** (*auto simp add*: *algebra-simps*)

have *abs* (*V w g Y*) − *1* = *4*∗ *abs w* ∗*g*∗*Y* − *1*
unfolding *V-def* **using** *assms* **by** (*auto simp add*: *abs-mult*)
hence *abs* (*V w g Y*)−*1* ≥ *2*∗*abs w* ∗ *g* ∗ *Y*
using *mult-mono*[*of 2 4*∗*abs w* ∗*g 1 Y*] *mult-mono*[*of 2 4*∗*abs w 1 g*] *mult-mono*[*of 2 4 1 abs w*]
assms w-nonzero **by** *auto*
hence *real-of-int* (*abs* (*V w g Y*) − *1*) ≥ *2*∗*abs w* ∗ *g* ∗ *Y* **by** *presburger*
hence *ineq-Vm1-wgY*: (*abs* (*V w g Y*)−*1*)/*2* ≥ *abs w* ∗ *g* ∗ *Y*
using *divide-right-mono*[*of 2*∗*abs w* ∗ *g* ∗ *Y abs* (*V w g Y*) − *1 2*] **by** *auto*
have *abs w* ∗ *g* ∗ *Y* ≥ *abs w* ∗ *b*
using *assms mult-mono*[*of abs w abs w* ∗ *g b Y*] *mult-left-mono*[*of 1 g abs w*] **by** *auto*
hence *abs w* ∗ *g* ∗ *Y* ≥ *abs* (*W w b*)
unfolding *W-def* **using** *assms abs-mult*[*of w b*] **by** (*auto simp add*: *algebra-simps*)
hence (*abs* (*V w g Y*)−*1*)/*2* ≥ *abs* (*W w b*) **using** *ineq-Vm1-wgY* **by** *linarith*
hence *real-of-int* (*abs* (*A l w g Y X*)) > (*abs* (*W w b*)) ^(*nat X* + *1*)
using *power-mono*[*of abs* (*W w b*) (*abs* (*V w g Y*)−*1*)/*2 nat X* +*1*] *absA-B-Vm1* **by** *auto*
hence *abs* (*A l w g Y X*) > (*abs* (*W w b*)) ^(*nat X* + *1*) **by** *presburger*
hence *abs* (*A l w g Y X*) > (*abs* (*W w b*)) ^(*nat X* + *1*) ∧ *nat X* +*1* ≥ *4* ∧ *abs* (*W w b*) ≥ *1*
using *XBe3 W-nonzero* **by** *auto*
hence *abs* (*A l w g Y X*) > (*abs* (*W w b*)) ^(*4*)
using *power-increasing*[*of 4 nat X* +*1 abs* (*W w b*)] **by** *auto*
hence *A-B-W4*: *abs* (*A l w g Y X*) > (*W w b*) ^*4* **by** *auto*

have *abs w* ∗ *g* ∗ *Y* ≥ *2*^*8* **using** *mult-mono*[*of 1 abs w*∗*g 2*^*8 Y*] *mult-mono*[*of 1 abs w 1 g*]
w-nonzero assms **by** *auto*
hence *real-of-int* (*abs w* ∗ *g* ∗ *Y*) ≥ *2*^*8* **using** *numeral-power-le-of-int-cancel-iff* **by** *blast*
hence (*abs* (*V w g Y*)−*1*)/*2* ≥ *2*^*8* **using** *ineq-Vm1-wgY* **by** *auto*
hence *real-of-int* (*abs* (*A l w g Y X*)) > (*2*^*8*) ^(*nat X* + *1*)
using *power-mono*[*of 2*^*8* (*abs* (*V w g Y*)−*1*)/*2 nat X* +*1*] *absA-B-Vm1* **by** *auto*
hence *abs* (*A l w g Y X*) > (*2*^*8*) ^(*nat X* + *1*)
by (*metis of-int-less-iff of-int-numeral of-int-power*)
hence *abs* (*A l w g Y X*) > *2*^(*8*∗(*nat X* +*1*))
using *power-mult*[*of 2 8 nat X* +*1*] **by** *metis*
hence *abs* (*A l w g Y X*) > *2*^(*8*∗(*nat X* +*1*)) ∧ *4*∗*nat* (*B X*) ≤ *8*∗(*nat X* +*1*)

unfolding *B-def* **using** *XBe3* **by** *auto*
hence *A-B-2po4B*: *abs* (*A l w g Y X*) > *2*^(*4*∗*nat* (*B X*))
using *power-increasing*[*of 4*∗*nat* (*B X*) *8*∗(*nat X* +*1*) *2*]
by (*smt* (*verit*, *ccfv-SIG*))

have (*3*∗*W w b*∗*C l w h g Y X* − *2*∗((*W w b*)^*2* − *1*)) *mod* (*2*∗*A l w g Y X* − *5*) = *0 mod* (*2*∗*A l w g Y X* − *5*)
using *sat-c* **unfolding** *d2c-def S-def T-def* **by** *auto*
hence *3*∗*W w b*∗*ψ* (*A l w g Y X*) (*nat* (*B X*)) *mod* (*2*∗*A l w g Y X* − *5*) = *2*∗((*W w b*)^*2* − *1*) *mod* (*2*∗*A l w g Y X* − *5*)
using *C-is-ψBA mod-add-cong*[*of 3*∗*W w b*∗*C l w h g Y X* − *2*∗((*W w b*)^*2* − *1*) *2*∗*A l w g Y X* − *5*
0 2∗((*W w b*)^*2* − *1*) *2*∗((*W w b*)^*2* − *1*)] *ψ-int-def*[*of A l w g Y X B X*] *B-Be7* **by** *auto*
hence *W w b* = *2*^(*nat* (*B X*))
using *lucas-diophantine-rec*[*of nat* (*B X*) *W w b A l w g Y X*] *A-B-W4* *A-B-2po4B B-Be7* **by** *auto*
hence *bw-pos*: *b*∗*w* = *2*^(*nat* (*B X*)) ∧ *w* > *0* ∧ *b* > *0* **unfolding** *W-def* **using** *assms*
by (*metis nat-0-iff nat-less-eq-zless not-less0 zero-le-mult-iff zero-le-square zero-less-mult-iff zero-less-numeral zero-less-power*)
hence *nat b*∗*nat w* = *2*^(*nat* (*B X*)) **by** (*metis b-def nat-eq-numeral-power-cancel-iff nat-mult-distrib*)
hence *is-power2-b*: *is-power2* (*nat b*) **unfolding** *is-power2-def*
using *prime-power-mult-nat*[*of 2 nat b nat w nat* (*B X*)] *two-is-prime-nat* **by** *auto*

have *V w g Y* ≥ *4*∗*w*∗*g*∗*b* **unfolding** *V-def* **using** *mult-left-mono*[*of b Y 4*∗*w*∗*g*] *bw-pos assms* **by** *auto*
hence *V w g Y* ≥ *4*∗(*b*∗*w*)∗*g* **by** (*auto simp add*: *algebra-simps*)
hence *V w g Y* ≥ *4*∗*2*^(*nat* (*B X*))∗*g* **using** *bw-pos* **by** *auto*
hence *V w g Y* ≥ *4*∗*2*^(*nat* (*B X*))∗*g* ∧ (*4*::*int*)∗*2*^(*nat* (*B X*)) ≥ *0* **using** *zero-less-power*[*of 2 nat* (*B X*)]
zero-less-mult-iff[*of 4 2*^(*nat* (*B X*))] **by** *simp*
hence *V w g Y* ≥ *4*∗*2*^(*nat* (*B X*)) **using** *mult-left-mono*[*of 1 g 4*∗*2*^(*nat* (*B X*))] *assms* **by** *presburger*
hence *V w g Y* ≥ *4*∗*2*^(*2*∗*nat X* +*1*)
unfolding *B-def* **by** (*simp add*: ‹*nat* (*2* ∗ *X* + *1*) = *2* ∗ *nat X* + *1* ∧ *nat* (*X* + *1*) = *nat X* + *1*›)
hence *V-Be82is-power2X*: *V w g Y* ≥ *8*∗*2*^(*2*∗*nat X*) **by** (*simp add*: *power-add*)
hence *VBe8*: *V w g Y* ≥ *8*
using ‹*4* ∗ *2* ^ *nat* (*B X*) ∗ *g* ≤ *V w g Y* ∧ *0* ≤ *4* ∗ *2* ^ *nat* (*B X*)› ‹*4* ∗ *2* ^ *nat* (*B X*) ≤ *V w g Y*› *absV-Be8*
by *linarith*

define *ϱ* **where** *ϱ* = (*V w g Y* +*1*)^(*2*∗*nat X*)/(*V w g Y*)^(*nat X*)
have *ϱ* = (∑ *i*≤*2*∗*nat X*. (*2*∗*nat X choose i*)∗*V w g Y*^*i*)/(*V w g Y*)^(*nat X*) **unfolding** *ϱ-def*
using *binomial-ring*[*of V w g Y 1 2*∗*nat X*] **by** *auto*

hence $\varrho = (\sum i{\leq}2{*}nat\ X.\ (2{*}nat\ X\ choose\ (nat\ (int\ i)))*V\ w\ g\ Y\hat{}(nat\ (int\ i)))/(V\ w\ g\ Y)\hat{}(nat\ X)$
by *auto*
hence $\varrho = (sum\ (\lambda i.\ (2{*}nat\ X\ choose\ nat\ i)*V\ w\ g\ Y\ \hat{}(nat\ i))\ (set[0..int\ (2{*}nat\ X)]))/(V\ w\ g\ Y)\hat{}(nat\ X)$
using *change-sum*[*of* $\lambda i.\ (2{*}nat\ X\ choose\ (nat\ i))*V\ w\ g\ Y\hat{}(nat\ i)\ 2{*}nat\ X$] **by** *presburger*
hence $\varrho = (sum\ (\lambda i.\ (2{*}nat\ X\ choose\ nat\ i)*V\ w\ g\ Y\ \hat{}(nat\ i))\ (set[0..2{*}X]))/(V\ w\ g\ Y)\hat{}(nat\ X)$
using *XBe3* **by** *auto*
hence $\varrho = (sum\ (\lambda i.\ real\ (2{*}nat\ X\ choose\ nat\ i)*V\ w\ g\ Y\ \hat{}(nat\ i))\ (set[0..2{*}X]))/(V\ w\ g\ Y)\hat{}(nat\ X)$
by *auto*
hence *ϱ-binom1*: $\varrho = (sum\ (\lambda i.\ real\ (2{*}nat\ X\ choose\ nat\ i)*V\ w\ g\ Y\ \hat{}(nat\ i)\ /(V\ w\ g\ Y)\hat{}(nat\ X))$
$(set[0..2{*}X]))$
using *sum-divide-distrib* **by** *blast*
have $(set[0..2{*}X]) = (set[0..X-1]) \cup (set[X..2{*}X]) \wedge (set[0..X-1]) \cap (set[X..2{*}X]) = \{\}$
$\wedge\ finite\ (set[0..X-1]) \wedge finite\ (set[0..2{*}X])$ **by** *auto*
hence *ϱ-binom2*: $\varrho = (sum\ (\lambda i.\ real\ (2{*}nat\ X\ choose\ nat\ i)*V\ w\ g\ Y\ \hat{}(nat\ i)\ /(V\ w\ g\ Y)\hat{}(nat\ X))\ (set[0..X-1]))$
$+\ (sum\ (\lambda i.\ real\ (2{*}nat\ X\ choose\ nat\ i)*V\ w\ g\ Y\ \hat{}(nat\ i)\ /(V\ w\ g\ Y)\hat{}(nat\ X))\ (set[X..2{*}X]))$
using *ϱ-binom1* **by** (*auto simp add*: *sum.union-disjoint*)
have $i{\in}\ (set[0..X-1]) \Longrightarrow real\text{-}of\text{-}int\ (V\ w\ g\ Y\ \hat{}(nat\ i))\ /(V\ w\ g\ Y)\hat{}(nat\ X) = 1/(V\ w\ g\ Y) * 1/(V\ w\ g\ Y)\hat{}(nat\ (X\ -\ i\ -\ 1))$ **for** i
proof −
fix i
assume *hyp*: $i{\in}\ (set[0..X-1])$
have *fact1*: $real\text{-}of\text{-}int\ (V\ w\ g\ Y\ \hat{}(nat\ i))*(V\ w\ g\ Y)\hat{}(nat\ X\ -\ nat\ i) = (V\ w\ g\ Y)\hat{}(nat\ X)$
using *power-add*[*of* $V\ w\ g\ Y\ nat\ i\ nat\ X\ -\ nat\ i$] *XBe3 hyp*
apply *simp*
by (*metis add-diff-inverse-nat less-SucI nat-less-eq-zless not-less-eq power-add*)
have $Suc\ (nat\ (X\ -\ i\ -\ 1)) = nat\ X\ -\ nat\ i$ **using** *hyp* **by** (*auto simp add*: *algebra-simps*)
hence $(V\ w\ g\ Y)\hat{}(nat\ i)*real\text{-}of\text{-}int\ (V\ w\ g\ Y) * (V\ w\ g\ Y)\hat{}(nat\ (X\ -\ i\ -\ 1)) = (V\ w\ g\ Y)\hat{}(nat\ X)$
using *power-Suc*[*of* $V\ w\ g\ Y\ nat\ X\ -\ nat\ i\ -\ 1$] *hyp XBe3 fact1*
apply *simp*
by (*metis* (*no-types*) *Nat.add-0-right* ‹$Suc\ (nat\ (X\ -\ i\ -\ 1)) = nat\ X\ -\ nat\ i$›
left-add-mult-distrib power-0 power-Suc power-add power-one-right)
hence *eq1*: $(V\ w\ g\ Y)\hat{}(nat\ i)*real\text{-}of\text{-}int\ (V\ w\ g\ Y) * (V\ w\ g\ Y)\hat{}(nat\ (X\ -\ i\ -\ 1))$
$/((V\ w\ g\ Y)\hat{}(nat\ X)*real\text{-}of\text{-}int\ (V\ w\ g\ Y)*(V\ w\ g\ Y)\hat{}(nat\ (X\ -\ i\ -\ 1)))$
$= (V\ w\ g\ Y)\hat{}(nat\ X)\ /\ ((V\ w\ g\ Y)\hat{}(nat\ X)*real\text{-}of\text{-}int\ (V\ w\ g\ Y)*(V\ w\ g\ Y)\hat{}(nat\ (X\ -\ i\ -\ 1)))$

by *auto*
have $(V\ w\ g\ Y)\hat{}(nat\ X)\ /\ ((V\ w\ g\ Y)\hat{}(nat\ X)*real\text{-}of\text{-}int\ (V\ w\ g\ Y)*(V\ w\ g\ Y)\hat{}(nat\ (X-i-1)))$
$= (V\ w\ g\ Y)\hat{}(nat\ X)\ /\ ((V\ w\ g\ Y)\hat{}(nat\ X)*(real\text{-}of\text{-}int\ (V\ w\ g\ Y)*(V\ w\ g\ Y)\hat{}(nat\ (X-i-1))))$
using *absV-Be8* **by** (*auto simp add*: *algebra-simps*)
have *triv-simp*: $a \neq 0 \Longrightarrow a/(a*c) = (1::real)/c$ **for** *a c* **by** *simp*
hence *eq2*: $(V\ w\ g\ Y)\hat{}(nat\ X)\ /\ ((V\ w\ g\ Y)\hat{}(nat\ X)*real\text{-}of\text{-}int\ (V\ w\ g\ Y)*(V\ w\ g\ Y)\hat{}(nat\ (X-i-1)))$
$= 1\ /\ (real\text{-}of\text{-}int\ (V\ w\ g\ Y)*(V\ w\ g\ Y)\hat{}(nat\ (X-i-1)))$ **using** *absV-Be8 triv-simp*[*of* $(V\ w\ g\ Y)\hat{}(nat\ X)$
real-of-int $(V\ w\ g\ Y)*(V\ w\ g\ Y)\hat{}(nat\ (X-i-1))$] **by** *auto*
hence $(V\ w\ g\ Y)\hat{}(nat\ X)\ /\ ((V\ w\ g\ Y)\hat{}(nat\ X)*real\text{-}of\text{-}int\ (V\ w\ g\ Y)*(V\ w\ g\ Y)\hat{}(nat\ (X-i-1)))$
$= 1\ /\ (real\text{-}of\text{-}int\ (V\ w\ g\ Y))*\ 1/(V\ w\ g\ Y)\hat{}(nat\ (X-i-1))$ **by** *auto*
have $(V\ w\ g\ Y)\hat{}(nat\ i)*real\text{-}of\text{-}int\ (V\ w\ g\ Y)\ *\ (V\ w\ g\ Y)\hat{}(nat\ (X-i-1))/((V\ w\ g\ Y)\hat{}(nat\ X)$
$*real\text{-}of\text{-}int\ (V\ w\ g\ Y)*(V\ w\ g\ Y)\hat{}(nat\ (X-i-1)))$
$= (V\ w\ g\ Y)\hat{}(nat\ i)*(real\text{-}of\text{-}int\ (V\ w\ g\ Y)\ *\ (V\ w\ g\ Y)\hat{}(nat\ (X-i-1))/(real\text{-}of\text{-}int\ (V\ w\ g\ Y)$
$*(V\ w\ g\ Y)\hat{}(nat\ (X-i-1))*(V\ w\ g\ Y)\hat{}(nat\ X)))$
by (*auto simp add*: *algebra-simps*)
hence $(V\ w\ g\ Y)\hat{}(nat\ i)*real\text{-}of\text{-}int\ (V\ w\ g\ Y)\ *\ (V\ w\ g\ Y)\hat{}(nat\ (X-i-1))/((V\ w\ g\ Y)\hat{}(nat\ X)$
$*real\text{-}of\text{-}int\ (V\ w\ g\ Y)*(V\ w\ g\ Y)\hat{}(nat\ (X-i-1)))$
$= (V\ w\ g\ Y)\hat{}(nat\ i)/((V\ w\ g\ Y)\hat{}(nat\ X))$
using *triv-simp*[*of real-of-int* $(V\ w\ g\ Y)\ *\ (V\ w\ g\ Y)\hat{}(nat\ (X-i-1))\ (V\ w\ g\ Y)\hat{}(nat\ X)$]
absV-Be8 **by** *auto*
thus *real-of-int* $(V\ w\ g\ Y\ \hat{}(nat\ i))\ /(V\ w\ g\ Y)\hat{}(nat\ X) = 1/(real\text{-}of\text{-}int\ (V\ w\ g\ Y))*1/(V\ w\ g\ Y)\hat{}(nat\ (X-i-1))$
using *eq1 eq2* **by** *auto*
qed
hence $i\in (set[0..X-1]) \Longrightarrow real\ (2*nat\ X\ choose\ nat\ i)*(V\ w\ g\ Y\ \hat{}(nat\ i)\ /(V\ w\ g\ Y)\hat{}(nat\ X))$
$= real\ (2*nat\ X\ choose\ nat\ i)\ *(1/(real\text{-}of\text{-}int\ (V\ w\ g\ Y))*1/(V\ w\ g\ Y)\hat{}(nat\ (X-i-1)))$ **for** *i*
by *auto*
hence $i\in (set[0..X-1]) \Longrightarrow real\ (2*nat\ X\ choose\ nat\ i)*V\ w\ g\ Y\ \hat{}(nat\ i)\ /(V\ w\ g\ Y)\hat{}(nat\ X)$
$= real\ (2*nat\ X\ choose\ nat\ i)\ *1/(real\text{-}of\text{-}int\ (V\ w\ g\ Y))*1/(V\ w\ g\ Y)\hat{}(nat\ (X-i-1))$ **for** *i*
by (*auto simp add*: *algebra-simps*)
hence $\varrho = (sum\ (\lambda i.\ real\ (2*nat\ X\ choose\ nat\ i)\ *1/(real\text{-}of\text{-}int\ (V\ w\ g\ Y))*1/(V\ w\ g\ Y)\hat{}(nat\ (X-i-1)))\ (set[0..X-1]))$
$+\ (sum\ (\lambda i.\ real\ (2*nat\ X\ choose\ nat\ i)*V\ w\ g\ Y\ \hat{}(nat\ i)\ /(V\ w\ g\ Y)\hat{}(nat\ X))\ (set[X..2*X]))$
using *ϱ-binom2* **by** *auto*
hence $\varrho = (sum\ (\lambda i.\ 1/(real\text{-}of\text{-}int\ (V\ w\ g\ Y))*(real\ (2*nat\ X\ choose\ nat\ i)$

$*1/(V\ w\ g\ Y)\ \hat{}(nat\ (X - i - 1)))))\ (set[0..X-1]))$
$+ (sum\ (\lambda i.\ real\ (2*nat\ X\ choose\ nat\ i)*V\ w\ g\ Y\ \hat{}(nat\ i)\ /(V\ w\ g\ Y)\ \hat{}(nat\ X))\ (set[X..2*X]))$
by (*auto simp add*: *algebra-simps*)
hence *ϱ-binom3*: $\varrho = 1/(real\text{-}of\text{-}int\ (V\ w\ g\ Y))*(sum\ (\lambda i.\ real\ (2*nat\ X\ choose\ nat\ i)\ *1/(V\ w\ g\ Y)\ \hat{}(nat\ (X - i - 1)))\ (set[0..X-1]))$
$+ (sum\ (\lambda i.\ real\ (2*nat\ X\ choose\ nat\ i)*V\ w\ g\ Y\ \hat{}(nat\ i)\ /(V\ w\ g\ Y)\ \hat{}(nat\ X))\ (set[X..2*X]))$
using *sum-distrib-left*[*of* $1/(real\text{-}of\text{-}int\ (V\ w\ g\ Y))\ \lambda i.\ real\ (2*nat\ X\ choose\ nat\ i)\ *1/(V\ w\ g\ Y)\ \hat{}(nat\ (X - i - 1))$
$set[0..X-1]$] **by** *auto*
have $set[X..2*X] = \{X\}\cup(set[X+1..2*X]) \wedge \{X\}\cap(set[X+1..2*X]) = \{\} \wedge finite\ \{X\} \wedge finite\ (set[X+1..2*X])$
using *XBe3* **by** *auto*
hence $\varrho = 1/(real\text{-}of\text{-}int\ (V\ w\ g\ Y))*(sum\ (\lambda i.\ real\ (2*nat\ X\ choose\ nat\ i)\ *1/(V\ w\ g\ Y)\ \hat{}(nat\ (X - i - 1)))\ (set[0..X-1]))$
$+ real\ (2*nat\ X\ choose\ nat\ X)*V\ w\ g\ Y\ \hat{}(nat\ X)\ /(V\ w\ g\ Y)\ \hat{}(nat\ X) + (sum\ (\lambda i.\ real\ (2*nat\ X\ choose\ nat\ i)$
$*\ V\ w\ g\ Y\ \hat{}(nat\ i)\ /(V\ w\ g\ Y)\ \hat{}(nat\ X))\ (set[X+1..2*X]))$
using *ϱ-binom3 sum.union-disjoint*[*of* $\{X\}\ set[X+1..2*X]$
$\lambda i.\ real\ (2*nat\ X\ choose\ nat\ i)*V\ w\ g\ Y\ \hat{}(nat\ i)\ /(V\ w\ g\ Y)\ \hat{}(nat\ X)$] **by** *auto*
hence *ϱ-binom4*: $\varrho = 1/(real\text{-}of\text{-}int\ (V\ w\ g\ Y))*(sum\ (\lambda i.\ real\ (2*nat\ X\ choose\ nat\ i)\ *1/(V\ w\ g\ Y)\ \hat{}(nat\ (X - i - 1)))\ (set[0..X-1]))$
$+ real\ (2*nat\ X\ choose\ nat\ X) + (sum\ (\lambda i.\ real\ (2*nat\ X\ choose\ nat\ i)*V\ w\ g\ Y\ \hat{}(nat\ i)\ /(V\ w\ g\ Y)\ \hat{}(nat\ X))\ (set[X+1..2*X]))$
using *absV-Be8* **by** *auto*

have *invV-pos*: $1/(real\text{-}of\text{-}int\ (V\ w\ g\ Y)) \geq 0$ **using** *VBe8* **by** *auto*
have *binom-is-pos*: $i\in(set[0..X-1])\cup(set[0..2*X]) \Longrightarrow real\ (2*nat\ X\ choose\ nat\ i) \geq 0$ **for** *i*
using *of-nat-0-le-iff* **by** *blast*
have $i\in\ (set[0..X-1]) \Longrightarrow 1/(V\ w\ g\ Y)\ \hat{}(nat\ (X - i - 1)) \geq 0$ **for** *i* **using** *VBe8* **by** *force*
hence $i\in\ (set[0..X-1]) \Longrightarrow real\ (2*nat\ X\ choose\ nat\ i)\ *1/(V\ w\ g\ Y)\ \hat{}(nat\ (X - i - 1)) \geq 0$ **for** *i*
using *binom-is-pos* **by** *simp*
hence *Vfrac-ϱ-pos*: $(sum\ (\lambda i.\ real\ (2*nat\ X\ choose\ nat\ i)\ *1/(V\ w\ g\ Y)\ \hat{}(nat\ (X - i - 1)))\ (set[0..X-1])) \geq 0$
using *sum-nonneg*[*of* $set[0..X-1]\ \lambda i.\ real\ (2*nat\ X\ choose\ nat\ i)\ *1/(V\ w\ g\ Y)\ \hat{}(nat\ (X - i - 1))$]
by *simp*
hence *frac-ϱ-pos*: $1/(real\text{-}of\text{-}int\ (V\ w\ g\ Y))*(sum\ (\lambda i.\ real\ (2*nat\ X\ choose\ nat\ i)$
$*\ 1/(V\ w\ g\ Y)\ \hat{}(nat\ (X - i - 1)))\ (set[0..X-1])) \geq 0$
using *invV-pos* **by** *simp*

have *obv-pos*: $1/(8::real)*1/2\ \hat{}(nat\ (2*X)) > 0$ **by** *force*
have $2*nat\ X = nat\ (2*X)$ **using** *XBe3* **by** (*auto simp add*: *algebra-simps*)

hence $2*X > X-1 \wedge X-1 > 0 \wedge 2*X \geq 0 \wedge real\ (2*nat\ X\ choose\ nat\ (2*X)) = 1$ **using** *XBe3* **by** *auto*
hence $2*X \in (set[0..2*X]) \wedge 2*X \notin (set[0..X-1]) \wedge real\ (2*nat\ X\ choose\ nat\ (2*X)) > 0$
$\wedge\ set[0..X-1] \subseteq set[0..2*X]$ **by** *auto*
hence *ineq-will-be-strict*: $2*X \in (set[0..2*X]) - (set[0..X-1]) \wedge real\ (2*nat\ X\ choose\ nat\ (2*X)) > 0$
$\wedge$ *finite* $(set[0..2*X]) \wedge set[0..X-1] \subseteq set[0..2*X]$ **using** *XBe3 Diff-iff*[*of* $2*X$ $set[0..2*X]$ $set[0..X-1]$]
by *blast*
have $i \in (set[0..X-1]) \Longrightarrow (V\ w\ g\ Y)\hat{}(nat\ (X - i - 1)) \geq 1$ **for** i **using** *VBe8* **by** *force*
hence $i \in (set[0..X-1]) \Longrightarrow 1/real\text{-}of\text{-}int\ ((V\ w\ g\ Y)\hat{}(nat\ (X-i-1))) \leq 1$ **for** i
by (*smt* (*verit*, *del-insts*) *divide-right-mono divide-self of-int-1-le-iff*)
hence $i \in (set[0..X-1]) \Longrightarrow real\ (2*nat\ X\ choose\ nat\ i) * 1/(V\ w\ g\ Y)\hat{}(nat\ (X - i - 1)) \leq real\ (2*nat\ X\ choose\ nat\ i)$ **for** i
using *binom-is-pos mult-left-mono*[*of* $1/(V\ w\ g\ Y)\hat{}(nat\ (X - i - 1))$ *1 real* $(2*nat\ X\ choose\ nat\ i)$]
by *simp*
hence $(sum\ (\lambda i.\ real\ (2*nat\ X\ choose\ nat\ i) * 1/(V\ w\ g\ Y)\hat{}(nat\ (X - i - 1)))\ (set[0..X-1]))$
$\leq (sum\ (\lambda i.\ real\ (2*nat\ X\ choose\ nat\ i)))\ (set[0..X-1])$ **using** *sum-mono*[*of* $set[0..X-1]$
$\lambda i.\ real\ (2*nat\ X\ choose\ nat\ i) * 1/(V\ w\ g\ Y)\hat{}(nat\ (X - i - 1))$ $\lambda i.\ real\ (2*nat\ X\ choose\ nat\ i)$]
by *auto*
hence $(sum\ (\lambda i.\ real\ (2*nat\ X\ choose\ nat\ i) * 1/(V\ w\ g\ Y)\hat{}(nat\ (X - i - 1)))\ (set[0..X-1]))$
$< (sum\ (\lambda i.\ real\ (2*nat\ X\ choose\ nat\ i)))\ (set[0..2*X])$ **using** *sum-strict-mono2*[*of* $set[0..2*X]$
$set[0..X-1]$ $2*X$ $\lambda i.\ real\ (2*nat\ X\ choose\ nat\ i)$] *ineq-will-be-strict binom-is-pos*
by *simp*
hence $(sum\ (\lambda i.\ real\ (2*nat\ X\ choose\ nat\ i) * 1/(V\ w\ g\ Y)\hat{}(nat\ (X - i - 1)))\ (set[0..X-1]))$
$< real\text{-}of\text{-}int\ ((sum\ (\lambda i.\ int\ (2*nat\ X\ choose\ nat\ i)))\ (set[0..2*X]))$ **by** *simp*
hence $(sum\ (\lambda i.\ real\ (2*nat\ X\ choose\ nat\ i) * 1/(V\ w\ g\ Y)\hat{}(nat\ (X - i - 1)))\ (set[0..X-1]))$
$< real\text{-}of\text{-}int\ (\sum i \leq nat(2*X).\ int\ (2*nat\ X\ choose\ i))$
using *change-sum*[*of* $\lambda i.\ int\ (2*nat\ X\ choose\ nat\ i)$ *nat* $(2*X)$] *XBe3* **by** *auto*
hence *hello*: $(sum\ (\lambda i.\ real\ (2*nat\ X\ choose\ nat\ i) * 1/(V\ w\ g\ Y)\hat{}(nat\ (X - i - 1)))\ (set[0..X-1]))$
$< real\ (\sum i \leq nat(2*X).\ 2*nat\ X\ choose\ i) \wedge 2*nat\ X = nat\ (2*X)$
by *auto*
hence $(sum\ (\lambda i.\ real\ (2*nat\ X\ choose\ nat\ i) * 1/(V\ w\ g\ Y)\hat{}(nat\ (X - i - 1)))\ (set[0..X-1]))$
$< real\ (\sum i \leq nat(2*X).\ nat\ (2*X)\ choose\ i)$
by *auto*
hence $(sum\ (\lambda i.\ real\ (2*nat\ X\ choose\ nat\ i) * 1/(V\ w\ g\ Y)\hat{}(nat\ (X - i - 1)))$

$(set[0..X-1]))$
$< real\ ((1+1)\ \hat{}(nat\ (2*X)))$ **using** *binomial*[*of 1 1 nat* $(2*X)$]
by *auto*
hence *sum-coeff-binom*: $(sum\ (\lambda i.\ real\ (2*nat\ X\ choose\ nat\ i)\ *1/(V\ w\ g\ Y)\ \hat{}(nat\ (X\ -\ i\ -\ 1)))\ (set[0..X-1]))$
$< 2\hat{}(nat\ (2*X))$
by *auto*
have $8*2\hat{}(2*nat\ X) \leq real\text{-}of\text{-}int\ (V\ w\ g\ Y)$ **using** *V-Be82is-power2X*
by (*metis* (*mono-tags*) *numeral-power-eq-of-int-cancel-iff of-int-le-iff of-int-mult of-int-numeral*)
hence $1/(real\text{-}of\text{-}int\ (V\ w\ g\ Y)) \leq 1/(8*2\hat{}(2*nat\ (X)))$
using *V-Be82is-power2X* **using** *inv-decr*[*of* $8*2\hat{}(2*nat\ X)$ *real-of-int* $(V\ w\ g\ Y)$] **by** *simp*
hence $1/(real\text{-}of\text{-}int\ (V\ w\ g\ Y)) \leq 1/8*1/2\hat{}(2*nat\ (X))$ **by** *auto*
hence $1/(real\text{-}of\text{-}int\ (V\ w\ g\ Y)) \leq 1/8*1/2\hat{}(nat\ (2*X))$ **using** *hello* **by** *auto*
hence $1/(real\text{-}of\text{-}int\ (V\ w\ g\ Y))*(sum\ (\lambda i.\ real\ (2*nat\ X\ choose\ nat\ i)\ *1/(V\ w\ g\ Y)\ \hat{}(nat\ (X\ -\ i\ -\ 1)))\ (set[0..X-1]))$
$< (1/8*1/2\hat{}(nat(2*X)))*2\hat{}(nat(2*X))$
using *sum-coeff-binom Vfrac-ϱ-pos obv-pos mult-strict-mono*[*of* $1/(real\text{-}of\text{-}int\ (V\ w\ g\ Y))$ $1/8*1/2\hat{}(nat\ (2*X))$
$(sum\ (\lambda i.\ real\ (2*nat\ X\ choose\ nat\ i)\ *1/(V\ w\ g\ Y)\ \hat{}(nat\ (X\ -\ i\ -\ 1)))\ (set[0..X-1]))\ 2\hat{}(nat\ (2*X))$]
by *fastforce*
hence *frac-ϱ-L8*: $1/(real\text{-}of\text{-}int\ (V\ w\ g\ Y))*(sum\ (\lambda i.\ real\ (2*nat\ X\ choose\ nat\ i)\ *\ 1/(V\ w\ g\ Y)\ \hat{}(nat\ (X\ -\ i\ -\ 1)))\ (set[0..X-1]))$
$< 1/8$ **using** *divide-self*[*of* $2\hat{}(nat(2*X))$] **by** *auto*

have $i \in set[X+1..2*X] \Longrightarrow real\text{-}of\text{-}int\ (V\ w\ g\ Y)\ \hat{}(nat\ i)\ /(real\text{-}of\text{-}int\ (V\ w\ g\ Y))\ \hat{}(nat\ X)$
$= V\ w\ g\ Y\ *\ real\text{-}of\text{-}int\ (V\ w\ g\ Y)\ \hat{}(nat\ i\ -\ nat\ X\ -\ 1)$ **for** i
using *VBe8 div-pow*[*of nat X nat i real-of-int* $(V\ w\ g\ Y)$] **by** *auto*
hence $i \in set[X+1..2*X] \Longrightarrow V\ w\ g\ Y\ \hat{}(nat\ i)\ /(V\ w\ g\ Y)\ \hat{}(nat\ X)$
$= V\ w\ g\ Y\ *\ real\text{-}of\text{-}int\ (V\ w\ g\ Y)\ \hat{}(nat\ i\ -\ nat\ X\ -\ 1)$ **for** i **by** *auto*
hence $i \in set[X+1..2*X] \Longrightarrow real\ (2*nat\ X\ choose\ nat\ i)*(V\ w\ g\ Y\ \hat{}(nat\ i)\ /(V\ w\ g\ Y)\ \hat{}(nat\ X))$
$= V\ w\ g\ Y*(real\ (2*nat\ X\ choose\ nat\ i)*V\ w\ g\ Y\ \hat{}(nat\ i\ -\ nat\ X\ -\ 1))$ **for** i
by (*auto simp add*: *algebra-simps*)
hence $(sum\ (\lambda i.\ real\ (2*nat\ X\ choose\ nat\ i)*V\ w\ g\ Y\ \hat{}(nat\ i)\ /(V\ w\ g\ Y)\ \hat{}(nat\ X))\ (set[X+1..2*X]))$
$= (sum\ (\lambda i.\ V\ w\ g\ Y*(real\ (2*nat\ X\ choose\ nat\ i)*V\ w\ g\ Y\ \hat{}(nat\ i\ -\ nat\ X\ -\ 1)))\ (set[X+1..2*X]))$
by *auto*
hence $(sum\ (\lambda i.\ real\ (2*nat\ X\ choose\ nat\ i)*V\ w\ g\ Y\ \hat{}(nat\ i)\ /(V\ w\ g\ Y)\ \hat{}(nat\ X))\ (set[X+1..2*X]))$
$= V\ w\ g\ Y*(sum\ (\lambda i.\ (real\ (2*nat\ X\ choose\ nat\ i)*V\ w\ g\ Y\ \hat{}(nat\ i\ -\ nat\ X\ -\ 1)))\ (set[X+1..2*X]))$
by (*auto simp add*: *sum-distrib-left*)
hence $(sum\ (\lambda i.\ real\ (2*nat\ X\ choose\ nat\ i)*V\ w\ g\ Y\ \hat{}(nat\ i)\ /(V\ w\ g\ Y)\ \hat{}(nat\ X))\ (set[X+1..2*X]))$

$= V\ w\ g\ Y * (sum\ (\lambda i.\ (int\ (2 * nat\ X\ choose\ nat\ i) * V\ w\ g\ Y\ \hat{}\ (nat\ i - nat\ X - 1)))\ (set[X+1..2*X]))$
by *auto*
hence *ϱ-binom5*: $\varrho = 1/(real\text{-}of\text{-}int\ (V\ w\ g\ Y)) * (sum\ (\lambda i.\ real\ (2 * nat\ X\ choose\ nat\ i)$
$* 1/(V\ w\ g\ Y)\ \hat{}\ (nat\ (X - i - 1)))\ (set[0..X-1])) + real\ (2 * nat\ X\ choose\ nat\ X)$
$+ V\ w\ g\ Y * (sum\ (\lambda i.\ (int\ (2 * nat\ X\ choose\ nat\ i) * V\ w\ g\ Y\ \hat{}\ (nat\ i - nat\ X - 1)))\ (set[X+1..2*X]))$
using *ϱ-binom4* **by** *auto*

define *ϱ-int* **where** $\varrho\text{-}int = int\ (2 * nat\ X\ choose\ nat\ X)$
$+ V\ w\ g\ Y * (sum\ (\lambda i.\ (int\ (2 * nat\ X\ choose\ nat\ i) * V\ w\ g\ Y\ \hat{}\ (nat\ i - nat\ X - 1)))\ (set[X+1..2*X]))$
define *ϱ-frac* **where** $\varrho\text{-}frac = 1/(real\text{-}of\text{-}int\ (V\ w\ g\ Y))$
$* (sum\ (\lambda i.\ real\ (2 * nat\ X\ choose\ nat\ i) * 1/(V\ w\ g\ Y)\ \hat{}\ (nat\ (X - i - 1)))\ (set[0..X-1]))$
have *ϱ-is-intpfrac*: $\varrho = \varrho\text{-}int + \varrho\text{-}frac$ **using** *ϱ-binom5 ϱ-int-def ϱ-frac-def* **by** *simp*
have *ineq-ϱ-frac*: $0 \leq \varrho\text{-}frac \wedge \varrho\text{-}frac < 1/8$ **using** *frac-ϱ-L8 frac-ϱ-pos ϱ-frac-def* **by** *simp*
hence *getting-int*: $abs\ (real\text{-}of\text{-}int\ q - \varrho) < 1-1/8 \Longrightarrow q = \varrho\text{-}int$ **for** *q*::*int*
proof −
assume *hyp*: $abs\ (real\text{-}of\text{-}int\ q - \varrho) < 1-1/8$
have $abs\ (real\text{-}of\text{-}int\ q - real\text{-}of\text{-}int\ \varrho\text{-}int) \leq abs\ (real\text{-}of\text{-}int\ q - \varrho) + abs\ (\varrho - \varrho\text{-}int)$
by *auto*
hence $abs\ (real\text{-}of\text{-}int\ q - real\text{-}of\text{-}int\ \varrho\text{-}int) < 1 - 1/8 + abs\ \varrho\text{-}frac$ **using** *hyp ϱ-is-intpfrac*
by *auto*
hence $abs\ (real\text{-}of\text{-}int\ q - real\text{-}of\text{-}int\ \varrho\text{-}int) < 1$ **using** *ineq-ϱ-frac* **by** *auto*
thus *?thesis* **by** *linarith*
qed

have $Y\ dvd\ V\ w\ g\ Y$ **unfolding** *V-def* **by** *auto*
hence $Y\ dvd\ V\ w\ g\ Y * (sum\ (\lambda i.\ (int\ (2 * nat\ X\ choose\ nat\ i) *$
$V\ w\ g\ Y\ \hat{}\ (nat\ i - nat\ X - 1)))\ (set[X+1..2*X]))$ **by** *auto*
hence *Y-dvd-iff*: $Y\ dvd\ (int\ (2 * nat\ X\ choose\ nat\ X)) = (Y\ dvd\ \varrho\text{-}int)$ **unfolding** *ϱ-int-def*
using *dvd-add-right-iff*[*of* $Y\ V\ w\ g\ Y * (sum\ (\lambda i.\ (int\ (2 * nat\ X\ choose\ nat\ i)$
$* V\ w\ g\ Y\ \hat{}\ (nat\ i - nat\ X - 1)))\ (set[X+1..2*X]))\ int\ (2 * nat\ X\ choose\ nat\ X)$] **by** *presburger*

have $abs\ (2 * C\ l\ w\ h\ g\ Y\ X - 2 * L\ l\ Y * K\ k\ l\ w\ g\ Y\ X) < abs\ (K\ k\ l\ w\ g\ Y\ X)$
using *sat-f* **unfolding** *d2f-def*
using *abs-le-square-iff linorder-not-less* **by** *blast*
hence $abs\ (2 * C\ l\ w\ h\ g\ Y\ X - 2 * L\ l\ Y * K\ k\ l\ w\ g\ Y\ X)\ /\ abs\ (K\ k\ l\ w\ g\ Y\ X) < 1$

using *of-int-add* **by** (*smt* (*verit*, *ccfv-SIG*) *divide-less-eq-1 of-int-less-iff*)
hence *abs 2 * abs (C l w h g Y X − L l Y * K k l w g Y X) / abs (K k l w g Y X) < 1*
using *abs-mult*[*of 2*::*int C l w h g Y X − L l Y * K k l w g Y X*]
by (*metis Groups.mult-ac*(*1*) *int-distrib*(*4*))
hence *abs (C l w h g Y X − L l Y * K k l w g Y X) / abs (K k l w g Y X) < 1/2*
by *linarith*
hence *6*: *abs ((C l w h g Y X − L l Y * K k l w g Y X) / K k l w g Y X) < 1/2*
using *of-int-abs of-int-diff Fields.field-abs-sgn-class.abs-divide* **by** *simp*
have *real-of-int (C l w h g Y X − L l Y * K k l w g Y X) / real-of-int (K k l w g Y X)*
= real-of-int (C l w h g Y X) / real-of-int (K k l w g Y X) − real-of-int (L l Y)
** real-of-int (K k l w g Y X) / real-of-int (K k l w g Y X)*
using *Fields.division-ring-class.diff-divide-distrib of-int-diff of-int-mult* **by** *metis*
hence *abs (C l w h g Y X / K k l w g Y X − L l Y) < 1/2*
using *K-nonzero 6* **by** *force*
hence *maj-LmCovK*: *abs (L l Y − C l w h g Y X / K k l w g Y X) < 1/2*
by *linarith*

have *abs (ϱ − (real-of-int (C l w h g Y X))/(K k l w g Y X)) ≤ 1 − 1/8 − 1/2 ⟹ ϱ-int = L l Y*
proof −
assume *hyp*: *abs (ϱ − (real-of-int (C l w h g Y X))/(K k l w g Y X)) ≤ 1 − 1/8 − 1/2*
have *abs (ϱ − L l Y) ≤ abs (ϱ − (real-of-int (C l w h g Y X))/(K k l w g Y X))*
+ abs ((real-of-int (C l w h g Y X))/(K k l w g Y X) − L l Y) **by** *auto*
hence *abs (ϱ − L l Y) < 1 − 1/8* **using** *maj-LmCovK hyp* **by** *argo*
thus *ϱ-int = L l Y* **using** *getting-int*[*of L l Y*] **by** *linarith*
qed
hence *abs (ϱ − (real-of-int (C l w h g Y X))/(K k l w g Y X)) ≤ 1 − 1/8 − 1/2*
*⟹ Y dvd (int (2*nat X choose nat X))*
unfolding *L-def* **using** *Y-dvd-iff* **by** *auto*
hence *ineq-required*: *abs (ϱ − (real-of-int (C l w h g Y X))/(K k l w g Y X)) ≤ 1/4*
*⟹ Y dvd (int (2*nat X choose nat X))* **by** *auto*

have *L-nonzero*: *L l Y ≠ 0* **unfolding** *L-def* **using** *lx-nonzero assms* **by** *auto*
hence *1+1/(2*abs (L l Y)) = (2*abs (L l Y))/(2*abs (L l Y)) + 1/(2*abs (L l Y))*
using *divide-self*[*of 2*abs (L l Y)*] **by** *auto*
hence *obv-eq-div2L*: *1+1/(2*abs (L l Y)) = (2*abs (L l Y) + 1)/(2*abs (L l Y))*
using *add-divide-distrib*[*of 2*abs (L l Y) 1 2*abs (L l Y)*] **by** *auto*

have *polyploidisation*: $1+1/(2*abs\ (L\ l\ Y)) > 0$
by (*smt* (*verit*, *ccfv-threshold*) *absL-Be1 of-int-pos zero-less-divide-1-iff*)

have *absU-expr*: $abs(U\ l\ X\ Y) = 2 * int(nat\ X) * abs\ (L\ l\ Y)$ **unfolding** *U-def* **using** *abs-mult XBe3*
by (*metis* (*mono-tags*) *abs-ge-zero abs-numeral abs-of-nat int-nat-eq order-trans*)
hence *aUBe2X*:$2 * int\ (nat\ X) \leq abs(U\ l\ X\ Y)$ **using** *absL-Be1 XBe3* **by** *auto*
have *VBe1*: $V\ w\ g\ Y \geq 1$ **using** *V-Be82is-power2X* **by** (*smt* (*verit*) *zero-less-power*)
have *abs2X1*: $nat\ (abs\ (2*X+1)) = 2*nat\ X+1$ **using** *XBe3* **by** *simp*
have $C\ l\ w\ h\ g\ Y\ X = \psi\text{-}int\ (U\ l\ X\ Y*(V\ w\ g\ Y+1))\ (2\ *\ X\ +\ 1)$ **using** *C-is-ψBA A-def B-def* **by** *metis*
hence *C-ψnat*:$C\ l\ w\ h\ g\ Y\ X = \psi\ (U\ l\ X\ Y*(V\ w\ g\ Y+1))\ (2\ *\ nat\ X\ +\ 1)$ **using** *ψ-int-def abs2X1 XBe3*
using *Groups.mult-ac*(1) *left-minus-one-mult-self one-add-one one-power2 power-0 power-add*
power-one-right **by** *force*
have $K\ k\ l\ w\ g\ Y\ X = \psi\text{-}int\ (U\ l\ X\ Y\hat{}2*V\ w\ g\ Y)\ (X+1)$ **using** *K-is-ψU2V* **by** *metis*
hence *K-ψnat*: $K\ k\ l\ w\ g\ Y\ X = \psi\ (U\ l\ X\ Y\hat{}2*V\ w\ g\ Y)\ (nat\ X+1)$ **using** *ψ-int-def XBe3 nat2X12X*
by *auto*
have $(C\ l\ w\ h\ g\ Y\ X)\ /\ (K\ k\ l\ w\ g\ Y\ X)$
$= (\psi\ (U\ l\ X\ Y*(V\ w\ g\ Y+1))\ (2\ *\ nat\ X\ +\ 1))\ /\ \psi\ (U\ l\ X\ Y\hat{}2*V\ w\ g\ Y)\ (nat\ X+1)$
using *C-ψnat K-ψnat* **by** *simp*
hence *absCKϱ-expl*:$abs\ (C\ l\ w\ h\ g\ Y\ X\ /\ K\ k\ l\ w\ g\ Y\ X - \varrho)$
$= abs\ ((\psi\ (U\ l\ X\ Y*(V\ w\ g\ Y+1))\ (2\ *\ nat\ X\ +\ 1))\ /\ \psi\ (U\ l\ X\ Y\hat{}2*V\ w\ g\ Y)\ (nat\ X+1) - \varrho)$
by *simp*
have $abs\ ((\psi\ (U\ l\ X\ Y*(V\ w\ g\ Y+1))\ (2\ *\ nat\ X\ +\ 1))\ /\ \psi\ (U\ l\ X\ Y\hat{}2*V\ w\ g\ Y)\ (nat\ X+1) - \varrho)$
$\leq (2\ *\ int(nat\ X))* \varrho\ /\ (abs(U\ l\ X\ Y) * V\ w\ g\ Y)$
using *XBe3 VBe1 aUBe2X ϱ-def lemma-4-4-cor-rho-abs*[*of nat X U l X Y V w g Y ϱ*] **by** *fastforce*
hence *majCKϱ1*: $abs\ ((C\ l\ w\ h\ g\ Y\ X)\ /\ (K\ k\ l\ w\ g\ Y\ X) - \varrho) \leq 2 * X * \varrho\ /\ (abs(U\ l\ X\ Y) * V\ w\ g\ Y)$
using *absCKϱ-expl XBe3* **by** *auto*

have $abs(V\ w\ g\ Y\ +\ 1) = V\ w\ g\ Y\ +\ 1$ **using** *VBe1* **by** *simp*
hence $abs\ (V\ w\ g\ Y\ +\ 1) - 1\ /\ real\text{-}of\text{-}int\ (abs\ (U\ l\ X\ Y)) \geq V\ w\ g\ Y$
using *aUBe2X XBe3* **by** *fastforce*
hence *aV1invU-maj*: $(real\text{-}of\text{-}int\ (abs(V\ w\ g\ Y\ +\ 1)) - 1\ /\ real\text{-}of\text{-}int\ (abs\ (U\ l\ X\ Y)))\hat{}(2 * nat\ X)$
$\geq (real\text{-}of\text{-}int\ (V\ w\ g\ Y))\hat{}(2 * nat\ X)$
using *XBe3 VBe1* **by** *simp*
have $real\text{-}of\text{-}int\ (abs\ (U\ l\ X\ Y)\ \hat{}\ (2 * nat\ X)) \geq 0$ **using** *aUBe2X XBe3* **by** *simp*

hence *real-of-int* (*abs* (*U l X Y*) ^ (2 * *nat X*)) * (*real-of-int* (*abs*(*V w g Y* + *1*))
− *1* / *real-of-int* (*abs* (*U l X Y*))^(2 * *nat X*)
≥ *real-of-int* (*abs*(*U l X Y*) ^ (2 * *nat X*)) * (*real-of-int* (*V w g Y*))^(2 * *nat X*)
using *aV1invU-maj mult-left-mono* **by** *fast*
hence *other-ineq-ψAB-3*: *real-of-int* (*ψ* (*abs* (*A l w g Y X*)) (*2*nat X+1*))
≥ *real-of-int* ((*abs* (*U l X Y*))^(*2*nat X*))*real-of-int((*V w g Y*)^(*2*nat X*))
using *other-ineq-ψAB* **by** *simp*
hence *ψ* (*abs* (*A l w g Y X*)) (*2*nat X+1*) / *ψ* (*abs* (*U l X Y*^2*V w g Y*)) (*nat X+1*)
≥ *real-of-int* ((*abs* (*U l X Y*))^(*2*nat X*))*real-of-int((*V w g Y*)^(*2*nat X*))
/ (*real-of-int* ((*abs* (*U l X Y*))^(*2*nat X*))*real-of-int ((*abs* (*V w g Y*))^(*nat X*)))
using *ψAB-is-pos ψU2V-is-pos other-ineq-ψAB-3 frac-le ineq-ψU2V* **by** *blast*
hence *abs* (*real-of-int* (*C l w h g Y X*)/(*K k l w g Y X*))
≥ *real-of-int* ((*abs* (*U l X Y*))^(*2*nat X*))*real-of-int((*V w g Y*)^(*2*nat X*))
/ (*real-of-int* ((*abs* (*U l X Y*))^(*2*nat X*))*real-of-int ((*abs* (*V w g Y*))^(*nat X*)))
using *eq2-CoverK* **by** *presburger*
hence *abs* (*real-of-int* (*C l w h g Y X*)/(*K k l w g Y X*))
≥ *real-of-int*((*V w g Y*)^(*2*nat X*)) / *real-of-int* ((*abs* (*V w g Y*))^(*nat X*))
using *several-pow-pos* **by** *fastforce*
hence *abs* (*real-of-int* (*C l w h g Y X*)/(*K k l w g Y X*))
≥ *real-of-int*((*V w g Y*)^(*2*nat X*)) / *real-of-int* ((*V w g Y*)^(*nat X*))
using *VBe1* **by** *fastforce*
hence *abs* (*real-of-int* (*C l w h g Y X*)/(*K k l w g Y X*))
≥ *real-of-int*(*V w g Y*)^(*2*nat X*) / *real-of-int* (*V w g Y*)^(*nat X*)
by *force*
hence *aCoK-Be-VpoX*: *abs* (*real-of-int* (*C l w h g Y X*)/(*K k l w g Y X*))
≥ *real-of-int*(*V w g Y*)^(*nat X*)
using *VBe1 div-pow'* **by** *force*
hence *abs*(*L l Y*) + *1/2* ≥ (*V w g Y*)^(*nat X*)
using *ineq3-CmLK* **by** *fastforce*
hence *ineq-absL*: ((*V w g Y*)^(*nat X*)/(*abs* (*L l Y*)+*1/2*)) ≤ *1*
using *polyploidisation divide-self*[*of abs* (*L l Y*)+*1/2*]
divide-right-mono[*of abs* (*L l Y*) + *1/2* (*V w g Y*)^(*nat X*) *abs* (*L l Y*)+*1/2*]
by *simp*

have *abs*(*U l X Y*) = *2* * *X* * *abs*(*L l Y*) **using** *absU-expr XBe3* **by** *simp*
hence *2* * *X* * *ϱ* / (*abs*(*U l X Y*) * *V w g Y*) = *2* * *X* * *ϱ* / (*2* * *X* * *abs*(*L l Y*) * *V w g Y*) **by** *simp*
hence *2* * *X* * *ϱ* / (*abs*(*U l X Y*) * *V w g Y*) = *ϱ* / (*abs*(*L l Y*) * *V w g Y*)
using *of-int-mult*[*of 2* * *X abs* (*L l Y*) * *V w g Y*] *XBe3 Fields.field-class.mult-divide-mult-cancel-left*
by *force*
hence *rewrite-2Xϱ*: *2* * *X* * *ϱ* / (*abs*(*U l X Y*) * *V w g Y*) = *ϱ* / (*abs*(*L l Y*)) * *1*/(*V w g Y*) **by** *auto*

have *simp-abs-L*: *abs* (*L l Y*)*(*1*+*1*/(*2*abs* (*L l Y*))) = *abs* (*L l Y*) + *1/2*

using *L-nonzero divide-self*[*of abs* (*L l Y*)] *distrib-left*[*of abs* (*L l Y*) *1 1*/(*2*∗*abs* (*L l Y*))]
by *auto*
have *ϱ / abs* (*L l Y*) = (*ϱ*/(*V w g Y*) ^(*nat X*))∗((*V w g Y*) ^(*nat X*)
/(*abs* (*L l Y*)∗(*1*+*1*/(*2*∗*abs* (*L l Y*)))))∗(*1*+*1*/(*2*∗*abs* (*L l Y*)))
using *VBe8 polyploidisation* **by** *fastforce*
hence *ϱ / abs* (*L l Y*) = (*ϱ*/(*V w g Y*) ^(*nat X*))∗((*V w g Y*) ^(*nat X*)
/(*abs* (*L l Y*)+*1*/*2*))∗(*1*+*1*/(*2*∗*abs* (*L l Y*)))
using *simp-abs-L* **by** *metis*
hence *rewrite-ϱL*: *ϱ / abs* (*L l Y*)
= (*ϱ*/(*V w g Y*) ^(*nat X*))∗(*1*+*1*/(*2*∗*abs* (*L l Y*)))∗((*V w g Y*) ^(*nat X*)/(*abs* (*L l Y*)+*1*/*2*)) **by** *auto*
have *ϱ-pos*: *ϱ* > *0* **unfolding** *ϱ-def* **using** *VBe1* **by** *simp*
hence (*ϱ*/(*V w g Y*) ^(*nat X*))∗(*1*+*1*/(*2*∗*abs* (*L l Y*))) ≥ *0* **using** *VBe8 polyploidisation* **by** *auto*
hence *ineq-ϱL1*: *ϱ / abs* (*L l Y*) ≤ (*ϱ*/(*V w g Y*) ^(*nat X*))∗(*1*+*1*/(*2*∗*abs* (*L l Y*)))
using *ineq-absL rewrite-ϱL*
mult-left-mono[*of* ((*V w g Y*) ^(*nat X*)/(*abs* (*L l Y*)+*1*/*2*)) *1* (*ϱ*/(*V w g Y*) ^(*nat X*))∗(*1*+*1*/(*2*∗*abs* (*L l Y*)))]
by *auto*

have *abs* (*L l Y*) = *abs l* ∗ *Y* **unfolding** *L-def* **using** *assms abs-mult*[*of l Y*] **by** *auto*
hence *abs* (*L l Y*) ≥ *2* **using** *lx-nonzero assms mult-mono*[*of 1 abs l 2 Y*] **by** *auto*
hence *real-of-int* (*2*∗*abs* (*L l Y*)) ≥ *4* **by** *auto*
hence *1*/(*real-of-int* (*2*∗*abs* (*L l Y*))) ≤ *1*/*4*
using *inv-decr*[*of 4 real-of-int* (*2*∗*abs* (*L l Y*))] **by** *auto*
hence *1* + *1*/(*real-of-int* (*2*∗*abs* (*L l Y*))) ≤ *5*/*4* **by** *auto*
hence *ineq-ϱL2*: *ϱ / abs* (*L l Y*) ≤ (*ϱ*/(*V w g Y*) ^(*nat X*))∗*5*/*4*
using *mult-left-mono*[*of 1* + *1*/(*real-of-int* (*2*∗*abs* (*L l Y*))) *5*/*4* (*ϱ*/(*V w g Y*) ^(*nat X*))]
ineq-ϱL1 ϱ-pos VBe8 **by** *auto*

have *bounding-1oV*: *0* < *1*/(*V w g Y*) ∧ *1*/(*V w g Y*) ≤ *1* **using** *VBe1 inv-decr*[*of 1 V w g Y*] **by** *auto*
have *ϱ*/(*V w g Y*) ^(*nat X*) = (*V w g Y* +*1*) ^(*2*∗*nat X*)/((*V w g Y*) ^(*nat X*)∗(*V w g Y*) ^(*nat X*))
unfolding *ϱ-def* **by** *auto*
hence *ϱ*/(*V w g Y*) ^(*nat X*) = (*V w g Y* +*1*) ^(*2*∗*nat X*)/(*V w g Y*) ^(*nat X* + *nat X*)
using *power-add*[*of V w g Y nat X nat X*] **by** *auto*
hence *ϱ*/(*V w g Y*) ^(*nat X*) = (*real-of-int* (*V w g Y* +*1*)) ^(*2*∗*nat X*)/(*real-of-int* (*V w g Y*)) ^(*2*∗*nat X*)
by (*simp add*: *mult-2*)
hence *ϱ*/(*V w g Y*) ^(*nat X*) = ((*V w g Y* +*1*)/(*V w g Y*)) ^(*2*∗*nat X*)
using *power-divide*[*of real-of-int* (*V w g Y* + *1*) *real-of-int* (*V w g Y*) *2*∗*nat X*] **by** *auto*

hence $\varrho/(V\ w\ g\ Y)\hat{}(nat\ X) = ((V\ w\ g\ Y)/(V\ w\ g\ Y) + 1/(V\ w\ g\ Y))\hat{}(2*nat\ X)$
using *VBe8 add-divide-distrib*[*of* $V\ w\ g\ Y\ 1\ V\ w\ g\ Y$] **by** *auto*
hence $\varrho/(V\ w\ g\ Y)\hat{}(nat\ X) = (1+1/(V\ w\ g\ Y))\hat{}(2*nat\ X)$
using *divide-self*[*of* $V\ w\ g\ Y$] *VBe8* **by** *auto*
hence $\varrho/(V\ w\ g\ Y)\hat{}(nat\ X) \leq 1+(2\hat{}(2*nat\ X)-1)*(1/(V\ w\ g\ Y))$
using *bounding-1oV power-majoration*[*of* $1/(V\ w\ g\ Y)\ 2*nat\ X$] **by** *simp*
hence *ineq-ϱV*: $\varrho/(V\ w\ g\ Y)\hat{}(nat\ X) \leq 1+2\hat{}(2*nat\ X)/(V\ w\ g\ Y)$
using *bounding-1oV mult-right-mono*[*of* $2\hat{}(2*nat\ X)-1\ 2\hat{}(2*nat\ X)\ 1/(V\ w\ g\ Y)$] **by** *auto*
have $V\ w\ g\ Y \geq 2\hat{}(2*nat\ X)$ **using** *VBe1 V-Be82is-power2X* **by** *force*
hence $1/(V\ w\ g\ Y) \leq 1/2\hat{}(2*nat\ X)$ **using** *inv-decr*[*of* $2\hat{}(2*nat\ X)\ V\ w\ g\ Y$] **by** *auto*
hence $2\hat{}(2*nat\ X)/(V\ w\ g\ Y) \leq 1$ **using** *divide-self*[*of* $2\hat{}(2*nat\ X)$]
by (*smt* (*z3*) ‹$8 * 2\ \hat{}\ (2 * nat\ X) \leq real\text{-}of\text{-}int\ (V\ w\ g\ Y)$› *bounding-1oV divide-nonneg-nonpos less-divide-eq-1-pos*)
hence $\varrho/(V\ w\ g\ Y)\hat{}(nat\ X) \leq 2$ **using** *ineq-ϱV* **by** *auto*
hence $\varrho\ /\ abs\ (L\ l\ Y) < 4$ **using** *ineq-ϱL2* **by** *auto*

hence $2 * X * \varrho\ /\ (abs(U\ l\ X\ Y) * V\ w\ g\ Y) \leq 4*1/(V\ w\ g\ Y)$
using *VBe1 rewrite-2Xϱ mult-right-mono*[*of* $\varrho/(abs\ (L\ l\ Y))\ 4\ 1/(V\ w\ g\ Y)$] **by** *auto*
hence *majCKϱ2*: $abs\ ((C\ l\ w\ h\ g\ Y\ X)\ /\ (K\ k\ l\ w\ g\ Y\ X) - \varrho) \leq 4/V\ w\ g\ Y$ **using** *majCKϱ1* **by** *auto*
have $V\ w\ g\ Y \geq 16$ **using** *V-Be82is-power2X XBe3 power-increasing*[*of* $1\ 2*nat\ X\ 2$]
by (*smt* (*verit*, *del-insts*) ‹$2 * nat\ X = nat\ (2 * X)$› *numeral-eq-one-iff numeral-power-le-nat-cancel-iff power-one-right*)
hence $4/(V\ w\ g\ Y) \leq 1/4$ **using** *inv-decr*[*of* $16\ V\ w\ g\ Y$] **by** *auto*
hence $abs\ ((C\ l\ w\ h\ g\ Y\ X)\ /\ (K\ k\ l\ w\ g\ Y\ X) - \varrho) \leq 1/4$ **using** *majCKϱ2* **by** *auto*
hence $abs\ (\varrho - (C\ l\ w\ h\ g\ Y\ X)\ /\ (K\ k\ l\ w\ g\ Y\ X)) \leq 1/4$ **by** *argo*
hence $Y\ dvd\ 2*nat\ X\ choose\ nat\ X$ **using** *ineq-required* **by** *simp*

then show *statement1 b Y X* **unfolding** *statement1-def*
using ‹$2 * nat\ X = nat\ (2 * X)$› ‹$b \geq 0$› *is-power2-b* **by** *auto*
qed

6.4 Proof of implication $(2a) \Longrightarrow (2)$

lemma (**in** *bridge-variables*) *theorem-II-3-2*:
assumes *b-def*:$(b::int) \geq 0$ **and** *Y-def*:$(Y::int) \geq b \wedge Y \geq 2\hat{}8$ **and** *X-def*:$(X::int) \geq 3*b$ **and** *g-def*:$(g::int) \geq 1$
shows $(statement2a\ b\ Y\ X\ g) \Longrightarrow (statement2\ b\ Y\ X\ g)$
proof −
assume *statement2a b Y X g*
then obtain $h\ k\ l\ w\ x\ y$ **where** *stat3*:$(d2a\ l\ w\ h\ x\ y\ g\ Y\ X) \wedge$

```
        (d2b k l w x g Y X)∧(d2c l w h b g Y X)∧(d2e k l w h g Y X)
        ∧ (h≥b)∧(k≥b)∧(l≥b)∧(w≥b)∧(x≥b)∧(y≥b) unfolding statement2a-def
by auto
  then have d2a-b-c:(d2a l w h x y g Y X)∧(d2b k l w x g Y X)∧(d2c l w h b g Y
X) by simp


  have W-nonzero: W w b ≠ 0
  proof −
    have W w b = 0 ⟹ False
    proof −
      assume hyp: W w b = 0
       hence (2∗A l w g Y X−5) dvd 2 using d2a-b-c unfolding d2c-def S-def
T-def by auto
      hence abs (2∗A l w g Y X − 5) ≤ 2 using dvd-imp-le-int by presburger
      hence 2∗A l w g Y X ≤ 7 ∧ 2∗A l w g Y X ≥ 3 by auto
      hence A-is-2-or-3: A l w g Y X ≤ 3 ∧ A l w g Y X ≥ 2 by auto
      have abs (A l w g Y X) = abs (U l X Y) ∗ abs (V w g Y + 1)
        unfolding A-def U-def using abs-mult by auto
      hence eq: abs (A l w g Y X) = 2∗X∗Y∗abs l ∗ abs (V w g Y +1)
          unfolding U-def L-def using abs-mult[of 2∗X l∗Y] abs-mult[of 2 X]
abs-mult[of l Y] assms
        by auto
      hence X-nonzero: X ≠ 0 using A-is-2-or-3 by auto
      have l≠0 using A-is-2-or-3 unfolding A-def U-def L-def by force
     hence several-ineq: X ≥ 1 ∧ Y ≥ 256 ∧ abs l ≥ 1 ∧ abs (V w g Y +1) ≥ 0
        using assms X-nonzero by auto
      hence 2∗X∗Y ≥ 2∗256 ∧ abs l ≥ 1 ∧ abs (V w g Y +1) ≥ 0
        using mult-mono[of 1 X 256 Y] mult-left-mono[of 256 X∗Y 2] by force
      hence (2∗X∗Y)∗abs l∗abs (V w g Y +1) ≥ (2∗256)∗abs (V w g Y +1)
        using mult-mono[of 2∗256 2∗X∗Y 1 abs l]
          mult-right-mono[of 2∗256 2∗X∗Y∗abs l abs (V w g Y +1)] by linarith
     hence ineq: 2∗X∗Y∗abs l ∗ abs (V w g Y +1) ≥ 2∗256∗abs (V w g Y +1)
        by (auto simp add: mult-mono)
       hence abs (A l w g Y X) ≥ 2∗256∗abs (V w g Y +1) using ineq eq by
presburger
        hence 3 ≥ 2∗256∗abs (V w g Y +1) ∧ abs (V w g Y +1) ≥ 0 using
A-is-2-or-3 by auto
      hence abs (V w g Y +1) = 0
         by (smt (verit, best) comm-semiring-class.distrib distrib-left distrib-right
mult-cancel-left2
          mult-le-0-iff mult-mono ring-class.ring-distribs(1) ring-class.ring-distribs(2))
      hence V w g Y +1 = 0 by auto
      hence A l w g Y X = 0 unfolding A-def by auto
      then show False using A-is-2-or-3 by auto
    qed
    then show ?thesis by auto
  qed
  hence bB0: b>0 unfolding W-def using assms by auto
```

then have *lxn0*: $l*x \neq 0$ **using** *stat3* **by** *simp*
have *0*: $(4 * g * C\ l\ w\ h\ g\ Y\ X - 4 * g * L\ l\ Y * K\ k\ l\ w\ g\ Y\ X)^2 < (K\ k\ l\ w\ g\ Y\ X)^2$
using *stat3* **unfolding** *d2e-def bB0* **by** *simp*
then have $(4 * g * C\ l\ w\ h\ g\ Y\ X - 4 * g * L\ l\ Y * K\ k\ l\ w\ g\ Y\ X)^2$
$= (2 * g)\hat{}2 *(2 * C\ l\ w\ h\ g\ Y\ X - 2 * L\ l\ Y * K\ k\ l\ w\ g\ Y\ X)^2$ **by** *algebra*
then have *1*: $(2 * g)\hat{}2 *(2 * C\ l\ w\ h\ g\ Y\ X - 2 * L\ l\ Y * K\ k\ l\ w\ g\ Y\ X)^2 < (K\ k\ l\ w\ g\ Y\ X)^2$
using *0* **by** *simp*
have $(2 * g)\hat{}2 \geq 1$ **using** *stat3 bB0* **by** (*smt* (*z3*) *g-def one-le-power*)
then have $(2 * C\ l\ w\ h\ g\ Y\ X - 2 * L\ l\ Y * K\ k\ l\ w\ g\ Y\ X)^2$
$\leq (2 * g)\hat{}2 * (2 * C\ l\ w\ h\ g\ Y\ X - 2 * L\ l\ Y * K\ k\ l\ w\ g\ Y\ X)^2$
using *mult-right-mono* **by** *fastforce*
then have $(2 * C\ l\ w\ h\ g\ Y\ X - 2 * L\ l\ Y * K\ k\ l\ w\ g\ Y\ X)^2 < (K\ k\ l\ w\ g\ Y\ X)^2$ **using** *1* **by** *simp*
then have *d2f k l w h g Y X* **unfolding** *d2f-def* **by** *simp*
then show *?thesis* **unfolding** *statement2-def* **using** *lxn0 d2a-b-c* **by** *fast*
qed

end
theory *Bridge-Theorem*
imports *Bridge-Theorem-Rev*
begin

theorem (**in** *bridge-variables*) *theorem-II*:
fixes *b Y X g* :: *int*
assumes $b \geq 0$ **and** $Y \geq b \wedge Y \geq 2\hat{}8$ **and** $X \geq 3*b$ **and** $g \geq 1$
shows *statement2 b Y X g = statement1 b Y X*
and *statement2a b Y X g = statement1 b Y X*
using *theorem-II-1-3*[*of b Y X g*] *theorem-II-3-2*[*of b Y X g*] *theorem-II-2-1*[*of b Y X g*] *assms* **by** *argo+*

definition (**in** *bridge-variables*) *bridge-relations*
where *bridge-relations k l w h x y b g Y X* ≡
is-square (*D l w h g Y X * F l w h x g Y X * I l w h x y g Y X*)
∧ *is-square* ((*U l X Y*^*4*V w g Y*^*2−4*)*J k l w g Y X*^*2+4*)
∧ *S l w g Y X dvd T l w h g Y X b*
∧ ((*4*g*(*C l w h g Y X*)−*4*g*l*Y*(*J k l w g Y X*))^*2 < (*J k l w g Y X*)^*2*)

theorem (**in** *bridge-variables*) *bridge-theorem-simplified*:
fixes *b Y X g* :: *int*
assumes $b \geq 0$ **and** $Y \geq b$ **and** $Y \geq 2\hat{}8$ **and** $X \geq 3*b$ **and** $g \geq 1$
shows (*is-power2 b* ∧ *Y dvd int* (*2 * nat X choose nat X*))
= (∃ *h k l w x y* :: *int*. *bridge-relations k l w h x y b g Y X*
∧ (*h*≥*b*)∧(*k*≥*b*)∧(*l*≥*b*)∧(*w*≥*b*)∧(*x*≥*b*)∧(*y*≥*b*))
unfolding *bridge-relations-def J-def*

using *assms theorem-II*(*2*) *statement1-def*
statement2a-def[*unfolded d2a-def d2b-def d2c-def d2e-def L-def*]
by (*auto simp*: *mult.assoc*)

end
theory *Algebra-Basics*
imports *Main* ../*Lucas-Sequences*/*Lucas-Sequences*
HOL−*Computational-Algebra.Primes Complex-Main HOL.NthRoot*
begin

7 Relation Combining

In this section the Matiyasevich-Robinson polynomial is formalized. The formal proof follows their paper [5]: first, auxiliary polynomials J_3 and Π are defined and then M_3 can be constructed from them.

7.1 Algebra Preliminaries

lemma *coercion-coherent*: *complex-of-real* (*of-rat q*) = *of-rat q*
proof −
obtain *a b* **where** *q-def*: (*a*,*b*) = *quotient-of q* **by** (*metis small-lazy'.cases*)
hence *eq1*: *real-of-rat q* = (*of-int a*) / (*of-int b*) ∧ *of-rat q* = (*of-int a*) / (*of-int b*)
by (*metis of-rat-divide of-rat-of-int-eq quotient-of-div*)
hence *eq*: *of-real* (*of-rat q*) = *of-real* ((*of-int a*) / (*of-int b*))
by *force*
have *b* ≠ *0* **using** *q-def* **by** (*smt* (*verit*, *best*) *quotient-of-denom-pos*)
hence *real-of-int b* ≠ *0* **using** *of-int-eq-0-iff* **by** *blast*
hence *complex-of-real* (*of-rat q*) = (*of-real* (*of-int a*)) / (*of-real* (*of-int b*)) **using** *eq*
nonzero-of-real-divide[*of of-int b of-int a*] **by** *force*
hence *complex-of-real* (*of-rat q*) = (*of-int a*) / (*of-int b*) **by** *simp*
thus *?thesis* **using** *eq1* **by** *metis*
qed

definition *C*::*complex set* **where** *C* = {*x*. *True*}
definition *Q*::*complex set* **where** *Q* = {*x*. ∃ *q*::*rat*. *x* = *of-rat q*}
definition *Qi*::*complex set* **where** *Qi* = {*x*. *Re x* ∈ *Q* ∧ *Im x* ∈ *Q*}

definition *cpx-sqrt*:: *int* ⇒ *complex* **where**
cpx-sqrt n = (*if* (*n* ≥ *0*) *then* (*sqrt n*) *else* (i ∗ *sqrt* (−*n*)))

lemma *norm-cpx-sqrt*: *norm* (*cpx-sqrt x*) = *sqrt* (*norm x*)
unfolding *cpx-sqrt-def* **by** (*simp add*: *cmod-def*)

lemma *square-sqrt*: (*cpx-sqrt n*) ^*2* = *n*
proof (*cases n*≥*0*)
case *True*

```
  then show ?thesis unfolding cpx-sqrt-def apply simp
    by (metis of-int-nonneg of-real-of-int-eq of-real-power real-sqrt-pow2)
next
  case False
  then show ?thesis unfolding cpx-sqrt-def apply simp
     by (smt (verit, del-insts) i-squared i-times-eq-iff mult.assoc mult-minus-right
of-int-0-le-iff
      of-real-minus of-real-of-int-eq of-real-sqrt power2-csqrt power2-i power-mult-distrib)
qed


fun field:: int list ⇒ complex set where
field [] = Q |
field (a # l) = {x. ∃ q∈(field l). ∃ r∈(field l). x = q + r∗cpx-sqrt a}

lemma Qi-is-m1: Qi = field [−1]
proof −
  have x ∈ Qi ⟹ x ∈ field [−1] for x
  proof −
    assume hyp: x ∈ Qi
    have ecr-x: x = Re x + Im x ∗ i by (simp add: complex-eq mult.commute)
    have Re x ∈ field [] ∧ Im x ∈ field [] using Qi-def hyp by force
    thus x ∈ field [−1] using cpx-sqrt-def ecr-x by fastforce
  qed
  hence inc1: Qi ⊆ field [−1] by blast
  have x ∈ field [−1] ⟹ x ∈ Qi for x
  proof −
    assume hyp: x ∈ field [−1]
    then obtain q r where x-def1: x = q + r ∗ cpx-sqrt (−1) ∧ q ∈ Q ∧ r ∈ Q
by auto
    hence x-def: x = q + r∗i ∧ q ∈ Q ∧ r ∈ Q using cpx-sqrt-def by fastforce
    hence ReIm-x: Re x = Re q − Im r ∧ Im x = Im q + Re r by simp
     have ReIm-rat: ⋀q. Im (of-rat q) = 0 ∧ complex-of-real (Re (of-rat q)) =
of-rat q
    proof −
      fix q::rat
      obtain a b where q-def: (a,b) = quotient-of q by (metis small-lazy'.cases)
      hence Im (of-rat q) = Im (of-int a / (of-int b))
        by (metis of-rat-divide of-rat-of-int-eq quotient-of-div)
      hence Im (of-rat q) = Im (of-int a) / (of-int b) by simp
      hence Im (of-rat q) = 0 by simp
      thus Im (of-rat q) = 0 ∧ complex-of-real (Re (of-rat q)) = of-rat q
        by (simp add: complex-is-Real-iff)
    qed
    have ⋀q. q ∈ Q ⟹ Im q = 0 ∧ complex-of-real (Re q) = q
    proof −
      fix q::complex
      assume hypo: q ∈ Q
      then obtain r where q = of-rat r using Q-def by force
```

```
    thus Im q = 0 ∧ complex-of-real (Re q) = q using ReIm-rat by simp
   qed
   hence Re x = q ∧ Im x = r using ReIm-x x-def1 by fastforce
   thus x ∈ Qi using Qi-def x-def1 by blast
  qed
  thus ?thesis using inc1 by fastforce
qed

lemma Zero-in-field: 0 ∈ field l
proof (induction l)
  case Nil
  have 0 = of-rat 0 by simp
  then show ?case using Q-def by force
next
  case (Cons a l)
  note t = this
  have 0 = 0 + 0 * cpx-sqrt a by simp
  then show ?case using t by force
qed

lemma field-incr: field l ⊆ field (a#l)
proof −
  have x ∈ field l ⟹ x ∈ field (a#l) for x
  proof −
   assume hyp: x ∈ field l
   have x = x + 0 * cpx-sqrt a by simp
   thus x ∈ field (a#l) using Zero-in-field hyp by force
  qed
  thus ?thesis by blast
qed

lemma int-in-field: ⋀x::int. of-int x ∈ field l
proof (induction l)
  case Nil
  fix x::int
  have x = rat-of-int x by auto
  then show of-int x ∈ field [] using Q-def by fastforce
next
  case (Cons a l)
  then show ?case using field-incr by blast
qed

lemma field-add-mult: ⋀x y. x ∈ field l ∧ y ∈ field l ⟹ (x + y ∈ field l ∧ x*y
∈ field l)
proof (induction l)
  case Nil
  assume hyp: x ∈ field [] ∧ y ∈ field []
  then obtain q r where x = of-rat q ∧ y = of-rat r using Q-def by force
   hence x + y = of-rat (q+r) ∧ x*y = of-rat (q*r) by (simp add: of-rat-add
```

of-rat-mult)
then show $x + y \in$ *field* [] $\wedge$ $x*y \in$ *field* [] **using** *Q-def* **by** *force*
next
case (*Cons a l*)
note $t =$ *this*
assume *hyp*: $x \in$ *field* $(a\#l) \wedge y \in$ *field* $(a\#l)$
have *a-in-field*: $a \in$ *field* l **using** *int-in-field* **by** *auto*
obtain *x1 x2* **where** *x-def*: $x =$ *x1* $+$ *x2* $*$ *cpx-sqrt a* $\wedge$ *x1* $\in$ *field l* $\wedge$ *x2* $\in$ *field l* **using** *hyp* **by** *force*
obtain *y1 y2* **where** *y-def*: $y =$ *y1* $+$ *y2* $*$ *cpx-sqrt a* $\wedge$ *y1* $\in$ *field l* $\wedge$ *y2* $\in$ *field l* **using** *hyp* **by** *force*
have *in-field*: *x1* $+$ *y1* $\in$ *field l* $\wedge$ *x2* $+$ *y2* $\in$ *field l* $\wedge$ *x1*$*$*y1* $+$ *a*$*$*x2*$*$*y2* $\in$ *field l* $\wedge$ *x1*$*$*y2* $+$ *x2*$*$*y1* $\in$ *field l*
using *x-def y-def a-in-field* **by** (*auto simp add*: *t*(*1*))
have $x + y =$ (*x1* $+$ *y1*) $+$ (*x2* $+$ *y2*)$*$*cpx-sqrt a* $\wedge$ $x*y =$ (*x1*$*$*y1* $+$ *a*$*$*x2*$*$*y2*) $+$ (*x1*$*$*y2* $+$ *x2*$*$*y1*)$*$*cpx-sqrt a*
using *power2-eq-square*[*of cpx-sqrt a*] *square-sqrt*[*of a*] *x-def y-def* **by** *algebra*
then show $x + y \in$ *field* $(a\#l) \wedge x*y \in$ *field* $(a\#l)$ **using** *in-field* **by** *force*
qed

lemma *field-square*: $x \in$ *field* $l \Longrightarrow x\hat{}2 \in$ *field* l **using** *power2-eq-square field-add-mult* **by** *metis*

lemma *field-opp*: $x \in$ *field* $l \Longrightarrow -x \in$ *field* l
proof $-$
assume *hyp*: $x \in$ *field* l
have $-1 =$ *of-int* (-1) **by** *auto*
hence $-1 \in$ *field* $l \wedge -x = (-1)*x$ **using** *int-in-field* **by** (*smt* (*verit*, *best*) *mult-minus1*)
thus $-x \in$ *field* l **using** *hyp field-add-mult* **by** *force*
qed

lemma *field-inv-div*: $\bigwedge x\ y.\ x \in$ *field* $l \wedge y \in$ *field* $l \Longrightarrow$ ($1/x \in$ *field* $l \wedge y/x \in$ *field* l)
proof (*induction l*)
case *Nil*
assume *hyp*: $x \in$ *field* [] $\wedge$ $y \in$ *field* []
then obtain q **where** *q-def*: $x =$ *of-rat* q **using** *Q-def* **by** *force*
hence *inverse* $x =$ *of-rat* (*inverse* q)
using *of-rat-inverse*[*of q*] **by** (*smt* (*verit*, *best*))
hence $1/x =$ *of-rat* $(1/q)$ **by** (*simp add*: *inverse-eq-divide*)
hence *inv*: $1/x \in Q$ **using** *Q-def* **by** *force*
have $y/x = y*(1/x)$ **by** *auto*
then show $1/x \in$ *field* [] $\wedge$ $y/x \in$ *field* [] **using** *hyp inv field-add-mult* **by** *fastforce*
next
case (*Cons a l*)
note *HR* $=$ *this*
assume *hyp*: $x \in$ *field* $(a\#l) \wedge y \in$ *field* $(a\#l)$

then obtain $q\ r$ **where** *x-def*: $x = q + r * \mathit{cpx\text{-}sqrt}\ a \wedge q \in \mathit{field}\ l \wedge r \in \mathit{field}\ l$ **by** *force*
then show $1/x \in \mathit{field}\ (a\#l) \wedge y/x \in \mathit{field}\ (a\#l)$
proof (*cases* $q - r * \mathit{cpx\text{-}sqrt}\ a = 0$)
case *True*
note $t = \mathit{this}$
then show *?thesis*
proof (*cases* $r = 0$)
case *True*
hence $q = 0 \wedge r = 0$ **using** t **by** *simp*
hence $x = 0$ **using** *x-def* **by** *simp*
then show *?thesis* **using** *hyp* **by** *presburger*
next
case *False*
hence $\mathit{cpx\text{-}sqrt}\ a = q/r$ **using** t **by** *auto*
hence $\mathit{cpx\text{-}sqrt}\ a \in \mathit{field}\ l$ **using** *HR*(*1*)[*of r q*] *x-def* **by** *force*
hence $x \in \mathit{field}\ l$ **using** *x-def field-add-mult* **by** *auto*
hence $1/x \in \mathit{field}\ l$ **using** *HR* **by** *force*
hence $1/x \in \mathit{field}\ (a\#l)$ **using** *field-incr* **by** *blast*
hence $1/x \in \mathit{field}\ (a\#l) \wedge y/x = y*(1/x)$ **by** *simp*
then show *?thesis* **using** *hyp field-add-mult* **by** *metis*
qed
next
case *False*
note $t = \mathit{this}$
have *a-in-field*: $a \in \mathit{field}\ l$ **using** *int-in-field* **by** *auto*
have *calc*: $(q + r * \mathit{cpx\text{-}sqrt}\ a) * (q - r * \mathit{cpx\text{-}sqrt}\ a) = q*q - a*r*r$
using *power2-eq-square*[*of cpx-sqrt a*] *square-sqrt*[*of a*] **by** (*auto simp add*: *algebra-simps*)
have $x = ((q + r * \mathit{cpx\text{-}sqrt}\ a) * (q - r * \mathit{cpx\text{-}sqrt}\ a))/(q - r* \mathit{cpx\text{-}sqrt}\ a)$ **using** *divide-self t*
by (*simp add*: *x-def*)
hence $1/x = (q - r * \mathit{cpx\text{-}sqrt}\ a)/(q*q-a*r*r)$ **using** *calc* **by** *auto*
hence *ecrx*: $1/x = q/(q*q-a*r*r) + (-r)/(q*q-a*r*r)*\mathit{cpx\text{-}sqrt}\ a$
by (*metis* (*no-types*, *lifting*) *ab-group-add-class.ab-diff-conv-add-uminus add-divide-eq-if-simps*(*2*) *eq-divide-eq mult-minus-left times-divide-eq-left*)
have $q*q-a*r*r \in \mathit{field}\ l \wedge -r \in \mathit{field}\ l$ **using** *a-in-field x-def field-add-mult field-opp*
apply *simp* **by** (*metis ab-group-add-class.ab-diff-conv-add-uminus*)
hence $q/(q*q-a*r*r) \in \mathit{field}\ l \wedge (-r)/(q*q-a*r*r) \in \mathit{field}\ l$ **using** *x-def HR*(*1*) **by** *blast*
hence $1/x \in \mathit{field}\ (a\#l) \wedge y/x = y*(1/x)$ **using** *ecrx*
by (*metis* (*mono-tags*, *lifting*) *field.simps*(*2*) *mem-Collect-eq mult.right-neutral times-divide-eq-right*)
thus $1/x \in \mathit{field}\ (a\#l) \wedge y/x \in \mathit{field}\ (a\#l)$ **using** *field-add-mult hyp* **by** *presburger*
qed
qed

lemma *field-sum*: $(\forall x{\in}S.\ f\ x{\in}\mathit{field}\ l) \Longrightarrow \mathit{finite}\ S \Longrightarrow (\sum x{\in}S.\ f\ x) \in \mathit{field}\ l$
proof (*induction card S arbitrary*:*S*)
case *0*
then show *?case* **using** *Zero-in-field* **by** *simp*
next
case (*Suc n*)
assume *assm*: $(\forall x{\in}S.\ f\ x{\in}\mathit{field}\ l)$
obtain *y* **where** *y-prop*: $y{\in}S$ **using** *Suc* **by** *fastforce*
define S' **where** $S' = S - \{y\}$
hence *card-S'*:*card* $S' = n$ **using** *Suc* **by** (*simp add*: *y-prop*)
have *disj-un-S*: $S = S'{\cup}\{y\} \wedge S'{\cap}\{y\}{=}\{\}{\wedge}\mathit{finite}\ S$ **using** *y-prop S'-def Suc* **by** *force*
hence $(\sum x{\in}S.\ f\ x) = (\sum x{\in}S'.\ f\ x) + f\ y$
by (*simp add*: *S'-def add.commute sum.remove y-prop*)
also have *...* $\in \mathit{field}\ l$ **using** *Suc*(*1*)[*of S'*] *card-S' assm field-add-mult disj-un-S* **by** *blast*
finally show $(\sum x{\in}S.\ f\ x) \in \mathit{field}\ l$ **by** *simp*
qed

lemma *field-comm*: $\mathit{field}\ (a\#b\#l) = \mathit{field}\ (b\#a\#l)$
proof −
have $x \in \mathit{field}\ (c\#d\#l) \Longrightarrow x \in \mathit{field}\ (d\#c\#l)$ **for** *c d x*
proof −
assume *hyp*: $x \in \mathit{field}\ (c\#d\#l)$
then obtain *q r* **where** *x-def*: $x = q + r * \mathit{cpx\text{-}sqrt}\ c \wedge q \in \mathit{field}\ (d\#l) \wedge r \in \mathit{field}\ (d\#l)$ **by** *force*
obtain *q1 q2* **where** *q-def*: $q = q1 + q2 * \mathit{cpx\text{-}sqrt}\ d \wedge q1 \in \mathit{field}\ l \wedge q2 \in \mathit{field}\ l$ **using** *x-def* **by** *force*
obtain *r1 r2* **where** *r-def*: $r = r1 + r2 * \mathit{cpx\text{-}sqrt}\ d \wedge r1 \in \mathit{field}\ l \wedge r2 \in \mathit{field}\ l$ **using** *x-def* **by** *force*
define *s* **where** $s = q1 + r1 * \mathit{cpx\text{-}sqrt}\ c$
define *t* **where** $t = q2 + r2 * \mathit{cpx\text{-}sqrt}\ c$
have *are-in*: $s \in \mathit{field}\ (c\#l) \wedge t \in \mathit{field}\ (c\#l)$ **unfolding** *s-def t-def* **using** *q-def r-def* **by** *force*
have $x = s + t * \mathit{cpx\text{-}sqrt}\ d$ **unfolding** *s-def t-def* **using** *x-def q-def r-def* **by** *algebra*
thus $x \in \mathit{field}\ (d\#c\#l)$ **using** *are-in* **by** *force*
qed
hence $\mathit{field}\ (c\#d\#l) \subseteq \mathit{field}\ (d\#c\#l)$ **for** *c d* **by** *blast*
thus *?thesis* **by** *blast*
qed

lemma *field-idempot1*: $\mathit{field}\ (a\#a\#l) = \mathit{field}\ (a\#l)$
proof −
have *inc1*: $\mathit{field}\ (a\#l) \subseteq \mathit{field}\ (a\#a\#l)$ **using** *field-incr* **by** *blast*
have $x \in \mathit{field}\ (a\#a\#l) \Longrightarrow x \in \mathit{field}\ (a\#l)$ **for** *x*
proof −
assume *hyp*: $x \in \mathit{field}\ (a\#a\#l)$
then obtain *q r* **where** *x-def*: $x = q + r * \mathit{cpx\text{-}sqrt}\ a \wedge q \in \mathit{field}\ (a\#l) \wedge r$

$\in$ *field* (*a*#*l*) **by** *force*
obtain *q1 q2* **where** *q-def*: *q* = *q1* + *q2* ∗ *cpx-sqrt a* ∧ *q1* ∈ *field l* ∧ *q2* ∈ *field l* **using** *x-def* **by** *force*
obtain *r1 r2* **where** *r-def*: *r* = *r1* + *r2* ∗ *cpx-sqrt a* ∧ *r1* ∈ *field l* ∧ *r2* ∈ *field l* **using** *x-def* **by** *force*
define *s* **where** *s* = *q1* + *a*∗*r2*
define *t* **where** *t* = *q2* + *r1*
have *a-in*: *a* ∈ *field l* **using** *int-in-field* **by** *force*
hence *are-in*: *s* ∈ *field* (*l*) ∧ *t* ∈ *field* (*l*) **unfolding** *s-def t-def* **using** *q-def r-def*
field-add-mult **by** *force*
have *x* = *s* + *t* ∗ *cpx-sqrt a* **unfolding** *s-def t-def* **using** *x-def q-def r-def*
power2-eq-square[*of cpx-sqrt a*] *square-sqrt*[*of a*] **by** *algebra*
thus *x* ∈ *field* (*a*#*l*) **using** *are-in* **by** *force*
qed
hence *field* (*a*#*a*#*l*) ⊆ *field* (*a*#*l*) **by** *blast*
thus *?thesis* **using** *inc1* **by** *blast*
qed

lemma *field-idempot*: *a* ∈ *set l* ⟹ *field* (*a*#*l*) = *field l*
proof (*induction l*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons b l*)
note *HR* = *this*
then show *?case*
proof (*cases a*=*b*)
case *True*
then show *?thesis* **using** *field-idempot1* **by** *force*
next
case *False*
hence *field* (*a*#*l*) = *field l* **using** *HR* **by** *simp*
hence *field* (*b*#*a*#*l*) = *field* (*b*#*l*) **by** *simp*
then show *?thesis* **using** *field-comm* **by** *simp*
qed
qed

lemma *disjoint-field-extensions-no-prime-roots*:
fixes *p-l*::*int list*
shows ⋀(*q-s*::*int set*). ((*finite q-s* ∧ *q-s* ≠ {} ∧ *q-s*∩(*set p-l*) = {} ∧ (∀ *q*∈*q-s*. *prime q*)
∧ (∀ *p*∈(*set p-l*). *prime p*)) ⟹ *prod* (λ*x*. *cpx-sqrt x*) *q-s* ∉ *field* (*p-l*@[−1]))
proof (*induction p-l*)
case *Nil*
note *t*=*this*
hence ∀ *q*∈*q-s*. (*cpx-sqrt q*) ∈ ℝ **using** *t* **unfolding** *cpx-sqrt-def* **by** (*simp add*: *prime-ge-0-int*)
hence *Im* (*prod cpx-sqrt q-s*) = *0*

using *Real-Vector-Spaces.prod-in-Reals*[*of q-s* $\lambda x.$ *cpx-sqrt x*] *complex-is-Real-iff*
by *blast*
hence *implQ*: (*prod cpx-sqrt q-s* $\notin$ *Q*) $\Longrightarrow$ (*prod* ($\lambda x.$ *cpx-sqrt x*) *q-s* $\notin$ *Qi*)
using *Qi-def complex-is-Real-iff* **by** *force*
have (*prod cpx-sqrt q-s*) ^2 = *prod* ($\lambda x.$ (*cpx-sqrt x*) ^2) *q-s*
using *prod-power-distrib* **by** *blast*
hence *eq-square*: (*prod cpx-sqrt q-s*) ^2 = *prod* ($\lambda x.$ *x*) *q-s* **using** *square-sqrt* **by**
auto
have (*prod cpx-sqrt q-s* $\in$ *Q*) $\Longrightarrow$ *False*
proof −
assume *prod cpx-sqrt q-s* $\in$ *Q*
then obtain *r* **where** *r-def*: *prod cpx-sqrt q-s* = *of-rat r* **using** *Q-def* **by** *blast*
then obtain *a b* **where** *ab-def*: (*a*,*b*) = *quotient-of r* **using** *quotient-of-def*
by (*metis eq-snd-iff*)
then have *b* ∗ *prod cpx-sqrt q-s* = *a* **using** *r-def*
by (*smt* (*verit*) *mult-of-int-commute nonzero-eq-divide-eq of-int-eq-0-iff of-rat-divide*
of-rat-of-int-eq quotient-of-denom-pos quotient-of-div)
then have *eq*: *b* ^2 ∗ *prod* ($\lambda x.$ *x*) *q-s* = *a* ^2 **using** *eq-square*
by (*metis* (*no-types*, *lifting*) *of-int-eq-iff of-int-mult of-int-power power-mult-distrib*)
obtain *q* **where** *q*∈*q-s* **using** *t* **by** *blast*
then have *q dvd prod* ($\lambda x.$ *x*) *q-s* **using** *t* **by** *blast*
then have *q dvd a* ^2 **using** *eq* **by** *algebra*
then have *qdvda*: *q dvd a* **using** *t* ‹*q* ∈ *q-s*› *prime-dvd-power* **by** *blast*
then have *q* ^2 *dvd a* ^2 **by** *force*
then have *q2dvd*: *q* ^2 *dvd b* ^2 ∗ *prod* ($\lambda x.$ *x*) *q-s* **using** *eq* **by** *simp*
have *prod* ($\lambda x.$ *x*) *q-s* = *prod* ($\lambda x.$ *x*) (*q-s* − {*q*})∗*q* **using** *t*
by (*metis* ‹*q* ∈ *q-s*› *mult.commute prod.remove*)
then have *eq2*: *q*∗*q dvd* (*b* ^2 ∗ *prod* ($\lambda x.$ *x*) (*q-s* − {*q*}))∗*q* **using** *q2dvd* **by**
algebra
have *qB1*: *q* > *1* **using** *t* ‹*q*∈*q-s*› **using** *prime-gt-1-int* **by** *blast*
then have *eq3*: *q dvd b* ^2 ∗ *prod* ($\lambda x.$ *x*) (*q-s* − {*q*}) **using** *eq2* **by** *force*
have *coprime q* (*prod* ($\lambda x.$ *x*) (*q-s* − {*q*})) **using** *t*
by (*metis DiffD1 Diff-insert-absorb* ‹*q* ∈ *q-s*› *mk-disjoint-insert primes-coprime*
prod-coprime-right)
then have *q dvd b* ^2 **using** *eq3 coprime-dvd-mult-left-iff* **by** *blast*
then have *qdvdb*:*q dvd b* **using** *t* ‹*q* ∈ *q-s*› *prime-dvd-power* **by** *blast*
then show *False* **using** *qdvda ab-def qB1* **unfolding** *quotient-of-def*
by (*metis ab-def coprime-def quotient-of-coprime zdvd-not-zless zero-less-one*)
qed
then show *?case* **using** *implQ Qi-is-m1* **by** *force*
next
case (*Cons a p-l*)
note *HR* = *this*
then show *?case*
proof (*cases a* ∈ *set p-l*)
case *True*
then show *?thesis* **using** *field-idempot HR* **by** *auto*
next
case *False*

hence $\{a\} \neq \{\} \wedge \{a\} \cap$ *set p-l* $= \{\} \wedge$ *finite* $\{a\} \wedge (\forall x{\in}\{a\}.$ *prime* $x) \wedge (\forall x{\in}$*set p-l*. *prime* $x)$
$\wedge$ *prod cpx-sqrt* $\{a\} =$ *cpx-sqrt* a **using** *HR* **by** *simp*
hence *notin*: *cpx-sqrt* $a \notin$ *field* (*p-l*@$[-1]$) **using** *HR*(1)[*of* $\{a\}$] **by** *presburger*
have *prod cpx-sqrt q-s* $\in$ *field* (a#*p-l*@$[-1]$) $\Longrightarrow$ *False*
proof $-$
assume *prod cpx-sqrt q-s* $\in$ *field* (a#*p-l*@$[-1]$)
then obtain x y **where** *xy-def*: *prod cpx-sqrt q-s* $= x + y *$ *cpx-sqrt* $a \wedge x \in$ *field* (*p-l*@$[-1]$)
$\wedge\ y \in$ *field* (*p-l*@$[-1]$) **by** *force*
have *sq1*: (*prod cpx-sqrt q-s*)^2 $=$ *prod* $(\lambda x.\ x)$ *q-s* **using** *square-sqrt*
by (*smt* (*verit*) *of-int-prod prod.cong prod-power-distrib*)
have *prod-in*: *prod* $(\lambda x.\ x)$ *q-s* $\in$ *field* (*p-l*@$[-1]$) **using** *int-in-field* **by** *blast*
have *sq2*: $(x + y *$ *cpx-sqrt* $a)$^2 $= x$^2 $+ a{*}y$^2 $+ 2{*}x{*}y{*}$*cpx-sqrt* a **using** *square-sqrt*
power2-sum[*of* x $y{*}$*cpx-sqrt* a] **by** (*simp add*: *power-mult-distrib*)
have *a-in*: $a \in$ *field* (*p-l*@$[-1]$) $\wedge\ 2 \in$ *field* (*p-l*@$[-1]$) **using** *int-in-field* **by** (*metis of-int-numeral*)
hence x^2 $+ a{*}y$^2 $\in$ *field* (*p-l*@$[-1]$) $\wedge\ 2{*}x{*}y \in$ *field* (*p-l*@$[-1]$) **using** *xy-def*
field-square[*of* x *p-l*@$[-1]$] *field-square*[*of* y *p-l*@$[-1]$] *field-add-mult* **by** *presburger*
hence *prod* $(\lambda x.\ x)$ *q-s* $- x$^2 $- a{*}y$^2 $\in$ *field* (*p-l*@$[-1]$) $\wedge\ 2{*}x{*}y \in$ *field* (*p-l*@$[-1]$)
using *prod-in field-add-mult field-opp* **apply** *simp*
by (*metis int-in-field power2-eq-square uminus-add-conv-diff xy-def*)
hence *coord-in*: (*prod* $(\lambda x.\ x)$ *q-s* $- x$^2 $- a{*}y$^2)/$(2{*}x{*}y) \in$ *field* (*p-l*@$[-1]$)

using *field-inv-div* **by** *blast*
have *prod* $(\lambda x.\ x)$ *q-s* $= x$^2 $+ a{*}y$^2 $+ 2{*}x{*}y{*}$*cpx-sqrt* a **using** *sq1 sq2 xy-def* **by** *simp*
hence $(2{*}x{*}y)/(2{*}x{*}y) *$ *cpx-sqrt* $a =$ (*prod* $(\lambda x.\ x)$ *q-s* $- x$^2 $- a{*}y$^2)/$(2{*}x{*}y)$ **by** *force*
hence *ecr-cpx*: $(2{*}x{*}y)/(2{*}x{*}y) *$ *cpx-sqrt* $a \in$ *field* (*p-l*@$[-1]$) **using** *coord-in* **by** *simp*
then show *?thesis*
proof (*cases* $2{*}x{*}y{=}0$)
case *True*
note *xy-0* $=$ *this*
then show *?thesis*
proof (*cases* $y{=}0$)
case *True*
then show *?thesis* **using** *xy-def HR*(1)[*of q-s*] *HR*(2) **by** *force*
next
case *False*
hence $x = 0$ **using** *xy-0* **by** *simp*
hence *prod cpx-sqrt q-s* $= y{*}$*cpx-sqrt* a **using** *xy-def* **by** *force*
hence *ecr1*: *prod cpx-sqrt q-s* $*$ *cpx-sqrt* $a = y * a$ **using** *square-sqrt*[*of* a] *power2-eq-square*

```
          by (metis mult.assoc)
          have q-s ∩ {a} = {} ∧ finite q-s ∧ finite {a} ∧ prod cpx-sqrt {a} =
cpx-sqrt a using HR
          by simp
         hence prod cpx-sqrt (q-s∪{a}) = y*a using ecr1 prod.union-disjoint by
metis
          hence in-field: prod cpx-sqrt (q-s∪{a}) ∈ field (p-l@[−1]) using xy-def
a-in field-add-mult by simp
        have q-s∪{a} ≠ {} ∧ finite (q-s∪{a}) ∧ (q-s∪{a})∩(set p-l) = {} ∧ (∀ z
∈ q-s∪{a}. prime z)
           using HR ‹{a} ≠ {} ∧ {a} ∩ set p-l = {} ∧ finite {a} ∧ (∀ x∈{a}.
prime x)
          ∧ (∀ x∈set p-l. prime x) ∧ prod cpx-sqrt {a} = cpx-sqrt a› by auto
       hence prod cpx-sqrt (q-s∪{a}) ∉ field (p-l@[−1]) using HR(1)[of q-s∪{a}]
HR(2) by simp
        then show ?thesis using in-field by simp
       qed
     next
       case False
       then show ?thesis using notin ecr-cpx divide-self[of 2*x*y] by simp
     qed
   qed
   then show ?thesis by (metis append-Cons)
 qed
qed

definition prime-list::int ⇒ int list
  where prime-list n = sorted-list-of-set ({p. prime p ∧ p dvd n})

lemma correct-prime-list: n ≠ 0 ⟹ set (prime-list n) = {p. prime p ∧ p dvd n}

  unfolding prime-list-def using finite-prime-divisors by simp

lemma field-prod: field ((a*b)#l) ⊆ field (a#b#l)
proof −
  have x ∈ field ((a*b)#l) ⟹ x ∈ field (a#b#l) for x
  proof −
    assume hyp: x ∈ field ((a*b)#l)
   then obtain q r where x-def: x = q + r * cpx-sqrt (a*b) ∧ q ∈ field l ∧ r ∈
field l by force
    consider (pp) a≥0 ∧ b≥0 | (pm) a≥0 ∧ b<0 | (mp) a<0 ∧ b≥0 | (mm)
a<0 ∧ b<0 by linarith
   then show ?thesis
   proof (cases)
     case pp
   hence prod-sqrt: cpx-sqrt (a*b) = cpx-sqrt a * cpx-sqrt b unfolding cpx-sqrt-def
       using real-sqrt-mult by fastforce
     have fact: r*cpx-sqrt b = 0 + r * cpx-sqrt b ∧ 0 ∈ field l ∧ r ∈ field l
       using x-def Zero-in-field by simp
```

hence $x = q + (r*cpx\text{-}sqrt\ b)*cpx\text{-}sqrt\ a \wedge q \in field\ (b\#l) \wedge r*cpx\text{-}sqrt\ b \in field\ (b\#l)$
using *x-def field-incr prod-sqrt fact* **by** (*auto*, *blast+*)
then show *?thesis* **by** *auto*
next
case *pm*
hence *prod-sqrt*: $cpx\text{-}sqrt\ (a*b) = cpx\text{-}sqrt\ a * cpx\text{-}sqrt\ b$ **unfolding** *cpx-sqrt-def*
using *real-sqrt-mult* **apply** (*simp add*: *zero-le-mult-iff*)
by (*metis mult-minus-right of-real-mult real-sqrt-mult*)
have *fact*: $r*cpx\text{-}sqrt\ b = 0 + r * cpx\text{-}sqrt\ b \wedge 0 \in field\ l \wedge r \in field\ l$
using *x-def Zero-in-field* **by** *simp*
hence $x = q + (r*cpx\text{-}sqrt\ b)*cpx\text{-}sqrt\ a \wedge q \in field\ (b\#l) \wedge r*cpx\text{-}sqrt\ b \in field\ (b\#l)$
using *x-def field-incr prod-sqrt fact* **by** (*auto*, *blast+*)
then show *?thesis* **by** *auto*
next
case *mp*
hence *prod-sqrt*: $cpx\text{-}sqrt\ (a*b) = cpx\text{-}sqrt\ a * cpx\text{-}sqrt\ b$ **unfolding** *cpx-sqrt-def*
using *real-sqrt-mult* **apply** (*simp add*: *zero-le-mult-iff*)
by (*metis mult-minus-left of-real-mult real-sqrt-mult*)
have *fact*: $r*cpx\text{-}sqrt\ b = 0 + r * cpx\text{-}sqrt\ b \wedge 0 \in field\ l \wedge r \in field\ l$
using *x-def Zero-in-field* **by** *simp*
hence $x = q + (r*cpx\text{-}sqrt\ b)*cpx\text{-}sqrt\ a \wedge q \in field\ (b\#l) \wedge r*cpx\text{-}sqrt\ b \in field\ (b\#l)$
using *x-def field-incr prod-sqrt fact* **by** (*auto*, *blast+*)
then show *?thesis* **by** *auto*
next
case *mm*
hence *prod-sqrt*: $cpx\text{-}sqrt\ (a*b) = -\ cpx\text{-}sqrt\ a * cpx\text{-}sqrt\ b$ **unfolding** *cpx-sqrt-def*
using *real-sqrt-mult* **apply** (*simp add*: *zero-le-mult-iff*)
by (*simp add*: *mult.assoc mult.left-commute real-sqrt-minus*)
have *in-field*: $y \in field\ (t) \wedge z \in field\ (t) \Longrightarrow y + z * cpx\text{-}sqrt\ a \in field\ (a\#t)$ **for** *y z t* **by** *force*
have *fact*: $-r*cpx\text{-}sqrt\ b = 0 + (-r) * cpx\text{-}sqrt\ b \wedge 0 \in field\ l \wedge r \in field\ l$
using *x-def Zero-in-field* **by** *simp*
hence $x = q + (-r*cpx\text{-}sqrt\ b)*cpx\text{-}sqrt\ a \wedge q \in field\ (b\#l) \wedge -r*cpx\text{-}sqrt\ b \in field\ (b\#l)$
using *x-def field-incr prod-sqrt field-opp*
by (*smt* (*verit*, *ccfv-threshold*) *add.right-neutral field.simps*(2) *mem-Collect-eq mult.commute*
mult.left-commute mult-minus-right mult-zero-left)
then show *?thesis* **using** *in-field*[*of q b*#*l* $-r*cpx\text{-}sqrt\ b$] **by** *presburger*
qed
qed
thus *?thesis* **by** *blast*
qed

lemma *field-add-in*: $cpx\text{-}sqrt\ a \in field\ l \Longrightarrow field\ (a\#l) = field\ l$

proof −
  assume *hyp*: *cpx-sqrt a* ∈ *field l*
  have *x* ∈ *field* (*a*#*l*) ⟹ *x* ∈ *field l* **for** *x*
  proof −
    assume *x-def*: *x* ∈ *field* (*a*#*l*)
    then obtain *p q* **where** *x* = *p* + *q*∗*cpx-sqrt a* ∧ *p* ∈ *field l* ∧ *q* ∈ *field l* **by** *force*
    thus *x* ∈ *field l* **using** *field-add-mult hyp* **by** *presburger*
  qed
  thus *?thesis* **using** *field-incr* **by** *blast*
qed

lemma *field-add-0*: *field* (*0*#*l*) = *field l*
  using *field-add-in*[*of 0 l*] *Zero-in-field cpx-sqrt-def* **by** *force*

lemma *field-remove-zeros*: ∃ *l*′::*int list*. *set l*′ = *set l* − {*0*} ∧ *field l*′ = *field l*
proof (*induction l*)
  case *Nil*
  then show *?case* **by** *simp*
next
  case (*Cons a l*)
  note *IH*=*this*
  then show *?case*
  proof (*cases a*=*0*)
    case *True*
    hence *field-l*: *field* (*a*#*l*) = *field l* **using** *field-add-0* **by** *blast*
    obtain *l*′ **where** *set l*′ = *set l* − {*0*} ∧ *field l*′ = *field l*
      using *IH* **by** *blast*
    hence *set l*′= *set* (*a*#*l*) − {*0*} ∧ *field l*′ = *field* (*a*#*l*) **using** *True field-l* **by** *simp*
    then show *?thesis* **by** *blast*
  next
    case *False*
    obtain *l*′ **where** *l*′*-prop*: *set l*′ = *set l* − {*0*} ∧ *field l*′ = *field l*
      using *IH* **by** *blast*
    hence *f*: *field* (*a*#*l*′) = *field* (*a*#*l*) **by** *simp*
    have *set* (*a*#*l*′) = *set* (*a*#*l*) − {*0*} **using** *False l*′*-prop* **by** *force*
    then show *?thesis* **using** *f* **by** *blast*
  qed
qed

lemma *sqrt-in-field*: *x* ∈ *set l* ⟹ *cpx-sqrt x* ∈ *field l*
proof (*induction l*)
  case *Nil*
  then show *?case* **by** *simp*
next
  case (*Cons a l*)
  note *t* = *this*
  then show *?case*

proof (*cases* $x = a$)
case *True*
hence *cpx-sqrt* $x = 0 + 1 * cpx\text{-}sqrt\ a \wedge 0 \in field\ l \wedge 1 \in field\ l$
using *int-in-field*[*of* 1 *l*] *Zero-in-field* **by** *force*
then show *?thesis* **by** *force*
next
case *False*
hence *cpx-sqrt* $x \in field\ l$ **using** t **by** *force*
then show *?thesis* **using** *field-incr* **by** *force*
qed
qed

lemma *field-incr2*: $field\ l \subseteq field\ m \Longrightarrow field\ (a\#l) \subseteq field\ (a\#m)$
proof −
assume *hyp*: $field\ l \subseteq field\ m$
have $x \in field\ (a\#l) \Longrightarrow x \in field\ (a\#m)$ **for** x
proof −
assume $x \in field\ (a\#l)$
then obtain $q\ r$ **where** *x-def*: $x = q + r * cpx\text{-}sqrt\ a \wedge q \in field\ l \wedge r \in field\ l$ **by** *force*
hence $q \in field\ m \wedge r \in field\ m$ **using** *hyp* **by** *force*
thus $x \in field\ (a\#m)$ **using** *x-def* **by** *force*
qed
thus $field\ (a\#l) \subseteq field\ (a\#m)$ **by** *blast*
qed

lemma *min-set-induct*: $(P{::}int\ set \Rightarrow bool)\ \{\}$
$\Longrightarrow (\bigwedge X.\ \bigwedge x.\ finite\ X \Longrightarrow x = Min\ (X \cup \{x\}) \Longrightarrow P\ X \Longrightarrow P\ (X \cup \{x\})) \Longrightarrow (\bigwedge X.\ finite\ X \Longrightarrow P\ X)$
proof −
assume *empty*: $P\ \{\}$
assume *induct*: $\bigwedge X.\ \bigwedge x.\ finite\ X \Longrightarrow x = Min\ (X \cup \{x\}) \Longrightarrow P\ X \Longrightarrow P\ (X \cup \{x\})$
have $\bigwedge X.\ card\ X = n \wedge finite\ X \Longrightarrow P\ X$ **for** n
proof (*induction* n)
case *0*
fix $X{::}int\ set$
assume $card\ X = 0 \wedge finite\ X$
hence $X = \{\}$ **using** *card-0-eq* **by** *blast*
then show $P\ X$ **using** *empty* **by** *force*
next
case (*Suc n*)
note $t = this$
fix $X{::}int\ set$
assume *hyp*: $card\ X = Suc\ n \wedge finite\ X$
define Y **where** $Y = X - \{Min\ X\}$
hence *eqX*: $X = Y \cup \{Min\ X\}$ **using** *hyp*
by (*metis Min-in Un-insert-right card-gt-0-iff insert-Diff sup-bot-right zero-less-Suc*)
hence *eqMin*: $Min\ X = Min\ (Y \cup \{Min\ X\})$ **by** *simp*

have *finY*: *finite Y* **using** *hyp* **by** (*simp add*: *Y-def*)
have *card Y* = *n* **unfolding** *Y-def* **using** *hyp*
by (*metis Min-in card-Diff-singleton-if card-gt-0-iff diff-Suc-1 zero-less-Suc*)
then show *P X* **using** *induct*[*of Y Min X*] *t*(*1*)[*of Y*] *finY eqMin eqX* **by** *simp*
qed
thus ⋀*X*. *finite X* ⟹ *P X* **by** *presburger*
qed

Sorting sets

lemma *sorting-set1*:
fixes *b*::*int* **and** *C*::*int set*
assumes ∀ *c*∈*C*. *b* < *c* **and** *finite C*
shows *sorted-list-of-set* ({*b*}∪*C*) = *b*#(*sorted-list-of-set C*)
by (*smt* (*verit*, *best*) *Diff-insert-absorb Min.union Min-in Min-singleton Un-insert-left assms*(*1*)
assms(*2*) *finite.emptyI finite.insertI insert-absorb insert-not-empty min.absorb3*
sorted-list-of-set-nonempty sup-bot-left)

lemma *sorting-set2*:
fixes *B*::*int set* **and** *C*::*int set*
assumes *finite B*
shows (∀ *b*∈*B*. ∀ *c*∈*C*. *b* < *c*) ∧ *finite B* ∧ *finite C*
⟹ *sorted-list-of-set* (*B*∪*C*) = (*sorted-list-of-set B*)@(*sorted-list-of-set C*)
proof (*induction B rule*: *min-set-induct*)
case *1*
then show *?case* **by** *simp*
next
case (*2 B x*)
note *t* = *this*
assume *hyp*: (∀ *b*∈(*B*∪{*x*}). ∀ *c*∈*C*. *b* < *c*) ∧ *finite* (*B*∪{*x*}) ∧ *finite C*
hence *finB*: *finite B* **by** *blast*
have *BlC*: ∀ *b*∈*B*. ∀ *c*∈*C*. *b* < *c* **using** *hyp* **by** *fast*
then show *?case*
proof (*cases x* ∈ *B*)
case *True*
hence *B*∪{*x*} = *B* **by** *blast*
then show *?thesis* **using** *t finB BlC* **by** *simp*
next
case *False*
hence *xlB*: ∀ *b*∈*B*. *x* < *b* **using** *t* **by** (*metis Un-insert-right eq-Min-iff insert-not-empty*
linorder-le-less-linear order-antisym-conv subset-iff subset-insertI sup-bot-right)
hence *xlBC*: ∀ *y*∈*B*∪*C*. *x* < *y* **using** *t hyp* **by** *fast*
hence *xMin*: *x* = *Min* (*B*∪{*x*}) ∧ *x* = *Min* (*B*∪{*x*}∪*C*) **using** *xlB*
by (*metis Min.union Min-in Un-empty hyp insert-not-empty min.absorb3 sup-bot-right t*(*2*))
have *B* = *B*∪{*x*} − {*x*} ∧ *B*∪*C* = *B*∪{*x*}∪*C* − {*x*} **using** *xlB xlBC* **by** *fast*
hence *sorted-list-of-set* (*B*∪{*x*}) = *x*#(*sorted-list-of-set B*)
∧ *sorted-list-of-set* (*B*∪{*x*}∪*C*) = *x*#(*sorted-list-of-set* (*B*∪*C*)) **using** *xMin*

by (*metis hyp infinite-Un insert-not-empty sorted-list-of-set-nonempty sup-eq-bot-iff*)
then show *?thesis* **using** *t* **by** *simp*
qed
next
case *3*
then show *?case* **using** *assms* **by** *simp*
qed

corollary *sorting-set3*:
fixes *B*::*int set* **and** *C*::*int set*
assumes $\forall\, b{\in}B.\ \forall\, c{\in}C.\ b < c$ **and** *finite B* **and** *finite C*
shows *sorted-list-of-set* $(B \cup C)$ = (*sorted-list-of-set B*)@(*sorted-list-of-set C*)
using *sorting-set2 assms* **by** *blast*

lemma *min-mset-induct*: (*P*::*int multiset* ⇒ *bool*) {#}
$\Longrightarrow$ ($\bigwedge X.\ \bigwedge x.\ x$ = *Min-mset* (X + {#x#}) $\Longrightarrow$ *P X* $\Longrightarrow$ *P* (X + {#x#}))
$\Longrightarrow$ ($\bigwedge X.$ *P X*)
proof −
assume *empty*: *P* {#}
assume *induct*: $\bigwedge X.\ \bigwedge x.\ x$ = *Min-mset* (X + {#x#}) $\Longrightarrow$ *P X* $\Longrightarrow$ *P* (X + {#x#})
have $\bigwedge X.$ *size X* = *n* $\Longrightarrow$ *P X* **for** *n*
proof (*induction n*)
case *0*
fix *X*::*int multiset*
assume *size X* = *0*
hence *X* = {#} **by** *blast*
then show *P X* **using** *empty* **by** *force*
next
case (*Suc n*)
note *t* = *this*
fix *X*::*int multiset*
assume *hyp*: *size X* = *Suc n*
hence *set-mset X* ≠ {} **by** *force*
hence *min-in*: *Min-mset X* ∈# *X* **by** *simp*
define *Y* **where** *Y* = *X* − {#(*Min-mset X*)#}
hence *eqX*: *X* = *Y* + {#(*Min-mset X*)#} **using** *min-in* **by** *force*
hence *eqMin*: *Min-mset X* = *Min-mset* (*Y* + {#(*Min-mset X*)#}) **by** *simp*
have *size Y* = *n* **unfolding** *Y-def* **using** *hyp eqX*
by (*metis Suc-inject min-in size-Suc-Diff1*)
then show *P X* **using** *induct*[*of Min-mset X Y*] *t*(*1*)[*of Y*] *eqMin eqX* **by** *simp*
qed
thus $\bigwedge X.$ *P X* **by** *presburger*
qed

lemma *field-prod2*: *field* ((*prod-mset A*)#*l*) ⊆ *field* ((*sorted-list-of-set* (*set-mset A*))@*l*)
proof (*induction A rule*: *min-mset-induct*)

```
  case 1
  have of-int 1 = 1 by simp
  hence one-in: 1 ∈ field l using int-in-field by metis
  have 1 = cpx-sqrt 1 unfolding cpx-sqrt-def by fastforce
  hence field (1#l) = field l using one-in field-add-in by presburger
  then show ?case by fastforce
next
  case (2 A x)
  note HR = this
  then show ?case
  proof (cases x ∈# A)
    case True
   hence eq1: sorted-list-of-set (set-mset A) = sorted-list-of-set (set-mset (add-mset
x A))
      by (simp add: insert-absorb)
    have add-eq: add-mset x A = A + {#x#} by simp
    have field ((prod-mset (add-mset x A))#l) ⊆ field (x#(prod-mset A)#l)
      using field-prod[of x prod-mset A l] by simp
     hence field ((prod-mset (add-mset x A))#l) ⊆ field (x#(sorted-list-of-set
(set-mset A))@l)
     using field-incr2[of (prod-mset A)#l (sorted-list-of-set (set-mset A))@l x] HR
by fast
   hence field ((prod-mset (add-mset x A))#l) ⊆ field ((sorted-list-of-set (set-mset
A))@l)
      using field-idempot True by simp
    then show ?thesis using eq1 add-eq by argo
  next
    case False
    hence x = Min-mset (A + {#x#}) ∧ x ∉ set-mset A using HR by linarith
    hence set-mset A = set-mset (A + {#x#}) − {x} ∧ x = Min (set-mset (A
+ {#x#})) by simp
   hence sorted-list-of-set (set-mset (A + {#x#})) = x#(sorted-list-of-set (set-mset
A))
      by (metis finite-set-mset multi-self-add-other-not-self set-mset-eq-empty-iff
        sorted-list-of-set-nonempty union-eq-empty)
   hence eq-sort: sorted-list-of-set (set-mset (A + {#x#}))@l = x#(sorted-list-of-set
(set-mset A))@l
      by force
    have prod-mset (A + {#x#}) = x ∗ prod-mset A by simp
    then show ?thesis using field-prod[of x prod-mset A l] eq-sort HR(2)
      field-incr2[of prod-mset A # l sorted-list-of-set (set-mset A) @ l x] by simp
  qed
qed

lemma field-n-in-field-prime:
  fixes n::int and l::int list
  assumes n ≠ 0
  shows field (n#l) ⊆ field ((−1)#(prime-list n)@l)
proof −
```

obtain A **where** *A-def*: *normalize* (*prod-mset* A) = *normalize* $n \wedge (\forall x{\in}\#A.$ *prime* x)
using *assms* **by** (*meson prime-factorization-exists'*)
hence *abs* (*prod-mset* A) = *abs* n **by** *simp*
then obtain s **where** *s-def*: $s \in \{1,-1\} \wedge n = s * prod\text{-}mset\ A$
by (*metis abs-eq-iff comm-monoid-mult-class.mult-1 insert-iff mult-minus1*)
hence n = *prod-mset* (*add-mset s A*) **by** *simp*
hence *field* ($n\#l$) ⊆ *field* ((*sorted-list-of-set* (*set-mset* (*add-mset s A*)))@l)
using *field-prod2* **by** *blast*
hence *field-inc*: *field* ($n\#l$) ⊆ *field* ((*sorted-list-of-set* ({s}∪(*set-mset A*)))@l)
by *auto*
have *all-fin*: *finite* {s} ∧ *finite* (*set-mset A*) **by** *fast*
have $\forall x \in\# A.\ x > s$ **using** *A-def s-def*
by (*metis abs-if-raw abs-neg-one dual-order.strict-trans ex-in-conv*
insert-iff prime-gt-0-int prime-gt-1-int)
hence $\forall x \in$ *set-mset A*. $\forall y \in \{s\}.\ x > y$ **by** *blast*
hence *sorted-list-of-set* ({s}∪(*set-mset A*))
= (*sorted-list-of-set* {s})@(*sorted-list-of-set* (*set-mset A*)) **using** *all-fin sorting-set3*
by *metis*
hence *sorted-list-of-set* ({s}∪(*set-mset A*)) = s#(*sorted-list-of-set* (*set-mset A*))
by *simp*
hence (*sorted-list-of-set* (*set-mset* (*add-mset s A*)))@l = s#(*sorted-list-of-set*
(*set-mset A*))@l
by *simp*
hence *field-inc2*: *field* ($n\#l$) ⊆ *field* (s#(*sorted-list-of-set* (*set-mset A*))@l)
using *field-inc* **by** *simp*
have *set-mset A* = {p. *prime p* ∧ *p dvd n*}
proof −
have $p \in$ *set-mset A* ⟹ $p \in \{p.$ *prime p* ∧ *p dvd n*$\}$ **for** p
proof −
assume *p-def*: $p \in$ *set-mset A*
define B **where** $B = A - \{\#p\#\}$
hence $A = B + \{\#p\#\}$ **using** *p-def* **by** *simp*
hence *prod-mset A* = *prod-mset B* ∗ *p* **by** *simp*
hence $n = s{*}prod\text{-}mset\ B * p$ **using** *s-def* **by** *simp*
hence *p dvd n* ∧ *prime p* **using** *A-def p-def* **by** *simp*
thus $p \in \{p.$ *prime p* ∧ *p dvd n*$\}$ **by** *blast*
qed
hence *inc1*: *set-mset A* ⊆ {p. *prime p* ∧ *p dvd n*} **by** *blast*
have {p. *prime p* ∧ *p dvd n*} ⊆ *set-mset A*
proof *safe*
fix p **assume** *p*: *prime p* **and** *dvd*: *p dvd n*
from *dvd* **have** *p dvd normalize n* **by** *simp*
also from *A-def* **have** *normalize n* = *normalize* (*prod-mset A*) **by** *simp*
finally have *p dvd prod-mset A*
by *simp*
thus $p \in\# A$ **using** *p A-def prime-dvd-prod-mset-primes-iff* **by** *blast*
qed
thus *?thesis* **using** *inc1* **by** *blast*

```
  qed
 hence field-inc3: field (n#l) ⊆ field (s#(prime-list n)@l) using field-inc2 prime-list-def
by simp
  thus ?thesis
  proof (cases s = -1)
    case True
    then show ?thesis using field-inc3 by simp
  next
    case False
    have of-int 1 = 1 by simp
    hence one-in: 1 ∈ field ((prime-list n)@l) using int-in-field by metis
    have sqrt-1: cpx-sqrt 1 = 1 using cpx-sqrt-def by force
    have s = 1 using False s-def by simp
    hence field (s#(prime-list n)@l) = field ((prime-list n)@l)
      using field-add-in one-in sqrt-1 by presburger
    then show ?thesis using field-incr field-inc3 by fast
  qed
qed

lemma field-n-in-field-prime2:
  fixes n::int and l::int list
  shows field (n#l) ⊆ field ((-1)#(prime-list n)@l)
proof (cases n=0)
  case True
  hence eqn: {p. prime p ∧ p dvd n} = {p. prime p} by simp
  have finite {p::int. prime p} ⟹ False
  proof -
    assume hyp: finite {p::int. prime p}
    define M where M = Max {p::int. prime p}
    hence ∀ p::int. prime p ⟶ p ≤ M using hyp by fastforce
    hence ∀ p::nat. prime p ⟶ p ≤ nat M using prime-int-nat-transfer
      by (metis dual-order.trans int-eq-iff le-nat-iff)
    hence finite {p::nat. prime p} using finite-nat-set-iff-bounded-le by auto
    then show ?thesis by (simp add: primes-infinite)
  qed
  hence infinite {p. prime p ∧ p dvd n} using eqn by presburger
  hence primen: prime-list n = [] unfolding prime-list-def by fastforce
  have cpx-sqrt n = 0 unfolding cpx-sqrt-def using True by simp
  hence field (n#l) = field l using Zero-in-field field-add-in by presburger
  thus ?thesis using field-incr primen by simp
next
  case False
  then show ?thesis using field-n-in-field-prime by simp
qed

fun prime-list-list::int list ⇒ int list where
prime-list-list [] = [] |
prime-list-list (a#l) = (prime-list a)@(prime-list-list l)
```

lemma *field-list-in-field-primes*:
fixes *l*::*int list*
shows *field* (*l*) ⊆ *field* ((*prime-list-list l*)@[−*1*])
proof (*induction l*)
case *Nil*
then show *?case* **using** *field-incr* **by** *fastforce*
next
case (*Cons a l*)
note *t* = *this*
have *field* ((−*1*)#*prime-list a* @ *prime-list-list l*@[−*1*])
= *field* (*prime-list a* @ *prime-list-list l*@[−*1*]) **using** *field-idempot* **by** *simp*
then show *?case* **using** *field-n-in-field-prime2*[*of a prime-list-list l* @ [−*1*]]
field-incr2[*of l prime-list-list l* @ [−*1*] *a*] *t* **by** *simp*
qed

Corollary

corollary *root-p-not-in-field-extension*:
fixes *B*::*int list* **and** *p*::*int*
assumes *prime p* **and** ∀ *b*∈(*set B*). ¬ *p dvd b*
shows *cpx-sqrt p* ∉ *field B*
proof −
have (∀ *d*∈(*set D*). ¬ *p dvd d*) ⟹ *p* ∉ *set* (*prime-list-list D*) **for** *D*
proof (*induction D*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a D*)
note *t* = *this*
assume *hyp*: ∀ *d*∈(*set* (*a*#*D*)). ¬ *p dvd d*
hence *hypD*: ∀ *d*∈(*set D*). ¬ *p dvd d* **by** *simp*
have *an0*: *a* ≠ *0* **using** *hyp* **by** *force*
have ¬ *p dvd a* **using** *hyp* **by** *simp*
hence *d dvd a* ⟹ ¬ *p dvd d* **for** *d* **by** *fastforce*
hence *d dvd a* ⟹ *p* ≠ *d* **for** *d* **by** *force*
hence *p* ∉ *set* (*prime-list a*) **using** *an0 correct-prime-list* **by** *blast*
then show *?case* **using** *hypD t* **by** *simp*
qed
hence *p* ∉ *set* (*prime-list-list B*) **using** *assms* **by** *simp*
hence *p-notin*: {*p*} ∩ (*set* (*prime-list-list B*)) = {} ∧ {*p*} ≠ {} ∧ *finite* {*p*} ∧
(∀ *q*∈{*p*}. *prime q*)
∧ *prod cpx-sqrt* {*p*} = *cpx-sqrt p* **using** *assms* **by** *force*
have ∀ *d*∈(*set* (*prime-list-list D*)). *prime d* **for** *D*
proof (*induction D*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a D*)
have *set* (*prime-list-list* (*a*#*D*)) = *set* (*prime-list a*) ∪ (*set* (*prime-list-list D*))
by *simp*

then show *?case* **unfolding** *prime-list-def* **by** (*metis* (*no-types*, *lifting*) *Un-iff local.Cons*
mem-Collect-eq self-append-conv2 set-append sorted-list-of-set.fold-insort-key.infinite

sorted-list-of-set.set-sorted-key-list-of-set)
qed
hence $\forall\, b{\in}(set\ (prime\text{-}list\text{-}list\ B)).\ prime\ b$ **by** *simp*
thus *?thesis*
using *disjoint-field-extensions-no-prime-roots*[*of* {*p*} *prime-list-list B*] *field-list-in-field-primes*[*of B*] *p-notin* **by** *auto*
qed

lemma *sqrt-int-smaller*:
fixes *a*::*int* **assumes** $a{\geq}0$
shows $sqrt\ a \leq a$
proof (*cases a=0*)
case *True*
then show *?thesis* **by** *force*
next
case *False*
hence $a{\geq}1$ **using** *assms* **by** *simp*
then show *?thesis* **by** (*simp add*: *power2-eq-square real-le-lsqrt*)
qed

end
theory *J3-Polynomial*
imports *Main Algebra-Basics Polynomials.More-MPoly-Type ../MPoly-Utils/More-More-MPoly-Type ../Coding/Utils*
abbrevs $pA1 = \mathcal{A}_1$
and $pA2 = \mathcal{A}_2$
and $pA3 = \mathcal{A}_3$
and $pX3 = \mathcal{X}_3$
begin

7.2 The J_3 polynomial

locale *section5-given*
begin

definition x :: *int mpoly* **where** $x \equiv Var\ 0$

definition $\mathcal{A}_1$ :: *int mpoly* **where** $\mathcal{A}_1 \equiv Var\ 1$
definition $\mathcal{A}_2$ :: *int mpoly* **where** $\mathcal{A}_2 \equiv Var\ 2$
definition $\mathcal{A}_3$ :: *int mpoly* **where** $\mathcal{A}_3 \equiv Var\ 3$
definition $\mathcal{X}_3$ :: *int mpoly* **where** $\mathcal{X}_3 \equiv Const\ 1 + \mathcal{A}_1\hat{}2 + \mathcal{A}_2\hat{}2 + \mathcal{A}_3\hat{}2$

lemmas *defs* = *x-def* $\mathcal{A}_1$*-def* $\mathcal{A}_2$*-def* $\mathcal{A}_3$*-def* $\mathcal{X}_3$*-def*

Functions on triples

fun *fst3*:: $'a \times 'a \times 'a \Rightarrow 'a$ **where** *fst3* $(a,\ b,\ c) = a$
fun *snd3*:: $'a \times 'a \times 'a \Rightarrow 'a$ **where** *snd3* $(a,\ b,\ c) = b$
fun *trd3*:: $'a \times 'a \times 'a \Rightarrow 'a$ **where** *trd3* $(a,\ b,\ c) = c$
fun *fun3*:: $'a \times 'a \times 'a \Rightarrow nat \Rightarrow 'a$::*zero* **where**
fun3 $(a,b,c)\ k = (if\ k=1\ then\ a\ else\ (if\ k=2\ then\ b\ else\ (if\ k=3\ then\ c\ else\ 0)))$

lemma *fun3-1-eq-fst3*: *fun3 a 1 = fst3 a* **by** (*metis fst3.elims fun3.simps*)
lemma *fun3-2-eq-snd3*: *fun3 a 2 = snd3 a* **by** (*metis even-plus-one-iff fun3.simps one-dvd snd3.elims*)
lemma *fun3-3-eq-trd3*: *fun3 a 3 = trd3 a*
by (*metis Suc-eq-plus1 add-cancel-right-left eval-nat-numeral*(*3*) *even-plus-one-iff fun3.simps*
numeral-eq-Suc trd3.elims zero-eq-add-iff-both-eq-0)

lemmas *fun3-def = fun3-1-eq-fst3 fun3-2-eq-snd3 fun3-3-eq-trd3*

definition *J3* :: *int mpoly* **where**
J3 $= ((x\hat{}2+\mathcal{A}_1+\mathcal{A}_2*\mathcal{X}_3\hat{}2-\mathcal{A}_3*\mathcal{X}_3\hat{}4)\hat{}2 + 4*x\hat{}2*\mathcal{A}_1 - 4*x\hat{}2*\mathcal{A}_2*\mathcal{X}_3\hat{}2 - 4*\mathcal{A}_1*\mathcal{A}_2*\mathcal{X}_3\hat{}2)\hat{}2$
$- \mathcal{A}_1 * ((4*x*(x\hat{}2+\mathcal{A}_1+\mathcal{A}_2*\mathcal{X}_3\hat{}2 - \mathcal{A}_3*\mathcal{X}_3\hat{}4) - 8*x*\mathcal{A}_2*\mathcal{X}_3\hat{}2))\hat{}2$

definition *r* **where** *r = MPoly-Type.degree J3 0*

lemma *J3-vars*: *vars J3* $\subseteq \{0,\ 1,\ 2,\ 3\}$
unfolding *J3-def defs diff-conv-add-uminus*
by *mpoly-vars*

Key lemma about J3

definition $\mathcal{E} \equiv \{-1,\ 1::int\}\times\{-1,\ 1::int\}\times\{-1,\ 1::int\}$

lemma *J3-fonction-eq-polynomial*:
fixes *f*::*nat*⇒*int*
defines *X* $\equiv 1 + (f\ 1)\hat{}2 + (f\ 2)\hat{}2 + (f\ 3)\hat{}2$
shows *of-int* (*insertion f J3*) =
$(\prod \varepsilon\in\mathcal{E}.$ *of-int* $(f\ 0)$
+ *fst3* ε $*$ *cpx-sqrt*$(f\ 1)$
+ *snd3* ε $*$ *cpx-sqrt*$(f\ 2)$ $*$ *of-int X*
+ *trd3* ε $*$ *cpx-sqrt*$(f\ 3)$ $*$ *of-int* $(X\hat{}2))$
proof −
define *inser* **where** *inser = insertion f*
have *inser-X*: *inser* $\mathcal{X}_3 = X$
unfolding *inser-def X-def defs power2-eq-square* **by** *simp*
have $\mathcal{E} = \{(-1,-1,-1), (-1,-1,1), (-1,1,-1), (-1,1,1), (1,-1,-1), (1,-1,1), (1,1,-1), (1,1,1)\}$
unfolding $\mathcal{E}$*-def* **by** *force*
hence $(\prod \varepsilon\in\mathcal{E}.$
(*of-int*$(f\ 0)$ + *fst3* ε $*$ *cpx-sqrt*$(f\ 1)$ + *snd3* ε $*$ *cpx-sqrt*$(f\ 2)$ $*$ *of-int X*
+ *trd3* ε $*$ *cpx-sqrt*$(f\ 3)$ $*$ *of-int* $(X\hat{}2)))$

$= (\prod \varepsilon{::}int{\times}int{\times}int{\in}\{(-1,-1,-1), (-1,-1,1), (-1,1,-1), (-1,1,1), (1,-1,-1), (1,-1,1), (1,1,-1), (1,1,1)\}.$
*(of-int(f 0) + fst3 ε * cpx-sqrt(f 1) + snd3 ε * cpx-sqrt(f 2) * of-int X + trd3 ε * cpx-sqrt(f 3) * of-int (X^2)))* **by** *argo*
also have ...
*= (of-int(f 0) − cpx-sqrt(f 1) − cpx-sqrt(f 2) * of-int X − cpx-sqrt(f 3) * of-int (X^2))*
** (of-int(f 0) − cpx-sqrt(f 1) − cpx-sqrt(f 2) * of-int X + cpx-sqrt(f 3) * of-int (X^2))*
** (of-int(f 0) − cpx-sqrt(f 1) + cpx-sqrt(f 2) * of-int X − cpx-sqrt(f 3) * of-int (X^2))*
** (of-int(f 0) − cpx-sqrt(f 1) + cpx-sqrt(f 2) * of-int X + cpx-sqrt(f 3) * of-int (X^2))*
** (of-int(f 0) + cpx-sqrt(f 1) − cpx-sqrt(f 2) * of-int X − cpx-sqrt(f 3) * of-int (X^2))*
** (of-int(f 0) + cpx-sqrt(f 1) − cpx-sqrt(f 2) * of-int X + cpx-sqrt(f 3) * of-int (X^2))*
** (of-int(f 0) + cpx-sqrt(f 1) + cpx-sqrt(f 2) * of-int X − cpx-sqrt(f 3) * of-int (X^2))*
** (of-int(f 0) + cpx-sqrt(f 1) + cpx-sqrt(f 2) * of-int X + cpx-sqrt(f 3) * of-int (X^2))*
by *simp*
also have ... *= ((of-int(f 0) − cpx-sqrt(f 1) − cpx-sqrt(f 2) * of-int X) ^2 − (cpx-sqrt(f 3) * of-int (X^2)) ^2)*
** ((of-int(f 0) − cpx-sqrt(f 1) + cpx-sqrt(f 2) * of-int X) ^2 − (cpx-sqrt(f 3) * of-int (X^2)) ^2)*
** ((of-int(f 0) + cpx-sqrt(f 1) − cpx-sqrt(f 2) * of-int X) ^2 − (cpx-sqrt(f 3) * of-int (X^2)) ^2)*
** ((of-int(f 0) + cpx-sqrt(f 1) + cpx-sqrt(f 2) * of-int X) ^2 − (cpx-sqrt(f 3) * of-int (X^2)) ^2)*
by *algebra*
also have ... *= ((of-int(f 0) − cpx-sqrt(f 1) − cpx-sqrt(f 2) * of-int X) ^2 − of-int (f 3) * of-int (X^2) ^2)*
** ((of-int(f 0) − cpx-sqrt(f 1) + cpx-sqrt(f 2) * of-int X) ^2 − of-int (f 3) * of-int (X^2) ^2)*
** ((of-int(f 0) + cpx-sqrt(f 1) − cpx-sqrt(f 2) * of-int X) ^2 − of-int (f 3) * of-int (X^2) ^2)*
** ((of-int(f 0) + cpx-sqrt(f 1) + cpx-sqrt(f 2) * of-int X) ^2 − of-int (f 3) * of-int (X^2) ^2)*
using *square-sqrt[of f 3] power-mult-distrib[of - - 2]* **by** (*smt* (*verit*, *best*))
also have ... *= ((of-int(f 0) − cpx-sqrt(f 1) − cpx-sqrt(f 2) * of-int X) ^2 − of-int (f 3 * X^4))*
** ((of-int(f 0) − cpx-sqrt(f 1) + cpx-sqrt(f 2) * of-int X) ^2 − of-int (f 3 * X^4))*
** ((of-int(f 0) + cpx-sqrt(f 1) − cpx-sqrt(f 2) * of-int X) ^2 − of-int (f 3 * X^4))*
** ((of-int(f 0) + cpx-sqrt(f 1) + cpx-sqrt(f 2) * of-int X) ^2 − of-int (f 3 * X^4))*
using *power2-eq-square of-int-mult* **by** *simp*

also have ... = (*of-int*(*f 0*)^*2* + *cpx-sqrt*(*f 1*)^*2* + *cpx-sqrt*(*f 2*)^*2* * (*of-int X*)^*2* − *f 3* * *X*^*4*
− *2* * *of-int*(*f 0*) * *cpx-sqrt*(*f 1*) − *2* * *of-int*(*f 0*) * *cpx-sqrt*(*f 2*) * *of-int X*
+ *2* * *cpx-sqrt*(*f 1*) * *cpx-sqrt*(*f 2*) * *of-int X*)
* (*of-int*(*f 0*)^*2* + *cpx-sqrt*(*f 1*)^*2* + *cpx-sqrt*(*f 2*)^*2* * (*of-int X*)^*2* − *f 3* * *X*^*4*
− *2* * *of-int*(*f 0*) * *cpx-sqrt*(*f 1*) + *2* * *of-int*(*f 0*) * *cpx-sqrt*(*f 2*) * *of-int X*
− *2* * *cpx-sqrt*(*f 1*) * *cpx-sqrt*(*f 2*) * *of-int X*)
* (*of-int*(*f 0*)^*2* + *cpx-sqrt*(*f 1*)^*2* + *cpx-sqrt*(*f 2*)^*2* * (*of-int X*)^*2* − *f 3* * *X*^*4*
+ *2* * *of-int*(*f 0*) * *cpx-sqrt*(*f 1*) − *2* * *of-int*(*f 0*) * *cpx-sqrt*(*f 2*) * *of-int X*
− *2* * *cpx-sqrt*(*f 1*) * *cpx-sqrt*(*f 2*) * *of-int X*)
* (*of-int*(*f 0*)^*2* + *cpx-sqrt*(*f 1*)^*2* + *cpx-sqrt*(*f 2*)^*2* * (*of-int X*)^*2* − *f 3* * *X*^*4*
+ *2* * *of-int*(*f 0*) * *cpx-sqrt*(*f 1*) + *2* * *of-int*(*f 0*) * *cpx-sqrt*(*f 2*) * *of-int X*
+ *2* * *cpx-sqrt*(*f 1*) * *cpx-sqrt*(*f 2*) * *of-int X*)
by *algebra*
also have ... = ((*of-int*(*f 0*)^*2* + *cpx-sqrt*(*f 1*)^*2* + *cpx-sqrt*(*f 2*)^*2* * (*of-int X*)^*2* −*f 3* * *X*^*4*
− *2* * *of-int*(*f 0*) * *cpx-sqrt*(*f 1*))^*2* − (*2* * *of-int*(*f 0*) * *cpx-sqrt*(*f 2*) * *of-int X*
− *2* * *cpx-sqrt*(*f 1*) * *cpx-sqrt*(*f 2*) * *of-int X*)^*2*)
* ((*of-int*(*f 0*)^*2* + *cpx-sqrt*(*f 1*)^*2* + *cpx-sqrt*(*f 2*)^*2* * (*of-int X*)^*2* − *f 3* * *X*^*4*
+ *2* * *of-int*(*f 0*) * *cpx-sqrt*(*f 1*))^*2* − (*2* * *of-int*(*f 0*) * *cpx-sqrt*(*f 2*) * *of-int X*
+ *2* * *cpx-sqrt*(*f 1*) * *cpx-sqrt*(*f 2*) * *of-int X*)^*2*)
by *algebra*
also have ... = (((*f 0*)^*2* + *f 1* + *f 2* * (*of-int X*)^*2* − *f 3* * *X*^*4*
− *2* * *of-int*(*f 0*) * *cpx-sqrt*(*f 1*))^*2* − (*2* * *of-int*(*f 0*) * *cpx-sqrt*(*f 2*) * *of-int X*
− *2* * *cpx-sqrt*(*f 1*) * *cpx-sqrt*(*f 2*) * *of-int X*)^*2*)
* (((*f 0*)^*2* + *f 1* + *f 2* * (*of-int X*)^*2* − *f 3* * *X*^*4*
+ *2* * *of-int*(*f 0*) * *cpx-sqrt*(*f 1*))^*2* − (*2* * *of-int*(*f 0*) * *cpx-sqrt*(*f 2*) * *of-int X*
+ *2* * *cpx-sqrt*(*f 1*) * *cpx-sqrt*(*f 2*) * *of-int X*)^*2*)
using *square-sqrt* **by** *simp*
also have ... = (*of-int*((*f 0*)^*2* + *f 1* + *f 2* * (*of-int X*)^*2* − *f 3* * *X*^*4*)^*2*
+ *4* * *of-int*(*f 0*)^*2* * *cpx-sqrt*(*f 1*)^*2*
− *4* * *of-int*(*f 0*) * *cpx-sqrt*(*f 1*) * ((*f 0*)^*2* + *f 1* + *f 2* * (*of-int X*)^*2* − *f 3* * *X*^*4*)
− *4* * *of-int*(*f 0*)^*2* * *cpx-sqrt*(*f 2*)^*2* * (*of-int X*)^*2*
− *4* * *cpx-sqrt*(*f 1*)^*2* * *cpx-sqrt*(*f 2*)^*2* * (*of-int X*)^*2*
+ *8* * *of-int*(*f 0*) * *cpx-sqrt*(*f 2*) * *of-int X* * *cpx-sqrt*(*f 1*) * *cpx-sqrt*(*f 2*) * *of-int X*)

$* (of\text{-}int((f\ 0)\ \hat{}2 + f\ 1 + f\ 2 * (of\text{-}int\ X)\ \hat{}2 - f\ 3 * X\hat{}4)\ \hat{}2$
$+ 4 * of\text{-}int(f\ 0)\ \hat{}2 * cpx\text{-}sqrt(f\ 1)\ \hat{}2$
$+ 4 * of\text{-}int(f\ 0) * cpx\text{-}sqrt(f\ 1) * ((f\ 0)\ \hat{}2 + f\ 1 + f\ 2 * (of\text{-}int\ X)\ \hat{}2 - f\ 3 * X\hat{}4)$
$- 4 * of\text{-}int(f\ 0)\ \hat{}2 * cpx\text{-}sqrt(f\ 2)\ \hat{}2 * (of\text{-}int\ X)\ \hat{}2$
$- 4 * cpx\text{-}sqrt(f\ 1)\ \hat{}2 * cpx\text{-}sqrt(f\ 2)\ \hat{}2 * (of\text{-}int\ X)\ \hat{}2$
$- 8 * of\text{-}int(f\ 0) * cpx\text{-}sqrt(f\ 2) * of\text{-}int\ X * cpx\text{-}sqrt(f\ 1) * cpx\text{-}sqrt(f\ 2) * of\text{-}int\ X)$
by *algebra*
also have $... = (((f\ 0)\ \hat{}2 + f\ 1 + f\ 2 * X\hat{}2 - f\ 3 * X\hat{}4)\ \hat{}2$
$+ 4 * (f\ 0)\ \hat{}2 * f\ 1$
$- 4 * f\ 0 * cpx\text{-}sqrt(f\ 1) * ((f\ 0)\ \hat{}2 + f\ 1 + f\ 2 * X\hat{}2 - f\ 3 * X\hat{}4)$
$- 4 * (f\ 0)\ \hat{}2 * f\ 2 * X\hat{}2$
$- 4 * f\ 1 * f\ 2 * X\hat{}2$
$+ 8 * f\ 0 * f\ 2\ * cpx\text{-}sqrt(f\ 1) * X * X)$
$* (((f\ 0)\ \hat{}2 + f\ 1 + f\ 2 * X\hat{}2 - f\ 3 * X\hat{}4)\ \hat{}2$
$+ 4 * (f\ 0)\ \hat{}2 * f\ 1$
$+ 4 * f\ 0 * cpx\text{-}sqrt(f\ 1) * ((f\ 0)\ \hat{}2 + f\ 1 + f\ 2 * X\hat{}2 - f\ 3 * X\hat{}4)$
$- 4 * (f\ 0)\ \hat{}2\ *f\ 2 * X\hat{}2$
$- 4 * f\ 1 * f\ 2 * X\hat{}2$
$- 8 * f\ 0 * cpx\text{-}sqrt(f\ 2) * X * cpx\text{-}sqrt(f\ 1) * cpx\text{-}sqrt(f\ 2) * X)$
using *square-sqrt of-int-mult power2-eq-square*[*of cpx-sqrt*(*f 2*)] **by** *force*
also have $... = (((f\ 0)\ \hat{}2 + f\ 1 + f\ 2 * X\hat{}2 - f\ 3 * X\hat{}4)\ \hat{}2 + 4 * (f\ 0)\ \hat{}2 * f\ 1$
$- 4 * f\ 0 * cpx\text{-}sqrt(f\ 1) * ((f\ 0)\ \hat{}2 + f\ 1 + f\ 2 * X\hat{}2 - f\ 3 * X\hat{}4)$
$- 4 * (f\ 0)\ \hat{}2 * f\ 2 * X\hat{}2 - 4 * f\ 1 * f\ 2 * X\hat{}2$
$+ 8 * f\ 0 * f\ 2\ * cpx\text{-}sqrt(f\ 1) * X\hat{}2)$
$* (((f\ 0)\ \hat{}2 + f\ 1 + f\ 2 * X\hat{}2 - f\ 3 * X\hat{}4)\ \hat{}2 + 4 * (f\ 0)\ \hat{}2 * f\ 1$
$+ 4 * f\ 0 * cpx\text{-}sqrt(f\ 1) * ((f\ 0)\ \hat{}2 + f\ 1 + f\ 2 * X\hat{}2 - f\ 3 * X\hat{}4)$
$- 4 * (f\ 0)\ \hat{}2 * f\ 2 * X\hat{}2 - 4 * f\ 1 * f\ 2 * X\hat{}2$
$- 8 * f\ 0 * f\ 2 * cpx\text{-}sqrt(f\ 1) * X\hat{}2)$
using *power2-eq-square square-sqrt mult.commute of-int-mult*
by (*smt* (*verit*, *del-insts*) *Groups.mult-ac*(*3*))
also have $... = (((f\ 0)\ \hat{}2 + f\ 1 + f\ 2 * X\hat{}2 - f\ 3 * X\hat{}4)\ \hat{}2 + 4 * (f\ 0)\ \hat{}2 * f\ 1$
$- 4 * (f\ 0)\ \hat{}2 * f\ 2 * X\hat{}2 - 4 * f\ 1 * f\ 2 * X\hat{}2$
$- (4 * f\ 0 * cpx\text{-}sqrt(f\ 1) * ((f\ 0)\ \hat{}2 + f\ 1 + f\ 2 * X\hat{}2 - f\ 3 * X\hat{}4)$
$- 8 * f\ 0 * f\ 2\ * cpx\text{-}sqrt(f\ 1) * X\hat{}2))$
$* (((f\ 0)\ \hat{}2 + f\ 1 + f\ 2 * X\hat{}2 - f\ 3 * X\hat{}4)\ \hat{}2 + 4 * (f\ 0)\ \hat{}2 * f\ 1$
$- 4 * (f\ 0)\ \hat{}2 * f\ 2 * X\hat{}2 - 4 * f\ 1 * f\ 2 * X\hat{}2$
$+ (4 * f\ 0 * cpx\text{-}sqrt(f\ 1) * ((f\ 0)\ \hat{}2 + f\ 1 + f\ 2 * X\hat{}2 - f\ 3 * X\hat{}4)$
$- 8 * f\ 0 * f\ 2\ * cpx\text{-}sqrt(f\ 1) * X\hat{}2))$
by (*simp add*:*algebra-simps*)
also have $... = (of\text{-}int(((f\ 0)\ \hat{}2 + f\ 1 + f\ 2 * X\hat{}2 - f\ 3 * X\hat{}4)\ \hat{}2 + 4 * (f\ 0)\ \hat{}2 * f\ 1$
$- 4 * (f\ 0)\ \hat{}2 * f\ 2 * X\hat{}2 - 4 * f\ 1 * f\ 2 * X\hat{}2))\ \hat{}2$
$- cpx\text{-}sqrt(f\ 1)\ \hat{}2 * (of\text{-}int\ (4 * f\ 0) * of\text{-}int((f\ 0)\ \hat{}2 + f\ 1 + f\ 2 * X\hat{}2 - f\ 3 * X\hat{}4)$
$- \ of\text{-}int(8 * f\ 0 * f\ 2)\ * of\text{-}int(X\hat{}2))\ \hat{}2$

by *algebra*
also have ... = $(((f\ 0)\hat{}2 + f\ 1 + f\ 2 * X\hat{}2 - f\ 3 * X\hat{}4)\hat{}2 + 4 * (f\ 0)\hat{}2 * f\ 1$
$- 4 * (f\ 0)\hat{}2 * f\ 2 * X\hat{}2 - 4 * f\ 1 * f\ 2 * X\hat{}2)\hat{}2$
$- (f\ 1) * (4 * f\ 0 * ((f\ 0)\hat{}2 + f\ 1 + f\ 2 * X\hat{}2 - f\ 3 * X\hat{}4)$
$- 8 * f\ 0 * f\ 2 * X\hat{}2)\hat{}2$
using *square-sqrt of-int-mult of-int-add* **by** *simp*
also have ... = $(((inser\ x)\hat{}2 + inser\ \mathcal{A}_1 + inser\ \mathcal{A}_2 * (inser\ \mathcal{X}_3)\hat{}2 - inser\ \mathcal{A}_3 * (inser\ \mathcal{X}_3)\hat{}4)\hat{}2 + 4 * (inser\ x)\hat{}2 * inser\ \mathcal{A}_1$
$- 4 * (inser\ x)\hat{}2 * inser\ \mathcal{A}_2 * (inser\ \mathcal{X}_3)\hat{}2 - 4 * inser\ \mathcal{A}_1 * inser\ \mathcal{A}_2 * (inser\ \mathcal{X}_3)\hat{}2)\hat{}2$
$- inser\ \mathcal{A}_1 * (4 * inser\ x * ((inser\ x)\hat{}2 + inser\ \mathcal{A}_1 + inser\ \mathcal{A}_2 * (inser\ \mathcal{X}_3)\hat{}2 - inser\ \mathcal{A}_3 * (inser\ \mathcal{X}_3)\hat{}4)$
$- 8 * inser\ x * inser\ \mathcal{A}_2 * (inser\ \mathcal{X}_3)\hat{}2)\hat{}2$
using *inser-X* **unfolding** *defs inser-def* **by** *simp*

also have ... = $((inser\ (x\hat{}2) + inser\ \mathcal{A}_1 + inser\ (\mathcal{A}_2 * \mathcal{X}_3\hat{}2) - inser\ (\mathcal{A}_3 * \mathcal{X}_3\hat{}4))\hat{}2 + 4 * inser\ (x\hat{}2) * inser\ \mathcal{A}_1$
$- 4 * inser\ (x\hat{}2) * inser\ \mathcal{A}_2 * inser\ (\mathcal{X}_3\hat{}2) - 4 * inser\ \mathcal{A}_1 * inser\ \mathcal{A}_2 * inser\ (\mathcal{X}_3\hat{}2))\hat{}2$
$- inser\ \mathcal{A}_1 * (4 * inser\ x * (inser\ (x\hat{}2) + inser\ \mathcal{A}_1 + inser\ \mathcal{A}_2 * inser\ (\mathcal{X}_3\hat{}2) - inser\ \mathcal{A}_3 * inser\ (\mathcal{X}_3\hat{}4))$
$- 8 * inser\ x * inser\ \mathcal{A}_2 * inser\ (\mathcal{X}_3\hat{}2))\hat{}2$
unfolding *inser-def power2-eq-square power4-eq-xxxx* **using** *insertion-mult*[*of f*] **by** *presburger*
also have ... = $((inser\ (x\hat{}2) + inser\ \mathcal{A}_1 + inser\ (\mathcal{A}_2 * \mathcal{X}_3\hat{}2) - inser\ (\mathcal{A}_3 * \mathcal{X}_3\hat{}4))\hat{}2 + 4 * inser\ (x\hat{}2 * \mathcal{A}_1)$
$- 4 * inser\ (x\hat{}2 * \mathcal{A}_2 * \mathcal{X}_3\hat{}2) - 4 * inser\ (\mathcal{A}_1 * \mathcal{A}_2 * \mathcal{X}_3\hat{}2))\hat{}2$
$- inser\ \mathcal{A}_1 * (4 * inser\ x * (inser\ (x\hat{}2) + inser\ \mathcal{A}_1 + inser\ (\mathcal{A}_2 * \mathcal{X}_3\hat{}2) - inser\ (\mathcal{A}_3 * \mathcal{X}_3\hat{}4))$
$- 8 * inser\ (x * \mathcal{A}_2 * \mathcal{X}_3\hat{}2))\hat{}2$
unfolding *inser-def* **using** *insertion-mult* [*of f*] **by** (*simp add*: *mult.assoc*)
also have ... = $((inser\ (x\hat{}2) + inser\ \mathcal{A}_1 + inser\ (\mathcal{A}_2 * \mathcal{X}_3\hat{}2) + inser\ (-\mathcal{A}_3 * \mathcal{X}_3\hat{}4))\hat{}2 + 4 * inser\ (x\hat{}2 * \mathcal{A}_1)$
$+ 4 * inser\ (-(x\hat{}2 * \mathcal{A}_2 * \mathcal{X}_3\hat{}2)) + 4 * inser\ (-\mathcal{A}_1 * \mathcal{A}_2 * \mathcal{X}_3\hat{}2))\hat{}2$
$+ inser\ (-\mathcal{A}_1) * (4 * inser\ x * (inser\ (x\hat{}2) + inser\ \mathcal{A}_1 + inser\ (\mathcal{A}_2 * \mathcal{X}_3\hat{}2) + inser\ (-\mathcal{A}_3 * \mathcal{X}_3\hat{}4))$
$+ 8 * inser\ (-x * \mathcal{A}_2 * \mathcal{X}_3\hat{}2))\hat{}2$
unfolding *inser-def* **by** *simp*
also have ... = $((inser\ (x\hat{}2) + inser\ \mathcal{A}_1 + inser\ (\mathcal{A}_2 * \mathcal{X}_3\hat{}2) + inser\ (-\mathcal{A}_3 * \mathcal{X}_3\hat{}4))\hat{}2 + inser\ (4 * (x\hat{}2 * \mathcal{A}_1))$
$+ inser\ (4 * (-(x\hat{}2 * \mathcal{A}_2 * \mathcal{X}_3\hat{}2))) + inser\ (4 * (-\mathcal{A}_1 * \mathcal{A}_2 * \mathcal{X}_3\hat{}2)))\hat{}2$
$+ inser\ (-\mathcal{A}_1) * (inser\ (4 * x) * (inser\ (x\hat{}2) + inser\ \mathcal{A}_1 + inser\ (\mathcal{A}_2 * \mathcal{X}_3\hat{}2) + inser\ (-\mathcal{A}_3 * \mathcal{X}_3\hat{}4))$
$+ inser\ (8 * (-x * \mathcal{A}_2 * \mathcal{X}_3\hat{}2)))\hat{}2$
unfolding *inser-def* **by** *simp*
also have ... = $((inser\ (x\hat{}2) + inser\ \mathcal{A}_1 + inser\ (\mathcal{A}_2 * \mathcal{X}_3\hat{}2) + inser\ (-\mathcal{A}_3 * \mathcal{X}_3\hat{}4))\hat{}2 + inser\ (4 * x\hat{}2 * \mathcal{A}_1)$

```
        + inser (−4 *x^2 * 𝒜2 * 𝒳3 ^2) + inser (−4 *  𝒜1 * 𝒜2 * 𝒳3 ^2)) ^2
      + inser (− 𝒜1) * (inser (4 * x) * (inser (x^2) + inser 𝒜1 + inser (𝒜2 *
𝒳3 ^2) + inser (−𝒜3 * 𝒳3 ^4))
        +  inser (−8 * x * 𝒜2 * 𝒳3 ^2)) ^2
   by (simp add: mult.assoc)
 also have ... = ((inser (x^2 + 𝒜1 + 𝒜2 * 𝒳3 ^2 + (− 𝒜3 * 𝒳3 ^4))) ^2 + inser
(4 * x^2 * 𝒜1
        + (−4 *x^2 * 𝒜2 * 𝒳3 ^2) + (−4 *  𝒜1 * 𝒜2 * 𝒳3 ^2))) ^2
      + inser (− 𝒜1) * (inser (4 * x) * inser (x^2 + 𝒜1 + 𝒜2 * 𝒳3 ^2 + (−𝒜3
* 𝒳3 ^4))
        +  inser (−8 * x * 𝒜2 * 𝒳3 ^2)) ^2
   unfolding inser-def using insertion-add[of f] by (smt (verit))
 also have ... = (inser ((x^2 + 𝒜1 + 𝒜2 * 𝒳3 ^2 − 𝒜3 * 𝒳3 ^4) ^2
        + (4 * x^2 * 𝒜1 − 4 *x^2 * 𝒜2 * 𝒳3 ^2 − 4 *  𝒜1 * 𝒜2 * 𝒳3 ^2))) ^2
      + inser (− 𝒜1) * (inser (4 * x * (x^2 + 𝒜1 + 𝒜2 * 𝒳3 ^2 − 𝒜3 * 𝒳3 ^4))
        +  inser (−8 * x * 𝒜2 * 𝒳3 ^2)) ^2
  unfolding inser-def power2-eq-square using insertion-mult[of f] insertion-add[of
f] by force
 also have ... = inser ( (((x^2+𝒜1+𝒜2*𝒳3 ^2−𝒜3*𝒳3 ^4) ^2 + (4*x^2*𝒜1 −
4*x^2*𝒜2*𝒳3 ^2 − 4*𝒜1*𝒜2*𝒳3 ^2))) ^2
      + (−𝒜1)*(4*x*(x^2+𝒜1+𝒜2*𝒳3 ^2−𝒜3*𝒳3 ^4) +  (−8*x*𝒜2*𝒳3 ^2)) ^2)
  unfolding inser-def power2-eq-square using insertion-mult[of f] insertion-add[of
f] by presburger
 also have ... = inser J3 unfolding J3-def by (simp add: add-diff-eq)
 finally show ?thesis unfolding ℰ-def inser-def by presburger
qed

lemma J3-cancels-iff:
 fixes f::nat⇒int
 defines X ≡ 1 + (f 1) ^2 + (f 2) ^2 + (f 3) ^2
 shows (insertion f J3 = 0) = (∃ε∈ℰ.
   of-int(f 0) + of-int (fst3 ε) * cpx-sqrt(f 1) + of-int (snd3 ε) * cpx-sqrt(f 2) *
of-int(X)
      + of-int (trd3 ε) * cpx-sqrt(f 3) * of-int (X^2) = 0)
proof −
 have fin: finite ℰ unfolding ℰ-def by blast
 thus ?thesis
   unfolding X-def using J3-fonction-eq-polynomial prod-zero-iff
   by (metis (no-types, lifting) of-int-0-eq-iff)
qed


lemma J3-zeros-bound:
 fixes A1 A2 A3
 defines X ≡ 1 + A1^2 + A2^2 + A3^2
 defines I ≡ X^3
 shows (∀x. insertion ((λ-. 0)(0:=x, 1:=A1, 2:=A2, 3:=A3)) J3 = 0 ⟶ x >
−I)
proof −
```

```
{
  fix x :: int
  define A :: nat ⇒ int where A ≡ (λ-. 0)(1:=A1, 2:=A2, 3:=A3)
  define f :: nat ⇒ int  where f ≡ (λ-. 0)(0:=x, 1:=A1, 2:=A2, 3:=A3)

  have X ≥ 0 unfolding X-def by simp
  have X > 0 unfolding X-def power2-eq-square
    by (smt (verit, best) mult-eq-0-iff sum-squares-gt-zero-iff)
  hence I ≥ 0 unfolding I-def by simp

  have aux0: sqrt (abs (A j)) * norm (X^(j-1))
            ≤ |A j| * norm (X^(j-1)) for j
    apply (intro mult-right-mono)
    apply (rule sqrt-int-smaller[of |A -|])
    by auto

  have sq-le: x ≤ x^2 for x::int
    apply (cases x = 0, auto)
    by (smt (verit, del-insts) mult-cancel-left mult-left-mono numeral-2-eq-2
        power2-eq-square power2-less-eq-zero-iff power-0 power-Suc)
  have aux1: |A j| ≤ (A j)^2 for j
    using sq-le[of |A j|] by auto

  assume insertion f J3 = 0
  hence of-int (insertion f J3) = 0
    by simp
  hence ∃ε∈E. 0 = of-int (f 0)
      + of-int (fst3 ε) * cpx-sqrt(f 1)
       + of-int (snd3 ε) * cpx-sqrt(f 2) * of-int (1 + (f 1)^2 + (f 2)^2 + (f
3)^2)
       + of-int (trd3 ε) * cpx-sqrt(f 3) * of-int ((1 + (f 1)^2 + (f 2)^2 + (f
3)^2)^2)
    by (subst (asm) J3-fonction-eq-polynomial E-def) (simp add: E-def)
  then obtain ε :: int×int×int where
      ε-def: ε∈E and
      ε-form: 0 = of-int (f 0) + of-int (fst3 ε) * cpx-sqrt(f 1)
          + of-int (snd3 ε) * cpx-sqrt(f 2) * of-int (1 + (f 1)^2 + (f 2)^2 + (f
3)^2)
           + of-int (trd3 ε) * cpx-sqrt(f 3) * of-int ((1 + (f 1)^2 + (f 2)^2 +
(f 3)^2)^2)
    by (auto simp only: prod-zero-iff)
  have ε-abs: ∀j∈{1,2,3}. norm (fun3 ε j) = 1 using ε-def unfolding E-def
by auto

  have x > -I
  proof (cases A1 = 0 ∧ A2 = 0 ∧ A3 = 0)
    case True
    then show ?thesis
      using ε-form unfolding f-def cpx-sqrt-def I-def
```

```
    by (simp add: ‹X > 0›)
  next
    case False
    hence X > 1 unfolding X-def power2-eq-square
      by (smt (verit) not-sum-squares-lt-zero sum-squares-gt-zero-iff)

    from ε-def ε-form have x-def:
      of-int x = - (∑j∈{1,2,3}. fun3 ε j * cpx-sqrt(A j) * X^(j-1))
      unfolding f-def A-def X-def apply (simp add: fun3-def del: One-nat-def)
    by (smt (verit, ccfv-SIG) add.commute add-0 add-diff-cancel-right' add-uminus-conv-diff)
   also have norm (...) ≤ (∑j∈{1,2,3::nat}. norm (fun3 ε j) * norm (cpx-sqrt
(A j))
                                          * norm X^(j-1))
      apply (subst norm-minus-cancel)
      apply (intro sum-norm-le)
      by (simp add: norm-mult norm-power)
    also have ... ≤ (∑j∈{1,2,3::nat}. norm (cpx-sqrt (A j)) * norm X^(j-1))
     using ε-abs by auto
    also have ... ≤ (∑j∈{1,2,3::nat}. sqrt (abs (A j)) * norm (X^(j-1)))
      by (auto simp: norm-cpx-sqrt)
    also have ... ≤ (∑j∈{1,2,3::nat}. |A j| * norm (X^(j-1)))
      apply (intro sum-mono)
      using aux0 by auto
    also have ... = (∑j∈{1,2,3::nat}. |A j| * X^(j-1))
      unfolding X-def by auto
    also have ... ≤ (∑j∈{1,2,3::nat}. (A j)^2 * X^(j-1))
    proof -
      have (∑j∈{1, 2, 3}. |A j| * X ^ (j - 1))
          ≤ (∑j∈{1,2,3::nat}. (A j)^2 * X^(j-1))
        apply (intro sum-mono mult-right-mono)
        by (auto simp: aux1 X-def)
      thus ?thesis by linarith
    qed

    also have ... < abs I
    proof -
      have ∀j. (A j)^2 < X
        unfolding A-def X-def power2-eq-square apply simp
        by (smt (verit, del-insts) not-sum-power2-lt-zero power2-eq-square)
      hence ∀j. (A (Suc j))^2 < X
        by auto
      hence (∑j=0..2. (A (Suc j))^2 * X ^ j) < I
        unfolding I-def
        using digit-repr-lt[of X power2 ∘ A ∘ Suc 2] ‹X > 1›
        by auto
      moreover have (∑j=0..2. (A (Suc j))^2 * X ^ j) = (∑j∈{1..3}. (A j)^2
* X ^ (j - 1))
        apply (simp add: sum.atLeast1-atMost-eq)
        apply (rule sum.cong)
```

```
      by auto
    moreover have {1..3} = {1::nat, 2, 3} by auto
    ultimately have (∑ j∈{1, 2, 3}. (A j)^2 * X ^ (j - 1)) < I
      by auto
    thus ?thesis
      unfolding X-def by linarith
  qed

  finally have abs x < abs I
    by auto
  then show ?thesis
    by (simp add: ‹0 ≤ I›)
 qed
}
thus ?thesis by auto
qed

declare single-numeral[simp del]

end

end
theory J3-Relations
 imports J3-Polynomial ../Coding/Utils
begin
```

7.3 Properties of the J_3 polynomial

```
context section5-given
begin
```

Helper lemmas

```
lemma cpx-sqrt-of-square:
 cpx-sqrt (k^2) = of-int (abs k)
proof −
 have k^2≥0 by simp
 hence cpx-sqrt (k^2) = sqrt (of-int (k^2)) unfolding cpx-sqrt-def by simp
 also have ... = of-int (abs k)
  by (metis of-int-abs of-int-mult of-real-of-int-eq power2-eq-square real-sqrt-abs2)
 finally show ?thesis by blast
qed

lemma sqrt-is-int-iff-square:
 (∃ k::int. cpx-sqrt n = of-int k) ⟷ (∃ a. n = a^2)
proof −
 have 0: (∃ a. n = a^2) ⟹ (∃ k::int. cpx-sqrt n = of-int k)
 proof −
  assume ∃ a. n = a^2
  then obtain a where a-def: n=a^2 by blast
```

```
    hence n≥0 by force
    hence cpx-sqrt n = of-int (abs a) using cpx-sqrt-of-square a-def by blast
    thus ∃ k::int. cpx-sqrt n = of-int k by simp
  qed
  have (∃ k::int. cpx-sqrt n = of-int k) ⟹ (∃ a. n = a^2)
  proof −
    assume ∃ k::int. cpx-sqrt n = of-int k
    then obtain k where k-def: cpx-sqrt n = of-int k by blast
    hence 1: Im (cpx-sqrt n) = 0 by simp
    have n<0 ⟹ False
    proof −
      assume assm: n<0
      hence cpx-sqrt n = i * sqrt (−n) unfolding cpx-sqrt-def by simp
      hence Im (cpx-sqrt n) ≠ 0 using assm by force
      thus False using 1 by simp
    qed
    hence n≥0 by force
    hence complex-of-real (sqrt n) = of-int k using k-def unfolding cpx-sqrt-def
by simp
    hence k^2 = n by (metis k-def of-int-eq-iff of-int-power square-sqrt)
    thus ∃ a. n = a^2 by blast
  qed
  thus ?thesis using 0 by blast
qed

abbreviation divd (infixl divd 70) where (divd) ≡ Rings.divide-class.divide

lemma square-odd-mult-prime:
  assumes b≥0
  shows ¬is-square b ⟹ ∃ p. prime p ∧ odd(multiplicity p b)
proof (cases b=0)
  case True
  then show ¬is-square b ⟹ ∃ p. prime p ∧ odd(multiplicity p b) unfolding
is-square-def by simp
next
  case False
  hence hyp: b>0 using assms by simp
  have (∀ p. prime p ⟶ even(multiplicity p b)) ⟹ is-square b
  proof −
    assume assm: (∀ p. prime p ⟶ even(multiplicity p b))
    define S where S = prime-factors b
    define a where a = (∏ p∈S. p^(multiplicity p b divd 2))
    hence a^2 = (∏ p∈S. (p^(multiplicity p b divd 2))^2) unfolding a-def
      using prod-power-distrib by blast
    also have ... = (∏ p∈S. (p^(multiplicity p b divd 2 * 2)))
      by (simp add: power-mult)
    also have ... = (∏ p∈S. (p^(multiplicity p b))) using S-def assm
      by (metis (no-types, lifting) dvd-div-mult-self in-prime-factors-imp-prime
prod.cong)
```

also have ... = *b*
using *S-def assms prime-factorization-int hyp* **by** *presburger*
finally show *?thesis* **unfolding** *is-square-def* **by** *blast*
qed
thus ¬*is-square b* ⟹ ∃ *p*. *prime p* ∧ *odd*(*multiplicity p b*) **by** *blast*
qed

lemma *square-even-multiplicity*: *prime p* ∧ *is-square a* ⟹ *even* (*multiplicity p a*)
proof −
assume *assm*: *prime p* ∧ *is-square a*
then obtain *b* **where** *a* = *b*^*2* **unfolding** *is-square-def* **by** *blast*
hence *multiplicity p a* = *2* ∗ *multiplicity p b* **using** *assm*
by (*metis mult-0-right multiplicity-zero prime-elem-multiplicity-power-distrib prime-imp-prime-elem zero-power2*)
thus *even* (*multiplicity p a*) **by** *simp*
qed

J3 correctly encodes the three squares

lemma *J3-encodes-three-squares*:
fixes *a1*::*int* **and** *a2*::*int* **and** *a3*::*int*
defines *f* ≡ (λ*y*. (λ-. *0*)(*0*:=*y*, *1*:=*a1*, *2*:=*a2*, *3*:=*a3*))
shows (*is-square a1* ∧ *is-square a2* ∧ *is-square a3*) ⟷ (∃ *y*::*int*. *insertion* (*f y*) *J3* = *0*)
proof −

have *dir-imp*: (*is-square a1* ∧ *is-square a2* ∧ *is-square a3*) ⟹ (∃ *y*::*int*. *insertion* (*f y*) *J3* = *0*)
proof −
assume *assm*: (*is-square a1* ∧ *is-square a2* ∧ *is-square a3*)
then obtain *k1*::*int* **and** *k2*::*int* **and** *k3*::*int*
where *a1-sq*: *a1* = *k1*^*2* **and** *a2-sq*: *a2* = *k2*^*2* **and** *a3-sq*: *a3* = *k3*^*2*
unfolding *is-square-def* **by** *blast*
define *X3*::*int* **where** *X3* = *1* + *a1*^*2* + *a2*^*2* + *a3*^*2*
define *y*::*int* **where** *y* = *abs k1* + *abs k2* ∗ *X3* + *abs k3* ∗ *X3*^*2*
define ε::*int*×*int*×*int* **where** ε = (−*1*, −*1*, −*1*)
have *eps*: ε∈{−*1*, *1*::*int*}×{−*1*, *1*::*int*}×{−*1*, *1*::*int*} **unfolding** *ε-def* **by** *blast*
have *X3-prop*: *X3* = *1* + (*f y 1*)^*2* + (*f y 2*)^*2* + (*f y 3*)^*2* **using** *f-def X3-def* **by** *force*
have *of-int*((*f y*) *0*) + *fst3* ε ∗ *cpx-sqrt*((*f y*) *1*) + *snd3* ε ∗ *cpx-sqrt*((*f y*) *2*) ∗ *of-int X3*
+ *trd3* ε ∗ *cpx-sqrt*((*f y*) *3*) ∗ *of-int*(*X3*^*2*)
= *of-int y* − *cpx-sqrt a1* − *cpx-sqrt a2* ∗ *of-int X3* − *cpx-sqrt a3* ∗ *of-int*(*X3*^*2*)
unfolding *ε-def f-def* **by** *simp*
also have ... = *of-int* (*abs k1* + *abs k2* ∗ *X3* + *abs k3* ∗ *X3*^*2*)
− *of-int* (*abs k1*) − *of-int* (*abs k2*) ∗ *of-int X3* − *of-int* (*abs k3*) ∗ *of-int*(*X3*^*2*)
unfolding *y-def a1-sq a2-sq a3-sq* **using** *cpx-sqrt-of-square* **by** *presburger*

also have ... = *0* **using** *of-int-add of-int-mult* **by** *simp*
finally have *insertion* (*f y*) *J3* = *0* **using** *J3-cancels-iff eps*
using *X3-prop* $\mathcal{E}$*-def* **by** *blast*
thus ∃ *y*::*int*. *insertion* (*f y*) *J3* = *0* **by** *blast*
qed

have (∃ *y*::*int*. *insertion* (*f y*) *J3* = *0*) ∧ ¬ (*is-square a1* ∧ *is-square a2* ∧ *is-square a3*) ⟹ *False*
proof −
assume *assm*: (∃ *y*::*int*. *insertion* (*f y*) *J3* = *0*) ∧ ¬ (*is-square a1* ∧ *is-square a2* ∧ *is-square a3*)
then obtain *y* **where** *J3-eq-0*: *insertion* (*f y*) *J3* = *0* **by** *blast*
define *A*::*nat*⇒*int* **where** *A* = (λ*k*. *f y k*)
define *X3* **where** *X3* ≡ *1* + (*f y 1*) ^ *2* + (*f y 2*) ^ *2* + (*f y 3*) ^ *2*
obtain ε::*int*×*int*×*int*
where *ε-prop*: ε∈{−*1*,*1*::*int*}×{−*1*,*1*::*int*}×{−*1*,*1*::*int*}
and *of-int*(*f y 0*) + *fst3* ε ∗ *cpx-sqrt*(*f y 1*) + *snd3* ε ∗ *cpx-sqrt*(*f y 2*) ∗ *of-int*(*X3*)
+ *trd3* ε ∗ *cpx-sqrt*(*f y 3*) ∗ *of-int* (*X3* ^ *2*) = *0*
using *J3-eq-0 J3-cancels-iff*[*of f y*] *X3-def* $\mathcal{E}$*-def* **by** *blast*
hence *fact-0*: *of-int y* + *fst3* ε ∗ *cpx-sqrt a1* + *snd3* ε ∗ *cpx-sqrt a2* ∗ *of-int*(*X3*)
+ *trd3* ε ∗ *cpx-sqrt a3* ∗ *of-int* (*X3* ^ *2*) = *0* **using** *f-def* **by** *force*
hence *of-int y* + *fun3* ε *1* ∗ *cpx-sqrt a1* + *fun3* ε *2* ∗ *cpx-sqrt a2* ∗ *of-int*(*X3*)
+ *fun3* ε *3* ∗ *cpx-sqrt a3* ∗ *of-int* (*X3* ^ *2*) = *0*
using *fact-0 fun3-1-eq-fst3 fun3-2-eq-snd3 fun3-3-eq-trd3* **by** *metis*
hence *0* = *of-int y* + *fun3* ε *1* ∗ *cpx-sqrt* (*A 1*) ∗ *of-int* (*X3* ^ (*1*−*1*))
+ *fun3* ε *2* ∗ *cpx-sqrt* (*A 2*) ∗ *of-int*(*X3* ^ (*2*−*1*))
+ *fun3* ε *3* ∗ *cpx-sqrt* (*A 3*) ∗ *of-int* (*X3* ^ (*3*−*1*))
unfolding *f-def A-def* **by** *fastforce*
also have ... = *of-int y* +(∑ *j*∈{*1*,*2*,*3*}. *fun3* ε *j* ∗ *cpx-sqrt*(*A j*) ∗ *of-int* (*X3* ^ (*j*−*1*)))
by *simp*
finally have *fact-0-sum*:
0 = *of-int y* +(∑ *j*∈{*1*,*2*,*3*}. *fun3* ε *j* ∗ *cpx-sqrt*(*A j*) ∗ *of-int* (*X3* ^ (*j*−*1*)))
by *blast*
have *X3-pos*: *X3*≥*1* **unfolding** *X3-def* **by** *simp*

have *abs-sqrt-Le-X3*: *n*∈{*a1*, *a2*, *a3*} ⟹ *abs*(*Im*(*cpx-sqrt n*)) ≤ *X3*−*1* **for** *n*
proof −
assume *assm*: *n*∈{*a1*, *a2*, *a3*}
have *abs*(*Im*(*cpx-sqrt n*)) ≤ *cmod* (*cpx-sqrt n*) **by** (*simp add*: *abs-Im-le-cmod*)
also have ... = *abs* (*sqrt n*)
unfolding *cpx-sqrt-def* **by** (*simp add*: *norm-mult real-sqrt-minus*)
also have ... = *sqrt*(*abs n*)
by (*metis of-int-abs real-sqrt-abs2 real-sqrt-mult*)

also have ... $\leq$ *abs n* **using** *sqrt-int-smaller*
using *abs-ge-zero* **by** *blast*
also have ... = *abs* (*sqrt* (*n^2*)) **by** *force*
also have ... $\leq$ *abs* (*n^2*) **using** *sqrt-int-smaller*
by (*smt* (*verit*, *ccfv-SIG*) *of-int-nonneg real-sqrt-ge-zero zero-le-power2*)
also have ... $\leq$ *X3−1* **unfolding** *X3-def f-def* **apply** *simp* **using** *assm* **by** *fastforce*
finally show *?thesis* **by** *simp*
qed
have *a3<0* $\Longrightarrow$ *False*
proof −
assume *assm*: *a3<0*
hence *sqrt3*:*cpx-sqrt a3* = i ∗ *sqrt* (−*a3*) **unfolding** *cpx-sqrt-def* **by** *simp*
have *sqBe1*: *abs*(*sqrt*(−*a3*)) $\geq$ *1* **using** *assm* **by** *simp*
have *0* = *Im* (*of-int y* + *fst3 ε* ∗ *cpx-sqrt a1* + *snd3 ε* ∗ *cpx-sqrt a2* ∗ *of-int*(*X3*)
+ *trd3 ε* ∗ *cpx-sqrt a3* ∗ *of-int* (*X3^2*)) **using** *fact-0* **by** *simp*
also have ... = *Im*(*fst3 ε* ∗ *cpx-sqrt a1* + *snd3 ε* ∗ *cpx-sqrt a2* ∗ *of-int*(*X3*)
+ *trd3 ε* ∗ *of-int* (*X3^2*) ∗ *cpx-sqrt a3*) **by** *force*
also have ... = *Im*(*fst3 ε* ∗ *cpx-sqrt a1* + *snd3 ε* ∗ *cpx-sqrt a2* ∗ *of-int*(*X3*))
+ *trd3 ε* ∗ *of-int* (*X3^2*) ∗ *sqrt*(−*a3*) **using** *sqrt3* **by** *force*
finally have *0*: *0* $\geq$ − (*abs*(*Im*(*fst3 ε* ∗ *cpx-sqrt a1*)) + *abs*(*Im*(*snd3 ε* ∗ *cpx-sqrt a2* ∗ *of-int*(*X3*))))
+ *abs* (*trd3 ε* ∗ *X3^2* ∗ *sqrt*(−*a3*)) **by** *force*

have *abs*(*Im*(*fst3 ε* ∗ *cpx-sqrt a1*)) + *abs*(*Im*(*snd3 ε* ∗ *cpx-sqrt a2* ∗ *of-int*(*X3*)))
= *abs*(*Im*(*cpx-sqrt a1*)) + *abs*(*Im* (*cpx-sqrt a2*)) ∗ *abs* (*of-int*(*X3*))
using *ε-prop*(*1*) *abs-mult* **by** *force*
also have ... $\leq$ *of-int* (*X3* − *1*) + *of-int* (*X3* −*1*) ∗ *abs*(*of-int X3*) **using** *abs-sqrt-Le-X3*
by (*smt* (*verit*, *best*) *insertCI mult-right-mono*)
also have ... = *X3−1* + (*X3−1*)∗*X3* **using** *X3-pos of-int-add of-int-mult* **by** *force*
also have ... = *X3^2−1* **by** *algebra*
finally have *1*: *abs*(*Im*(*fst3 ε* ∗ *cpx-sqrt a1*)) + *abs*(*Im*(*snd3 ε* ∗ *cpx-sqrt a2* ∗ *of-int*(*X3*)))
$\leq$ *X3^2−1* **by** *blast*

have *abs* (*trd3 ε* ∗ *X3^2* ∗ *sqrt*(−*a3*)) = *abs* (*of-int* (*trd3 ε*)) ∗ *abs* (*of-int* (*X3^2*)) ∗ *abs* (*sqrt*(−*a3*))
using *abs-mult of-int-mult* **by** *metis*
also have ... = *X3^2* ∗ *abs* (*sqrt*(−*a3*))
using *ε-prop*(*1*) **by** *fastforce*
also have ... $\geq$ *X3^2* **using** *sqBe1 mult-left-mono*[*of 1 abs*(*sqrt*(−*a3*)) *X3^2*] **by** *force*
finally have *0* $\geq$ −(*X3^2−1*) + *X3^2* **using** *0 1* **by** *force*
thus *False* **by** *simp*
qed

hence *a3-pos*: *a3≥0* **by** *fastforce*
have *a2<0* ⟹ *False*
proof −
assume *assm*: *a2<0*
hence *sqrt2*:*cpx-sqrt a2* = i ∗ *sqrt* (−*a2*) **unfolding** *cpx-sqrt-def* **by** *simp*
have *sqBe1*: *abs*(*sqrt*(−*a2*)) ≥ *1* **using** *assm* **by** *simp*
have *0* = *Im* (*of-int y* + *fst3 ε* ∗ *cpx-sqrt a1* + *snd3 ε* ∗ *cpx-sqrt a2* ∗ *of-int*(*X3*)
+ *trd3 ε* ∗ *cpx-sqrt a3* ∗ *of-int* (*X3*^*2*)) **using** *fact-0* **by** *simp*
also have ... = *Im*(*fst3 ε* ∗ *cpx-sqrt a1* + *snd3 ε* ∗ *cpx-sqrt a2* ∗ *of-int*(*X3*))
using *a3-pos cpx-sqrt-def* **by** *force*
also have ... = *Im*(*fst3 ε* ∗ *cpx-sqrt a1*) + *snd3 ε* ∗ *sqrt* (−*a2*) ∗ *of-int*(*X3*)
using *sqrt2* **by** *force*
finally have *0*: *0* ≥ − *abs*(*Im*(*fst3 ε* ∗ *cpx-sqrt a1*)) + *abs*(*snd3 ε* ∗ *X3* ∗ *sqrt* (−*a2*))
by (*smt* (*verit*, *del-insts*) *mult.assoc mult.commute of-int-mult*)

have *abs*(*Im*(*fst3 ε* ∗ *cpx-sqrt a1*)) = *abs*(*Im*(*cpx-sqrt a1*))
using *ε-prop*(*1*) *abs-mult* **by** *force*
also have ... ≤ *of-int* (*X3* − *1*) **using** *abs-sqrt-Le-X3*
by (*smt* (*verit*, *best*) *insertCI mult-right-mono*)
finally have *1*: *abs*(*Im*(*fst3 ε* ∗ *cpx-sqrt a1*)) ≤ *X3*−*1* **by** *blast*

have *abs* (*snd3 ε* ∗ *X3* ∗ *sqrt*(−*a2*)) = *abs* (*of-int* (*snd3 ε*)) ∗ *abs* (*of-int* (*X3*)) ∗ *abs* (*sqrt*(−*a2*))
using *abs-mult of-int-mult* **by** *metis*
also have ... = *X3* ∗ *abs* (*sqrt*(−*a2*)) **using** *ε-prop*(*1*) *X3-pos* **by** *fastforce*
also have ... ≥ *X3* **using** *sqBe1 X3-pos mult-left-mono* **by** *force*
finally have *0* ≥ −(*X3*−*1*) + *X3* **using** *0 1* **by** *force*
thus *False* **by** *simp*
qed
hence *a2-pos*: *a2≥0* **by** *fastforce*
have *a1<0* ⟹ *False*
proof −
assume *assm*: *a1<0*
hence *sqrt1*:*cpx-sqrt a1* = i ∗ *sqrt* (−*a1*) **unfolding** *cpx-sqrt-def* **by** *simp*
have *sqBe1*: *abs*(*sqrt*(−*a1*)) ≥ *1* **using** *assm* **by** *simp*
have *0* = *Im* (*of-int y* + *fst3 ε* ∗ *cpx-sqrt a1* + *snd3 ε* ∗ *cpx-sqrt a2* ∗ *of-int*(*X3*)
+ *trd3 ε* ∗ *cpx-sqrt a3* ∗ *of-int* (*X3*^*2*)) **using** *fact-0* **by** *simp*
also have ... = *Im*(*fst3 ε* ∗ *cpx-sqrt a1*)
using *a3-pos a2-pos cpx-sqrt-def* **by** *force*
also have ... = *fst3 ε* ∗ *sqrt* (−*a1*) **using** *sqrt1* **by** *simp*
finally show *False* **using** *assm ε-prop*(*1*) **by** *force*
qed
hence *a1-pos*: *a1≥0* **by** *fastforce*

define k::*nat* **where** $k = (if\ \neg is\text{-}square\ a3\ then\ 3\ else$
$(if\ \neg is\text{-}square\ a2\ then\ 2\ else\ 1))$
hence *k123*: $k \in \{1,2,3\}$ **by** *simp*
consider $(3)\ \neg is\text{-}square\ a3 \mid (2)\ is\text{-}square\ a3 \wedge \neg is\text{-}square\ a2$
$\mid (1)\ is\text{-}square\ a3 \wedge is\text{-}square\ a2$ **by** *blast*
hence *Ak-bigger-not-sq*: $\neg is\text{-}square\ (A\ k) \wedge (\forall j \in \{k+1..3\}.\ is\text{-}square\ (A\ j))$
proof (*cases*)
case *3* **then show** *?thesis* **using** *k-def A-def f-def* **by** *simp*
next
case *2* **then show** *?thesis* **using** *k-def A-def f-def* **by** *simp*
next
case *1* **then show** *?thesis* **using** *assm k-def A-def f-def* **by** *simp*
qed
have *A-pos*: $j \in \{1,2,3\} \Longrightarrow A\ j \geq 0$ **for** j
using *a1-pos a2-pos a3-pos* **unfolding** *A-def f-def* **by** *fastforce*
obtain p::*int* **where** *p-prime*: *prime p* **and** *p-mult*: $odd(multiplicity\ p\ (A\ k))$
using *square-odd-mult-prime Ak-bigger-not-sq A-pos*[*of k*] *k123* **by** *blast*
have *sqrtpA*: $sqrt(A\ j) = sqrt(p) * sqrt(A\ j/p)$ **for** j **using** *real-sqrt-mult*[*of real-of-int p*]
by (*metis mult.commute nonzero-divide-eq-eq not-prime-0 of-int-0-eq-iff p-prime*)
define *Se* **where** $Se = \{x \in \{1,2,3\}.\ even(multiplicity\ p\ (A\ x))\}$
define *So* **where** $So = \{x \in \{1,2,3\}.\ odd(multiplicity\ p\ (A\ x))\}$
have *disj-Se-So*: $Se \cap So = \{\} \wedge Se \cup So = \{1,2,3\}$ **using** *Se-def So-def* **by** *fast*
hence *eq-two-sums*: $0 = of\text{-}int\ y + (\sum j \in Se.\ fun3\ \varepsilon\ j * cpx\text{-}sqrt(A\ j) * of\text{-}int\ (X3\hat{}(j-1)))$
$+ (\sum j \in So.\ fun3\ \varepsilon\ j * cpx\text{-}sqrt(A\ j) * of\text{-}int\ (X3\hat{}(j-1)))$
using *fact-0-sum sum.union-disjoint*
by (*smt* (*verit*, *best*) *add.commute finite.emptyI finite-Un finite-insert group-cancel.add2*)
also have $... = of\text{-}int\ y + (\sum j \in Se.\ fun3\ \varepsilon\ j * complex\text{-}of\text{-}real\ (sqrt(A\ j)) * of\text{-}int\ (X3\hat{}(j-1)))$
$+ (\sum j \in So.\ fun3\ \varepsilon\ j * complex\text{-}of\text{-}real\ (sqrt(A\ j)) * of\text{-}int\ (X3\hat{}(j-1)))$
using *A-pos* **unfolding** *Se-def So-def cpx-sqrt-def* **by** *force*
also have $... = of\text{-}int\ y + (\sum j \in Se.\ fun3\ \varepsilon\ j * sqrt(A\ j) * of\text{-}int\ (X3\hat{}(j-1)))$
$+ (\sum j \in So.\ fun3\ \varepsilon\ j * sqrt(A\ j) * of\text{-}int\ (X3\hat{}(j-1)))$
using *of-real-mult of-real-add* **by** *force*
also have *eq-sum-random1*: $... = of\text{-}int\ y + (\sum j \in Se.\ fun3\ \varepsilon\ j * sqrt(A\ j) * of\text{-}int\ (X3\hat{}(j-1)))$
$+ (\sum j \in So.\ sqrt(p) * fun3\ \varepsilon\ j * sqrt(A\ j/p) * of\text{-}int\ (X3\hat{}(j-1)))$
using *sqrtpA* **by** (*simp add:algebra-simps*)
also have *eq-sum-random2*: $... = of\text{-}int\ y + (\sum j \in Se.\ fun3\ \varepsilon\ j * sqrt(A\ j) * of\text{-}int\ (X3\hat{}(j-1)))$
$+ sqrt\ (p) * (\sum j \in So.\ fun3\ \varepsilon\ j * sqrt(A\ j/p) * of\text{-}int\ (X3\hat{}(j-1)))$
using *sum-distrib-left*[*of sqrt* (*real-of-int p*) ($\lambda j.$ *real-of-int* (*fun3* ε *j*) *
sqrt (*real-of-int* (*A j*) / *real-of-int p*) * *real-of-int* (*X3* ^ (*j* − 1))) *So*]
by (*metis* (*no-types*, *lifting*) *mult.assoc sum.cong*)
finally have *fact-0-sum-real*:

$0 = \ of\text{-}int\ y + (\sum j{\in}Se.\ fun3\ \varepsilon\ j * sqrt(A\ j) * of\text{-}int\ (X3\hat{}(j-1))) + sqrt\ (p) * (\sum j{\in}So.\ fun3\ \varepsilon\ j * sqrt(A\ j/p) * of\text{-}int\ (X3\hat{}(j-1)))$
using *of-real-eq-0-iff* **by** *fastforce*

obtain *b1*::*int* **where** *b1-def*: $a1 = b1 * p\hat{}(multiplicity\ p\ a1)$
by (*metis dvd-div-mult-self multiplicity-dvd*)
obtain *b2*::*int* **where** *b2-def*: $a2 = b2 * p\hat{}(multiplicity\ p\ a2)$
by (*metis dvd-div-mult-self multiplicity-dvd*)
obtain *b3*::*int* **where** *b3-def*: $a3 = b3 * p\hat{}(multiplicity\ p\ a3)$
by (*metis dvd-div-mult-self multiplicity-dvd*)
have *b1-pos*: $b1 \geq 0$ **using** *b1-def a1-pos*
by (*smt* (*verit*) *p-prime prime-gt-0-int zero-le-mult-iff zero-less-power*)
have *b2-pos*: $b2 \geq 0$ **using** *b2-def a2-pos*
by (*smt* (*verit*) *p-prime prime-gt-0-int zero-le-mult-iff zero-less-power*)
have *b3-pos*: $b3 \geq 0$ **using** *b3-def a3-pos*
by (*smt* (*verit*) *p-prime prime-gt-0-int zero-le-mult-iff zero-less-power*)
define *B*::*nat*⇒*int* **where** $B = (\lambda\text{-}.\ 0)(1{:=}b1,\ 2{:=}b2,\ 3{:=}b3)$
hence *B-prop*: $j{\in}\{1,2,3\} \Longrightarrow A\ j = B\ j * p\hat{}(multiplicity\ p\ (A\ j))$ **for** *j*
unfolding *A-def B-def f-def* **using** *b1-def b2-def b3-def* **by** *fastforce*
have *B-pos*: $j{\in}\{1,2,3\} \Longrightarrow B\ j \geq 0$ **for** *j* **using** *B-def b1-pos b2-pos b3-pos* **by** *fastforce*
have *pnB0*:$p\hat{}n>0$ **for** *n* **by** (*simp add*: *p-prime prime-gt-0-int*)
hence *even n* $\Longrightarrow sqrt(p\hat{}n) = p\hat{}(n\ divd\ 2)$ **for** *n*
by (*simp add*: *p-prime prime-ge-0-int real-sqrt-power real-sqrt-power-even*)
hence $\forall j{\in}Se.\ sqrt(p\hat{}(multiplicity\ p\ (A\ j))) = p\hat{}((multiplicity\ p\ (A\ j))\ divd\ 2)$
using *Se-def* **by** *blast*
hence *sqrtAj*: $\forall j{\in}Se.\ sqrt\ (A\ j) = sqrt(B\ j) * p\hat{}(multiplicity\ p\ (A\ j)\ divd\ 2)$
using *B-prop real-sqrt-mult* **by** (*metis* (*no-types*, *lifting*) *Se-def mem-Collect-eq of-int-mult*)
have *sqrtBj-field*: $\forall j{\in}\{1,2,3\}.\ sqrt(B\ j) \in field([B\ 1,\ B\ 2,\ B\ 3])$
using *cpx-sqrt-def sqrt-in-field B-pos*
by (*metis insert-iff list.simps*(15) *singletonD*)
hence $\forall j{\in}Se.\ sqrt(A\ j) \in field\ [B\ 1,\ B\ 2,\ B\ 3]$
using *int-in-field field-add-mult sqrtAj*
by (*metis UnCI disj-Se-So of-real-mult of-real-of-int-eq*)
hence *Se-field*: $\forall j{\in}Se.\ fun3\ \varepsilon\ j * sqrt(A\ j) * of\text{-}int\ (X3\hat{}(j-1)) \in field\ [B\ 1,\ B\ 2,\ B\ 3]$
using *int-in-field field-add-mult* **by** (*metis of-real-mult of-real-of-int-eq*)

have *odd n* $\Longrightarrow sqrt(p\hat{}n) = sqrt\ p * p\hat{}(n\ divd\ 2)$ **for** *n*
using *pnB0* **by** (*smt* (*z3*) *Suc-eq-plus1 odd-two-times-div-two-succ of-int-nonneg of-int-power*
p-prime power.simps(2) *power-mult prime-ge-0-int real-sqrt-abs real-sqrt-power*)
hence $j{\in}So \Longrightarrow sqrt(p\hat{}(multiplicity\ p\ (A\ j))) = sqrt\ p * p\hat{}((multiplicity\ p\ (A\ j))\ divd\ 2)$ **for** *j*

using *So-def* **by** *blast*
hence $j \in So \Longrightarrow sqrt\ (A\ j) = sqrt(B\ j) * sqrt\ p * p\hat{}(multiplicity\ p\ (A\ j)\ divd\ 2)$ **for** j
using *B-prop real-sqrt-mult*
by (*metis* (*no-types*, *lifting*) *UnCI disj-Se-So mult.assoc of-int-mult*)
hence $j \in So \Longrightarrow sqrt\ (A\ j/p) = sqrt(B\ j) * p\hat{}(multiplicity\ p\ (A\ j)\ divd\ 2)$ **for** j
using *real-sqrt-mult p-prime real-sqrt-divide* **by** *force*
hence *So-field*: $j \in So \Longrightarrow sqrt(A\ j/p) \in field\ [B\ 1,\ B\ 2,\ B\ 3]$ **for** j
using *int-in-field field-add-mult sqrtBj-field*
by (*metis* (*no-types*, *lifting*) *So-def mem-Collect-eq of-real-mult of-real-of-int-eq*)
hence *So-field*: $\forall j \in So.\ fun3\ \varepsilon\ j * sqrt(A\ j/p) * of\text{-}int\ (X3\hat{}(j-1)) \in field\ [B\ 1,\ B\ 2,\ B\ 3]$
using *int-in-field field-add-mult* **by** (*metis of-real-mult of-real-of-int-eq*)

define *sum-Se* **where** $sum\text{-}Se = of\text{-}int\ y + (\sum j \in Se.\ fun3\ \varepsilon\ j * sqrt(A\ j) * of\text{-}int\ (X3\hat{}(j-1)))$
define *sum-So* **where** $sum\text{-}So = (\sum j \in So.\ fun3\ \varepsilon\ j * sqrt(A\ j/p) * of\text{-}int\ (X3\hat{}(j-1)))$
have *fact-0bis*: $sum\text{-}Se + sqrt\ p * sum\text{-}So = 0$
unfolding *sum-Se-def sum-So-def* **using** *fact-0-sum-real* **by** *presburger*
have $(\sum j \in Se.\ complex\text{-}of\text{-}real(fun3\ \varepsilon\ j * sqrt(A\ j) * of\text{-}int\ (X3\hat{}(j-1)))) \in field\ [B\ 1,\ B\ 2,\ B\ 3]$
using *Se-field field-sum*
by (*metis* (*no-types*, *lifting*) *disj-Se-So finite.emptyI finite.insertI finite-Un*)
hence $(\sum j \in Se.\ fun3\ \varepsilon\ j * sqrt(A\ j) * of\text{-}int\ (X3\hat{}(j-1))) \in field\ [B\ 1,\ B\ 2,\ B\ 3]$
using *of-real-add* **by** *force*
hence *sum-Se-field*: $sum\text{-}Se \in field\ [B\ 1,\ B\ 2,\ B\ 3]$
unfolding *sum-Se-def* **using** *int-in-field field-add-mult*
by (*metis* (*no-types*, *lifting*) *of-real-add of-real-of-int-eq*)
have $(\sum j \in So.\ complex\text{-}of\text{-}real(fun3\ \varepsilon\ j * sqrt(A\ j/p) * of\text{-}int\ (X3\hat{}(j-1)))) \in field\ [B\ 1,\ B\ 2,\ B\ 3]$
using *So-field field-sum*
by (*metis* (*no-types*, *lifting*) *disj-Se-So finite.emptyI finite.insertI finite-Un*)
hence *sum-So-field*: $sum\text{-}So \in field\ [B\ 1,\ B\ 2,\ B\ 3]$
unfolding *sum-So-def* **using** *of-real-add* **by** *force*

obtain *B-no-0* **where** *B-no-0-prop*: $set\ B\text{-}no\text{-}0 = set\ [B\ 1,\ B\ 2,\ B\ 3] - \{0\} \wedge field\ B\text{-}no\text{-}0 = field\ [B\ 1,\ B\ 2,\ B\ 3]$
using *field-remove-zeros* **by** *blast*

have $sum\text{-}So \neq 0 \Longrightarrow False$
proof −
assume $sum\text{-}So \neq 0$
hence $sqrt\ p = -sum\text{-}Se\ /\ sum\text{-}So$
using *fact-0bis* **by** (*smt* (*verit*, *best*) *nonzero-divide-eq-eq*)
hence *sqrtp-in*: $sqrt\ p \in field\ [B\ 1,\ B\ 2,\ B\ 3]$ **using** *sum-So-field sum-Se-field*
by (*metis field-inv-div field-opp of-real-divide of-real-minus*)

have $\forall j{\in}\{1,2,3\}$. *multiplicity p* $(A\ j)$ = *multiplicity p* $(B\ j)$ + *multiplicity p* $(A\ j)$
using *B-prop*
by (*metis add-cancel-left-left multiplicity-prime-power multiplicity-zero not-prime-0 p-prime power-not-zero prime-elem-multiplicity-mult-distrib prime-imp-prime-elem*)
hence $\forall j{\in}\{1,2,3\}$. *multiplicity p* $(B\ j) = 0$ **by** *simp*
hence $\forall j{\in}\{1,2,3\}$. $B\ j \neq 0 \longrightarrow \neg p$ *dvd* $(B\ j)$ **using** *prime-elem-multiplicity-eq-zero-iff*
using *p-prime* **by** *blast*
hence $\forall b{\in}$*set B-no-0*. $\neg p$ *dvd b* **using** *B-no-0-prop* **by** *force*
hence *cpx-sqrt p* $\notin$ *field B-no-0* **using** *root-p-not-in-field-extension p-prime* **by** *simp*
hence *sqrt p* $\notin$ *field* [*B 1*, *B 2*, *B 3*] **using** *B-no-0-prop* **unfolding** *cpx-sqrt-def*
by (*metis p-prime prime-ge-0-int*)
thus *False* **using** *sqrtp-in* **by** *blast*
qed
hence *sum-So = 0* **by** *blast*
hence *sqrt p* $*$ *sum-So = 0* **by** *simp*
hence $(\sum j{\in}So.$ *sqrt p* $*$ *fun3* $\varepsilon\ j *$ *sqrt*$(A\ j/p) *$ *of-int* $(X3\hat{}(j-1))) = 0$
unfolding *sum-So-def* **by** (*smt* (*verit*) *Re-complex-of-real eq-sum-random2*)
hence *sum-So-0*: $(\sum j{\in}So.$ *fun3* $\varepsilon\ j *$ *sqrt*$(A\ j) *$ *of-int* $(X3\hat{}(j-1))) = 0$
by (*smt* (*verit*, *best*) *Re-complex-of-real eq-sum-random1*)

have $\{k{+}1..3\} \subseteq$ *Se* **using** *Ak-bigger-not-sq square-even-multiplicity*[*of p*] *p-prime*
unfolding *Se-def* **by** *fastforce*
hence $\forall j{\in}So.\ j{\in}\{1,2,3\} \wedge j{\notin}\{k{+}1..3\}$ **using** *disj-Se-So* **by** *blast*
hence *So-Le-k*: $So \subseteq \{1..k\}$ **by** *fastforce*
hence *So-L-k*: $So{-}\{k\} \subseteq \{1..k{-}1\}$ **by** *fastforce*
hence *So-k-un-disj*: $(So{-}\{k\}) \cup (\{1..k{-}1\}{-}(So{-}\{k\})) = \{1..k{-}1\} \wedge (So{-}\{k\}) \cap (\{1..k{-}1\}{-}(So{-}\{k\})) = \{\}$
by *blast*
have *fin-So-k*: *finite* $(So{-}\{k\}) \wedge$ *finite* $(\{1..k{-}1\}{-}(So{-}\{k\}))$ **using** *So-L-k*
by (*meson finite-Diff finite-atLeastAtMost finite-subset*)

have *fin-So*: *finite So* **unfolding** *So-def* **by** *simp*
have *So-123*: $So{\subseteq}\{1,2,3\}$ **using** *So-def* **by** *fast*
hence *So-k-123*: $So{-}\{k\}{\subseteq}\{1,2,3\}$ **by** *blast*
have *tok-123*: $\{1..k{-}1\}{\subseteq}\{1,2,3\}$ **using** *k123* **by** *force*
have *k-in-So*: $k{\in}So$ **unfolding** *So-def* **using** *k123 p-mult* **by** *blast*
hence $(\sum j{\in}So.$ *fun3* $\varepsilon\ j *$ *sqrt*$(A\ j) *$ *of-int* $(X3\hat{}(j-1)))$
$= (\sum j{\in}So{-}\{k\}.$ *fun3* $\varepsilon\ j *$ *sqrt*$(A\ j) *$ *of-int* $(X3\hat{}(j-1)))$
$+$ *fun3* $\varepsilon\ k *$ *sqrt*$(A\ k) *$ *of-int* $(X3\hat{}(k-1))$
using *fin-So* **by** (*simp add*: *sum.remove*)
hence *fun3* $\varepsilon\ k *$ *sqrt*$(A\ k) *$ *of-int* $(X3\hat{}(k-1))$
$= -\ (\sum j{\in}So{-}\{k\}.$ *fun3* $\varepsilon\ j *$ *sqrt*$(A\ j) *$ *of-int* $(X3\hat{}(j-1)))$
using *sum-So-0* **by** *simp*

hence *abs* (*fun3* ε *k* $*$ *sqrt*(*A k*) $*$ *of-int* (*X3*^(*k*−1)))
= *abs* ($\sum$ *j*∈*So*−{*k*}. *fun3* ε *j* $*$ *sqrt*(*A j*) $*$ *of-int* (*X3*^(*j*−1)))
by *simp*
hence *eq0-abs*: *abs* (*of-int* (*fun3* ε *k*)) $*$ *abs* (*sqrt*(*A k*) $*$ *of-int* (*X3*^(*k*−1)))
= *abs* ($\sum$ *j*∈*So*−{*k*}. *fun3* ε *j* $*$ *sqrt*(*A j*) $*$ *of-int* (*X3*^(*j*−1)))
using *abs-mult*[*of of-int* (*fun3* ε *k*) *sqrt*(*A k*) $*$ *of-int* (*X3*^(*k*−1))] **by** *argo*
have *fun3-eps-1*: ∀ *j*∈{*1*,*2*,*3*}. *fun3* ε *j*∈{−*1*, *1*} **using** ε*-prop k123* **by** *fastforce*
hence *fun3-epsk-1*: *fun3* ε *k*∈{−*1*, *1*} **using** *k123* **by** *blast*
have *abs-fun3*: ∀ *j*∈*So*−{*k*}. *abs*(*fun3* ε *j*) = *1* **using** *So-k-123 fun3-eps-1* **by** *force*
have *abs-sqrt-X3*: ∀ *j*∈{*1*,*2*,*3*}. *abs* (*sqrt*(*A j*) $*$ *of-int* (*X3*^(*j*−1))) = *sqrt*(*A j*) $*$ *X3*^(*j*−1)
using *abs-mult A-pos X3-pos* **by** *simp*
hence *random-eq*: *abs* (*sqrt*(*A k*) $*$ *of-int* (*X3*^(*k*−1))) = *sqrt*(*A k*) $*$ *X3*^(*k*−1) **using** *k123* **by** *blast*
hence *sqrt*(*A k*) $*$ *of-int* (*X3*^(*k*−1))
= *abs* ($\sum$ *j*∈*So*−{*k*}. *fun3* ε *j* $*$ *sqrt*(*A j*) $*$ *of-int* (*X3*^(*j*−1)))
using *eq0-abs fun3-epsk-1* **by** *force*
also have ... ≤ ($\sum$ *j*∈*So*−{*k*}. *abs* (*fun3* ε *j* $*$ *sqrt*(*A j*) $*$ *of-int* (*X3*^(*j*−1))))
by *blast*
also have ... = ($\sum$ *j*∈*So*−{*k*}. *abs* (*fun3* ε *j*) $*$ *abs*(*sqrt*(*A j*) $*$ *of-int* (*X3*^(*j*−1))))
using *abs-mult* **by** (*metis* (*no-types*, *opaque-lifting*) *mult.commute mult.left-commute of-int-abs*)
also have ... = ($\sum$ *j*∈*So*−{*k*}. *abs* (*sqrt*(*A j*) $*$ *of-int* (*X3*^(*j*−1))))
using *abs-fun3* **by** *force*
also have ... ≤ ($\sum$ *j*∈*So*−{*k*}. *abs* (*sqrt*(*A j*) $*$ *of-int* (*X3*^(*j*−1))))
+ ($\sum$ *j*∈{*1*..*k*−*1*}−(*So*−{*k*}). *abs* (*sqrt*(*A j*) $*$ *of-int* (*X3*^(*j*−1))))
by *force*
also have ... = ($\sum$ *j*∈{*1*..*k*−*1*}. *abs* (*sqrt*(*A j*) $*$ *of-int* (*X3*^(*j*−1))))
using *So-k-un-disj fin-So-k sum.union-disjoint* **by** (*smt* (*verit*, *best*))
also have ... = ($\sum$ *j*∈{*1*..*k*−*1*}. *sqrt*(*A j*) $*$ *of-int* (*X3*^(*j*−1)))
using *abs-sqrt-X3 tok-123* **by** (*meson subsetD sum.cong*)
also have ... ≤ ($\sum$ *j*∈{*1*..*k*−*1*}. *sqrt*(*A j*) $*$ *of-int* (*X3*^(*k*−*2*)))
proof −
have ∀ *j*∈{*0*..*k*−*2*}. *X3*^*j* ≤ *X3*^(*k*−*2*) **using** *X3-pos*
by (*meson atLeastAtMost-iff power-increasing*)
hence ∀ *j*∈{*1*..*k*−*1*}. *X3*^(*j*−*1*) ≤ *X3*^(*k*−*2*) **by** *fastforce*
hence ∀ *j*∈{*1*..*k*−*1*}. *sqrt* (*A j*) $*$ *X3*^(*j*−*1*) ≤ *sqrt*(*A j*) $*$*X3*^(*k*−*2*) **using** *A-pos tok-123*
by (*meson mult-left-mono of-int-le-iff of-int-nonneg real-sqrt-ge-zero subsetD*)
thus *?thesis* **by** (*meson sum-mono*)
qed
also have ... = ($\sum$ *j*∈{*1*..*k*−*1*}. *sqrt*(*A j*)) $*$ *X3*^(*k*−*2*) **using** *sum-distrib-right* **by** *metis*
finally have *ineq1*: *sqrt* (*real-of-int* (*A k*)) $*$ *real-of-int* (*X3* ^ (*k* − *1*))
≤ ($\sum$ *j* = *1*..*k* − *1*. *sqrt* (*real-of-int* (*A j*))) $*$ *real-of-int* (*X3* ^ (*k* − *2*))
by *simp*
have *sqrt* (*A k*) $*$ *of-int*(*X3*) ≤ ($\sum$ *j* = *1*..*k* − *1*. *sqrt* (*real-of-int* (*A j*)))
proof (*cases k*=*1*)

```
    case True
    then show ?thesis using ineq1 random-eq by simp
  next
    case False
    hence k-23: k∈{2,3} using k123 by blast
    hence 1+(k−2) = k−1 by force
    hence X3 ^ (k − 1) = X3 ∗ X3 ^ (k − 2)
      by (metis power-add power-one-right)
    hence real-of-int (X3 ^ (k − 1)) = X3 ∗ real-of-int (X3 ^ (k − 2)) by simp
    hence sqrt (A k) ∗ of-int(X3) ∗ of-int (X3 ^ (k − 2))
    ≤ (∑j = 1..k − 1. sqrt (real-of-int (A j))) ∗ of-int (X3 ^ (k − 2)) using
ineq1 by simp
    then show ?thesis using X3-pos by simp
  qed
  also have ... ≤ (∑j = 1..k − 1. abs (sqrt (A j)))
    by (simp add: sum-mono)
  also have ... ≤ (∑j = 1..k − 1. abs (sqrt (A j))) +(∑j = k..3. abs (sqrt (A
j))) by force
  also have ... ≤ (∑j∈{1,2,3}. abs (sqrt (A j)))
  proof −
    have {1..k−1}∪{k..3} = {1,2,3} ∧ {1..k−1}∩{k..3} = {} using k123 by
fastforce
    thus ?thesis by (smt (verit, best) finite-atLeastAtMost sum.union-disjoint)
  qed
  also have ... ≤ (∑j∈{1,2,3}. A j)
  by (smt (verit, best) A-pos of-int-nonneg of-int-sum real-sqrt-ge-0-iff sqrt-int-smaller
sum-mono)
  also have ... ≤ (∑j∈{1,2,3}. A j^2) using A-pos
  proof −
    have (m::int)≥0 ⟹ m≤m^2 for m
      by (meson of-int-power-le-of-int-cancel-iff sqrt-int-smaller sqrt-le-D)
    thus ?thesis using A-pos by (meson of-int-le-iff sum-mono)
  qed
  also have ... < 1 + (A 1)^2 + (A 2)^2 + (A 3)^2 by simp
  also have ... ≤ X3 using X3-def A-def by blast
  finally have sqrt (A k) < 1 using X3-pos by simp
  hence A k = 0 using A-pos k123 by force
  thus False using Ak-bigger-not-sq using is-square-def by force
 qed
 thus ?thesis using dir-imp by blast
qed

end

end
theory Pi-Relations
 imports J3-Relations
begin
```

7.4 The Π polynomial

lemma *finite-when*: *finite* $\{x.\ (f\ x\ when\ x = c) \neq 0\}$
by *auto*

locale *section5-variables*
begin

definition $\mathcal{S}$ :: *int mpoly* **where** $\mathcal{S} \equiv Var\ 4$
definition $\mathcal{T}$ :: *int mpoly* **where** $\mathcal{T} \equiv Var\ 5$
definition $\mathcal{R}$ :: *int mpoly* **where** $\mathcal{R} \equiv Var\ 6$
definition $\mathcal{I}$ :: *int mpoly* **where** $\mathcal{I} \equiv Var\ 7$
definition $\mathcal{Y}$ :: *int mpoly* **where** $\mathcal{Y} \equiv Var\ 8$
definition $\mathfrak{Z}$ :: *int mpoly* **where** $\mathfrak{Z} \equiv Var\ 9$
definition $\mathcal{J}$ :: *int mpoly* **where** $\mathcal{J} \equiv Var\ 10$
definition n :: *int mpoly* **where** n ≡ *Var 11*

definition $\mathcal{U}$ ::*int mpoly* **where** $\mathcal{U} = \mathcal{S}\hat{}2 * (2*\mathcal{R}-1)$
definition $\mathcal{V}$ ::*int mpoly* **where** $\mathcal{V} = \mathcal{T}\hat{}2 + \mathcal{S}\hat{}2 *$ n

lemmas *defs* = $\mathcal{S}$-*def* $\mathcal{T}$-*def* $\mathcal{R}$-*def* $\mathcal{I}$-*def* $\mathcal{Y}$-*def* $\mathfrak{Z}$-*def* $\mathcal{J}$-*def* n-*def* $\mathcal{U}$-*def* $\mathcal{V}$-*def*

end

locale *pi-relations* = *section5-variables* +
fixes $\mathcal{F}$ :: *int mpoly*
and *v* :: *nat*
assumes *F-monom-over-v*: *vars* $\mathcal{F} \subseteq \{v\}$
and *F-one*: *coeff* $\mathcal{F}$ (*Abs-poly-mapping* $(\lambda k.\ (degree\ \mathcal{F}\ v\ when\ k = v))) = 1$
begin

definition *coeff-F* **where**
coeff-F d = *coeff* $\mathcal{F}$ (*Abs-poly-mapping* $(\lambda k.\ (d\ when\ k = v))$)
definition *q* :: *nat* **where**
q = *degree* $\mathcal{F}$ *v*
definition *G* :: *int mpoly* **where**
G = $(\sum i{=}0..q.\ of\text{-}int\ (coeff\text{-}F\ i) * (\mathcal{Y}-\mathcal{I}-\mathcal{T}\hat{}2)\hat{}i)$
definition *G-sub* :: *nat*⇒*int mpoly* **where**
G-sub j = $(\sum i{=}j..q.\ of\text{-}int\ (coeff\text{-}F\ i) * of\text{-}nat\ (i\ choose\ j) * (-\mathcal{I}-\mathcal{T}\hat{}2)\hat{}(i-j))$
definition *H* :: *int mpoly* **where**
H = $(\sum j{=}0..q.\ G\text{-}sub\ j * \mathfrak{Z}\hat{}j * \mathcal{J}\hat{}(q-j))$
definition Π :: *int mpoly* **where**
Π = $(\sum j{=}0..q.\ G\text{-}sub\ j * (\mathcal{S}\hat{}2*\text{n} + \mathcal{T}\hat{}2)\hat{}j * (\mathcal{S}\hat{}2*(2*\mathcal{R}-1))\hat{}(q-j))$

lemma *eval-F*:
insertion f $\mathcal{F}$ = $(\sum d = 0..q.\ coeff\text{-}F\ d * (f\ v\ \hat{}\ d))$
proof −
have *aux1*: $\{a.\ lookup\ (mapping\text{-}of\ \mathcal{F})\ a * (\prod v.\ f\ v\ \hat{}\ lookup\ a\ v) \neq 0\}$
$\subseteq \{a \in range\ (\lambda d.\ Abs\text{-}poly\text{-}mapping\ (\lambda k.\ d\ when\ k = v)).$
$lookup\ (mapping\text{-}of\ \mathcal{F})\ a * (\prod v.\ f\ v\ \hat{}\ lookup\ a\ v) \neq 0\}$

```
  proof −
    have lookup (mapping-of ℱ) a ≠ 0 ⟹
        (∏ v. f v ^ lookup a v) ≠ 0 ⟹
        a ∈ range (λd. Abs-poly-mapping (λk. d when k = v)) for a
    proof −
      assume lookup (mapping-of ℱ) a ≠ 0
      hence a ∈ keys (mapping-of ℱ)
        unfolding keys.rep-eq
        by simp
      hence ⋀v'. v ≠ v' ⟹ lookup a v' = 0
        subgoal for v'
        proof (cases lookup a v' = 0)
          case True
          then show ?thesis by simp
        next
          case False
          assume lookup a v' ≠ 0
          hence v' ∈ ⋃ (keys ` keys (mapping-of ℱ))
            using ‹a ∈ keys (mapping-of ℱ)›
            unfolding keys.rep-eq
            apply simp
            apply (rule exI[where ?x=a])
            by simp
          hence 0: v' = v
            using F-monom-over-v vars-def
            by auto
          assume 1: v ≠ v'
          from 0 and 1 show ?thesis by simp
        qed
        done
      hence a = Abs-poly-mapping (λk. (lookup a v) when k = v)
        apply (subst lookup-inverse[symmetric])
        apply (rule arg-cong[where ?f=Abs-poly-mapping])
        by (metis (mono-tags, opaque-lifting) when(1) when(2))
      thus a ∈ range (λd. Abs-poly-mapping (λk. d when k = v))
        by simp
    qed
    thus ?thesis by auto
  qed

  have aux2: lookup (mapping-of ℱ) (Abs-poly-mapping (λk. d when k = v)) ≠ 0
⟹
    (∏ va. f va ^ lookup (Abs-poly-mapping (λk. d when k = v)) va) ≠ 0 ⟹ d ∈
{0..q} for d
  proof −
    assume lookup (mapping-of ℱ) (Abs-poly-mapping (λk. d when k = v)) ≠ 0
    thus d ∈ {0..q}
      unfolding q-def degree.rep-eq keys.rep-eq
      apply (subst atLeastAtMost-iff)
```

```
    apply simp
    apply (rule Max-ge)
    subgoal by simp
    apply (rule insertI2)
    apply (rule image-eqI[where ?x=Abs-poly-mapping (λk. d when k = v)])
    subgoal
      apply (subst lookup-Abs-poly-mapping)
      subgoal using finite-when[where ?f=λ-. d] by simp
      by auto
    by simp
  qed

  have aux3: x ≤ q ⟹ y ≤ q ⟹
    Abs-poly-mapping (λk. x when k = v) = Abs-poly-mapping (λk. y when k = v)
⟹ x = y for x y
    proof −
      assume Abs-poly-mapping (λk. x when k = v) = Abs-poly-mapping (λk. y
when k = v)
    hence 0: (λk. x when k = v) = (λk. y when k = v)
      apply (subst Abs-poly-mapping-inject[symmetric])
      subgoal
        apply (rule CollectI)
        using finite-when[where ?f=λ-. x] by simp
      subgoal
        apply (rule CollectI)
        using finite-when[where ?f=λ-. y] by simp
      by simp
    have (x when v = v) = (y when v = v)
      by (rule fun-cong[OF 0, where ?x=v])
    thus ?thesis
      by simp
  qed

  have aux4: lookup (mapping-of F) (Abs-poly-mapping (λk. d when k = v)) ≠ 0
⟹
    (∏ va. f va ^ lookup (Abs-poly-mapping (λk. d when k = v)) va) ≠ 0 ⟹
    Abs-poly-mapping (λk. d when k = v) ∈ (λd. Abs-poly-mapping (λk. d when k
= v)) ‘ {0..q} for d
    apply (rule image-eqI[where ?x=d, OF refl])
    using aux2 by simp

  have insertion f F = (∑ m∈{a. lookup (mapping-of F) a * (∏ v. f v ^ lookup
a v) ≠ 0}.
      lookup (mapping-of F) m * (∏ v. f v ^ lookup m v))
    unfolding insertion.rep-eq insertion-aux.rep-eq insertion-fun-def
    unfolding Sum-any.expand-set
    by simp
  also have ... =
    (∑ m∈{a ∈ range (λd. Abs-poly-mapping (λk. d when k = v)).
```

```
        lookup (mapping-of F) a * (∏ v. f v ^ lookup a v) ≠ 0}.
      lookup (mapping-of F) m * (∏ v. f v ^ lookup m v))
    apply (rule sum.mono-neutral-right[symmetric])
    subgoal by auto
    subgoal using aux1 by auto
    subgoal by auto
    done
  also have ... =
    (∑ m∈{a ∈ (λd. Abs-poly-mapping (λk. d when k = v)) ` {0..q}.
        lookup (mapping-of F) a * (∏ v. f v ^ lookup a v) ≠ 0}.
      lookup (mapping-of F) m * (∏ v. f v ^ lookup m v))
    apply (rule arg-cong[where ?f=sum -])
    apply standard
    subgoal using aux4 by auto
    subgoal by auto
    done
  also have ... =
    (∑ m∈(λd. Abs-poly-mapping (λk. d when k = v)) ` {0..q}.
      lookup (mapping-of F) m * (∏ v. f v ^ lookup m v))
    apply (rule sum.mono-neutral-right[symmetric])
    by auto
  also have ... =
    sum
      ((λm. lookup (mapping-of F) m *
       (∏ v. f v ^ lookup m v)) ∘ (λd. Abs-poly-mapping (λk. d when k = v)))
      {0..q}
    apply (rule sum.reindex) unfolding inj-on-def using aux3 by auto
  also have ... =
    (∑ d∈{0..q}. lookup (mapping-of F) (Abs-poly-mapping (λk. d when k = v))
*
      (∏ va. f va ^ lookup (Abs-poly-mapping (λk. d when k = v)) va))
    by simp
  finally show ?thesis
    unfolding coeff-F-def coeff-def
    apply simp
    apply (rule arg-cong[where ?f=λf. (∑ d = 0..q. - d * f d)])
    apply (rule ext)
    subgoal for d
      apply (subst lookup-Abs-poly-mapping)
      subgoal using finite-when[where ?f=λ-. d] by simp
    proof (cases d)
      case 0
      then show (∏ va. f va ^ (d when va = v)) = f v ^ d
        apply (subst ‹d = 0›)
        by simp
    next
      case (Suc nat)
      then show (∏ va. f va ^ (d when va = v)) = f v ^ d
        apply (subst pow-when)
```

```
      by simp-all
    qed
    done
qed


lemma triangular-sum: (∑k=0..(n::nat). (∑j=0..k. f j k)) = (∑j=0..(n::nat).
(∑k=j..n. f j k))
proof −
  have 0: k∈{0..n} ⟹ (∑j=0..k. f j k) = (∑j=0..n. (if j≤k then f j k else 0))
for k
  proof −
    assume k-prop: k∈{0..n}
    hence (∑j=0..k. f j k) = (∑j=0..k. f j k) + (∑j=(k+1)..n. 0) by simp
     also have ... = (∑j=0..k. (if j≤k then f j k else 0)) + (∑j=(k+1)..n. (if
j≤k then f j k else 0))
      by simp
    also have ... = (∑j=0..n. (if j≤k then f j k else 0))
      using sum.union-disjoint[of {0..k} {(k+1)..n}]
     by (smt (verit) atLeastAtMost-iff k-prop le-iff-add sum.ub-add-nat zero-order(1))
    finally show ?thesis by blast
  qed
  have 1: j∈{0..n} ⟹ (∑k=j..n. f j k) = (∑k=0..n. (if j≤k then f j k else 0))
for j
  proof −
    assume j-prop: j∈{0..n}
    hence (∑k=j..n. f j k) = (∑k<j. 0) + (∑k=j..n. f j k) by simp
    also have ... = (∑k<j. (if j≤k then f j k else 0)) + (∑k=j..n. (if j≤k then
f j k else 0))
      by simp
    also have ... = (∑k=0..n. (if j≤k then f j k else 0))
      using sum.union-disjoint[of {0..<j} {j..n}]
     by (smt (verit, best) atLeast0LessThan atLeastAtMost-iff finite-atLeastAtMost
finite-atLeastLessThan ivl-disj-int-two(7) ivl-disj-un-two(7) j-prop)
    finally show ?thesis by blast
  qed
  hence (∑k=0..(n::nat). (∑j=0..k. f j k)) = (∑k=0..n. (∑j=0..n. (if j≤k
then f j k else 0)))
    using 0 by simp
   also have ... = (∑j=0..n. (∑k=0..n. (if j≤k then f j k else 0))) using
sum.swap by fast
  finally show ?thesis using 1 by simp
qed


lemma G-expr:
   G = (∑j=0..q. 𝒴^j * (∑i=j..q. of-int (coeff-F i) * of-nat (i choose j) *
(−ℐ−𝒯^2)^(i−j)))
proof −
```

```
  have 𝒴−ℐ−𝒯^2 = 𝒴 + (−ℐ−𝒯^2) by simp
  hence (𝒴−ℐ−𝒯^2)^i = (∑j≤i. of-nat (i choose j) * 𝒴^j * (−ℐ−𝒯^2)^(i−j))
for i
    using binomial-ring[of 𝒴 −ℐ−𝒯^2 i] by presburger
  hence G = (∑i=0..q. of-int (coeff-F i) * (∑j≤i. of-nat (i choose j) * 𝒴^j *
(−ℐ−𝒯^2)^(i−j)))
    using G-def by presburger
  also have ... = (∑i=0..q. (∑j=0..i. of-int (coeff-F i) * of-nat (i choose j) *
𝒴^j * (−ℐ−𝒯^2)^(i−j) ))
    (is (∑i=-..-. ?f i) = (∑i=-..-. ?g i))
  proof (rule sum.cong[OF refl])
    fix i
      have ?f i = of-int (coeff-F i) * (∑j=0..i. of-nat (i choose j) * 𝒴^j *
(−ℐ−𝒯^2)^(i−j) )
      using atMost-atLeast0 by simp
    thus ?f i = ?g i
      unfolding mult.assoc
      by (simp add: sum-distrib-left)
  qed
  also have ... = (∑j=0..q. (∑i=j..q. of-int (coeff-F i) * of-nat (i choose j) *
𝒴^j * (−ℐ−𝒯^2)^(i−j) ))
    using triangular-sum[of (λj. (λi. of-int (coeff-F i) * of-nat (i choose j) * 𝒴^j
* (−ℐ−𝒯^2)^(i−j))) q]
    by blast
  also have ... = (∑j=0..q. (∑i=j..q. 𝒴^j * of-int (coeff-F i) * of-nat (i choose
j) * (−ℐ−𝒯^2)^(i−j) ))
    apply (rule sum.cong[OF refl])
    apply (rule sum.cong[OF refl])
    by simp
  also have ... = (∑j=0..q. 𝒴^j * (∑i=j..q. of-int (coeff-F i) * of-nat (i choose
j) * (−ℐ−𝒯^2)^(i−j) ))
    (is (∑j=-..-. ?f j) = (∑j=-..-. ?g j))
    apply (rule sum.cong[OF refl])
    unfolding mult.assoc
    by (simp add: sum-distrib-left)
  finally show ?thesis by simp
qed


lemma G-in-Y: G = (∑j=0..q. 𝒴^j * G-sub j)
  unfolding G-sub-def using G-expr by blast

lemma Gq-eq-1: G-sub q = 1
proof −
  have coeff-F q = 1 using q-def F-one unfolding coeff-F-def by simp
  have G-sub q = of-int (coeff-F q) unfolding G-sub-def by simp
  also have ... = of-int 1 using q-def using ‹coeff-F q = 1› by presburger
  finally show ?thesis using Const₀-one one-mpoly-def by simp
qed
```

lemma *lemma-J-div-Z* :
(∑ *j′*=*0*..*q*. *insertion f* (*G-sub j′*) * *z*^*j′* * *j*^(*q*−*j′*)) = *0* ⟹ *j dvd z* **for** *f z j*
proof (*cases j*=*0* ∧ *z*=*0*)
case *True*
then show *?thesis* **by** *blast*
next
case *False*
assume *assm*: (∑ *j′*=*0*..*q*. *insertion f* (*G-sub j′*) * *z*^*j′* * *j*^(*q*−*j′*)) = *0*
define *d* **where** *d* = *gcd j z*
have *d dvd j* **using** *d-def* **by** *blast*
then obtain *j0* **where** *j0-def*: *j* = *d* * *j0* **by** *blast*
have *d dvd z* **using** *d-def* **by** *blast*
then obtain *z0* **where** *z0-def*: *z* = *d* * *z0* **by** *blast*
have *gcd j z* = *d* * *gcd j0 z0* **using** *d-def j0-def z0-def*
by (*metis abs-gcd-int gcd-mult-distrib-int*)
hence *coprime*: *gcd j0 z0* = *1* **using** *d-def* **using** *False* **by** *simp*

have *dn0*: *d*≠*0* **using** *False d-def* **by** *simp*

have *j′*≤*q* ⟹ *j′* + (*q*−*j′*) =*q* **for** *j′* **by** *simp*
hence *nat-pow*: *j′*≤*q* ⟹ *d*^*j′* * *d*^(*q*−*j′*) = *d*^*q* **for** *j′* **using** *power-add* **by** *metis*
have *z*^*j′* * *j*^(*q*−*j′*) = *d*^*j′* * *z0*^*j′* * *d*^(*q*−*j′*) * *j0*^(*q*−*j′*) **for** *j′*
using *j0-def z0-def* **by** (*simp add*: *power-mult-distrib*)
hence *j′*≤*q* ⟹ *z*^*j′* * *j*^(*q*−*j′*) = *d*^*q* * *z0*^*j′* * *j0*^(*q*−*j′*) **for** *j′*
using *power-add nat-pow* **by** *simp*
hence *0* = (∑ *j′*=*0*..*q*. *insertion f* (*G-sub j′*) * *d*^*q* * *z0*^*j′* * *j0*^(*q*−*j′*))
using *assm*
by (*metis* (*no-types*, *lifting*) *ab-semigroup-mult-class.mult-ac*(*1*) *atLeastAtMost-iff sum.cong*)
also have ... = (∑ *j′*=*0*..*q*. *d*^*q* * *insertion f* (*G-sub j′*) * *z0*^*j′* * *j0*^(*q*−*j′*))
using *mult.commute* **by** (*metis* (*no-types*, *opaque-lifting*))
also have ... = *d*^*q* * (∑ *j′*=*0*..*q*. *insertion f* (*G-sub j′*) * *z0*^*j′* * *j0*^(*q*−*j′*))
using *sum-distrib-left*[*of d*^*q* (λ*j′*. *insertion f* (*G-sub j′*) * *z0*^*j′* * *j0*^(*q*−*j′*)) {*0*..*q*}]
by (*metis* (*no-types*, *lifting*) *ab-semigroup-mult-class.mult-ac*(*1*) *sum.cong*)
finally have *pol-red-0*: (∑ *j′*=*0*..*q*. *insertion f* (*G-sub j′*) * *z0*^*j′* * *j0*^(*q*−*j′*)) = *0*
using *dn0* **by** *simp*

have *inser-Gq-1*: *insertion f* (*G-sub q*) = *1* **using** *Gq-eq-1* **by** *simp*
have *j0-pow*: *j′*<*q* ⟹ *j0*^(*q*−*j′*) = *j0*^(*q*−*j′*−*1*) * *j0* **for** *j′*
by (*metis power-minus-mult zero-less-diff*)
have (∑ *j′*=*0*..*q*. *insertion f* (*G-sub j′*) * *z0*^*j′* * *j0*^(*q*−*j′*))
= (∑ *j′*=*0*..<*q*. *insertion f* (*G-sub j′*) * *z0*^*j′* * *j0*^(*q*−*j′*)) + *insertion f* (*G-sub q*) * *z0*^*q* * *j0*^(*q*−*q*)
by (*simp add*: *sum.last-plus*)
also have ... = (∑ *j′*=*0*..<*q*. *insertion f* (*G-sub j′*) * *z0*^*j′* * *j0*^(*q*−*j′*)) + *z0*^*q*

using *inser-Gq-1* **by** *simp*
also have ... = ($\sum j'$=0..<q. *insertion f* (*G-sub j'*) * *z0^j'* * *j0^*(q−j'−1)*j0) + *z0^q*
using *j0-pow sum.cong* **by** (*simp add*: *ab-semigroup-mult-class.mult-ac*(1))
also have ... = ($\sum j'$=0..<q. *insertion f* (*G-sub j'*) * *z0^j'* * *j0^*(q−j'−1)) * *j0* + *z0^q*
using *sum-distrib-right*[*of* - - *j0*] **by** *metis*
finally have *z0^q* = −($\sum j'$=0..<q. *insertion f* (*G-sub j'*) * *z0^j'* * *j0^*(q−j'−1)) * *j0*
using *pol-red-0* **by** *simp*
hence *j0 dvd z0^q* **by** *simp*
hence *j0 dvd z0* **using** *coprime*
by (*metis coprime-dvd-mult-left-iff coprime-dvd-mult-right-iff coprime-power-right-iff is-unit-gcd one-dvd*)
thus *j dvd z* **using** *j0-def z0-def* **by** *simp*
qed

lemma *lemma-pos*:
assumes ($\sum j'$=0..q. *insertion f* (*G-sub j'*) * *z^j'* * *j^*(q−j')) = *0*
j≠*0 z* = *z'* * *j*
shows *insertion* (λ-. *z'* − *f 7* − (*f 5*)^*2*) $\mathcal{F}$ = *0*
proof −
have *z^j'* = *z'^j'* * *j^j'* **for** *j'* **using** *power-mult-distrib*[*of z' j j'*] *assms* **by** *blast*
hence *0*: *0* = ($\sum j'$=0..q. *insertion f* (*G-sub j'*) * *z'^j'* * *j^j'* * *j^*(q−j')) **using** *assms*
by (*metis* (*no-types*, *lifting*) *ab-semigroup-mult-class.mult-ac*(*1*) *sum.cong*)
also have ... = ($\sum j'$=0..q. *insertion f* (*G-sub j'*) * *z'^j'* * *j^*(*j'*+(q−*j'*)))
using *power-add*[*of j*] **by** (*metis* (*no-types*, *lifting*) *ab-semigroup-mult-class.mult-ac*(*1*))
also have ... = ($\sum j'$=0..q. *insertion f* (*G-sub j'*) * *z'^j'* * *j^q*) **by** *simp*
also have ... = ($\sum j'$=0..q. *insertion f* (*G-sub j'*) * *z'^j'*) * *j^q*
using *sum-distrib-right*[*of* - - *j^q*] **by** *metis*
finally have *G-is-0*: ($\sum j'$=0..q. *insertion f* (*G-sub j'*) * *z'^j'*) = *0* **using** *assms* **by** *simp*

define *f'* **where** *f'* = (λ*x*. *f x*)(*8*:=*z'*)
hence *insertion f'* $\mathcal{Y}$ = *z'*
by (*simp add*: $\mathcal{Y}$-*def*)
hence *ins-f'-Y*: *insertion f'* ($\mathcal{Y}$ ^*j'*) = *z'^j'* **for** *j'*::*nat* **using** *insertion-pow* **by** *blast*
have *ins-f'-Gsub*: *insertion f'* (*G-sub j'*) = *insertion f* (*G-sub j'*) **for** *j'*

proof (*rule insertion-irrelevant-vars*)
fix *v*
assume *H*: *v* ∈ *vars* (*G-sub j'*)
have *vars* (*G-sub j'*) ⊆ *vars* $\mathcal{I}$ ∪ *vars* $\mathcal{T}$
unfolding *G-sub-def*
apply (*rule subset-trans*[*OF vars-setsum*])
subgoal by *simp*

```
    apply (rule union-subset)
    subgoal for x
     apply (rule subset-trans[OF vars-mult])
     apply (rule Un-least)
      apply (rule subset-trans[OF vars-mult])
      apply (subst of-int-Const)
      apply (subst of-nat-Const)
     using vars-Const
     subgoal by simp
     apply (rule subset-trans[OF vars-pow])
     apply (subst diff-conv-add-uminus)
     apply (rule subset-trans[OF vars-add])
     apply (rule Un-mono)
     using vars-neg
     subgoal by auto
     apply (subst vars-neg)
     apply (subst power2-eq-square)
     apply (rule subset-trans[OF vars-mult])
     by simp
    done
   also have ... ⊆ {5, 7}
    unfolding defs
    unfolding MPoly-Type.Var.abs-eq[symmetric]
    by simp
   finally have v ≠ 8
    using H
    by auto
   thus f' v = f v
    unfolding f'-def
    by auto
  qed

  have (∑j'=0..q. insertion f' (G-sub j') * insertion f' (𝒴 ^ j')) = 0
   using ins-f'-Y ins-f'-Gsub G-is-0 by presburger
  hence (∑j'=0..q. insertion f' (G-sub j' * 𝒴 ^ j')) = 0
   using insertion-mult by (metis (no-types, lifting) sum.cong)
  hence insertion f' (∑j'=0..q. G-sub j' * 𝒴 ^ j') = 0 using insertion-sum[of
{0..q} f'] by simp
  hence insertion f' G = 0 using G-in-Y mult.commute
   by (metis (no-types, lifting) sum.cong)

  hence (∑i = 0..q. insertion f' (of-int (coeff-F i) * (𝒴 - ℐ - 𝒯²) ^ i)) = 0
   unfolding G-def using insertion-sum[of {0..q} f'] by simp
  hence (∑i = 0..q. coeff-F i * insertion f' ((𝒴 - ℐ - 𝒯²) ^ i)) = 0
   by simp
  hence 1: (∑i = 0..q. coeff-F i * (insertion f' (𝒴 - ℐ - 𝒯²)) ^ i) = 0
   using insertion-pow[of f'] by simp

  have insertion f' (𝒴 - ℐ - 𝒯²) = z' - f 7 - (f 5)^2
```

```
  proof −
    have insertion f' (𝒴 − ℐ − 𝒯^2) = insertion f' 𝒴 + insertion f' (−ℐ−𝒯*𝒯)
      using insertion-add[of f' 𝒴 −ℐ−𝒯*𝒯] power2-eq-square[of 𝒯] add.assoc
      by (metis ab-group-add-class.ab-diff-conv-add-uminus)
    also have ... = insertion f' 𝒴 − insertion f' ℐ − insertion f' (𝒯*𝒯)
      by simp
    also have ... = insertion f' 𝒴 − insertion f' ℐ − (insertion f' 𝒯) ^2
      using insertion-mult[of f' 𝒯 𝒯] power2-eq-square by metis
    also have ... = f' 8 − f' 7 − (f' 5) ^2
      unfolding 𝒴-def ℐ-def 𝒯-def by simp
    finally show ?thesis using f'-def by simp
  qed

  hence (∑ i = 0..q. coeff-F i * (z' − f 7 − (f 5) ^2) ^ i) = 0 using 1 by simp
  thus insertion (λ-. z' − f 7 − (f 5) ^2) ℱ = 0
    using q-def eval-F
    by simp
qed

lemma Π-encodes-correctly:
  fixes S T R I :: int
  assumes S ≠ 0
          (∀ x. insertion (λ-. x) ℱ = 0 ⟶ x > −I)
      and S dvd T
          R > 0
          insertion (λ-. y) ℱ = 0
  defines φ n ≡ (λ-. 0)(4:= S, 5:=T, 6:=R, 7:=I, 11:=int n)
  shows ∃ n :: nat. insertion (φ n) Π = 0 ∧ n = (2*R−1)*(y + I + T^2) − (T
div S) ^ 2
proof −
  obtain TdS where TdS-def: T = TdS * S using ‹S dvd T› by force
  define n_i where n_i = (2*R−1)*(y+I+T^2) − TdS^2
  have nBe0: n_i≥0
  proof −
    have 0: (2*R−1) ≥ 1 using assms by simp
    have y > −I using assms by blast
    hence 1: y+I+T^2 > T^2 by linarith
    have T^2 ≥0 by simp
    hence y+I+T^2 > 0 using 1 by linarith
    hence (2*R−1)*(y+I+T^2) ≥ y+I+T^2 using 0 by simp
    hence 2: (2*R−1)*(y+I+T^2) ≥ T^2 using 1 by simp

     have 3: T^2 = TdS^2 * S^2 unfolding TdS-def using power-mult-distrib
by blast
    have S^2 > 0 using assms by simp
    hence 4: S^2 ≥ 1 by linarith
    have TdS^2 ≥ 0 by simp
   hence TdS^2 * S^2 ≥ TdS^2 using 4 by (metis mult-cancel-left1 mult-left-mono)
    hence T^2 ≥ TdS^2 using 3 by simp
```

thus *?thesis* **unfolding** n_i*-def* **using** *2* **by** *simp*
qed
then obtain *n* **where** *n-def*: *int n* = n_i **using** *nonneg-int-cases* **by** *metis*

define *f*::*nat*⇒*int* **where** *f* = (λ-. *0*) (*4* := *S*, *5* := *T*, *6* := *R*, *7* := *I*, *11* := *int n*)
have *0*: *insertion f* Π
= (∑ *j* = *0*..*q*. *insertion f* (*G-sub j* * ($\mathcal{S}^2$ * n + $\mathcal{T}^2$) ^ *j* * ($\mathcal{S}^2$ * (*2* * $\mathcal{R}$ − *1*)) ^ (*q* − *j*)))
unfolding Π*-def* **using** *insertion-sum* **by** *blast*
have *1*: *insertion f* (*G-sub j* * ($\mathcal{S}^2$ * n + $\mathcal{T}^2$) ^ *j* * ($\mathcal{S}^2$ * (*2* * $\mathcal{R}$ − *1*)) ^ (*q* − *j*))
= *insertion f* (*G-sub j*) * (*S^2* * *n* + *T^2*) ^*j* * (*S^2* * (*2**R− *1*)) ^ (*q* − *j*) **for** *j*
proof −
have *0*: *insertion f* (*G-sub j* * ($\mathcal{S}^2$ * n + $\mathcal{T}^2$) ^ *j* * ($\mathcal{S}^2$ * (*2* * $\mathcal{R}$ − *1*)) ^ (*q* − *j*))
= *insertion f* (*G-sub j*) * *insertion f* (($\mathcal{S}^2$ * n + $\mathcal{T}^2$) ^ *j*) * *insertion f* (($\mathcal{S}^2$ * (*2* * $\mathcal{R}$ − *1*)) ^ (*q* − *j*))
using *insertion-mult*[*of f*] **by** *simp*
have *1*: *insertion f* (($\mathcal{S}^2$ * n + $\mathcal{T}^2$) ^ *j*) * *insertion f* (($\mathcal{S}^2$ * (*2* * $\mathcal{R}$ − *1*)) ^ (*q* − *j*))
= (*insertion f* ($\mathcal{S}^2$ * n + $\mathcal{T}^2$)) ^ *j* * (*insertion f* ($\mathcal{S}^2$ * (*2* * $\mathcal{R}$ − *1*))) ^ (*q* − *j*)
using *insertion-pow* **by** *metis*
have *insertion f* ($\mathcal{S}^2$ * n + $\mathcal{T}^2$) = *insertion f* ($\mathcal{S}$*$\mathcal{S}$) * *insertion f* n + *insertion f* ($\mathcal{T}$*$\mathcal{T}$)
using *insertion-add*[*of f*] *insertion-mult*[*of f*] *power2-eq-square* **by** *metis*
also have ... = (*insertion f* $\mathcal{S}$) ^*2* * *insertion f* n + (*insertion f* $\mathcal{T}$) ^*2*
using *insertion-mult*[*of f*] *power2-eq-square* **by** *metis*
also have ... = *S^2* * *n* + *T^2*
unfolding $\mathcal{S}$*-def* n*-def* $\mathcal{T}$*-def* **using** *f-def* **by** *simp*
finally have *2*: *insertion f* ($\mathcal{S}^2$ * n + $\mathcal{T}^2$) = *S^2* * *n* + *T^2* **by** *simp*
have *insertion f* ($\mathcal{S}^2$ * (*2* * $\mathcal{R}$ − *1*)) = (*insertion f* $\mathcal{S}$) ^*2* * *insertion f* (*2* * $\mathcal{R}$ − *1*)
using *insertion-mult*[*of f*] *power2-eq-square* **by** *metis*
also have ... = (*insertion f* $\mathcal{S}$) ^*2* * (*insertion f* (*2**$\mathcal{R}$) − *insertion f 1*)
by *simp*
also have ... = (*insertion f* $\mathcal{S}$) ^*2* * (*2* * *insertion f* $\mathcal{R}$ − *1*)
by (*simp add*: *insertion-mult*[*of f*])
also have ... = *S^2* * (*2***R*−*1*)
unfolding $\mathcal{S}$*-def* $\mathcal{R}$*-def* **using** *f-def* **by** *simp*
finally have *3*: *insertion f* ($\mathcal{S}^2$ * (*2* * $\mathcal{R}$ − *1*)) = *S^2* * (*2***R*−*1*) **by** *simp*
show *?thesis* **using** *0 1 2 3* **by** *simp*
qed
have *2*: *j*≤*q* ⟹ (*S^2* * *n* + *T^2*) ^*j* * (*S^2* * (*2***R*− *1*)) ^ (*q* − *j*)
= (*y*+*I*+*T^2*) ^*j* * (*S^2* * (*2***R*− *1*)) ^ *q* **for** *j*
proof −

assume *jLeq*: $j \leq q$
have $S\hat{}2 * n + T\hat{}2 = S\hat{}2*(2*R-1)*(y+I+T\hat{}2) - S\hat{}2 * TdS\hat{}2 + T\hat{}2$
unfolding *n-def* n_i*-def*
by (*simp add*: *right-diff-distrib*)
also have ... $= S\hat{}2*(2*R-1)*(y+I+T\hat{}2) - (S* TdS)\hat{}2 + T\hat{}2$ **using** *power-mult-distrib* **by** *metis*
also have ... $= S\hat{}2*(2*R-1)*(y+I+T\hat{}2)$ **using** *TdS-def* **by** (*simp add*: *mult.commute*)
finally have $(S\hat{}2 * n + T\hat{}2)\hat{}j * (S\hat{}2 * (2*R- 1)) \hat{} (q - j)$
$= (S\hat{}2*(2*R-1)*(y+I+T\hat{}2))\hat{}j * (S\hat{}2 * (2*R- 1)) \hat{} (q - j)$ **by** *simp*
also have ... $= (y+I+T\hat{}2)\hat{}j * (S\hat{}2*(2*R-1))\hat{}j * (S\hat{}2 * (2*R- 1)) \hat{} (q - j)$
using *power-mult-distrib mult.commute* **by** *metis*
also have ... $= (y+I+T\hat{}2)\hat{}j * (S\hat{}2 * (2*R- 1))\hat{}q$ **using** *power-add jLeq*
by (*metis* (*no-types*, *lifting*) *add-diff-cancel-left' mult.assoc nat-le-iff-add*)
finally show *?thesis* **by** *simp*
qed
hence *insertion f* Π
$= (\sum j = 0..q.\ insertion\ f\ (G\text{-}sub\ j) * (S\hat{}2 * n + T\hat{}2)\hat{}j * (S\hat{}2 * (2*R-1)) \hat{} (q - j))$
using *0 1* **by** *presburger*
also have ... $= (\sum j = 0..q.\ insertion\ f\ (G\text{-}sub\ j) * (y+I+T\hat{}2)\hat{}j * (S\hat{}2 * (2*R- 1)) \hat{} q)$
using *2* **by** (*metis* (*no-types*, *lifting*) *ab-semigroup-mult-class.mult-ac*(*1*) *atLeastAtMost-iff sum.cong*)
finally have *inser-expr-1*: *insertion f* Π =
$(\sum j = 0..q.\ insertion\ f\ (G\text{-}sub\ j) * (y+I+T\hat{}2)\hat{}j) * (S\hat{}2 * (2*R-1)) \hat{} q$
using *sum-distrib-right*[*of* - - $(S\hat{}2 * (2*R- 1)) \hat{} q$] **by** *metis*

have *inser-G-sub*: *insertion f* $(G\text{-}sub\ j) = (\sum i{=}j..q.\ coeff\text{-}F\ i * of\text{-}nat\ (i\ choose\ j) * (-I-T\hat{}2)\hat{}(i-j))$
for *j*
proof −
have *insertion f* $(G\text{-}sub\ j)$
$= insertion\ f\ (\sum i{=}j..q.\ of\text{-}int\ (coeff\text{-}F\ i) * of\text{-}nat\ (i\ choose\ j) * (-\mathcal{I}-\mathcal{T}\hat{}2)\hat{}(i-j))$
unfolding *G-sub-def* **by** *blast*
also have ... $= (\sum i{=}j..q.\ insertion\ f\ (of\text{-}int\ (coeff\text{-}F\ i) * of\text{-}nat\ (i\ choose\ j) * (-\mathcal{I}-\mathcal{T}\hat{}2)\hat{}(i-j)))$
using *insertion-sum* **by** *blast*
also have ... $= (\sum i{=}j..q.\ (coeff\text{-}F\ i) * of\text{-}nat\ (i\ choose\ j) * insertion\ f\ ((-\mathcal{I}-\mathcal{T}\hat{}2)\hat{}(i-j)))$
proof −
have *1*: *insertion f* $(of\text{-}int\ (coeff\text{-}F\ i) * of\text{-}nat\ (i\ choose\ j) * (-\mathcal{I}-\mathcal{T}\hat{}2)\hat{}(i-j))$
$= (coeff\text{-}F\ i) * insertion\ f\ (of\text{-}nat\ (i\ choose\ j) * (-\mathcal{I}-\mathcal{T}\hat{}2)\hat{}(i-j))$ **for** *i*
using *mult.assoc insertion-of-int-times* **by** *metis*
moreover have $(coeff\text{-}F\ i) * insertion\ f\ (of\text{-}nat\ (i\ choose\ j) * (-\mathcal{I}-\mathcal{T}\hat{}2)\hat{}(i-j))$
$= (coeff\text{-}F\ i) * (i\ choose\ j) * insertion\ f\ ((-\mathcal{I}-\mathcal{T}\hat{}2)\hat{}(i-j))$ **for** *i*
using *insertion-mult*

by (*smt* (*verit*) *ab-semigroup-mult-class.mult-ac*(*1*) *insertion-of-int-times mult.commute*
of-int-of-nat-eq)
ultimately show *?thesis* **by** *presburger*
qed
also have ... = ($\sum$ *i*=*j*..*q*. (*coeff-F i*) ∗ *of-nat* (*i choose j*) ∗ (*insertion f* (−$\mathcal{I}$−$\mathcal{T}$ ^*2*)) ^(*i*−*j*))
proof −
have *insertion f* ((−$\mathcal{I}$−$\mathcal{T}$ ^*2*) ^(*i*−*j*)) = (*insertion f* (−$\mathcal{I}$−$\mathcal{T}$ ^*2*)) ^(*i*−*j*) **for** *i*
using *insertion-pow* **by** *blast*
thus *?thesis* **by** *simp*
qed
also have ... = ($\sum$ *i*=*j*..*q*. (*coeff-F i*) ∗ *of-nat* (*i choose j*) ∗ (−*insertion f* $\mathcal{I}$ − (*insertion f* $\mathcal{T}$) ^*2*) ^(*i*−*j*))
proof −
have *insertion f* (−$\mathcal{I}$−$\mathcal{T}$ ^*2*) = −*insertion f* ($\mathcal{I}$ + $\mathcal{T}$ ^*2*) **by** *simp*
also have ... = −*insertion f* $\mathcal{I}$ − (*insertion f* ($\mathcal{T}$ ^*2*)) **by** *simp*
finally have *insertion f* (−$\mathcal{I}$−$\mathcal{T}$ ^*2*) = −*insertion f* $\mathcal{I}$ − (*insertion f* $\mathcal{T}$) ^*2*
using *insertion-mult*[*of f* $\mathcal{T}$ $\mathcal{T}$] *power2-eq-square* **by** *metis*
thus *?thesis* **by** *simp*
qed
also have ... = ($\sum$ *i*=*j*..*q*. *coeff-F i* ∗ *of-nat* (*i choose j*) ∗ (−*I*−*T* ^*2*) ^(*i*−*j*))
unfolding *defs* **using** *f-def* **by** *simp*
finally show *?thesis* **by** *blast*
qed

hence ($\sum$ *j* = *0*..*q*. *insertion f* (*G-sub j*) ∗ (*y*+*I*+*T* ^*2*) ^*j*)
= ($\sum$ *j* = *0*..*q*. ($\sum$ *i*=*j*..*q*. *coeff-F i* ∗ *of-nat* (*i choose j*) ∗ (−*I*−*T* ^*2*) ^(*i*−*j*)) ∗ (*y*+*I*+*T* ^*2*) ^*j*)
using *inser-G-sub* **by** *presburger*
also have ... = ($\sum$ *j* = *0*..*q*. ($\sum$ *i*=*j*..*q*. *coeff-F i* ∗ *of-nat* (*i choose j*) ∗ (−*I*−*T* ^*2*) ^(*i*−*j*) ∗ (*y*+*I*+*T* ^*2*) ^*j*))
proof −
have ($\sum$ *i*=*j*..*q*. *coeff-F i* ∗ *of-nat* (*i choose j*) ∗ (−*I*−*T* ^*2*) ^(*i*−*j*)) ∗ (*y*+*I*+*T* ^*2*) ^*j*
= ($\sum$ *i*=*j*..*q*. *coeff-F i* ∗ *of-nat* (*i choose j*) ∗ (−*I*−*T* ^*2*) ^(*i*−*j*) ∗ (*y*+*I*+*T* ^*2*) ^*j*) **for** *j*
using *sum-distrib-right* **by** *blast*
thus *?thesis* **by** *presburger*
qed
also have ... = ($\sum$ *i* = *0*..*q*. ($\sum$ *j*=*0*..*i*. *coeff-F i* ∗ *of-nat* (*i choose j*) ∗ (−*I*−*T* ^*2*) ^(*i*−*j*) ∗ (*y*+*I*+*T* ^*2*) ^*j*))
using *triangular-sum*[*of* (λ*j*. (λ*i*. *coeff-F i* ∗ *of-nat* (*i choose j*) ∗ (−*I*−*T* ^*2*) ^(*i*−*j*) ∗ (*y*+*I*+*T* ^*2*) ^*j*)) *q*]
by *argo*
also have ... = ($\sum$ *i* = *0*..*q*. *coeff-F i* ∗ ($\sum$ *j*=*0*..*i*. *of-nat* (*i choose j*) ∗ (−*I*−*T* ^*2*) ^(*i*−*j*) ∗ (*y*+*I*+*T* ^*2*) ^*j*))
proof −
have ($\sum$ *j*=*0*..*i*. *coeff-F i* ∗ *of-nat* (*i choose j*) ∗ (−*I*−*T* ^*2*) ^(*i*−*j*) ∗ (*y*+*I*+*T* ^*2*) ^*j*)
= *coeff-F i* ∗ ($\sum$ *j*=*0*..*i*. *of-nat* (*i choose j*) ∗ (−*I*−*T* ^*2*) ^(*i*−*j*) ∗

```
(y+I+T^2) ^j) for i
      using sum-distrib-left[of coeff-F i (λj. of-nat (i choose j) * (−I−T^2) ^(i−j)
* (y+I+T^2) ^j) {0..i}]
      by (metis (no-types, lifting) ab-semigroup-mult-class.mult-ac(1) sum.cong)
    thus ?thesis by presburger
  qed
  also have ... = (∑ i = 0..q. coeff-F i * y^i)
  proof −
    have (∑ j=0..i. of-nat (i choose j) * (−I−T^2) ^(i−j) * (y+I+T^2) ^j) =
y^i for i
      using binomial-ring[of y+I+T^2 −I−T^2 i]
        by (smt (verit, best) ab-semigroup-mult-class.mult-ac(1) atLeast0AtMost
mult.commute sum.cong)
    thus ?thesis by simp
  qed
  also have ... = insertion (λ-. y) 𝓕 using q-def eval-F by metis
  also have ... = 0 using assms by blast
  finally have insertion f Π = 0 using inser-expr-1 by presburger

  hence insertion (φ n) Π = 0
    unfolding f-def φ-def by auto

  moreover have n = (2*R−1)*(y + I + T^2) − (T div S) ^ 2
    unfolding n-def n_i-def TdS-def using ‹S ≠ 0› by auto

  ultimately show ?thesis by auto
qed



lemma Π-encodes-correctly-2:
  fixes S T R I :: int
  assumes S ≠ 0
          (∀ x. insertion (λ-. x) 𝓕 = 0 ⟶ x > −I)
  defines φ n ≡ (λ-. 0)(4:= S, 5:=T, 6:=R, 7:=I, 11:=int n)
  assumes ∃ n :: nat. insertion (φ n) Π = 0
  shows S dvd T ∧ R > 0 ∧ (∃ x::int. (insertion (λ-. x) 𝓕) = 0)
proof −
  from assms obtain n where
    prop-2-n: insertion ((λ-. 0) (4 := S, 5:=T, 6:=R, 7:= I, 11:= int n)) Π = 0
    unfolding φ-def by auto
  define subst::nat⇒int where subst = ((λ-. 0) (4 := S, 5 := T, 6 := R, 7 :=
I, 11 := n))
  define J::int where J = S^2 * (2*R−1)
  have 2*R−1 ≠ 0 by presburger
  hence Jnot0: J≠0 unfolding J-def using assms by simp

 have 0 = insertion subst (∑ j=0..q. G-sub j * (𝒮 ^2 * n + 𝒯 ^2) ^j * (𝒮 ^2*(2*ℛ−1)) ^(q−j))
    using prop-2-n unfolding Π-def[symmetric] subst-def by argo
```

also have ... = ($\sum j$=0..q. *insertion subst* (*G-sub j* * ($\mathcal{S}$ ^2 * n + $\mathcal{T}$ ^2) ^j * ($\mathcal{S}$ ^2*(2*$\mathcal{R}$−1)) ^(q−j)))
using *insertion-sum* **by** *blast*
also have ... = ($\sum j$=0..q. *insertion subst* (*G-sub j*) * *insertion subst* (($\mathcal{S}$ ^2 * n + $\mathcal{T}$ ^2) ^j * ($\mathcal{S}$ ^2*(2*$\mathcal{R}$−1)) ^(q−j)))
proof −
have *insertion subst* (*G-sub j* * ($\mathcal{S}$ ^2 * n + $\mathcal{T}$ ^2) ^j * ($\mathcal{S}$ ^2*(2*$\mathcal{R}$−1)) ^(q−j))
= *insertion subst* (*G-sub j*) * *insertion subst* (($\mathcal{S}$ ^2 * n + $\mathcal{T}$ ^2) ^j * ($\mathcal{S}$ ^2*(2*$\mathcal{R}$−1)) ^(q−j)) **for** *j*
using *insertion-mult*[*of subst*] **by** *simp*
thus *?thesis* **by** *presburger*
qed
also have ... = ($\sum j$=0..q. *insertion subst* (*G-sub j*) * (*S*^2 * *n* + *T*^2) ^j * (*S*^2*(2*R−1)) ^(q−j))
proof −
have *insertion subst* (($\mathcal{S}$ ^2 * n + $\mathcal{T}$ ^2) ^j * ($\mathcal{S}$ ^2*(2*$\mathcal{R}$−1)) ^(q−j)) = (*S*^2 * *n* + *T*^2) ^j * (*S*^2*(2*R−1)) ^(q−j) **for** *j*
proof −
have *0*: *insertion subst* (($\mathcal{S}^2$ * n + $\mathcal{T}^2$) ^ *j* * ($\mathcal{S}^2$ * (*2* * $\mathcal{R}$ − *1*)) ^ (*q* − *j*))
= *insertion subst* (($\mathcal{S}^2$ * n + $\mathcal{T}^2$) ^ *j*) * *insertion subst* (($\mathcal{S}^2$ * (*2* * $\mathcal{R}$ − *1*)) ^ (*q* − *j*))
using *insertion-mult*[*of subst*] **by** *simp*
have *1*: *insertion subst* (($\mathcal{S}^2$ * n + $\mathcal{T}^2$) ^ *j*) * *insertion subst* (($\mathcal{S}^2$ * (*2* * $\mathcal{R}$ − *1*)) ^ (*q* − *j*))
= (*insertion subst* ($\mathcal{S}^2$ * n + $\mathcal{T}^2$)) ^ *j* * (*insertion subst* ($\mathcal{S}^2$ * (*2* * $\mathcal{R}$ − *1*))) ^ (*q* − *j*)
using *insertion-pow* **by** *metis*
have *insertion subst* ($\mathcal{S}^2$ * n + $\mathcal{T}^2$) = *insertion subst* ($\mathcal{S}$*$\mathcal{S}$) * *insertion subst* n + *insertion subst* ($\mathcal{T}$*$\mathcal{T}$)
using *insertion-add*[*of subst*] *insertion-mult*[*of subst*] *power2-eq-square* **by** *metis*
also have ... = (*insertion subst* $\mathcal{S}$) ^2 * *insertion subst* n + (*insertion subst* $\mathcal{T}$) ^2
using *insertion-mult*[*of subst*] *power2-eq-square* **by** *metis*
also have ... = *S*^2 * *n* + *T*^2
unfolding $\mathcal{S}$-*def* n-*def* $\mathcal{T}$-*def* **using** *subst-def* **by** *simp*
finally have *2*: *insertion subst* ($\mathcal{S}^2$ * n + $\mathcal{T}^2$) = *S*^2 * *n* + *T*^2 **by** *simp*
have *insertion subst* ($\mathcal{S}^2$ * (*2* * $\mathcal{R}$ − *1*)) = (*insertion subst* $\mathcal{S}$) ^2 * *insertion subst* (*2* * $\mathcal{R}$ − *1*)
using *insertion-mult*[*of subst*] *power2-eq-square* **by** *metis*
also have ... = (*insertion subst* $\mathcal{S}$) ^2 * (*insertion subst* (*2*$\mathcal{R}$) − *insertion subst 1*)
by *simp*
also have ... = (*insertion subst* $\mathcal{S}$) ^2 * (*2* * *insertion subst* $\mathcal{R}$ − *1*)
using *insertion-of-int-times*[*of subst 2*] **by** *simp*
also have ... = *S*^2 * (*2**R−1)
unfolding $\mathcal{S}$-*def* $\mathcal{R}$-*def* **using** *subst-def* **by** *simp*
finally have *3*: *insertion subst* ($\mathcal{S}^2$ * (*2* * $\mathcal{R}$ − *1*)) = *S*^2 * (*2**R−1) **by** *simp*
show *?thesis* **using** *0 1 2 3* **by** *argo*

```
    qed
   thus ?thesis using sum.cong by (simp add: ab-semigroup-mult-class.mult-ac(1))
  qed
  finally have Π-eq-0: (∑j=0..q. insertion subst (G-sub j) * (S^2 * n + T^2) ^j
* (S^2*(2*R-1)) ^(q-j)) = 0
    by simp
  hence J-div-ST: J dvd S^2 * n + T^2 using lemma-J-div-Z unfolding J-def
by blast
  hence S^2 dvd S^2 * n + T^2 unfolding J-def by fastforce
  hence S^2 dvd (S^2*n + T^2) - S^2*n by fastforce
  hence S^2 dvd T^2 by simp
  hence S-dvd-T: S dvd T by simp

  obtain Z::int where Z-prop: S^2 * n + T^2 = Z * J using J-div-ST by
fastforce
  hence insertion (λ-. Z - subst 7 - (subst 5) ^2) F = 0
    using Π-eq-0 Jnot0 lemma-pos[of subst S^2*n+T^2 J Z]
    unfolding J-def by blast
  hence F-cancels: insertion (λ-. Z - I - T^2) F = 0
    unfolding subst-def by simp
  hence Z - I - T^2 > -I
    using assms by blast
  hence Z > T^2 by simp
  hence Z ≠ 0 by fastforce
  hence SnT-not-0: S^2 * n + T^2 ≠ 0 using Z-prop Jnot0 by simp
  have Snt: S^2 * n + T^2 ≥ 0 using prop-2-n by simp
  hence SnT-B-0: S^2 * n + T^2 > 0 using SnT-not-0 by linarith
  have T^2 ≥0 by simp
  hence ZB0:Z > 0 using ‹Z > T^2› by linarith
  hence J≤0 ⟹ False
  proof -
    assume J≤0
    hence J<0 using Jnot0 by simp
    hence S^2 * n + T^2 < 0 using Z-prop ZB0
      by (metis SnT-B-0 one-dvd plus-int-code(2) zdvd-not-zless zero-less-mult-pos
zless-add1-eq)
    thus False using Snt by simp
  qed
  hence JB0:J>0 by fastforce
  hence S^2 * (2 * R- 1) > 0 unfolding J-def by blast
  hence 2 * R- 1 > 0
    by (metis mult-eq-0-iff zero-eq-power2 zero-less-mult-pos zero-less-power2)
  hence 2*R > 0 by simp
  hence RB0: R > 0 by simp
  thus ?thesis using S-dvd-T RB0 F-cancels by blast
qed

end
```

end
theory *M3-Polynomial*
imports *Pi-Relations Polynomials.MPoly-Type-Univariate*
../MPoly-Utils/Poly-Extract ../MPoly-Utils/Poly-Degree
begin

7.5 The Matiyasevich-Robinson-Polynomial

For any appropriately typed function fn, we introduce the syntax $fn\ \pi \equiv fn\ A1\ A2\ A3\ S\ T\ R\ n$, as well as $(\lambda\pi.\ e) \equiv (\lambda A1\ A2\ A3\ S\ T\ R\ n.\ e)$.

syntax *pi* :: $(int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int) \Rightarrow int$ (- π [*1000*] *1000*)
syntax *pi-fn* :: $(int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int) \Rightarrow int$ ($\lambda\pi$. - [*0*] *0*)
parse-translation ‹
[
(
syntax-const ‹*pi*›,
fn ctxt => fn args =>
list-comb (
args |> *hd*,
[A_1, A_2, A_3, S, T, R, n] |> *map* (*fn name => Free* (*name*, @{*typ int*}))
)
),
(
syntax-const ‹*pi-fn*›,
fn ctxt => fn args =>
[A_1, A_2, A_3, S, T, R, n]
|> *List.foldr* (*fn* (*name*, *r*) => *Abs* (*name*, @{*typ int*}, *r*)) (*args* |> *hd*)
)
]
›

locale *section5 = section5-given + section5-variables*
begin

definition X_0 :: $int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$ **where**
$X_0\ \pi = 1 + A_1\hat{}2 + A_2\hat{}2 + A_3\hat{}2$
definition I_0 :: $int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$ **where**
$I_0\ \pi = (X_0\ \pi)\hat{}3$
definition U_0 :: $int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$ **where**
$U_0\ \pi = S\hat{}2 * (2 * R - 1)$
definition V_0 :: $int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$ **where**
$V_0\ \pi = S\hat{}2 * n + T\hat{}2$

poly-extract X_0
poly-extract I_0
poly-extract U_0
poly-extract V_0

```
definition M3 :: int ⇒ int ⇒ int ⇒ int ⇒ int ⇒ int ⇒ int where
  M3 A₁ A₂ A₃ S T R n =
    (
      ((
        (
          (
            ((V₀ π) − (U₀ π) * (I₀ π) − (U₀ π) * T^2) ^2
              + (U₀ π) ^2 * A₁
              + (U₀ π) ^2 * A₂ * (X₀ π) ^2
              − (U₀ π) ^2 * A₃ * (X₀ π) ^4
          )
        ) ^2
          + 4 * (U₀ π) ^2 * (V₀ π − U₀ π * I₀ π − U₀ π * T^2) ^2 * A₁
          − 4 * (U₀ π) ^2 * (V₀ π − U₀ π * I₀ π − U₀ π * T^2) ^2 * A₂ * (X₀
π) ^2
          − 4 * (U₀ π) ^4 * A₁ * A₂ * (X₀ π) ^2
      ) ^2)
        − A₁ * (U₀ π * (
        4
          * (V₀ π − U₀ π * I₀ π − U₀ π * T^2)
          * (
            ((V₀ π − U₀ π * I₀ π − U₀ π * T^2)) ^2
              + (U₀ π) ^2 * A₁
              + (U₀ π) ^2 * A₂ * (X₀ π) ^2
              − (U₀ π) ^2 * A₃ * (X₀ π) ^4
            )
              − 8
                * (U₀ π) ^2
                * (V₀ π − U₀ π * I₀ π − U₀ π * T^2)
                * A₂
                * (X₀ π) ^2
        )) ^2
    )


poly-extract M3

end

end
theory Pi-to-M3-rat
  imports Pi-Relations J3-Relations M3-Polynomial
begin
```

7.6 Relation between M_3 and Π

ML‹

```
Theory.setup (
  Method.setup binding ‹unfold-auto› (Attrib.thms >>
    (fn thms => fn ctxt =>
      SIMPLE-METHOD (
        REPEAT1 (
          CHANGED-PROP (
            (TRY (Clasimp.auto-tac ctxt)) THEN
            Local-Defs.unfold-tac ctxt thms THEN
            (TRY (Clasimp.auto-tac ctxt))
          )
        )
      )
    )
  ) repeated unfold and auto
)
›

locale matiyasevich-polynomial = section5
begin

type-synonym τ=real


definition x' :: τ mpoly where x' ≡ of-int-mpoly x
definition 𝒜₁' :: τ mpoly where 𝒜₁' ≡ of-int-mpoly 𝒜₁
definition 𝒜₂' :: τ mpoly where 𝒜₂' ≡ of-int-mpoly 𝒜₂
definition 𝒜₃' :: τ mpoly where 𝒜₃' ≡ of-int-mpoly 𝒜₃
definition 𝒳₃' :: τ mpoly where 𝒳₃' ≡ of-int-mpoly 𝒳₃


definition 𝒮' :: τ mpoly where 𝒮' ≡ of-int-mpoly 𝒮
definition 𝒯' :: τ mpoly where 𝒯' ≡ of-int-mpoly 𝒯
definition ℛ' :: τ mpoly where ℛ' ≡ of-int-mpoly ℛ
definition ℐ' :: τ mpoly where ℐ' ≡ of-int-mpoly ℐ
definition 𝒴' :: τ mpoly where 𝒴' ≡ of-int-mpoly 𝒴
definition 𝒵' :: τ mpoly where 𝒵' ≡ of-int-mpoly 𝔷
definition 𝒥' :: τ mpoly where 𝒥' ≡ of-int-mpoly 𝒥
definition n' :: τ mpoly where n' ≡ of-int-mpoly n

definition 𝒰' ::τ mpoly where 𝒰' = of-int-mpoly 𝒰
definition 𝒱' ::τ mpoly where 𝒱' = of-int-mpoly 𝒱
definition J3' :: τ mpoly where J3' = of-int-mpoly J3
definition ψ' :: nat ⇒ τ mpoly where ψ' = of-int-mpoly ∘ ((λ-. 0)(0 := 𝒯 ^2+𝒮 ^2*n
−𝒯 ^2*𝒰− 𝒳₃ ^3*𝒰, 1 := 𝒰 ^2*𝒜₁, 2 := 𝒰 ^2*𝒜₂, 3 := 𝒰 ^2*𝒜₃))
definition M3-poly' :: τ mpoly where M3-poly' ≡ of-int-mpoly M3-poly

lemma Pi-equals-M3-rationals:
  fixes A₁ A₂ A₃ R S T n :: int
  defines X ≡ X₀ π
```

defines $I \equiv I_0\ \pi$
defines $U \equiv U_0\ \pi$
defines $V \equiv V_0\ \pi$

defines $\varphi \equiv (\lambda\text{-}.\ 0)(0 := \mathit{Var}\ 0,\ 1 := \mathit{Const}\ A_1,\ 2 := \mathit{Const}\ A_2,\ 3 := \mathit{Const}\ A_3)$

defines $\varphi' \equiv \mathit{of\text{-}int\text{-}mpoly} \circ \varphi$
defines $\mathcal{F} \equiv \mathit{poly\text{-}subst}\ \varphi\ J3$
defines $\mathcal{F}' \equiv \mathit{of\text{-}int\text{-}mpoly}\ \mathcal{F} :: \tau\ \mathit{mpoly}$
defines $q \equiv \mathit{MPoly\text{-}Type.degree}\ \mathcal{F}\ 0$
defines $\Pi \equiv \mathit{pi\text{-}relations}.\Pi\ \mathcal{F}\ 0$

defines $\mathit{G\text{-}sub} \equiv \mathit{pi\text{-}relations.G\text{-}sub}\ \mathcal{F}\ 0$

defines $\mathit{G\text{-}sub}' \equiv \mathit{of\text{-}int\text{-}mpoly} \circ (\mathit{pi\text{-}relations.G\text{-}sub}\ \mathcal{F}\ 0)$

defines $\xi \equiv ((\lambda\text{-}.\ 0)(4 := S,\ 5 := T,\ 6{:=}R,\ 7{:=}X\hat{}3,\ 11{:=}n))$

defines $\xi' \equiv \mathit{of\text{-}int} \circ \xi$

assumes $U \neq 0$
assumes $q = 8$
assumes *F-monom-over-0*: $\mathit{vars}\ \mathcal{F} \subseteq \{0\}$
and *F-one-0*: $\mathit{MPoly\text{-}Type.coeff}\ \mathcal{F}\ (\mathit{Abs\text{-}poly\text{-}mapping}\ (\lambda k.\ (\mathit{MPoly\text{-}Type.degree}\ \mathcal{F}\ 0\ \mathit{when}\ k = 0))) = 1$

shows $\mathit{insertion}\ \xi\ \Pi = M3\ \pi$
proof −
have *U-ev*: $\mathit{insertion}\ \xi\ \mathcal{U} = U$ **unfolding** *defs U-def* U_0*-def* ξ*-def power2-eq-square*

by *auto*
have *V-ev*: $\mathit{insertion}\ \xi\ \mathcal{V} = V$ **unfolding** *defs V-def* V_0*-def* ξ*-def power2-eq-square*
by *auto*

have *j-in-sum*: $\bigwedge j.\ j \leq q \longrightarrow \mathit{of\text{-}int}\ (\mathit{insertion}\ \xi\ ((\mathit{G\text{-}sub}\ j) * \mathcal{V}\hat{}j * \mathcal{U}\hat{}(q-j))) = \mathit{insertion}\ \xi\ (\mathit{G\text{-}sub}\ j) * (V/U)\hat{}j * U\hat{}q$
proof
fix j
assume $j \leq q$
have $\mathit{insertion}\ \xi\ ((\mathit{G\text{-}sub}\ j) * \mathcal{V}\hat{}j * \mathcal{U}\hat{}(q-j))$
$= (\mathit{insertion}\ \xi\ ((\mathit{G\text{-}sub}\ j) * \mathcal{V}\hat{}j) * (\mathit{insertion}\ \xi\ \mathcal{U}\hat{}(q-j)))$ **by** *auto*
also have ... $= \mathit{insertion}\ \xi\ (\mathit{G\text{-}sub}\ j) * V\hat{}j * U\hat{}(q-j)$
by (*auto simp*: *U-ev V-ev*)
also have ... $= \mathit{insertion}\ \xi\ (\mathit{G\text{-}sub}\ j) * V\hat{}j * U\hat{}q\ /\ U\hat{}j$
using *power-diff*[*of U j q*] ‹$U \neq 0$› ‹$j \leq q$› **apply** *simp*
by (*metis* (*no-types*, *lifting*) *of-int-0-eq-iff of-int-power power-diff times-divide-eq-right*)
also have ... $= \mathit{insertion}\ \xi\ (\mathit{G\text{-}sub}\ j) * (V/U)\hat{}j * U\hat{}q$
using *power-divide*[*of V U j*] **by** *auto*
finally show $\mathit{insertion}\ \xi\ ((\mathit{G\text{-}sub}\ j) * \mathcal{V}\hat{}j * \mathcal{U}\hat{}(q-j)) = \mathit{insertion}\ \xi\ (\mathit{G\text{-}sub}$

```
j) * (V/U) ^j * U^q
      by auto
  qed

  have XI3: I = X^3
    unfolding I-def I_0-def X-def by simp

  interpret pi-relations: pi-relations 𝓕 0
    using F-monom-over-0 F-one-0
    by (unfold-locales; auto)

  have X1-unfold:(of-int-mpoly (Const 1 + 𝒜₁² + 𝒜₂² + 𝒜₃²))
      = (of-int-mpoly (Const 1) + (of-int-mpoly 𝒜₁)² + (of-int-mpoly 𝒜₂)² +
(of-int-mpoly 𝒜₃)²)
    unfolding of-int-mpoly-simps ..

  have aux0: (λx. insertion ((λ-. 0)(0 := real-of-int V / real-of-int U − real-of-int
I − (real-of-int T)²))
              (of-int-mpoly (((λ-. 0)(0 := Var 0, Suc 0 := Const A₁, 2 := Const
A₂, 3 := Const A₃)) x))) i
        = ((λ-. 0)(0 := real-of-int V / real-of-int U − real-of-int I − (real-of-int
T)²,
                  Suc 0 := real-of-int A₁, 2 := real-of-int A₂, 3 := real-of-int A₃)) i
for i
    apply (cases i = 0)
    by (auto simp: of-int-mpoly-simps)

  have G-sub-vars: vars (G-sub j) ⊆ {5, 7} for j
    unfolding G-sub-def pi-relations.G-sub-def 𝓘-def 𝓣-def
          defs diff-conv-add-uminus
    apply (rule subset-trans[OF vars-setsum]; simp)
    apply (rule UN-least)
    unfolding power2-eq-square of-int-Const of-nat-Const diff-conv-add-uminus
    by mpoly-vars
  have G-sub-aux0: insertion ξ (G-sub j)
                  = insertion ((λ-. 0)(5 := T, 7 := X ^ 3)) (G-sub j) for j
    unfolding ξ-def
    apply (rule insertion-irrelevant-vars)
    using G-sub-vars[of j] by auto

  {
    fix j
   have real-of-int (insertion ((λ-. 0)(5 := T, 7 := X ^ 3)) (local.pi-relations.G-sub
j))
      = insertion (of-int ∘ ((λ-. 0)(5 := T, 7 := X ^ 3))) (of-int-mpoly (G-sub j))
      unfolding G-sub-def[symmetric] insertion-of-int-mpoly ..
    also have ... = insertion
         ((λx. real-of-int (((λ-. 0)(5 := T, 7 := X ^ 3)) x))(8 := real-of-int V /
real-of-int U))
```

```
      (of-int-mpoly (G-sub j))
    apply (intro insertion-irrelevant-vars)
    using vars-of-int-mpoly[of G-sub j] G-sub-vars[of j] by auto
  also have ...
    = insertion ((λx. real-of-int
        (((λ-. 0)(5 := T, 7 := X ^ 3)) x))(8 := real-of-int V / real-of-int U))
      (of-int-mpoly (local.pi-relations.G-sub j))
    unfolding G-sub-def[symmetric] ..
  finally have G-sub-aux1: real-of-int (insertion ((λ-. 0)(5 := T, 7 := X ^ 3))
(local.pi-relations.G-sub j))
    = insertion ((λx. real-of-int (((λ-. 0)(5 := T, 7 := X ^ 3)) x))(8 := real-of-int
V / real-of-int U))
      (of-int-mpoly (local.pi-relations.G-sub j)) .
 }
 note G-sub-aux1 = this

 have degree-aux-lt: MPoly-Type.degree F' 0 ≤ q
  unfolding q-def F'-def
  by (simp add: le-funD)

 have vars-F': vars F' ⊆ {0}
  unfolding F'-def
  apply (rule subset-trans[OF vars-of-int-mpoly])
  by (auto simp add: F-monom-over-0)

 have coeff-F': MPoly-Type.coeff F' = MPoly-Type.coeff F
  unfolding F'-def of-int-mpoly-coeff ..

 have degree-aux-gt: MPoly-Type.degree F' 0 ≥ q
 proof −
  have MPoly-Type.coeff F' (Abs-poly-mapping (λk. (q when k = 0))) = 1
   using F-one-0 coeff-F' q-def by simp

  thus ?thesis
   unfolding MPoly-Type.degree.rep-eq apply (intro Max-ge)
   subgoal by simp
   apply (rule insertI2)
   unfolding keys.rep-eq image-iff bex-simps(6) coeff-def
   apply (rule exI[of - Poly-Mapping.single 0 q])
   by (auto simp: Poly-Mapping.single.abs-eq when-def)
 qed

 have degree-aux: MPoly-Type.degree F' 0 = q
  using degree-aux-lt degree-aux-gt by simp

 have univariate-expansion:
  F' = (∑ d = 0..MPoly-Type.degree F' 0. of-int (pi-relations.coeff-F d) * (Var
0) ^ d)
  apply (subst mpoly-univariate-expansion-sum[of F' 0, OF vars-F'])
```

apply (*rule sum.cong*, *simp*)
unfolding *pi-relations.coeff-F-def Poly-Mapping.single.abs-eq coeff-$\mathcal{F}'$ of-int-general-Const*
by (*simp add*: *of-int-mpoly-simps*)

have *X-unfold*:X = *insertion*
((λ-. *0*)
(*0* := *real-of-int V* / *real-of-int U* − *real-of-int I* − (*real-of-int T*)^2,
Suc 0 := *real-of-int* A_1, *2* := *real-of-int* A_2, *3* := *real-of-int* A_3)) $\mathcal{X}_3{}'$
unfolding *x'-def* $\mathcal{A}_1{}'$*-def* $\mathcal{A}_2{}'$*-def* $\mathcal{A}_3{}'$*-def* $\mathcal{X}_3{}'$*-def x-def* $\mathcal{A}_1$*-def* $\mathcal{A}_2$*-def* $\mathcal{A}_3$*-def* $\mathcal{X}_3$*-def X-def* X_0*-def X1-unfold of-int-mpoly-simps*
by *simp*

have *U-V-cancel*: *of-int U* ∗ (*of-int V*/*of-int U*) = (*of-int V* :: *real*)
by (*simp add*: ‹$U \neq 0$›)

have *insertion* ξ Π = *insertion* ξ (∑*j*=*0*..*q*. *G-sub j* ∗ $\mathcal{V}$ ^*j* ∗ $\mathcal{U}$ ^(*q*−*j*))
unfolding Π*-def pi-relations.*Π*-def*
unfolding *q-def pi-relations.q-def*
unfolding *G-sub-def pi-relations.G-sub-def*
unfolding *defs*
by (*auto simp*: *algebra-simps*)
also have ... = (∑*j*=*0*..*q*. *insertion* ξ ((*G-sub j*) ∗ $\mathcal{V}$ ^*j* ∗ $\mathcal{U}$ ^(*q*−*j*)))
by *auto*
also have *of-int* (...) = (∑*j*=*0*..*q*. *insertion* ξ (*G-sub j*) ∗ (*V*/*U*) ^*j* ∗ *U*^*q*)
apply (*subst of-int-sum*)
apply (*rule sum.cong*)
using *j-in-sum* **by** *auto*
also have ... = *U*^*q* ∗ (∑*j*=*0*..*q*. *insertion* ξ (*G-sub j*) ∗ (*V*/*U*) ^*j*)
apply (*subst sum-distrib-left*)
by (*simp add*: *algebra-simps*)

also have ... = *U*^*q* ∗ *insertion* ((λ-. *0*)(*5*:=*T*, *7*:=*I*, *8*:=*V*/*U*)) (*of-int-mpoly* (*pi-relations.G*))
unfolding *pi-relations.G-in-Y G-sub-aux0*
apply (*simp only*: *of-int-mpoly-simps* $\mathcal{Y}$*-def G-sub-def*)
using *insertion-of-int-mpoly*[**where** *?P* = *local.pi-relations.G-sub* -
and *?α* = ((λ-. *0*)(*4* := *S*, *5* := *T*, *6* := *R*, *7* := *X* ^ *3*, *11* := *n*))]
apply (*subst insertion-simps*)
subgoal by *simp*
unfolding *q-def pi-relations.q-def comp-def of-int-mpoly-simps* ‹*I* = *X*^*3*›
apply (*subst insertion-simps*)+
unfolding *G-sub-aux1*
by (*simp add*: *mult.commute*)
also have ... = *U*^*q* ∗ *insertion* ((λ-. *0*)(*0*:=*of-int V*/*of-int U* − *of-int I* − *of-int T*^*2*)) $\mathcal{F}'$
unfolding *pi-relations.G-def*
apply (*subst univariate-expansion*)
apply (*simp add*: *defs of-int-mpoly-simps pi-relations.q-def pi-relations.coeff-F-def*)

apply (*rule disjI2*)
apply (*rule sum.cong*)
subgoal by (*simp add*: *degree-aux q-def*)
unfolding *of-int-Const of-int-mpoly-Const*
apply (*simp only*: *insertion-simps*)
unfolding *of-int-general-Const of-int-mpoly-Const*
by *simp*
also have ... = $U\hat{}q * insertion\ ((\lambda\text{-}.\ 0)(0{:=}of\text{-}int\ V / of\text{-}int\ U - of\text{-}int\ I - of\text{-}int\ T\hat{}2,$
$1{:=}of\text{-}int\ A_1,\ 2{:=}of\text{-}int\ A_2,\ 3{:=}of\text{-}int\ A_3))\ J3'$
unfolding $\mathcal{F}'$*-def* $\mathcal{F}$*-def J3'-def* φ*-def*
apply *simp*
unfolding *of-int-mpoly-poly-subst*
unfolding *insertion-poly-subst comp-def*
apply (*subst aux0*)
by *simp*

also have
... =
(
((
(
(
(of-int V − of-int U * of-int I − of-int U * of-int T^2) ^2
+ (of-int U) ^2 * A_1
+ (of-int U) ^2 * A_2 * X^2
− (of-int U) ^2 * A_3 * X^4
)
) ^2
+ 4 * (of-int U) ^2 * ((of-int V) − (of-int U) * of-int I − (of-int U) * of-int T^2) ^2 * A_1
− 4 * (of-int U) ^2 * ((of-int V) − (of-int U) * of-int I − (of-int U) * of-int T^2) ^2 * A_2 * X^2
− 4 * (of-int U) ^4 * A_1 * A_2 * X^2
) ^2)
− A_1 * ((of-int U) * (
4
* (of-int V − (of-int U) * of-int I − (of-int U) * of-int T^2)
* (
((of-int V − (of-int U) * of-int I − (of-int U) * of-int T^2)) ^2
+ (of-int U) ^2 * A_1
+ (of-int U) ^2 * A_2 * X^2
− (of-int U) ^2 * A_3 * X^4
)
− 8
* (of-int U) ^2
* (of-int V − (of-int U) * of-int I − (of-int U) * of-int T^2)
* A_2
* X^2

```
    )) ^2
  )

proof −
  have
    ... =
    (of-int U) ^8 * (
      (
        (
          (of-int V / of-int U − of-int I − of-int T^2) ^2
            + A₁
            + A₂ * X^2 − A₃ * X^4
        ) ^2
          + 4 * (of-int V / of-int U − of-int I − of-int T^2) ^2 * A₁
          − 4 * (of-int V / of-int U − of-int I − of-int T^2) ^2 * A₂ * X^2
          − 4 * A₁ * A₂ * X^2
      ) ^2
        − A₁ * (
          4
            * (of-int V / of-int U − of-int I − of-int T^2)
            * (
              (of-int V / of-int U − of-int I − of-int T^2) ^2
                + A₁
                + A₂ * X^2
                − A₃ * X^4
              )
                − 8 * (of-int V / of-int U − of-int I − of-int T^2) * A₂ * X^2
      ) ^2
    )

      by (unfold-auto ‹q = 8› J3'-def x'-def 𝒜₁'-def 𝒜₂'-def 𝒜₃'-def 𝒳₃'-def
J3-def x-def 𝒜₁-def
        𝒜₂-def 𝒜₃-def 𝒳₃-def X-unfold insertion-simps of-int-mpoly-simps)
    also have
    ... =
    (((of-int U) ^4) ^2) * (
      (
        (
          (of-int V / of-int U − of-int I − of-int T^2) ^2
            + A₁
            + A₂ * X^2
            − A₃ * X^4
        ) ^2
          + 4 * (of-int V / of-int U − of-int I − of-int T^2) ^2 * A₁
          − 4 * (of-int V / of-int U − of-int I − of-int T^2) ^2 * A₂ * X^2
          − 4 * A₁ * A₂ * X^2
      ) ^2
        − A₁ * (
          4
```

```
      * (of-int V / of-int U - of-int I - of-int T^2)
      * (
        (of-int V / of-int U - of-int I - of-int T^2) ^2
          + A1
          + A2 * X^2
          - A3 * X^4
        )
          - 8 * (of-int V / of-int U - of-int I - of-int T^2) * A2 * X^2
    ) ^2
)

  by simp
also have
  ... =
(
  ((((of-int U) ^4) ^2) * (
    (
      (of-int V / of-int U - of-int I - of-int T^2) ^2
        + A1
        + A2 * X^2
        - A3 * X^4
    ) ^2
      + 4 * (of-int V / of-int U - of-int I - of-int T^2) ^2 * A1
      - 4 * (of-int V / of-int U - of-int I - of-int T^2) ^2 * A2 * X^2
      - 4 * A1 * A2 * X^2
  ) ^2)
    - (of-int U) ^8 * (A1 * (
      4
        * (of-int V / of-int U - of-int I - of-int T^2)
        * (
          (of-int V / of-int U - of-int I - of-int T^2) ^2
            + A1
            + A2 * X^2
            - A3 * X^4
          )
            - 8 * (of-int V / of-int U - of-int I - of-int T^2) * A2 * X^2
    ) ^2)
)

  unfolding right-diff-distrib by simp
also have
  ... =
(
  ((((of-int U) ^4) ^2) * (
    (
      (of-int V / of-int U - of-int I - of-int T^2) ^2
        + A1
        + A2 * X^2
        - A3 * X^4
```

```
    ) ^2
      + 4 * (of-int V / of-int U - of-int I - of-int T^2) ^2 * A₁
      - 4 * (of-int V / of-int U - of-int I - of-int T^2) ^2 * A₂ * X^2
      - 4 * A₁ * A₂ * X^2
  ) ^2)
    - A₁ * (((of-int U) ^4) * (
      4
        * (of-int V / of-int U - of-int I - of-int T^2)
        * (
          (of-int V / of-int U - of-int I - of-int T^2) ^2
            + A₁
            + A₂ * X^2
            - A₃ * X^4
          )
            - 8 * (of-int V / of-int U - of-int I - of-int T^2) * A₂ * X^2
    )) ^2
)

  unfolding power-mult-distrib by simp
also have
  ... =
(
  ((((of-int U) ^4) * (
    (
      (of-int V / of-int U - of-int I - of-int T^2) ^2
        + A₁
        + A₂ * X^2
        - A₃ * X^4
    ) ^2
      + 4 * (of-int V / of-int U - of-int I - of-int T^2) ^2 * A₁
      - 4 * (of-int V / of-int U - of-int I - of-int T^2) ^2 * A₂ * X^2
      - 4 * A₁ * A₂ * X^2
  )) ^2)
    - A₁ * (((of-int U) ^4) * (
      4
        * (of-int V / of-int U - of-int I - of-int T^2)
        * (
          (of-int V / of-int U - of-int I - of-int T^2) ^2
            + A₁
            + A₂ * X^2
            - A₃ * X^4
          )
            - 8 * (of-int V / of-int U - of-int I - of-int T^2) * A₂ * X^2
    )) ^2
)

  unfolding power-mult-distrib by simp
also have
  ... =
```

```
    (
      ((
        ((of-int U) ^2^2) * (
          (of-int V/of-int U − of-int I − of-int T^2) ^2
            + A1
            + A2 * X^2
            − A3 * X^4
        ) ^2
          + ((of-int U) ^4) * 4 * (of-int V/of-int U − of-int I − of-int T^2) ^2
* A1
          − ((of-int U) ^4) * 4 * (of-int V/of-int U − of-int I − of-int T^2) ^2
* A2 * X^2
          − ((of-int U) ^4) * 4 * A1 * A2 * X^2
      ) ^2)
        − A1 * (((of-int U) ^4) * (
          4
            * (of-int V/of-int U − of-int I − of-int T^2)
            * (
              (of-int V/of-int U − of-int I − of-int T^2) ^2
                + A1
                + A2 * X^2
                − A3 * X^4
              )
                − 8 * (of-int V/of-int U − of-int I − of-int T^2) * A2 * X^2
      )) ^2
    )

    unfolding distrib-left right-diff-distrib mult.assoc by simp
  also have
    ... =
    (
      ((
        (
          (of-int U) ^2 * (
            (of-int V/of-int U − of-int I − of-int T^2) ^2
              + A1
              + A2 * X^2
              − A3 * X^4
          )
        ) ^2
          + ((of-int U) ^4) * 4 * (of-int V/of-int U − of-int I − of-int T^2) ^2
* A1
          − ((of-int U) ^4) * 4 * (of-int V/of-int U − of-int I − of-int T^2) ^2
* A2 * X^2
          − ((of-int U) ^4) * 4 * A1 * A2 * X^2
      ) ^2)
        − A1 * (((of-int U) ^4) * (
          4
            * (of-int V/of-int U − of-int I − of-int T^2)
```

```
          * (
            (of-int V/of-int U − of-int I − of-int T^2) ^2
              + A1
              + A2 * X^2
              − A3 * X^4
            )
              − 8 * (of-int V/of-int U − of-int I − of-int T^2) * A2 * X^2
      )) ^2
  )

    unfolding power-mult-distrib by simp
  also have
    ... =
    (
      ((
        (
          (
            (of-int U) ^2 * (of-int V/of-int U − of-int I − of-int T^2) ^2
              + (of-int U) ^2 * A1
              + (of-int U) ^2 * A2 * X^2
              − (of-int U) ^2 * A3 * X^4
          )
        ) ^2
          + ((of-int U) ^4) * 4 * (of-int V/of-int U − of-int I − of-int T^2) ^2
* A1
          − ((of-int U) ^4) * 4 * (of-int V/of-int U − of-int I − of-int T^2) ^2
* A2 * X^2
          − ((of-int U) ^4) * 4 * A1 * A2 * X^2
      ) ^2)
        − A1 * (((of-int U) ^4) * (
          4
            * (of-int V/of-int U − of-int I − of-int T^2)
            * (
              (of-int V/of-int U − of-int I − of-int T^2) ^2
                + A1
                + A2 * X^2
                − A3 * X^4
              )
                − 8 * (of-int V/of-int U − of-int I − of-int T^2) * A2 * X^2
      )) ^2
    )

    unfolding distrib-left right-diff-distrib mult.assoc by simp
  also have
    ... =
    (
      ((
        (
          (
```

```
          ((of-int U) * (of-int V / of-int U - of-int I - of-int T^2)) ^ 2
            + (of-int U) ^ 2 * A₁
            + (of-int U) ^ 2 * A₂ * X^2
            - (of-int U) ^ 2 * A₃ * X^4
        )
      ) ^ 2
        + ((of-int U) ^ 4) * 4 * (of-int V / of-int U - of-int I - of-int T^2) ^ 2
* A₁
        - ((of-int U) ^ 4) * 4 * (of-int V / of-int U - of-int I - of-int T^2) ^ 2
* A₂ * X^2
        - ((of-int U) ^ 4) * 4 * A₁ * A₂ * X^2
    ) ^ 2)
      - A₁ * (((of-int U) ^ 4) * (
        4
          * (of-int V / of-int U - of-int I - of-int T^2)
          * (
            (of-int V / of-int U - of-int I - of-int T^2) ^ 2
              + A₁
              + A₂ * X^2
              - A₃ * X^4
            )
              - 8 * (of-int V / of-int U - of-int I - of-int T^2) * A₂ * X^2
    )) ^ 2
  )
```

unfolding *power-mult-distrib* **by** *simp*

also have

```
  ... =
  (
    ((
      (
        (
          (of-int V - of-int U * of-int I - of-int U * of-int T^2) ^ 2
            + (of-int U) ^ 2 * A₁
            + (of-int U) ^ 2 * A₂ * X^2
            - (of-int U) ^ 2 * A₃ * X^4
        )
      ) ^ 2
        + ((of-int U) ^ 4) * 4 * (of-int V / of-int U - of-int I - of-int T^2) ^ 2
* A₁
        - ((of-int U) ^ 4) * 4 * (of-int V / of-int U - of-int I - of-int T^2) ^ 2
* A₂ * X^2
        - ((of-int U) ^ 4) * 4 * A₁ * A₂ * X^2
    ) ^ 2)
      - A₁ * (((of-int U) ^ 4) * (
        4
          * (of-int V / of-int U - of-int I - of-int T^2)
          * (
            (of-int V / of-int U - of-int I - of-int T^2) ^ 2
```

```
            + A_1
            + A_2 * X^2
            - A_3 * X^4
          )
            - 8 * (of-int V / of-int U - of-int I - of-int T^2) * A_2 * X^2
      )) ^2
    )
```

unfolding *distrib-left right-diff-distrib U-V-cancel* **by** *simp*
also have

```
    ... =
    (
      ((
        (
          (
            (of-int V - of-int U * of-int I - of-int U * of-int T^2) ^2
              + (of-int U) ^2 * A_1
              + (of-int U) ^2 * A_2 * X^2
              - (of-int U) ^2 * A_3 * X^4
          )
        ) ^2
          + 4 * (of-int U) ^2 * ((of-int U) ^2 * (of-int V / of-int U - of-int I -
of-int T^2) ^2) * A_1
          - 4 * (of-int U) ^2 * ((of-int U) ^2 * (of-int V / of-int U - of-int I -
of-int T^2) ^2) * A_2 * X^2
          - 4 * ((of-int U) ^4) * A_1 * A_2 * X^2
      ) ^2)
        - A_1 * (((of-int U) ^4) * (
          4
            * (of-int V / of-int U - of-int I - of-int T^2)
            * (
              (of-int V / of-int U - of-int I - of-int T^2) ^2
                + A_1
                + A_2 * X^2
                - A_3 * X^4
            )
              - 8 * (of-int V / of-int U - of-int I - of-int T^2) * A_2 * X^2
      )) ^2
    )
```

unfolding *ab-semigroup-mult-class.mult.commute*[*of 4*] *mult.assoc* **by** *simp*
also have

```
    ... =
    (
      ((
        (
          (
            (of-int V - of-int U * of-int I - of-int U * of-int T^2) ^2
              + (of-int U) ^2 * A_1
```

```
          + (of-int U)^2 * A_2 * X^2
          - (of-int U)^2 * A_3 * X^4
        )
      )^2
       + 4 * (of-int U)^2 * ((of-int U) * (of-int V / of-int U - of-int I - of-int
T^2))^2 * A_1
       - 4 * (of-int U)^2 * ((of-int U) * (of-int V / of-int U - of-int I - of-int
T^2))^2 * A_2 * X^2
        - 4 * (of-int U)^4 * A_1 * A_2 * X^2
    )^2)
      - A_1 * ((of-int U)^4 * (
        4
          * (of-int V / of-int U - of-int I - of-int T^2)
          * (
            (of-int V / of-int U - of-int I - of-int T^2)^2
              + A_1
              + A_2 * X^2
              - A_3 * X^4
            )
              - 8 * (of-int V / of-int U - of-int I - of-int T^2) * A_2 * X^2
    ))^2
  )

  unfolding power-mult-distrib ..
also have
  ... =
  (
    ((
      (
        (
          (of-int V - of-int U * of-int I - of-int U * of-int T^2)^2
            + (of-int U)^2 * A_1
            + (of-int U)^2 * A_2 * X^2
            - (of-int U)^2 * A_3 * X^4
        )
      )^2
        + 4 * (of-int U)^2 * ((of-int V) - (of-int U) * of-int I - (of-int U) *
of-int T^2)^2 * A_1
        - 4 * (of-int U)^2 * ((of-int V) - (of-int U) * of-int I - (of-int U) *
of-int T^2)^2 * A_2 * X^2
        - 4 * (of-int U)^4 * A_1 * A_2 * X^2
    )^2)
      - A_1 * ((of-int U)^4 * (
        4
          * (of-int V / of-int U - of-int I - of-int T^2)
          * (
            (of-int V / of-int U - of-int I - of-int T^2)^2
              + A_1
              + A_2 * X^2
```

```
            − A3 * X^4
          )
            − 8 * (of-int V/of-int U − of-int I − of-int T^2) * A2 * X^2
      )) ^2
    )

    unfolding distrib-left right-diff-distrib U-V-cancel by simp
  also have
    ... =
    (
      ((
        (
          (
            (of-int V − of-int U * of-int I − of-int U * of-int T^2) ^2
              + (of-int U) ^2 * A1
              + (of-int U) ^2 * A2 * X^2
              − (of-int U) ^2 * A3 * X^4
          )
        ) ^2
          + 4 * (of-int U) ^2 * ((of-int V) − (of-int U) * of-int I − (of-int U) *
of-int T^2) ^2 * A1
          − 4 * (of-int U) ^2 * ((of-int V) − (of-int U) * of-int I − (of-int U) *
of-int T^2) ^2 * A2 * X^2
          − 4 * (of-int U) ^4 * A1 * A2 * X^2
      ) ^2)
        − A1 * ((of-int U) * ((of-int U) ^3 * (
          4
            * (of-int V/of-int U − of-int I − of-int T^2)
            * (
              (of-int V/of-int U − of-int I − of-int T^2) ^2
                + A1
                + A2 * X^2
                − A3 * X^4
              )
                − 8 * (of-int V/of-int U − of-int I − of-int T^2) * A2 * X^2
      ))) ^2
    )

    unfolding power-def mult.assoc by simp
  also have
    ... =
    (
      ((
        (
          (
            (of-int V − of-int U * of-int I − of-int U * of-int T^2) ^2
              + (of-int U) ^2 * A1
              + (of-int U) ^2 * A2 * X^2
              − (of-int U) ^2 * A3 * X^4
```

```
          )
        ) ^2
          + 4 * (of-int U) ^2 * ((of-int V) - (of-int U) * of-int I - (of-int U) *
of-int T^2) ^2 * A1
          - 4 * (of-int U) ^2 * ((of-int V) - (of-int U) * of-int I - (of-int U) *
of-int T^2) ^2 * A2 * X^2
          - 4 * (of-int U) ^4 * A1 * A2 * X^2
      ) ^2)
        - A1 * ((of-int U) * (
          (of-int U) ^3
            * (4
            * (of-int V / of-int U - of-int I - of-int T^2)
            * (
              (of-int V / of-int U - of-int I - of-int T^2) ^2
                + A1
                + A2 * X^2
                - A3 * X^4
              ))
                - (of-int U) ^3 * (8 * (of-int V / of-int U - of-int I - of-int T^2)
* A2 * X^2)
        )) ^2
    )

    unfolding right-diff-distrib ..
  also have
    ... =
    (
      ((
        (
          (
            (of-int V - of-int U * of-int I - of-int U * of-int T^2) ^2
              + (of-int U) ^2 * A1
              + (of-int U) ^2 * A2 * X^2
              - (of-int U) ^2 * A3 * X^4
          )
        ) ^2
          + 4 * (of-int U) ^2 * ((of-int V) - (of-int U) * of-int I - (of-int U) *
of-int T^2) ^2 * A1
          - 4 * (of-int U) ^2 * ((of-int V) - (of-int U) * of-int I - (of-int U) *
of-int T^2) ^2 * A2 * X^2
          - 4 * (of-int U) ^4 * A1 * A2 * X^2
      ) ^2)
        - A1 * ((of-int U) * (
          4
            * (of-int U) ^3
            * (of-int V / of-int U - of-int I - of-int T^2)
            * (
              (of-int V / of-int U - of-int I - of-int T^2) ^2
                + A1
```

```
            + A_2 * X^2
            - A_3 * X^4
          )
            - 8 * (of-int U)^3 * (of-int V/of-int U - of-int I - of-int T^2)
* A_2 * X^2
        ))^2
    )
```

unfolding *ab-semigroup-mult-class.mult.commute*[*of 4*] *ab-semigroup-mult-class.mult.commute*[*of 8*] *mult.assoc* **by** *simp*

also have

```
  ... =
  (
    ((
      (
        (
          (of-int V - of-int U * of-int I - of-int U * of-int T^2)^2
            + (of-int U)^2 * A_1
            + (of-int U)^2 * A_2 * X^2
            - (of-int U)^2 * A_3 * X^4
        )
      )^2
        + 4 * (of-int U)^2 * ((of-int V) - (of-int U) * of-int I - (of-int U) *
of-int T^2)^2 * A_1
        - 4 * (of-int U)^2 * ((of-int V) - (of-int U) * of-int I - (of-int U) *
of-int T^2)^2 * A_2 * X^2
        - 4 * (of-int U)^4 * A_1 * A_2 * X^2
      )^2)
      - A_1 * ((of-int U) * (
        4
          * (of-int U)
          * (of-int U)^2
          * (of-int V/of-int U - of-int I - of-int T^2)
          * (
            (of-int V/of-int U - of-int I - of-int T^2)^2
              + A_1
              + A_2 * X^2
              - A_3 * X^4
            )
              - 8
                * (of-int U)^2
                * (of-int U) * (of-int V/of-int U - of-int I - of-int T^2)
                * A_2
                * X^2
      ))^2
  )
```

unfolding *power2-eq-square power3-eq-cube mult.assoc* **..**

also have

```
    ... =
    (
      ((
        (
          (
            (of-int V − of-int U ∗ of-int I − of-int U ∗ of-int T^2) ^2
              + (of-int U) ^2 ∗ A₁
              + (of-int U) ^2 ∗ A₂ ∗ X^2
              − (of-int U) ^2 ∗ A₃ ∗ X^4
          )
        ) ^2
          + 4 ∗ (of-int U) ^2 ∗ ((of-int V) − (of-int U) ∗ of-int I − (of-int U) ∗
of-int T^2) ^2 ∗ A₁
          − 4 ∗ (of-int U) ^2 ∗ ((of-int V) − (of-int U) ∗ of-int I − (of-int U) ∗
of-int T^2) ^2 ∗ A₂ ∗ X^2
          − 4 ∗ (of-int U) ^4 ∗ A₁ ∗ A₂ ∗ X^2
      ) ^2)
        − A₁ ∗ ((of-int U) ∗ (
          4
            ∗ ((of-int U)
            ∗ (of-int V / of-int U − of-int I − of-int T^2))
            ∗ ((of-int U) ^2
            ∗ (
              (of-int V / of-int U − of-int I − of-int T^2) ^2
                + A₁
                + A₂ ∗ X^2
                − A₃ ∗ X^4
              ))
                − 8
                  ∗ (of-int U) ^2 ∗ (of-int U)
                  ∗ (of-int V / of-int U − of-int I − of-int T^2)
                  ∗ A₂
                  ∗ X^2
      )) ^2
    )
```

unfolding *ab-semigroup-mult-class.mult.commute*[*of* (*of-int U*) ^2] *mult.assoc*
by *simp*
also have

```
    ... =
    (
      ((
        (
          (
            (of-int V − of-int U ∗ of-int I − of-int U ∗ of-int T^2) ^2
              + (of-int U) ^2 ∗ A₁
              + (of-int U) ^2 ∗ A₂ ∗ X^2
              − (of-int U) ^2 ∗ A₃ ∗ X^4
          )
```

```
      ) ^2
        + 4 * (of-int U) ^2 * ((of-int V) - (of-int U) * of-int I - (of-int U) *
of-int T^2) ^2 * A1
        - 4 * (of-int U) ^2 * ((of-int V) - (of-int U) * of-int I - (of-int U) *
of-int T^2) ^2 * A2 * X^2
        - 4 * (of-int U) ^4 * A1 * A2 * X^2
      ) ^2)
      - A1 * ((of-int U) * (
        4
          * ((of-int U) * (of-int V / of-int U) - (of-int U) * of-int I - (of-int
U) * of-int T^2)
          * (
            (of-int U) ^2 * (of-int V / of-int U - of-int I - of-int T^2) ^2
              + (of-int U) ^2 * A1
              + (of-int U) ^2 * A2 * X^2
              - (of-int U) ^2 * A3 * X^4
            )
              - 8
                * (of-int U) ^2
                * ((of-int U) * (of-int V / of-int U - of-int I - of-int T^2))
                * A2
                * X^2
      )) ^2
  )

  unfolding distrib-left right-diff-distrib mult.assoc by simp
also have
  ... =
  (
    ((
      (
        (
          (of-int V - of-int U * of-int I - of-int U * of-int T^2) ^2
            + (of-int U) ^2 * A1
            + (of-int U) ^2 * A2 * X^2
            - (of-int U) ^2 * A3 * X^4
        )
      ) ^2
        + 4 * (of-int U) ^2 * ((of-int V) - (of-int U) * of-int I - (of-int U) *
of-int T^2) ^2 * A1
        - 4 * (of-int U) ^2 * ((of-int V) - (of-int U) * of-int I - (of-int U) *
of-int T^2) ^2 * A2 * X^2
        - 4 * (of-int U) ^4 * A1 * A2 * X^2
      ) ^2)
      - A1 * ((of-int U) * (
        4
          * ((of-int U) * (of-int V / of-int U) - (of-int U) * of-int I - (of-int
U) * of-int T^2)
          * (
```

```
          ((of-int U) * (of-int V/of-int U − of-int I − of-int T^2)) ^2
            + (of-int U) ^2 * A1
            + (of-int U) ^2 * A2 * X^2
            − (of-int U) ^2 * A3 * X^4
          )
            − 8
              * (of-int U) ^2
              * ((of-int U) * (of-int V/of-int U − of-int I − of-int T^2))
              * A2
              * X^2
      )) ^2
  )
```

unfolding *power-mult-distrib* **..**

also have

```
  ... =
  (
    ((
      (
        (
          (of-int V − of-int U * of-int I − of-int U * of-int T^2) ^2
            + (of-int U) ^2 * A1
            + (of-int U) ^2 * A2 * X^2
            − (of-int U) ^2 * A3 * X^4
        )
      ) ^2
        + 4 * (of-int U) ^2 * ((of-int V) − (of-int U) * of-int I − (of-int U) *
of-int T^2) ^2 * A1
        − 4 * (of-int U) ^2 * ((of-int V) − (of-int U) * of-int I − (of-int U) *
of-int T^2) ^2 * A2 * X^2
        − 4 * (of-int U) ^4 * A1 * A2 * X^2
      ) ^2)
      − A1 * ((of-int U) * (
        4
          * ((of-int U) * (of-int V/of-int U) − (of-int U) * of-int I − (of-int
U) * of-int T^2)
          * (
            (((of-int U) * (of-int V/of-int U) − (of-int U) * of-int I − (of-int
U) * of-int T^2)) ^2
              + (of-int U) ^2 * A1
              + (of-int U) ^2 * A2 * X^2
              − (of-int U) ^2 * A3 * X^4
            )
              − 8
                * (of-int U) ^2
              * ((of-int U) * (of-int V/of-int U) − (of-int U) * of-int I − (of-int
U) * of-int T^2)
                * A2
                * X^2
```

```
      )) ^2
    )

    unfolding distrib-left right-diff-distrib ..
  also have
    ... =
    (
      ((
        (
          (
            (of-int V − of-int U ∗ of-int I − of-int U ∗ of-int T^2) ^2
              + (of-int U) ^2 ∗ A₁
              + (of-int U) ^2 ∗ A₂ ∗ X^2
              − (of-int U) ^2 ∗ A₃ ∗ X^4
          )
        ) ^2
          + 4 ∗ (of-int U) ^2 ∗ ((of-int V) − (of-int U) ∗ of-int I − (of-int U) ∗
of-int T^2) ^2 ∗ A₁
          − 4 ∗ (of-int U) ^2 ∗ ((of-int V) − (of-int U) ∗ of-int I − (of-int U) ∗
of-int T^2) ^2 ∗ A₂ ∗ X^2
          − 4 ∗ (of-int U) ^4 ∗ A₁ ∗ A₂ ∗ X^2
      ) ^2)
        − A₁ ∗ ((of-int U) ∗ (
          4
            ∗ (of-int V − (of-int U) ∗ of-int I − (of-int U) ∗ of-int T^2)
            ∗ (
              ((of-int V − (of-int U) ∗ of-int I − (of-int U) ∗ of-int T^2)) ^2
                + (of-int U) ^2 ∗ A₁
                + (of-int U) ^2 ∗ A₂ ∗ X^2
                − (of-int U) ^2 ∗ A₃ ∗ X^4
              )
                − 8
                  ∗ (of-int U) ^2
                  ∗ (of-int V − (of-int U) ∗ of-int I − (of-int U) ∗ of-int T^2)
                  ∗ A₂
                  ∗ X^2
        )) ^2
    )

    unfolding U-V-cancel by simp
  finally show ?thesis .
qed

also have ... = M3 π unfolding M3-def X-def I-def U-def V-def by auto
finally show insertion ξ Π = M3 π by auto
qed

end
```

end
theory *Matiyasevich-Polynomial*
imports *M3-Polynomial Digit-Expansions.Binary-Operations Pi-to-M3-rat*
begin

7.7 The key property of M_3

context *matiyasevich-polynomial*
begin

lemma *relation-combining'*:
fixes *R S T* A_1 A_2 A_3 :: *int*
assumes *S≠0*

defines $\gamma' \equiv \lambda(n :: int)\ fn.\ fn\ A_1\ A_2\ A_3\ S\ T\ R\ n :: int$

shows (*S dvd T* ∧ *R* > *0* ∧ *is-square* A_1 ∧ *is-square* A_2 ∧ *is-square* A_3)
= (∃ *n*. *n*≥*0* ∧ (γ' *n*) *M3* = *0*) (**is** *?LHS* = *?RHS*)
proof −

have *U-not-0*: *S^2* ∗ (*2*∗*R* − *1*) ≠ *0*
using ‹*S* ≠ *0*› **apply** *simp* **by** *presburger*

define φ :: *nat* ⇒ *int mpoly* **where**
φ = (λ-. *0*)(*0* := *Var 0*, *1* := *Const* A_1, *2* := *Const* A_2, *3* := *Const* A_3)
define $\mathcal{F}$ **where** $\mathcal{F} \equiv$ *poly-subst* φ *J3*
define *r* **where** *r* ≡ *MPoly-Type.degree* $\mathcal{F}$ *0*
define Π **where** Π ≡ *pi-relations*.Π $\mathcal{F}$ *0*

define *X* **where** *X* ≡ *1* + A_1 *^2* + A_2 *^2* + A_3 *^2*
define *I* **where** *I* ≡ *X^3*

have *F-repr*: $\mathcal{F}$ = ((*x^2*+*Const* A_1+*Const* A_2∗(*Const X^2*)−*Const* A_3∗(*Const X^4*)) *^2* + *4*∗*x^2*∗*Const* A_1
− *4*∗*x^2*∗*Const* A_2∗*Const X^2* − *4*∗*Const* A_1∗*Const* A_2∗*Const X^2*) *^2*
− *Const* A_1 ∗ ((*4*∗*x*∗(*x^2*+*Const* A_1+*Const* A_2∗*Const X^2* − *Const* A_3∗*Const X^4*)
− *8*∗*x*∗*Const* A_2∗*Const X^2*)) *^2*
unfolding $\mathcal{F}$*-def* φ*-def J3-def X-def*
unfolding *power2-eq-square power4-eq-xxxx section5-given.defs*
by (*simp add*: *Const-add*[*symmetric*] *Const-mult*[*symmetric*])

have *F-zeros-bound*: ∀ *x*. *insertion* (λ-. *x*) $\mathcal{F}$ = *0* ⟶ − *I* < *x*
proof −
have *aux0*: ⋀*x*. (λ-. *0*::*int*)(*0* := *x*, *1* := A_1, *2* := A_2, *3* := A_3)
= *insertion* (λ-. *x*) ∘ φ
unfolding φ*-def* **by** *auto*

```
    show ?thesis
      using J3-zeros-bound aux0[symmetric]
      unfolding F-def I-def X-def insertion-poly-subst comp-def
      by simp
  qed

  have deg-phi: MPoly-Type.degree (φ i) 0 = (1 when i = 0) for i
      unfolding φ-def by simp

  have aux5: MPoly-Type.degree P2 v2 ≤ k ⟹
     MPoly-Type.degree Q2 v2 ≤ k ⟹ MPoly-Type.degree (Q2 + P2) v2 ≤ k for
k v2
    and P2 Q2 :: int mpoly
    by (simp add: le-trans[OF degree-add max.boundedI])

  have r ≤ 8
    unfolding r-def F-repr power2-eq-square
    apply (simp add: algebra-simps)
   apply (simp add: power2-eq-square[symmetric] power-Suc[symmetric] mult.assoc[symmetric])
   unfolding x-def Const-power Const-numeral[symmetric] mult.assoc Const-mult
    apply (rule le-trans[OF degree-diff max.boundedI])
    apply (rule aux5)+
    unfolding degree-Const apply simp-all
    apply (subst le-trans[OF degree-mult] le-trans[OF degree-pow]; simp)+
    apply (rule aux5)+
    unfolding degree-Const apply simp-all
    by (subst le-trans[OF degree-mult] le-trans[OF degree-pow]; simp)+

  have eq-coeff: MPoly-Type.coeff F (Poly-Mapping.single 0 8) = 1
    unfolding F-repr power2-eq-square
    apply (simp add: algebra-simps)
   unfolding Notation.coeff-minus[symmetric] More-MPoly-Type.coeff-add[symmetric]
   apply (simp add: power2-eq-square[symmetric] power-Suc[symmetric] mult.assoc[symmetric])
    unfolding x-def
    apply (subst coeff-var-power-eq)
    apply (subst mult.assoc)+
    apply (subst coeff-var-power-le; simp)+
    unfolding Const-power Const-mult Const-numeral[symmetric]
    by (subst coeff-const; simp)+

  hence MPoly-Type.degree F 0 ≥ 8
    unfolding MPoly-Type.degree.rep-eq apply (intro Max-ge)
    subgoal by simp
    apply (rule insertI2)
    unfolding keys.rep-eq image-iff bex-simps(6) coeff-def
    apply (rule exI[of - Poly-Mapping.single 0 8])
    by simp
```

with ‹$r \leq 8$› *r-def* **have** $r = 8$ **by** *auto*

interpret *pi-relations*: *pi-relations* $\mathcal{F}$ *0*
proof (*unfold-locales*)
show *vars* $\mathcal{F} \subseteq \{0\}$
unfolding $\mathcal{F}$*-def*
apply (*rule subset-trans*[*OF vars-poly-subst*[*of* φ *J3*]])
unfolding φ*-def*
using *J3-vars* **by** *auto*
next
have *MPoly-Type.coeff* $\mathcal{F}$ (*Poly-Mapping.single 0 r*) = *1*
by (*simp add*: ‹*r*=*8*› *eq-coeff*)

thus *MPoly-Type.coeff* $\mathcal{F}$ (*Abs-poly-mapping* (λk. *MPoly-Type.degree* $\mathcal{F}$ *0 when* $k = 0$)) = *1*
unfolding *r-def single.abs-eq* **by** *simp*
qed

define η :: *nat* ⇒ *nat* ⇒ *int* **where** $\eta \equiv \lambda n.$ (λ-. *0*)(*4*:=*S*, *5*:=*T*, *6*:=*R*, *7*:=*I*, *11*:=*n*)

have *insertion-F-squares*:
($\exists x.$ *insertion* (λ-. *x*) $\mathcal{F}$ = *0*) = (*is-square* A_1 ∧ *is-square* A_2 ∧ *is-square* A_3)
proof −
have *insertion-equivalence*: *insertion* (λ-. *x*) ∘ φ
= (λ-. *0*)(*0* := *x*, *1* := A_1, *2* := A_2, *3* := A_3) **for** *x*
unfolding φ*-def comp-def* **by** *auto*

have ($\exists x.$ *insertion* (λ-. *x*) $\mathcal{F}$ = *0*)
= ($\exists y.$ *insertion* ((λ-. *0*)(*0* := *y*, *1* := A_1, *2* := A_2, *3* := A_3)) *J3* = *0*) (**is** *?F* = *?J3*)
proof (*rule iffI*)
assume *?F*
then obtain *x* **where** *x*: *0* = *insertion* (λ-. *x*) (*poly-subst* φ *J3*)
unfolding $\mathcal{F}$*-def* **by** *auto*

hence *insertion* (*insertion* (λ-. *x*) ∘ φ) *J3* = *0*
using *insertion-poly-subst*[*of* λ-. *x* φ *J3*] **by** *auto*

with *insertion-equivalence* **have**
insertion ((λ-. *0*)(*0* := *x*, *1* := A_1, *2* := A_2, *3* := A_3)) *J3* = *0*
by *auto*

thus *?J3* **by** *auto*
next
assume *?J3*
then obtain *y* **where** *insertion* ((λ-. *0*)(*0* := *y*, *1* := A_1, *2* := A_2, *3* := A_3)) *J3* = *0*
by *auto*

hence *insertion* (λ-. *y*) (*poly-subst* φ *J3*) = *0*
unfolding *insertion-poly-subst insertion-equivalence* **by** *auto*

thus *?F* **unfolding** $\mathcal{F}$*-def* **by** *auto*
qed

thus *?thesis*
unfolding *J3-encodes-three-squares* **by** *auto*
qed

hence *M3*-Π*-equivalence*: *insertion* ((λ-. *0*)(*4*:=*S*, *5*:=*T*, *6*:=*R*, *7*:=*I*, *11*:=*n*))
Π
= *M3* π **for** *n*
unfolding Π*-def I-def X-def*
using U_0*-def Pi-equals-M3-rationals*[*of* A_1 A_2 A_3 *S T R*]
using *U-not-0*
unfolding U_0*-def* X_0*-def*
using *pi-relations.F-monom-over-v local.pi-relations.F-one*
unfolding $\mathcal{F}$*-def* φ*-def*
using ‹*r* = *8*›
unfolding *r-def* $\mathcal{F}$*-def* φ*-def*
by *simp*

have *direct-imp*: *?LHS* ⟶ *?RHS*
proof
assume *asm*: *?LHS*
have *S dvd T* **using** *asm* **by** *auto*
have *R* > *0* **using** *asm* **by** *auto*
have ∃ *y*. *insertion* (λ-. *y*) $\mathcal{F}$ = *0*
using *insertion-F-squares asm* **by** *auto*

then obtain *y* **where** *insertion* (λ-. *y*) $\mathcal{F}$ = *0* **by** *auto*

then obtain *n* **where** *pi-result*: *insertion* (η *n*) Π = *0*
∧ *int n* = (*2* ∗ *R* − *1*) ∗ (*y* + *I* + T^2) − (*T div S*)^2
using *pi-relations*.Π*-encodes-correctly*[*of S I T R*, *OF* ‹*S* ≠ *0*› *F-zeros-bound*
‹*S dvd T*› ‹*R* > *0*›]
unfolding η*-def* Π*-def* **by** *auto*

show *?RHS*
apply (*rule exI*[*of* - *int n*], *simp*)
unfolding γ′*-def*
unfolding *M3*-Π*-equivalence*[*symmetric*]
using *pi-result* **unfolding** η*-def* **by** *auto*
qed

have *recipr-imp*: *?RHS* ⟶ *?LHS*
proof (*rule impI*)

assume *?RHS*
then obtain n **where** $n{\geq}0 \wedge M3\ \pi = 0$
unfolding γ'*-def* **by** *auto*

hence *insertion* $(\eta\ (nat\ n))\ \Pi = 0$
unfolding η*-def* **using** *M3*-Π*-equivalence* **by** *auto*

hence *pi-relations*: *S dvd T* $\wedge$ *0* $<$ *R* $\wedge$ $(\exists x.\ insertion\ (\lambda\text{-}.\ x)\ \mathcal{F} = 0)$
using *pi-relations*.Π*-encodes-correctly-2*[*of S I T R*, *OF* ‹*S* $\neq$ *0*› *F-zeros-bound*]
unfolding Π*-def*[*symmetric*] η*-def* **by** *blast*

show *?LHS* **using** *pi-relations insertion-F-squares* **by** *auto*
qed

then show *?thesis* **using** *direct-imp* **by** *blast*
qed

lemma *relation-combining*:
assumes *S*$\neq$*0*
shows (*S dvd T* $\wedge$ *R* $>$ *0* $\wedge$ *is-square* A_1 $\wedge$ *is-square* A_2 $\wedge$ *is-square* A_3)
$= (\exists n{\geq}0.\ M3\ A_1\ A_2\ A_3\ S\ T\ R\ n = 0)$
using *relation-combining*$'$ *assms* **by** *auto*
end

end
theory *Nine-Unknowns-N-Z-Definitions*
imports *../Coding-Theorem/Coding-Theorem-Definitions ../Bridge-Theorem/Bridge-Theorem-Imp*
M3-Polynomial ../Coding/Suitable-For-Coding ../MPoly-Utils/Poly-Degree
HOL−Eisbach.Eisbach
begin

8 Nine Unknowns over $\mathbb{N}$ and $\mathbb{Z}$

8.1 Combining all previous constructions

For any appropriately typed function F, we introduce the syntax *fn* $\tau \equiv$ *fn* *a f g h k l w x y n*, as well as $(\lambda\tau.\ e) \equiv (\lambda f\ a\ f\ g\ h\ k\ l\ w\ x\ y\ n.\ e)$.

syntax *tau* :: (*int* $\Rightarrow$ *int* $\Rightarrow$ *int* $\Rightarrow$ *int* $\Rightarrow$ *int* $\Rightarrow$ *int* $\Rightarrow$ *int* $\Rightarrow$ *int* $\Rightarrow$ *int* $\Rightarrow$ *int*) $\Rightarrow$ *int* (- τ [*1000*] *1000*)
syntax *tau-fn* :: (*int* $\Rightarrow$ *int* $\Rightarrow$ *int* $\Rightarrow$ *int* $\Rightarrow$ *int* $\Rightarrow$ *int* $\Rightarrow$ *int* $\Rightarrow$ *int* $\Rightarrow$ *int* $\Rightarrow$ *int* $\Rightarrow$ *int*) $\Rightarrow$ *int* $(\lambda\tau.$ - [*0*] *0*)

parse-translation ‹
[
(
syntax-const ‹*tau*›,
fn ctxt => fn args =>
list-comb (

```
        args |> hd,
        [a, f, g, h, k, l, w, x, y, n] |> map (fn name => Free (name, @{typ int}))
      )
  ),
  (
    syntax-const ‹tau-fn›,
    fn ctxt => fn args =>
      [a, f, g, h, k, l, w, x, y, n]
        |> List.foldr (fn (name, r) => Abs (name, @{typ int}, r)) (args |> hd)
  )
 ]
›

locale combined-variables =
  fixes P :: int mpoly
begin
```

Copy of the polynomials from Theorem I

```
definition P₁ :: int mpoly where
  P₁ ≡ suitable-for-coding P

abbreviation δ ≡ coding-variables.δ P₁
abbreviation ν ≡ coding-variables.ν P₁

definition b :: int ⇒ int ⇒ int ⇒ int ⇒ int ⇒ int ⇒ int ⇒ int ⇒ int ⇒
int
  where b τ = coding-variables.b a f
definition X :: int ⇒ int ⇒ int ⇒ int ⇒ int ⇒ int ⇒ int ⇒ int ⇒ int ⇒ int
⇒ int
  where X τ = coding-variables.X P₁ a f g
definition Y :: int ⇒ int ⇒ int ⇒ int ⇒ int ⇒ int ⇒ int ⇒ int ⇒ int ⇒ int
⇒ int
  where Y τ = coding-variables.Y P₁ a f

poly-extract b


definition Y-poly :: int mpoly where
  Y-poly ≡ coding-variables.Y-poly P₁
lemma Y-correct: insertion f Y-poly =  Y (f 0) (f 1) (f 2) (f 3) (f 4) (f 5) (f 6)
(f 7) (f 8) (f 9)
  unfolding Y-def coding-variables.Y-correct Y-poly-def by simp



definition X-poly :: int mpoly where
  X-poly ≡ coding-variables.X-poly P₁
lemma X-correct: insertion f X-poly =  X (f 0) (f 1) (f 2) (f 3) (f 4) (f 5) (f 6)
```

(*f 7*) (*f 8*) (*f 9*)
unfolding *X-def coding-variables.X-correct X-poly-def* **by** *simp*

lemma *δ-gt0*: $\delta > 0$
unfolding *coding-variables.δ-def* P_1*-def*
by (*simp add*: *suitable-for-coding-total-degree*)

Polynomials from Theorem II

definition $L :: int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
where $L\ \tau = bridge\text{-}variables.L\ l\ (Y\ \tau)$
definition $U :: int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
where $U\ \tau = bridge\text{-}variables.U\ l\ (X\ \tau)\ (Y\ \tau)$
definition $V :: int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
where $V\ \tau = bridge\text{-}variables.V\ w\ g\ (Y\ \tau)$
definition $A :: int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
where $A\ \tau = bridge\text{-}variables.A\ l\ w\ g\ (Y\ \tau)\ (X\ \tau)$
definition $C :: int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
where $C\ \tau = bridge\text{-}variables.C\ l\ w\ h\ g\ (Y\ \tau)\ (X\ \tau)$
definition $D :: int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
where $D\ \tau = bridge\text{-}variables.D\ l\ w\ h\ g\ (Y\ \tau)\ (X\ \tau)$
definition $F :: int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
where $F\ \tau = bridge\text{-}variables.F\ l\ w\ h\ x\ g\ (Y\ \tau)\ (X\ \tau)$
definition $I :: int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
where $I\ \tau = bridge\text{-}variables.I\ l\ w\ h\ x\ y\ g\ (Y\ \tau)\ (X\ \tau)$
definition $W :: int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
where $W\ \tau = bridge\text{-}variables.W\ w\ (coding\text{-}variables.b\ a\ f)$
definition $K :: int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
where $K\ \tau = bridge\text{-}variables.K\ k\ l\ w\ g\ (Y\ \tau)\ (X\ \tau)$

poly-extract *L*
poly-extract *U*
poly-extract *V*
poly-extract *A*
poly-extract *C*
poly-extract *D*
poly-extract *F*
poly-extract *I*

poly-extract *W*
poly-extract *K*

Variables in the proof of Theorem III

definition *M3* :: *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int*
  where *M3* A_1 A_2 A_3 *S T Q′ m* = *insertion-list* [A_1, A_2, A_3, *S*, *T*, *Q′*, *m*]
section5.M3-poly

poly-extract *M3*

lemma *max-vars-M3*: *max-vars M3-poly* ≤ *6*
  unfolding *M3-poly-def*
  using *max-vars-id*[*of 6*, *unfolded upt-def*]
  by *auto*

definition μ :: *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int*
  where $\mu\ \tau$ = (*coding-variables.*γ P_1) ∗ ($b\ \tau$) ^ (*coding-variables.*α P_1)

poly-extract μ

definition A_1 :: *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int*
  where $A_1\ \tau = b\ \tau$
definition A_2 :: *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int*
  where $A_2\ \tau = (D\ \tau) * (F\ \tau) * (I\ \tau)$
definition A_3 :: *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int*
  where $A_3\ \tau = ((U\ \tau)\ \hat{}\ 4 * (V\ \tau)^2 - 4) * (K\ \tau)^2 + 4$
definition *S* :: *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int*
  where $S\ \tau$ = *bridge-variables.S l w g* ($Y\ \tau$) ($X\ \tau$)
definition *T* :: *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int*
  where $T\ \tau$ = *bridge-variables.T l w h g* ($Y\ \tau$) ($X\ \tau$) ($b\ \tau$)
definition *R* :: *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int*
  where $R\ \tau = f^2 * l^2 * x^2 * (8 * (\mu\ \tau)\hat{}3 * g * (K\ \tau)^2 - g^2 * (32 * ((C\ \tau) - (K\ \tau) * (L\ \tau))\hat{}2 * (\mu\ \tau)\hat{}3 + g^2 * (K\ \tau)^2))$
definition *m* :: *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int*
  where $m\ \tau = n$

poly-extract A_1
poly-extract A_2
poly-extract A_3

poly-extract S
poly-extract T
poly-extract R
poly-extract m

definition $\sigma :: (int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int) \Rightarrow int \Rightarrow int$
$\Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
where $(\sigma\ fn)\ \tau = fn\ (A_1\ \tau)\ (A_2\ \tau)\ (A_3\ \tau)\ (S\ \tau)\ (T\ \tau)\ (R\ \tau)\ (m\ \tau)$

definition $Q :: int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
$\Rightarrow int$
where $Q\ \tau = M3\ (A_1\ \tau)\ (A_2\ \tau)\ (A_3\ \tau)\ (S\ \tau)\ (T\ \tau)\ (R\ \tau)\ (m\ \tau)$

lemma *M-poly-degree-correct*: $total\text{-}degree\ (coding\text{-}variables.M\text{-}poly\ P_1) \leq (1+(\delta+1)\ \hat{}\ \nu) * 2*\delta$
using *δ-gt0 coding-variables.M-poly-degree-correct* **by** *blast*

lemma *D-poly-degree-correct*: $total\text{-}degree\ (coding\text{-}variables.D\text{-}poly\ P_1) \leq (\delta+1)\ \hat{}\ (\nu+1) * (2*\delta)$
using *δ-gt0 coding-variables.D-poly-degree-correct* **by** *blast*

lemma *K-poly-degree-correct*: $total\text{-}degree\ (coding\text{-}variables.K\text{-}poly\ P_1)$
$\leq max\ (\delta*(1+2*\delta) + (\delta+1)\ \hat{}\ (\nu+1) * 2*\delta)\ ((1 + (2*\delta+1)*(\delta+1)\ \hat{}\ \nu) * 2*\delta)$
using *δ-gt0 coding-variables.K-poly-degree-correct* **by** *blast*

poly-degree *X-poly*
poly-degree *Y-poly*

lemma *X-poly-degree-alt*: $X\text{-}poly\text{-}degree = 12 * \delta + 12 * \delta * Suc\ \delta\ \hat{}\ \nu + 12 * \delta^2 * Suc\ \delta\ \hat{}\ \nu$
if $\delta > 0$
proof −
have *step0*: $X\text{-}poly\text{-}degree$
$= \delta + \delta + Suc\ \delta\ \hat{}\ \nu * (\delta + \delta) + (\delta + \delta + (Suc\ \delta\ \hat{}\ \nu + 2 * \delta * Suc\ \delta\ \hat{}\ \nu) * (\delta + \delta)) +$
$max\ (max\ (max\ (Suc\ 0)$
$(max\ (\delta + \delta * (2 * \delta) + (Suc\ \delta\ \hat{}\ \nu * 2 + \delta * Suc\ \delta\ \hat{}\ \nu * 2) * \delta)$
$(\delta + (\delta + (Suc\ \delta\ \hat{}\ \nu * 2 + 4 * (\delta * Suc\ \delta\ \hat{}\ \nu)) * \delta)) +$
$(\delta + \delta + Suc\ \delta\ \hat{}\ \nu * (\delta + \delta))))$
$(max\ (2 * \delta + Suc\ \delta\ \hat{}\ \nu * 2 * \delta)$
$(\delta + \delta + (Suc\ \delta\ \hat{}\ \nu + \delta * Suc\ \delta\ \hat{}\ \nu) * (\delta + \delta) +$
$(\delta + \delta + Suc\ \delta\ \hat{}\ \nu * (\delta + \delta)))) +$
$(\delta + \delta + Suc\ \delta\ \hat{}\ \nu * (\delta + \delta) +$
$(\delta + \delta + (Suc\ \delta\ \hat{}\ \nu + 2 * \delta * Suc\ \delta\ \hat{}\ \nu) * (\delta + \delta))))$
$(max\ (2 * \delta + Suc\ \delta\ \hat{}\ \nu * 2 * \delta)$
$(\delta + \delta + (Suc\ \delta\ \hat{}\ \nu + \delta * Suc\ \delta\ \hat{}\ \nu) * (\delta + \delta) + (\delta + \delta + Suc\ \delta\ \hat{}\ \nu * (\delta$

$+ \delta))))$
by (*simp add*: *X-poly-degree-def*)

have *step1*: $max\ (2 * \delta + Suc\ \delta\ \hat{}\ \nu * 2 * \delta)$
$(\delta + \delta + (Suc\ \delta\ \hat{}\ \nu + \delta * Suc\ \delta\ \hat{}\ \nu) * (\delta + \delta) + (\delta + \delta + Suc\ \delta\ \hat{}\ \nu * (\delta + \delta)))$
$= 4 * \delta + 2 * (\delta \hat{} 2 + 2 * \delta) * Suc\ \delta\ \hat{}\ \nu$
using *that* **apply** (*simp add*: *ab-semigroup-mult-class.mult-ac*(*1*) *mult-2*)
by (*smt* (*verit*, *del-insts*) *ab-semigroup-mult-class.mult-ac*(*1*) *add.commute add.left-commute*
mult.commute mult-2-right mult-Suc power2-eq-square)

have *step2*: $max\ (\delta + \delta * (2 * \delta) + (Suc\ \delta\ \hat{}\ \nu * 2 + \delta * Suc\ \delta\ \hat{}\ \nu * 2) * \delta)$
$(\delta + (\delta + (Suc\ \delta\ \hat{}\ \nu * 2 + 4 * (\delta * Suc\ \delta\ \hat{}\ \nu)) * \delta))$
$= 2 * \delta\ + 2 * \delta * Suc\ \delta\ \hat{}\ \nu + 4 * \delta^2 * Suc\ \delta\ \hat{}\ \nu$
proof −
have *aux1*: $max\ (\delta + (\delta * (\delta * 2) + (\delta * (2 * Suc\ \delta\ \hat{}\ \nu) + \delta * (\delta * (2 * Suc\ \delta\ \hat{}\ \nu)))))$
$(\delta * 2 + (\delta * (2 * Suc\ \delta\ \hat{}\ \nu) + \delta * (\delta * (4 * Suc\ \delta\ \hat{}\ \nu))))$
$= \delta + \delta * (2 * Suc\ \delta\ \hat{}\ \nu) +$
$max\ (\delta * (\delta * 2) + \delta * (\delta * (2 * Suc\ \delta\ \hat{}\ \nu)))\ (\delta + \delta * (\delta * (4 * Suc\ \delta\ \hat{}\ \nu)))$
using *nat-add-max-right* **by** *presburger*
have *aux2*: $max\ (\delta * (\delta * 2) + \delta * (\delta * (2 * Suc\ \delta\ \hat{}\ \nu)))\ (\delta + \delta * (\delta * (4 * Suc\ \delta\ \hat{}\ \nu))) =$
$2 * \delta \hat{} 2 * Suc\ \delta\ \hat{}\ \nu + max\ (2 * \delta \hat{} 2)\ (\delta + 2 * \delta \hat{} 2 * Suc\ \delta\ \hat{}\ \nu)$
by (*simp add*: *mult.commute mult.left-commute power2-eq-square*)
have *aux3*: $max\ (2 * \delta^2)\ (\delta + 2 * \delta^2 * Suc\ \delta\ \hat{}\ \nu) = \delta + 2 * \delta^2 * Suc\ \delta\ \hat{}\ \nu$
apply (*rule linorder-class.max.absorb2*)
by (*simp add*: *trans-le-add2*)
show *?thesis*
apply (*simp add*: *algebra-simps*)
unfolding *aux1 aux2 aux3*
by (*simp add*: *algebra-simps*)
qed

have *step3*: $max\ (Suc\ 0)\ (2 * \delta + 2 * \delta * Suc\ \delta\ \hat{}\ \nu + 4 * \delta^2 * Suc\ \delta\ \hat{}\ \nu + (\delta + \delta + Suc\ \delta\ \hat{}\ \nu * (\delta + \delta)))$
$= \delta * 4 + \delta * 4 * Suc\ \delta\ \hat{}\ \nu + 4 * \delta^2 * Suc\ \delta\ \hat{}\ \nu$
proof −
have $max\ (Suc\ 0)\ (2 * \delta + 2 * \delta * Suc\ \delta\ \hat{}\ \nu + 4 * \delta^2 * Suc\ \delta\ \hat{}\ \nu + (\delta + \delta + Suc\ \delta\ \hat{}\ \nu * (\delta + \delta)))$
$= (2 * \delta + 2 * \delta * Suc\ \delta\ \hat{}\ \nu + 4 * \delta^2 * Suc\ \delta\ \hat{}\ \nu + (\delta + \delta + Suc\ \delta\ \hat{}\ \nu * (\delta + \delta)))$
apply (*rule linorder-class.max.absorb2*)
using *Suc-leI add-gr-0 that* **by** *presburger*
thus *?thesis*
by (*simp add*: *algebra-simps*)
qed

```
  show ?thesis
    unfolding step0
    unfolding step1
    unfolding step2
    unfolding step3
    apply (simp add: algebra-simps)
    by algebra
qed

poly-degree A1-poly
poly-degree A2-poly
poly-degree A3-poly
poly-degree S-poly
poly-degree T-poly
poly-degree R-poly

lemma A1-vars: max-vars A1-poly ≤ 8
  unfolding A1-poly-def b-poly-def coding-variables.b-poly-def
  apply (rule le-trans[OF max-vars-poly-subst-list-bounded])
  by (rule le-trans[OF max-vars-poly-subst-list-general], auto)

lemma h0: max-vars (4::int mpoly) ≤ 8
  by (metis Const-numeral max-vars-Const zero-le)

lemma h1: max-vars (8::int mpoly) ≤ 8
  by (metis Const-numeral max-vars-Const zero-le)

lemma h2: max-vars (32::int mpoly) ≤ 8
  by (metis Const-numeral max-vars-Const zero-le)

lemma A2-vars: max-vars A2-poly ≤ 8
  unfolding A2-poly-def apply (rule le-trans[OF max-vars-mult], auto)+
  unfolding D-poly-def F-poly-def I-poly-def
 unfolding X-poly-def coding-variables.X-poly-def Y-poly-def coding-variables.Y-poly-def
  unfolding power2-eq-square
  apply (all ‹rule le-trans[OF max-vars-poly-subst-list-bounded[of - 8]]›, auto)
  apply (all ‹rule le-trans[OF max-vars-poly-subst-list-general]›, simp-all, all ‹in-
tro conjI›)
  apply (all ‹rule le-trans[OF max-vars-poly-subst-list-bounded[of - 8]]›, auto)
  by (rule le-trans[OF max-vars-poly-subst-list-general]
            le-trans[OF max-vars-pow]
            le-trans[OF max-vars-mult]
            le-trans[OF max-vars-add]
            le-trans[OF max-vars-diff], auto)+

lemma A3-vars: max-vars A3-poly ≤ 8
  unfolding A3-poly-def power2-eq-square
  apply (rule le-trans[OF max-vars-pow]
            le-trans[OF max-vars-mult]
```

```
        le-trans[OF max-vars-add]
        le-trans[OF max-vars-diff]; auto simp: h0)+
  apply (all ‹rule le-trans[OF max-vars-pow] le-trans[OF max-vars-mult]›, auto)
  apply (all ‹rule le-trans[OF max-vars-poly-subst-list-bounded[of - 8]]›, auto)
  unfolding U-poly-def V-poly-def K-poly-def
 unfolding X-poly-def coding-variables.X-poly-def Y-poly-def coding-variables.Y-poly-def
  unfolding power2-eq-square
  apply (all ‹rule le-trans[OF max-vars-poly-subst-list-general]›, auto)
  apply (all ‹rule le-trans[OF max-vars-poly-subst-list-bounded[of - 8]]›, auto)
  by (rule le-trans[OF max-vars-poly-subst-list-general]
        le-trans[OF max-vars-pow]
        le-trans[OF max-vars-mult]
        le-trans[OF max-vars-add]
        le-trans[OF max-vars-diff], auto)+

lemma S-vars: max-vars S-poly ≤ 8
  unfolding S-poly-def
 unfolding X-poly-def coding-variables.X-poly-def Y-poly-def coding-variables.Y-poly-def
  unfolding power2-eq-square
  apply (all ‹rule le-trans[OF max-vars-poly-subst-list-general]›, auto)
  apply (all ‹rule le-trans[OF max-vars-poly-subst-list-bounded[of - 8]]›, auto)
  by (rule le-trans[OF max-vars-poly-subst-list-general]
        le-trans[OF max-vars-pow]
        le-trans[OF max-vars-mult]
        le-trans[OF max-vars-add]
        le-trans[OF max-vars-diff], auto)+

lemma T-vars: max-vars T-poly ≤ 8
  unfolding T-poly-def b-poly-def coding-variables.b-poly-def
 unfolding X-poly-def coding-variables.X-poly-def Y-poly-def coding-variables.Y-poly-def
  unfolding power2-eq-square
  apply (all ‹rule le-trans[OF max-vars-poly-subst-list-general]›, auto)
  apply (all ‹rule le-trans[OF max-vars-poly-subst-list-bounded[of - 8]]›, auto)
  by (rule le-trans[OF max-vars-poly-subst-list-general]
        le-trans[OF max-vars-pow]
        le-trans[OF max-vars-mult]
        le-trans[OF max-vars-add]
        le-trans[OF max-vars-diff], auto)+

lemma K-vars: max-vars K-poly ≤ 8
  unfolding K-poly-def
 unfolding X-poly-def coding-variables.X-poly-def Y-poly-def coding-variables.Y-poly-def
power2-eq-square
  apply ((rule le-trans[OF max-vars-pow]
          le-trans[OF max-vars-mult max.boundedI]
          le-trans[OF max-vars-add max.boundedI]
          le-trans[OF max-vars-diff' max.boundedI])
         | simp-all add: h1)+
  apply (all ‹rule le-trans[OF max-vars-poly-subst-list-general]›, auto)
```

apply (*all* ‹*rule le-trans*[*OF max-vars-poly-subst-list-bounded*[*of - 8*]]›, *auto*)
by (*rule le-trans*[*OF max-vars-poly-subst-list-general*]
le-trans[*OF max-vars-pow*]
le-trans[*OF max-vars-mult*]
le-trans[*OF max-vars-add*]
le-trans[*OF max-vars-diff*], *auto*)+

lemma *L-vars*: *max-vars L-poly* ≤ *8*
unfolding *L-poly-def*
unfolding *X-poly-def coding-variables.X-poly-def Y-poly-def coding-variables.Y-poly-def power2-eq-square*
apply ((*rule le-trans*[*OF max-vars-pow*]
le-trans[*OF max-vars-mult max.boundedI*]
le-trans[*OF max-vars-add max.boundedI*]
le-trans[*OF max-vars-diff' max.boundedI*])
| *simp-all add*: *h1*)+
apply (*all* ‹*rule le-trans*[*OF max-vars-poly-subst-list-general*]›, *auto*)
apply (*all* ‹*rule le-trans*[*OF max-vars-poly-subst-list-bounded*[*of - 8*]]›, *auto*)
by (*rule le-trans*[*OF max-vars-poly-subst-list-general*]
le-trans[*OF max-vars-pow*]
le-trans[*OF max-vars-mult*]
le-trans[*OF max-vars-add*]
le-trans[*OF max-vars-diff*], *auto*)+

lemma *C-vars*: *max-vars C-poly* ≤ *8*
unfolding *C-poly-def*
unfolding *X-poly-def coding-variables.X-poly-def Y-poly-def coding-variables.Y-poly-def power2-eq-square*
apply ((*rule le-trans*[*OF max-vars-pow*]
le-trans[*OF max-vars-mult max.boundedI*]
le-trans[*OF max-vars-add max.boundedI*]
le-trans[*OF max-vars-diff' max.boundedI*])
| *simp-all add*: *h1*)+
apply (*all* ‹*rule le-trans*[*OF max-vars-poly-subst-list-general*]›, *auto*)
apply (*all* ‹*rule le-trans*[*OF max-vars-poly-subst-list-bounded*[*of - 8*]]›, *auto*)
by (*rule le-trans*[*OF max-vars-poly-subst-list-general*]
le-trans[*OF max-vars-pow*]
le-trans[*OF max-vars-mult*]
le-trans[*OF max-vars-add*]
le-trans[*OF max-vars-diff*], *auto*)+

lemma *μ-vars*:
max-vars (*poly-subst-list* [*Var 0*, *Var* (*Suc 0*), *Var 2*, *Var 3*, *Var 4*, *Var 5*, *Var 6*, *Var 7*, *Var 8*, *Var 9*] *μ-poly*) ≤ *8*
apply (*all* ‹*rule le-trans*[*OF max-vars-poly-subst-list-bounded*[*of - 8*]]›, *auto*)
unfolding *μ-poly-def*
unfolding *X-poly-def coding-variables.X-poly-def Y-poly-def coding-variables.Y-poly-def*

b-poly-def power2-eq-square

apply ((*rule le-trans*[*OF max-vars-pow*]
le-trans[*OF max-vars-mult max.boundedI*]
le-trans[*OF max-vars-add max.boundedI*]
le-trans[*OF max-vars-diff' max.boundedI*])
| *simp-all add*: *h1*)+
apply (*all* ‹*rule le-trans*[*OF max-vars-poly-subst-list-bounded*[*of - 8*]]›, *auto*)
by (*all* ‹*rule le-trans*[*OF max-vars-poly-subst-list-general*]›, *auto*)

lemma *R-vars*: *max-vars R-poly* ≤ *8*
unfolding *R-poly-def*
unfolding *power2-eq-square*
by ((*rule le-trans*[*OF max-vars-pow*]
le-trans[*OF max-vars-mult max.boundedI*]
le-trans[*OF max-vars-add max.boundedI*]
le-trans[*OF max-vars-diff' max.boundedI*])
| *simp-all add*: *h1 h2 μ-vars*
| *rule le-trans*[*OF max-vars-poly-subst-list-bounded*[*of - 8*]], *simp add*: *K-vars C-vars L-vars*)+

lemma *list-vars*: *i* ≤ *8* ⟹ *max-vars*
([*Var 0*::*int mpoly*, *Var* (*Suc 0*), *Var* (*Suc* (*Suc 0*)), *Var* (*Suc* (*Suc* (*Suc 0*))),
Var (*Suc* (*Suc* (*Suc* (*Suc 0*)))), *Var* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc 0*))))),
Var (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc 0*)))))),
Var (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc 0*))))))),
Var (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc 0*)))))))),
Var (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc 0*))))))))),
Var (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc 0*)))))))))] !$_0$
i)
≤ *8*
apply (*simp add*: *nth0-def*)
by (*smt* (*z3*) *add.right-neutral add-Suc-right le-Suc-eq less-eq-nat.simps*(*1*) *max-vars-Var nle-le*
nth-Cons-0 nth-Cons-Suc numeral-1-eq-Suc-0 numeral-Bit0)

lemmas *aux-vars* = *A*$_1$*-vars A*$_2$*-vars A*$_3$*-vars S-vars T-vars R-vars*

lemma *total-degree-three-squares-rw*:
fixes *Pextra* :: *'a*::*comm-semiring-1 mpoly*
shows *ia* ≤ *8* ⟹
total-degree-env env
([*Var 0*, *Var* (*Suc 0*), *Var* (*Suc* (*Suc 0*)),
Var (*Suc* (*Suc* (*Suc 0*))), *Var* (*Suc* (*Suc* (*Suc* (*Suc 0*)))),
Var (*Suc* (*Suc* (*Suc* (*Suc* (*Suc 0*))))),
Var (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc 0*)))))),
Var (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc 0*))))))),
Var (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc 0*)))))))),

```
        Pextra] !0
       ia)
    = total-degree-env env
     ([Var 0 :: 'a mpoly , Var (Suc 0), Var (Suc (Suc 0)), Var (Suc (Suc (Suc
0))),
       Var (Suc (Suc (Suc (Suc 0)))), Var (Suc (Suc (Suc (Suc (Suc 0))))),
       Var (Suc (Suc (Suc (Suc (Suc (Suc 0)))))),
       Var (Suc (Suc (Suc (Suc (Suc (Suc (Suc 0)))))))),
       Var (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc 0))))))))] !0
       ia)
  unfolding total-degree-env-id[symmetric]
  apply (drule le-imp-less-Suc)
   using total-degree-env-reduce[of ia map Var [0::nat, 1, 2, 3, 4, 5, 6, 7, 8],
simplified]
  by (metis One-nat-def Suc-1 Suc-numeral semiring-norm(2,5,8))

lemma final: ⋀ia. ia ≤ 8 ⟹
       total-degree-env (λ-. Suc 0)
        ([Var 0::int mpoly, Var (Suc 0), Var (Suc (Suc 0)), Var (Suc (Suc (Suc
0))),
          Var (Suc (Suc (Suc (Suc 0)))), Var (Suc (Suc (Suc (Suc (Suc 0))))),
          Var (Suc (Suc (Suc (Suc (Suc (Suc 0)))))),
          Var (Suc (Suc (Suc (Suc (Suc (Suc (Suc 0))))))),
          Var (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc 0)))))))),
         Var (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc 0))))))))))
+
           (Var (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc 0)))))))))) *
            Var (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc 0)))))))))) +
           (Var (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc 0))))))))))) *
            Var (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc 0))))))))))) +
                 Var (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc
0)))))))))))) *
                 Var (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc
0))))))))))))] !0
          ia)
       ≤ Suc 0
  apply (simp add: total-degree-three-squares-rw)
  by (metis (no-types, lifting) Nil-is-map-conv One-nat-def le-add1 list.simps(9)
plus-1-eq-Suc
     total-degree-env-mono3-map-Var)


poly-extract Q


lemma Q-alt: Q = σ M3
  unfolding Q-def σ-def ..

lemma R-gt-0-consequences:
```

```
  fixes a :: nat and f g h k l w n :: int
  assumes R τ > 0 and b τ ≥ 0 and f > 0
  shows g ≥ 1 and g < 2 * μ τ and K τ ≠ 0
    and bridge-variables.d2f k l w h g (Y τ) (X τ)
proof −
  have helper: 0 < int α * β ^ γ if α > 0 and β > 0 for α β γ
    by (simp add: that(1−2))

  have b τ > 0
  proof −
    have b τ ≠ 0
    unfolding b-def coding-variables.b-def
    using ‹f > 0› apply simp
    by (smt (verit, best) mult-pos-pos of-nat-less-0-iff)

    thus b τ > 0 using ‹b τ ≥ 0› by auto
  qed

  have μ τ > 0
    unfolding μ-def
    using helper[of coding-variables.γ P₁ b τ,
                 OF coding-variables.γ-gt-0 ‹b τ > 0›]
    unfolding b-def by auto
  hence real-of-int (8*(μ τ)^3) > 0
    by auto

  let ?R' = 8 * (μ τ)^3 * g * (K τ)^2 − g^2 * (32 * ((C τ) − (K τ)*(L τ))^2
* (μ τ)^3 + g^2 * (K τ)^2)
  have R-gt-0-condition: real-of-int ?R' > 0
  proof −
    have rewrite: f^2 * l^2 * x^2 = (f * l * x) ^2
      by (simp add: power-mult-distrib)

    have helper: 0 < α^2 * β ⟹ 0 < β for α β :: int
      by (smt (verit) mult-nonneg-nonpos power2-less-0)

    have ?R' > 0
      using ‹R τ > 0›
      unfolding R-def μ-def K-def L-def C-def
      unfolding rewrite using helper[of f * l * x]
      by metis

    thus ?thesis
      using of-int-pos by blast
  qed

  hence g ≠ 0
    by (auto)
  hence real-of-int (g^2) > 0
```

```
    by auto

  have inequality-divided:
    (K τ)^2 / g > 4 * ((C τ) - (K τ)*(L τ))^2 + (g^2 * (K τ)^2) / (8*(μ
τ)^3)
    (is ?A / g > ?B + ?C / (8*(μ τ)^3))
  proof -
    have 0 < (8*(μ τ)^3) * g * (K τ)^2 / (8*(μ τ)^3) - g^2 * (32 * ((C τ) -
(K τ)*(L τ))^2 * (μ τ)^3 + g^2 * (K τ)^2) / (8*(μ τ)^3)
      using divide-strict-right-mono[of 0 real-of-int ?R' 8*(μ τ)^3, OF R-gt-0-condition
‹real-of-int (8*(μ τ)^3) > 0›]
        by (simp add: diff-divide-distrib)
    hence aux1: g * (K τ)^2 > g^2 * (32 * ((C τ) - (K τ)*(L τ))^2 * (μ τ)^3
+ g^2 * (K τ)^2) / (8*(μ τ)^3)
      using ‹0 < μ τ› by auto

    have (g * (K τ)^2) / g^2 > (32 * ((C τ) - (K τ)*(L τ))^2 * (μ τ)^3 +
g^2 * (K τ)^2) / (8*(μ τ)^3)
      using divide-strict-right-mono[OF aux1 ‹real-of-int (g^2) > 0›]
      by (auto simp: ‹g ≠ 0›)
    hence (K τ)^2 / g > 4 * ((C τ) - (K τ)*(L τ))^2 + (g^2 * (K τ)^2) /
(8*(μ τ)^3)
      using ‹μ τ > 0›
    by (auto simp add: add-divide-distrib) (metis power2-eq-square real-divide-square-eq)
    then show ?thesis
      by auto
  qed

  have 1: 4 * ((C τ) - (K τ)*(L τ))^2 + (g^2 * (K τ)^2) / (8*(μ τ)^3)
          ≥ (g^2 * (K τ)^2) / (8*(μ τ)^3)
      by simp

  have (K τ)^2 / g > 0
  proof -
    have 2: (g^2 * (K τ)^2) / (8*(μ τ)^3) ≥ 0
      using ‹(μ τ) > 0› by (simp add: pos-imp-zdiv-nonneg-iff)

    show ?thesis using 1 2 inequality-divided
      by linarith
  qed

  show (K τ) ≠ 0
    using ‹(K τ)^2 / g > 0› by fastforce


  have g > 0
    using ‹(K τ)^2 / g > 0› ‹(K τ) ≠ 0›
    by (simp add: zero-less-divide-iff)
  then show g ≥ 1
```

```
    using ‹g ≠ 0› by auto
  from ‹g > 0› have real-of-int g > 0
    by auto

  show g < 2 * μ τ
  proof −
    from inequality-divided have (K τ)^2 / g > (g^2 * (K τ)^2) / (8*(μ τ)^3)
      by auto (smt (verit) power2-less-0)
    hence (8*(μ τ)^3) * (K τ)^2 > real-of-int g^3 * (K τ)^2
      using ‹real-of-int (8*(μ τ)^3) > 0› using ‹real-of-int g > 0›
      by (auto simp: divide-simps algebra-simps power2-eq-square power3-eq-cube)
    hence real-of-int g < 2 * μ τ
      apply (simp add: ‹(K τ) ≠ 0›)
      apply (rule power-less-imp-less-base[of - 3])
      using ‹μ τ > 0› by auto
    then show ?thesis
      by auto
  qed

  from inequality-divided have 4 * ((C τ) − (K τ)*(L τ))^2 < (K τ)^2 / g
    using ‹g > 0› ‹μ τ > 0› ‹(K τ) ≠ 0› apply simp
    by (smt (verit, ccfv-SIG) diff-divide-distrib mult-sign-intros(1)
          of-int-1-le-iff power2-less-0 zero-le-power-eq-numeral zero-less-divide-iff)
  also have ... ≤ (K τ)^2
    using ‹g ≥ 1›
    by (auto simp: algebra-simps divide-simps mult-le-cancel-right1)
  finally have 4 * ((C τ) − (K τ)*(L τ))^2 < (K τ)^2
    by linarith
  then have (2 * (C τ) − 2 * (K τ)*(L τ))^2 < (K τ)^2
    by (auto simp: algebra-simps power2-eq-square)
  then show bridge-variables.d2f k l w h g (Y τ) (X τ)
    unfolding bridge-variables.d2f-def C-def L-def K-def
    by (auto simp: algebra-simps)
qed

lemma R-gt-0-necessary-condition:
  fixes a :: nat and f g h k l w x y :: int
  assumes f > 0 and x > 0 and l > 0 and g > 0 and g < μ τ and
        bridge-variables.d2e k l w h g (Y τ) (X τ)
  shows R τ > 0
proof −
  from assms have μ τ > 0
    by auto

  have (K τ)^2 ≥ 0
    by auto

  have 0: (4 * g * (C τ − K τ * L τ))^2 < (K τ)^2
    using assms(6)
```

unfolding *bridge-variables.d2e-def C-def K-def L-def*
by (*auto simp*: *algebra-simps*)
hence $(K\ \tau)\hat{}2 > 0$
by *auto*

from *0* **have** $2 * 4\hat{}2 * g\hat{}2 * (C\ \tau - K\ \tau * L\ \tau)\hat{}2 < 2 * (K\ \tau)\hat{}2$
by (*auto simp*: *algebra-simps power2-eq-square*)

moreover have $8{*}g\hat{}4\ /\ (8{*}(\mu\ \tau)\hat{}3) * (K\ \tau)\hat{}2 \leq g * (K\ \tau)\hat{}2$
proof −
have $8{*}g\hat{}4\ /\ (8{*}(\mu\ \tau)\hat{}3) < g$
apply (*simp add*: *power2-eq-square* ‹$g < \mu\ \tau$›)
using ‹$g < \mu\ \tau$› ‹$g > 0$› ‹$\mu\ \tau > 0$›
apply (*simp add*: *algebra-simps divide-simps power2-eq-square power4-eq-xxxx*)
using ‹$g < \mu\ \tau$› **apply** *simp*
unfolding *real-of-int-strict-inequality power3-eq-cube*
apply (*simp add*: *mult.commute mult.assoc mult-strict-right-mono*)
by (*smt* (*verit*, *best*) *mult-le-less-imp-less mult-nonneg-nonneg of-int-le-0-iff real-of-int-inequality*)
thus *?thesis*
using ‹$(K\ \tau)\hat{}2 \geq 0$›
apply *simp*
by (*metis less-eq-real-def mult-right-mono times-divide-eq-left zero-le-power2*)
qed

ultimately have $2 * 4\hat{}2 * g\hat{}2 * (C\ \tau - K\ \tau * L\ \tau)\hat{}2 + 8{*}g\hat{}4\ /\ (8{*}(\mu\ \tau)\hat{}\ 3) * (K\ \tau)\hat{}2 < 2 * (K\ \tau)\hat{}2 + g * (K\ \tau)\hat{}2$
unfolding *real-of-int-strict-inequality* **by** *auto*

also have $\ldots < 8 * g * (K\ \tau)\hat{}2$
using ‹$g > 0$› ‹$(K\ \tau)\hat{}2 > 0$› **by** (*auto simp*: *algebra-simps*)

finally have $4 * g\hat{}2 * (C\ \tau - K\ \tau * L\ \tau)\hat{}2 + g\hat{}4\ /\ (8{*}(\mu\ \tau)\ \hat{}\ 3) * (K\ \tau)\hat{}2 < g * (K\ \tau)\hat{}2$
by (*auto simp*: *algebra-simps*)

hence $8 * (\mu\ \tau)\hat{}3 * 4 * g\hat{}2 * (C\ \tau - K\ \tau * L\ \tau)\hat{}2 + g\hat{}4 * (K\ \tau)\hat{}2 < 8 * (\mu\ \tau)\hat{}3 * g * (K\ \tau)\hat{}2$
using ‹$\mu\ \tau > 0$› **unfolding** *real-of-int-strict-inequality*
by (*auto simp*: *divide-simps algebra-simps power2-eq-square*)

hence $8 * (\mu\ \tau)\hat{}3 * g * (K\ \tau)^2 - g^2 * (32 * (C\ \tau - K\ \tau * L\ \tau)\hat{}2 * (\mu\ \tau)\hat{}3 + g^2 * (K\ \tau)^2) > 0$
by (*auto simp*: *divide-simps power2-eq-square power4-eq-xxxx algebra-simps*)

thus *?thesis* **unfolding** *R-def* **using** *assms* **by** *auto*
qed

end

end
theory *Nine-Unknowns-N-Z*
imports *../Coding-Theorem/Coding-Theorem ../Bridge-Theorem/Bridge-Theorem*
../Coding-Theorem/Lemma-2-2 ../Coding-Theorem/Lower-Bounds
Nine-Unknowns-N-Z-Definitions Matiyasevich-Polynomial
begin

8.2 Proof of the Nine Unknowns Theorem

theorem *theorem-III-new-statement*:
fixes A :: *nat set*
and P
defines $\varphi\ a\ z_1\ z_2\ z_3\ z_4\ z_5\ z_6\ z_7\ z_8\ z_9 \equiv \lambda fn.\ fn\ (int\ a)\ z_1\ z_2\ z_3\ z_4\ z_5\ z_6\ z_7\ z_8\ z_9$
assumes *is-diophantine-over-N-with A P*
shows $a \in A = (\exists\, z_1\ z_2\ z_3\ z_4\ z_5\ z_6\ z_7\ z_8\ z_9.\ z_9 \geq 0 \wedge$
$\varphi\ a\ z_1\ z_2\ z_3\ z_4\ z_5\ z_6\ z_7\ z_8\ z_9\ (combined\text{-}variables.Q\ P) = 0)$
(**is** - = *?Pa-zero*)
proof −
interpret *thm3*: *combined-variables P*
by *unfold-locales*

show *?thesis*
proof (*rule iffI*)
assume $a \in A$

then obtain ys_0 **where** ys_0*-insertion*: *insertion* $(ys_0(0 := int\ a))\ P = 0$ **and**
ys_0*-nonnegative*: *is-nonnegative* ys_0
using ‹*is-diophantine-over-N-with A P*› **unfolding** *is-diophantine-over-N-with-def*
by *auto*

obtain *ys* **where** *ys-insertion*: *insertion* $(ys(0 := int\ a))\ (thm3.P_1) = 0$
and *ys-nonzero-unknown*: $\exists\, i{\in}\{1..fresh\text{-}var\ P\}.\ 0 < ys\ i$
and *ys-zero*: $\forall\, i{>}fresh\text{-}var\ P.\ ys\ i = 0$
and *ys-nonnegative*: *is-nonnegative ys*
unfolding $thm3.P_1$*-def*
using *suitable-for-coding-exists-positive-unknown*[*of A P a* ys_0,
OF ‹*is-diophantine-over-N-with A P*› ‹$a \in A$› ys_0*-insertion* ys_0*-nonnegative*]
by *auto*

define ys' **where** $ys' \equiv ys(0 := int\ a)$

hence ys': *insertion* ys' $thm3.P_1 = 0$
using *ys-insertion* **by** *auto*

have *vars-ne*: *vars* $thm3.P_1 \neq \{\}$
unfolding $thm3.P_1$*-def suitable-for-coding-degree-vars*
by *auto*

```
  have max-vars-P: max-vars thm3.P₁ = fresh-var P
    unfolding max-vars-of-nonempty[OF vars-ne]
    unfolding thm3.P₁-def suitable-for-coding-degree-vars
    unfolding fresh-var-def
    apply simp
    using ‹max-vars thm3.P₁ = Max (vars thm3.P₁)› combined-variables.P₁-def
      fresh-var-def suitable-for-coding-degree-vars(2) suitable-for-coding-max-vars
    by force

  define ys-max :: int where ys-max = Max (ys' ` {0..fresh-var P})

  have ys-max ≥ 0
  proof −
    have ys-max ≥ ys' 0
      unfolding ys-max-def by auto
    thus ?thesis
      unfolding ys'-def by simp
  qed

  have ys' i ≤ ys-max for i
  proof (cases i ≤ fresh-var P)
    case True
    then show ?thesis
      unfolding ys-max-def ys'-def by auto
  next
    case False
    then show ?thesis
      using ys-zero ‹ys-max ≥ 0›
      unfolding ys-max-def ys'-def
      by simp
  qed

  define b₀ where b₀ ≡ coding-variables.b a
  have ∃f. f > 0 ∧ is-square (b₀ f) ∧ is-power2 (b₀ f) ∧ b₀ f > ys-max
    using lemma-2-2-useful
    unfolding b₀-def thm3.b-def coding-variables.b-def
     using insertion-mult insertion-add insertion-Var insertion-Const ‹ys-max ≥
0› by auto

  then obtain f where f > 0 and is-square (b₀ f) and is-power2 (b₀ f)
                and b₀ f > ys-max
    by auto

interpret coding: coding-theorem thm3.P₁ a f
  proof (unfold-locales)
    show 0 ≤ int a by simp

    show 0 < f
      using ‹0 < f› by auto
```

```
    show is-power2 (coding-variables.b (int a) f)
      using ‹is-power2 (b₀ f)› unfolding b₀-def
      unfolding coding-variables.b-def
      by auto

    show 0 < coding-variables.δ thm3.P₁
      unfolding thm3.P₁-def coding-variables.δ-def
      using suitable-for-coding-total-degree by auto
  qed

  have p0: coding.P-coeff (replicate (coding.ν + 1) 0) > 0
    using suitable-for-coding-coeff0[of P]
    unfolding coding.P-coeff-def thm3.P₁-def coding-variables.P-coeff-def
          coding-variables.ν-def
    by simp

  have coding-1: coding.statement1-strong ys'
    unfolding coding.statement1-strong-def coding.statement1-weak-def
  proof (intro conjI allI)
    show ys' 0 = int a
      unfolding ys'-def by auto

    show 0 ≤ ys' i for i
      using ys-nonnegative unfolding is-nonnegative-def ys'-def by auto

    show ys' i < coding.b for i
      using ‹b₀ f > ys-max› ‹ys' i ≤ ys-max›
      unfolding b₀-def
      by (auto simp: coding.b-def)

    show insertion ys' thm3.P₁ = 0
      using ys' by auto

    show ∃ i∈{1..coding.ν}. ys' i ≠ 0
    proof −
      from ys-nonzero-unknown obtain i where i ∈ {1..fresh-var P} and ys-i: ys
i > 0 by auto
      hence ys' i ≠ 0 unfolding ys'-def by auto
      thus ?thesis
        using ‹i ∈ {1..fresh-var P}› ‹vars thm3.P₁ ≠ {}›
        unfolding coding.ν-def by (auto simp: max-vars-P)
    qed
  qed


  hence ∃ g. coding.statement2-strong g
    using coding-theorem.coding-theorem-direct[OF coding.coding-theorem-axioms]
by auto
```

then obtain g :: *int* **where** g: *coding.statement2-strong g* **by** *auto*

define b **where** $b \equiv$ *thm3.b a f g 0 0 0 0 0 0 0*
define X **where** $X \equiv$ *thm3.X a f g 0 0 0 0 0 0 0*
define Y **where** $Y \equiv$ *thm3.Y a f g 0 0 0 0 0 0 0*

from g **have** $g \geq b$
  unfolding *coding.statement2-strong-def*
  unfolding *b-def thm3.b-def*
  by (*auto simp*: *le-trans coding.b-def*)
from g **have** $g < coding.\gamma * b \hat{} coding.\alpha$
  unfolding *coding.statement2-strong-def*
  unfolding *b-def thm3.b-def*
  by (*auto simp*: *le-trans coding.b-def*)

have *is-square b*
  using ‹*is-square* (b_0 *f*)›
  unfolding b_0*-def b-def thm3.b-def*
  by *auto*

have *is-power2 b*
  using ‹*is-power2* (b_0 *f*)›
  unfolding b_0*-def b-def thm3.b-def*
  by *simp*

have *1*: $b \geq 0$
  using ‹*is-square b*› **unfolding** *is-square-def*
  by *auto*

have $b \geq 1$
  using ‹b_0 *f* > *ys-max*› ‹*ys-max* ≥ *0*›
  unfolding *coding.statement1-strong-def* b_0*-def b-def thm3.b-def*
  by *auto*

have *4*: $1 \leq g$
  using *g* ‹$b \geq 1$›
  unfolding *coding.statement2-strong-def b-def thm3.b-def*
  by (*auto simp*: *le-trans coding.b-def*)
hence $0 < g$ **by** *auto*

have *bridge-variables.statement2a b Y X g*
proof −
  from ‹$0 < g$› **have** $0 \leq g$ **by** *auto*
  have $0 \leq$ *int a* **by** *auto*
  have *2*: $b \leq Y \wedge 2 \hat{} 8 \leq Y$
    using *coding.lower-bounds*[*OF* ‹$0 \leq$ *int a*› ‹$0 < f$› ‹$0 \leq g$› *coding.δ-pos p0*]
    unfolding *b-def Y-def thm3.b-def thm3.Y-def* **by** *auto*

```
  have 3: 3 * b ≤ X
   using coding.lower-bounds[OF ‹0 ≤ int a› ‹0 < f› ‹0 ≤ g› coding.δ-pos p0]
   unfolding b-def X-def thm3.b-def thm3.X-def coding.b-def
   by (auto simp: coding.b-def)

  have Y dvd int (2 * nat X choose nat X)
  proof −
   have coding.Y dvd int (2 * nat (coding.X g) choose nat (coding.X g))
    using g unfolding coding.statement2-strong-def by auto

   thus ?thesis
    unfolding Y-def thm3.Y-def X-def thm3.X-def by auto
  qed

  hence statement1: bridge-variables.statement1 b Y X
   unfolding bridge-variables.statement1-def
   using ‹is-power2 b› by simp

  show ?thesis
   using bridge-variables.theorem-II-1-3[of b Y X g, OF 1 2 3 4 statement1] by
auto
 qed

 then obtain h k l w x y where thm2:
  bridge-variables.d2a l w h x y g Y X ∧
  bridge-variables.d2b k l w x g Y X ∧
  bridge-variables.d2c l w h b g Y X ∧
  bridge-variables.d2e k l w h g Y X ∧
  b ≤ h ∧ b ≤ k ∧ b ≤ l ∧ b ≤ w ∧ b ≤ x ∧ b ≤ y
  using bridge-variables.statement2a-def[of b Y X g] by metis


 define z0 where z0 ≡ λfn. fn (int a) f g h k l w x y (0 :: int) :: int

 define S where S ≡ z0 thm3.S
 define T where T ≡ z0 thm3.T
 define R where R ≡ z0 thm3.R
 define A1 where A1 ≡ z0 thm3.A1
 define A2 where A2 ≡ z0 thm3.A2
 define A3 where A3 ≡ z0 thm3.A3

 define γ' where
  γ' ≡ λ(n :: int) fn. fn A1 A2 A3 S T R n :: int


 have ∃n≥0. γ' n thm3.M3 = 0
 proof −
  have thm3-Y: thm3.Y (int a) f g h k l w x y 0 = Y
```

```
    unfolding thm3.Y-def Y-def ..
  have thm3-X: thm3.X (int a) f g h k l w x y 0 = X
    unfolding thm3.X-def X-def ..
  have thm3-b: thm3.b (int a) f g h k l w x y 0 = b
    unfolding thm3.b-def b-def ..

  from thm2 have d2a: bridge-variables.d2a l w h x y g Y X by simp
  from thm2 have d2b: bridge-variables.d2b k l w x g Y X by simp
  from thm2 have d2c: bridge-variables.d2c l w h b g Y X by simp
  from thm2 have d2e: bridge-variables.d2e k l w h g Y X by simp

  have S ≠ 0
    unfolding S-def z0-def thm3.S-def bridge-variables.S-def by presburger

  have S dvd T
   using d2c unfolding bridge-variables.d2c-def S-def T-def thm3.S-def thm3.T-def
z0-def
    by (simp add: thm3-Y thm3-X thm3-b)

  have R > 0
  proof −
    have l > 0 using thm2 ‹b ≥ 1› by auto
    have x > 0 using thm2 ‹b ≥ 1› by auto

    have g < thm3.μ (int a) f g h k l w x y 0
      unfolding thm3.μ-def coding.b-def
      using ‹g < coding.γ * b ^ coding.α›
      unfolding b-def thm3.b-def .

    show ?thesis
      using thm3.R-gt-0-necessary-condition[OF ‹f > 0› ‹x > 0› ‹l > 0› ‹g >
0›
                                              ‹g < thm3.μ (int a) f g h k l w x y 0›]
      unfolding R-def z0-def
      using d2e unfolding X-def Y-def thm3.Y-def thm3.X-def by auto
  qed

  have is-square A1
    unfolding A1-def z0-def thm3.A1-def thm3-b
    using ‹is-square b›
    by simp
  have is-square A2
    unfolding A2-def z0-def thm3.A2-def
   using d2a unfolding bridge-variables.d2a-def thm3.D-def thm3.I-def thm3.F-def
    by (simp add: thm3-Y thm3-X thm3-b)
  have is-square A3
    unfolding A3-def z0-def thm3.A3-def
   using d2b unfolding bridge-variables.d2b-def thm3.U-def thm3.V-def thm3.K-def
    by (simp add: thm3-Y thm3-X thm3-b)
```

```
    obtain n where n-def: n≥0 ∧ section5.M3 A₁ A₂ A₃ S T R n = 0
      using matiyasevich-polynomial.relation-combining[of S T R A₁ A₂ A₃, OF
‹S ≠ 0›]
      using ‹S dvd T› ‹R > 0› ‹is-square A₁› ‹is-square A₂› ‹is-square A₃›
      by auto

    show ?thesis
      apply (rule exI[of - n], simp add: n-def)
      unfolding thm3.M3-def γ'-def apply simp
      unfolding section5.M3-correct
      by (auto simp: n-def nth0-Cons)
  qed

  then obtain n where insertion-M3: γ' n thm3.M3 = 0 and n≥0
    by auto

  hence n ≥ 0 by auto


  define z where z ≡ φ a f g h k l w x y n

  have insertion-Pa: z thm3.Q = 0
  proof −
    have γ-eq-z-comp-σ: γ' n = z ∘ thm3.σ
      unfolding γ'-def thm3.σ-def comp-def z-def φ-def
      unfolding A₁-def A₂-def A₃-def S-def T-def R-def z₀-def
        unfolding thm3.A₁-def thm3.A₂-def thm3.A₃-def thm3.S-def thm3.T-def
thm3.R-def thm3.m-def
      unfolding thm3.μ-def
     unfolding thm3.L-def thm3.U-def thm3.V-def thm3.A-def thm3.C-def thm3.D-def
thm3.F-def
                thm3.I-def thm3.W-def thm3.K-def
      unfolding thm3.b-def thm3.X-def thm3.Y-def
      ..

    show ?thesis
    unfolding thm3.Q-alt fun-cong[OF γ-eq-z-comp-σ[symmetric, unfolded comp-def]]
      using insertion-M3 .
  qed

  show ?Pa-zero
    using insertion-Pa ‹n≥0› unfolding z-def by metis
next
  assume ?Pa-zero

  then obtain f g h k l w x y n where insertion-Q:
    φ a f g h k l w x y n thm3.Q = 0 and n ≥ 0
    unfolding thm3.Q-def by auto
```

define z **where** $z \equiv \varphi\ a\ f\ g\ h\ k\ l\ w\ x\ y\ n$

define S **where** $S \equiv z\ thm3.S$
define T **where** $T \equiv z\ thm3.T$
define R **where** $R \equiv z\ thm3.R$
define A_1 **where** $A_1 \equiv z\ thm3.A_1$
define A_2 **where** $A_2 \equiv z\ thm3.A_2$
define A_3 **where** $A_3 \equiv z\ thm3.A_3$

have S *dvd* T **and** $0 < R$ **and** *is-square* A_1 **and** *is-square* A_2 **and** *is-square* A_3
proof −
have $S \neq 0$
unfolding *S-def z-def* φ*-def thm3.S-def bridge-variables.S-def* **by** *presburger*

have *section5.M3* $A_1\ A_2\ A_3\ S\ T\ R\ n = 0$
using *insertion-Q*
unfolding φ*-def thm3.Q-def thm3.M3-def*
apply (*simp add*: *section5.M3-correct nth0-def*)
unfolding A_1*-def* A_2*-def* A_3*-def S-def T-def R-def z-def* φ*-def*
by (*auto simp add*: *thm3.m-def*)

thus S *dvd* T **and** $0 < R$ **and** *is-square* A_1 **and** *is-square* A_2 **and** *is-square* A_3
using *matiyasevich-polynomial.relation-combining*[*of* $S\ T\ R\ A_1\ A_2\ A_3$, *OF* ‹$S \neq 0$›] ‹$n \geq 0$›
unfolding *thm3.M3-def* φ*-def* **by** *auto*
qed

define b **where** $b \equiv thm3.b\ a\ f\ g\ h\ k\ l\ w\ x\ y\ n$
define X **where** $X \equiv thm3.X\ a\ f\ g\ h\ k\ l\ w\ x\ y\ n$
define Y **where** $Y \equiv thm3.Y\ a\ f\ g\ h\ k\ l\ w\ x\ y\ n$

have $b \geq 0$
using ‹*is-square* A_1› **unfolding** A_1*-def thm3.*A_1*-def thm3.b-def b-def z-def* φ*-def*
unfolding *is-square-def* **by** *auto*

have $thm3.R\ \tau > 0$ **using** ‹$R > 0$› **unfolding** *thm3.R-def R-def z-def* φ*-def* **by** *auto*
have $thm3.b\ \tau \geq 0$ **using** ‹$b \geq 0$› **unfolding** *thm3.b-def b-def z-def* φ*-def* **by** *auto*

have $f > 0$
proof −
have $f \neq 0$ **using** ‹$R > 0$› **unfolding** *R-def thm3.R-def z-def* φ*-def* **by** *auto*

show *?thesis*
using ‹$b \geq 0$› **unfolding** *b-def thm3.b-def coding-variables.b-def*

```
    using ‹f ≠ 0› apply simp
   by (smt (verit) mult-le-cancel-right1 of-nat-less-0-iff)
  qed

  have g ≥ 1
   using thm3.R-gt-0-consequences(1)[OF ‹thm3.R τ > 0› ‹thm3.b τ ≥ 0› ‹f >
0›] by simp

  have p0: coding-variables.P-coeff thm3.P_1
             (replicate (coding-variables.ν thm3.P_1 + 1) 0) > 0
   using suitable-for-coding-coeff0[of P]
   unfolding thm3.P_1-def coding-variables.P-coeff-def coding-variables.ν-def
   by simp


  have bridge-variables.statement1 b Y X
  proof −
   have g ≥ 0 using ‹g ≥ 1› by auto

   have int a ≥ 0
    by auto
   have delta: coding-variables.δ thm3.P_1 > 0
   by (simp add: suitable-for-coding-total-degree coding-variables.δ-def thm3.P_1-def)

   have Y: b ≤ Y ∧ 2 ^ 8 ≤ Y
    using coding-variables.lower-bounds(2−3)[OF ‹int a ≥ 0› ‹0 < f› ‹g ≥ 0›
delta p0]
    unfolding b-def Y-def thm3.b-def thm3.Y-def by auto

   have X: X ≥ 3 ∗ b
     using coding-variables.lower-bounds(1)[OF ‹int a ≥ 0› ‹0 < f› ‹g ≥ 0›
delta p0]
    unfolding b-def X-def thm3.b-def thm3.X-def by auto

   have bridge-variables.statement2 b Y X g
   proof −
    have l ∗ x ≠ 0
     using ‹R > 0› unfolding R-def thm3.R-def z-def φ-def by auto

    have d2a: bridge-variables.d2a l w h x y g Y X
     using ‹is-square A_2›
     unfolding bridge-variables.d2a-def A_2-def z-def thm3.A_2-def φ-def
          thm3.D-def thm3.I-def thm3.F-def Y-def X-def by auto

    have d2b: bridge-variables.d2b k l w x g Y X
     using ‹is-square A_3›
     unfolding bridge-variables.d2b-def A_3-def z-def thm3.A_3-def φ-def
          thm3.U-def Y-def X-def thm3.V-def thm3.K-def by auto
```

```
    have d2c: bridge-variables.d2c l w h b g Y X
      using ‹S dvd T› unfolding bridge-variables.d2c-def S-def T-def
                                thm3.S-def thm3.T-def z-def φ-def Y-def X-def b-def
      by auto

    have d2f: bridge-variables.d2f k l w h g Y X
      using thm3.R-gt-0-consequences(4)[OF ‹thm3.R τ > 0› ‹thm3.b τ ≥ 0›
‹f > 0›]
      unfolding X-def Y-def by simp

    show ?thesis
      unfolding bridge-variables.statement2-def
      using ‹l * x ≠ 0› d2a d2b d2c d2f by metis
  qed

  thus ?thesis
    using bridge-variables.theorem-II-2-1[of b Y X g, OF ‹b ≥ 0› Y X ‹g≥1›]
by auto
 qed

 hence is-power2 b and Y-dvd: Y dvd int (2 * nat X choose nat X)
  unfolding bridge-variables.statement1-def by auto

 interpret coding: coding-theorem thm3.P₁ int a f
 proof (unfold-locales)
  show 0 ≤ int a by simp

  show 0 < f using ‹f > 0› by auto

  show is-power2 (coding-variables.b (int a) f)
    using ‹is-power2 b› unfolding b-def thm3.b-def by auto

  show 0 < coding-variables.δ thm3.P₁
    unfolding thm3.P₁-def coding-variables.δ-def
    using suitable-for-coding-total-degree by auto
 qed

 have ∃ y. coding.statement1-weak y
 proof −
  have 0 ≤ g using ‹g ≥ 1› by auto

  have g-bound: g < 2 * int coding.γ * coding.b ^ coding.α
    using thm3.R-gt-0-consequences(2)[OF ‹thm3.R τ > 0› ‹thm3.b τ ≥ 0› ‹f
> 0›]
    unfolding thm3.μ-def thm3.b-def by auto

  have dvd-choose: coding.Y dvd int (2 * nat (coding.X g) choose nat (coding.X
g))
    using Y-dvd unfolding Y-def X-def thm3.Y-def thm3.X-def by auto
```

```
    show ?thesis
     using coding.coding-theorem-reverse[of g] unfolding coding.statement2-weak-def
      using ‹g≥0› g-bound dvd-choose
      by (metis)
  qed

  then obtain y where y-0: y 0 = int a
                and y-i: ∀ i. 0 ≤ y i ∧ y i < coding.b
                and insertion-y: insertion y thm3.P₁ = 0
    unfolding coding.statement1-weak-def by auto

  have is-nonnegative y
    using y-i unfolding is-nonnegative-def by auto

  have insertion (y(0 := int a)) thm3.P₁ = 0
  proof −
    have y-eq: (y(0 := int a)) = y
      using y-0 by auto

    show ?thesis using insertion-y unfolding y-eq by auto
  qed

  hence ∃ y₀. insertion (y₀(0 := int a)) P = 0 ∧ is-nonnegative y₀
    using suitable-for-coding-diophantine-equivalent ‹is-nonnegative y›
    unfolding thm3.P₁-def by auto

  thus a ∈ A
    using assms(2) unfolding is-diophantine-over-N-with-def by auto
qed

qed

theorem theorem-III:
  fixes A :: nat set and P :: int mpoly
  assumes is-diophantine-over-N-with A P
  shows a ∈ A = (∃ f g h k l w x y n. n ≥ 0 ∧
                insertion ((!) [int a, f, g, h, k, l, w, x, y, n])
                (combined-variables.Q-poly P) = 0)
  unfolding combined-variables.Q-correct
  unfolding theorem-III-new-statement[of A P a, OF assms(1)]
  by simp

theorem nine-unknowns-over-Z-and-N:
  fixes P :: int mpoly
  shows (∃ z :: nat ⇒ int. is-nonnegative z ∧
            insertion (z(0 := int a)) P = 0)
      = (∃ f g h k l w x y n. n ≥ 0 ∧
            insertion ((!) [int a, f, g, h, k, l, w, x, y, n])
```

(*combined-variables.Q-poly P*) = *0*)
proof −
define *A* **where** *A* ≡ {*a*. ∃ *z*. *insertion* (*z*(*0* := *int a*)) *P* = *0* ∧ *is-nonnegative z*}

show *?thesis*
using *theorem-III*[*of A P*] **unfolding** *A-def is-diophantine-over-N-with-def*
by *fastforce*
qed

end
theory *Eleven-Unknowns-Z*
imports *Nine-Unknowns-N-Z*/*Nine-Unknowns-N-Z Three-Squares.Three-Squares*
begin

9 Eleven Unknowns over ℤ

context
fixes *P* :: *int mpoly*
begin

definition *Q-tilde* :: *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int*
where *Q-tilde a* z_1 z_2 z_3 z_4 z_5 z_6 z_7 z_8 z_9 z_{10} z_{11} =
(*combined-variables.Q P*) *a* z_1 z_2 z_3 z_4 z_5 z_6 z_7 z_8 (z_9 ^*2* + z_{10} ^*2* + z_{11} ^*2* + z_{11})

poly-extract *Q-tilde*
poly-degree *Q-tilde-poly* | *combined-variables.aux-vars combined-variables.final combined-variables.list-vars*

lemma *Q-tilde-degree-in-X-Y*:
Q-tilde-poly-degree = *8352* ∗ *combined-variables.Y-poly-degree P*
+ (*15568* + (*4176* ∗ *combined-variables.X-poly-degree P*
+ *48* ∗ *coding-variables.α* (*combined-variables.*P_1 *P*)))
proof −
note *Q-tilde-degree-presimplified* = *Q-tilde-poly-degree-def*[*simplified Nat.max-nat.right-neutral*

Nat.max-nat.left-neutral Nat.plus-nat.add-0]
note A_2*-poly-degree-alt* = *combined-variables.*A_2*-poly-degree-def*[*simplified*, *simplified*
Nat.Suc-eq-plus1, *simplified Groups.ac-simps*(*1*), *unfolded one-add-one*,
simplified one-plus-numeral, *simplified*]

show *?thesis* **unfolding** *Q-tilde-degree-presimplified*
combined-variables.S-poly-degree-def[*simplified*]
combined-variables.T-poly-degree-def[*simplified*]
combined-variables.R-poly-degree-def[*simplified*]

combined-variables.A_1*-poly-degree-def*[*simplified*]
combined-variables.A_3*-poly-degree-def*[*simplified*]
A_2*-poly-degree-alt*
by *simp*
qed

definition *delta-P-suitable* (δ_s) **where**
delta-P-suitable ≡ *total-degree* (*suitable-for-coding P*)

definition *nu-P-suitable* (ν_s) **where**
nu-P-suitable ≡ *max-vars* (*suitable-for-coding P*)

definition eta_s **where**
$eta_s \equiv 15616 + 116928 * \delta_s + 116976 * \delta_s * Suc\ \delta_s\ \hat{}\ \nu_s + 116928 * \delta_s\hat{}2 * Suc\ \delta_s\ \hat{}\ \nu_s$

lemma *Q-tilde-degree-*eta_s: *Q-tilde-poly-degree* = eta_s
proof −
note *X-degree* = *combined-variables.X-poly-degree-alt*[*unfolded combined-variables.*P_1*-def*

coding-variables.δ*-def*, *OF suitable-for-coding-total-degree*,
folded coding-variables.δ*-def*, *folded combined-variables.*P_1*-def*]
have *degree-bound*: *coding-variables.*δ (*combined-variables.*P_1 *P*) = *delta-P-suitable*

unfolding *coding-variables.*δ*-def combined-variables.*P_1*-def delta-P-suitable-def*
by *simp*
have *vars*: *coding-variables.*ν (*combined-variables.*P_1 *P*) = *nu-P-suitable*
unfolding *coding-variables.*ν*-def combined-variables.*P_1*-def nu-P-suitable-def*
by *simp*
show *?thesis*
unfolding *Q-tilde-degree-in-X-Y combined-variables.Y-poly-degree-def*[*simplified*] *X-degree*
unfolding *coding-variables.*α*-def coding-variables.n-def*
apply *simp*
unfolding *vars degree-bound* eta_s*-def*
apply (*simp add*: *algebra-simps*)
using *Semiring-Normalization.comm-semiring-1-class.semiring-normalization-rules*(*29*)
by *metis*
qed

definition η **where**
$\eta\ \nu\ \delta \equiv 15616 + 233856 * \delta + 233952 * \delta * (2 * \delta + 1)\ \hat{}\ (\nu + 1) + 467712 * \delta\hat{}2 * (2 * \delta + 1)\ \hat{}\ (\nu + 1)$

lemma *Q-tilde-degree*:
assumes 0 < *total-degree P*
assumes *total-degree P* ≤ δ
assumes *max-vars P* ≤ ν

shows *Q-tilde-poly-degree* $\leq \eta\ \nu\ \delta$
proof −
have ν: $\nu_s \leq \nu + 1$
apply (*cases vars P* = {})
subgoal
by (*simp add*: *fresh-var-def max-vars-of-nonempty nu-P-suitable-def*
suitable-for-coding-degree-vars(*2*))
subgoal
unfolding *nu-P-suitable-def*
using *suitable-for-coding-max-vars*
by (*simp add*: *assms*(*3*))
done
have δ: $\delta_s \leq 2 * \delta$
unfolding *delta-P-suitable-def*
using *suitable-for-coding-total-degree-bound*[*OF assms*(*1*)] *assms*(*2*) **by** *auto*
have δ-*power*: *Suc* δ_s ^ $\nu \leq$ *Suc* (*2* ∗ δ) ^ ν
by (*simp add*: δ *power-mono*)
hence δ-*power2*: *Suc* δ_s ^ $\nu_s \leq$ *Suc* (*2* ∗ δ) ^ (ν + *1*)
using ν *power-mono le-trans*
by (*smt* (*verit*) *Suc-eq-plus1* δ *le-SucE mult-Suc mult-le-mono not-less-eq-eq power-Suc*
power-increasing trans-le-add1 trans-le-add2)
hence δ-*power3*: *Suc* δ_s ^ $\nu_s \leq$ *Suc* (*2* ∗ δ) ^ ν + *2* ∗ δ ∗ *Suc* (*2* ∗ δ) ^ ν
using δ-*power2* **by** (*algebra*, *simp*)
have *aux*: δ_s ∗ *Suc* δ_s ^ $\nu_s \leq$ *2* ∗ (δ ∗ (*Suc* (*2* ∗ δ) ^ ν + *2* ∗ δ ∗ *Suc* (*2* ∗ δ) ^ ν))
using δ-*power3 mult-le-mono* **by** (*metis* δ *more-arith-simps*(*11*))
have *bound*: *Q-tilde-poly-degree* $\leq$ *15616* + *116928* ∗ *2* ∗ δ + *116976* ∗ *2* ∗ δ ∗ *Suc* (*2* ∗ δ) ^ (ν + *1*)
+ *116928* ∗ *4* ∗ δ^*2* ∗ *Suc* (*2* ∗ δ) ^ (ν + *1*)
unfolding *Q-tilde-degree-eta*$_s$ *eta*$_s$-*def*
apply *simp*
apply (*rule add-mono mult-le-mono*, *simp add*: δ ν)+
subgoal by (*simp add*: *aux*)
subgoal
apply (*rule add-mono mult-le-mono power-mono*, *simp-all add*: δ ν δ-*power3*)
by (*metis* δ *numeral-Bit0-eq-double power2-eq-square power2-nat-le-eq-le power-mult-distrib*)
done
thus *?thesis* **unfolding** η-*def* **by** *auto*
qed

lemma *max-vars-Q-tilde*: *max-vars Q-tilde-poly* $\leq$ *11*
proof −
have *aux-three-squares*: *vars* ((*Var 9*)^2 + (*Var 10*)^2 + (*Var 11*)^2 + *Var 11*) $\subseteq$ {*9*, *10*, *11*}
unfolding *power2-eq-square*
by (*rule subset-trans*[*OF vars-add Un-least*]
| *rule subset-trans*[*OF vars-mult Un-least*]

```
      | subst vars-Var; simp)+

  show ?thesis
    unfolding Q-tilde-poly-def
    apply (rule le-trans[OF max-vars-poly-subst-list-general])
    apply (simp add: max-vars-def aux-three-squares)
    apply (rule Max.boundedI, auto simp: vars-finite)
    using in-mono[OF aux-three-squares] by fastforce
qed

lemma eleven-unknowns-over-Z:
  fixes A :: nat set
  assumes is-diophantine-over-N-with A P
  shows a ∈ A = (∃ z1 z2 z3 z4 z5 z6 z7 z8 z9 z10 z11.
    Q-tilde (int a) z1 z2 z3 z4 z5 z6 z7 z8 z9 z10 z11 = 0)
proof −

  have (∃ w1 w2 w3 w4 w5 w6 w7 w8 w9.
          w9 ≥ 0 ∧ (combined-variables.Q P) (int a) w1 w2 w3 w4 w5 w6 w7 w8
w9 = 0)
     = (∃ z1 z2 z3 z4 z5 z6 z7 z8 z9 z10 z11.
          Q-tilde (int a) z1 z2 z3 z4 z5 z6 z7 z8 z9 z10 z11 = 0)
  proof (intro iffI)
    assume asm1:(∃ w1 w2 w3 w4 w5 w6 w7 w8 w9. w9 ≥ 0 ∧
            (combined-variables.Q P) (int a) w1 w2 w3 w4 w5 w6 w7 w8 w9 = 0)
    obtain w1 w2 w3 w4 w5 w6 w7 w8 w9 where w-prop:w9 ≥ 0 ∧
            (combined-variables.Q P) (int a) w1 w2 w3 w4 w5 w6 w7 w8 w9 = 0
      using asm1 by auto
    have w9decomp:∃ u9 u10 u11. w9 = u9^2+u10^2+u11^2+u11
      using four-decomposition-int w-prop by presburger
    obtain u9 u10 u11 where u-prop:w9 = u9^2+u10^2+u11^2+u11
      using w9decomp by auto
    have cs1:Q-tilde (int a) w1 w2 w3 w4 w5 w6 w7 w8 u9 u10 u11 = 0
      using u-prop Q-tilde-def asm1 w-prop by presburger
    thus ∃ z1 z2 z3 z4 z5 z6 z7 z8 z9 z10 z11. Q-tilde (int a) z1 z2 z3 z4 z5 z6 z7
z8 z9 z10 z11 = 0 by blast
  next
    assume asm2: ∃ z1 z2 z3 z4 z5 z6 z7 z8 z9 z10 z11. Q-tilde (int a) z1 z2 z3
z4 z5 z6 z7 z8 z9 z10 z11 = 0
    obtain z1 z2 z3 z4 z5 z6 z7 z8 z9 z10 z11 where z-prop:
      Q-tilde (int a) z1 z2 z3 z4 z5 z6 z7 z8 z9 z10 z11 = 0 using asm2 by auto

    have cs2-0:
      (combined-variables.Q P) (int a) z1 z2 z3 z4 z5 z6 z7 z8 (z9 ^2 + z10 ^2 +
z11 ^2 + z11) = 0
      using z-prop Q-tilde-def by simp
    have z9 ^2 + z10 ^2 + z11 ^2 + z11 ≥0 by (meson four-decomposition-int)
    thus cs2:∃ w1 w2 w3 w4 w5 w6 w7 w8 w9. w9 ≥ 0 ∧
        (combined-variables.Q P) (int a) w1 w2 w3 w4 w5 w6 w7 w8 w9 = 0 using
```

cs2-0 **by** *blast*
qed

thus *?thesis*
using *theorem-III-new-statement assms* **by** *presburger*
qed

end

lemma *eleven-unknowns-over-Z-polynomial*:
fixes A :: *nat set* **and** P :: *int mpoly*
assumes *is-diophantine-over-N-with A P*
shows $a \in A = (\exists\, z_1\ z_2\ z_3\ z_4\ z_5\ z_6\ z_7\ z_8\ z_9\ z_{10}\ z_{11}.$
insertion ((!) [*int a*, z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8, z_9, z_{10}, z_{11}])
(*Q-tilde-poly P*) = *0*)
unfolding *Q-tilde-correct eleven-unknowns-over-Z*[*of A P a*, *OF assms*(*1*)] **by** *simp*

end
theory *Universal-Pairs*
imports *Eleven-Unknowns-Z*
begin

definition *universal-pair-N* ($'(\text{-}, \text{-}')_{\mathbb{N}}$ [*1000*]) **where**
universal-pair-N $\nu\ \delta \equiv$ ($\forall$ *A*::*nat set*. *is-diophantine-over-N A* $\longrightarrow$
($\exists$ *P*::*int mpoly*. *max-vars P* $\leq \nu \wedge$ *total-degree P* $\leq \delta \wedge$
is-diophantine-over-N-with A P))

definition *universal-pair-Z* ($'(\text{-}, \text{-}')_{\mathbb{Z}}$ [*1000*]) **where**
universal-pair-Z $\nu\ \delta \equiv$ ($\forall$ *A*::*nat set*. *is-diophantine-over-N A* $\longrightarrow$
($\exists$ *P*::*int mpoly*. *max-vars P* $\leq \nu \wedge$ *total-degree P* $\leq \delta \wedge$
is-diophantine-over-Z-with A P))

theorem *universal-pairs-Z-from-N*:
assumes $(\nu, \delta)_{\mathbb{N}}$
shows $(11, \eta\ \nu\ \delta)_{\mathbb{Z}}$
proof (*unfold universal-pair-Z-def*, (*rule allI impI*)+)
fix *A*
assume *is-diophantine-over-N A*
then obtain *P* **where** *P1*: *max-vars P* $\leq \nu$ **and**
P2: *total-degree P* $\leq \delta$ **and**
P3: *is-diophantine-over-N-with A P*
using *assms* **unfolding** *universal-pair-N-def* **by** *auto*

show $\exists$ *Q*. *max-vars Q* $\leq$ *11* $\wedge$ *total-degree Q* $\leq \eta\ \nu\ \delta \wedge$ *is-diophantine-over-Z-with A Q*
proof (*cases total-degree P* $>$ *0*)
case *True*

```
    define Q where Q ≡ Q-tilde-poly P

    have 1: max-vars Q ≤ 11
      using max-vars-Q-tilde
      unfolding Q-def by auto

    have 2: total-degree Q ≤ η ν δ
     using Q-tilde-degree[OF True P2 P1] Q-tilde-poly-degree-correct Q-def le-trans
by blast

    have 3: is-diophantine-over-Z-with A Q
      unfolding is-diophantine-over-Z-with-def Q-def
      unfolding eleven-unknowns-over-Z-polynomial[OF P3]
      apply (auto, metis fun-upd-triv nth-Cons-0)
      using Q-tilde-correct by force

    then show ?thesis
      using 1 2 3 by auto
  next
    case False
    then obtain c where c: P ≡ Const c
      using Total-Degree.total-degree-zero by blast

    have 1: max-vars P ≤ 11 unfolding c by simp

    have 2: total-degree P ≤ η ν δ unfolding c by simp

    have 3: is-diophantine-over-Z-with A P
      using P3 unfolding is-diophantine-over-N-with-def
      unfolding is-diophantine-over-Z-with-def
      unfolding c using is-nonnegative-def by auto

    show ?thesis
      by (rule exI[of - P], auto simp: 1 2 3)
  qed
qed

theorem universal-pair-1:
  assumes (58, 4)ℕ
 shows (11, 168104323522661991630118262451191852783413773370740844833553984 0)ℤ
  using universal-pairs-Z-from-N[of 58 4] assms unfolding η-def
  by (simp add: algebra-simps)




theorem universal-pair-2:
  assumes (32, 12)ℕ
```

shows $(11, 950817549694171759711025515571236610412597656252821888)_{\mathbb{Z}}$
using *universal-pairs-Z-from-N*[*of 32 12*] *assms* **unfolding** η*-def*
by (*simp add*: *algebra-simps*)

end

References

[1] J. Bayer and M. David. A Formal Proof of Complexity Bounds on Diophantine Equations. In *Proceedings of the 16th International Conference on Interactive Theorem Proving (ITP 2025)*, Leibniz International Proceedings in Informatics (LIPIcs), 2025. To appear.

[2] J. Bayer, M. David, M. Haßler, D. Schleicher, and Y. Matiyavsevich. Diophantine Equations over $\mathbb{Z}$: Universal Bounds and Parallel Formalization, 2025. https://arxiv.org/abs/2506.20909.

[3] A. Danilkin and L. Chevalier. Three Squares Theorem. *Archive of Formal Proofs*, 2023. https://isa-afp.org/entries/Three_Squares.html, Formal proof development.

[4] Y. Matiyasevich. *Hilbert's Tenth Problem*. Foundations of Computing Series. MIT Press, 1993.

[5] Y. Matiyasevich and J. Robinson. Reduction of an arbitrary Diophantine equation to one in 13 unknowns. *Acta Arithmetica*, 27:521–553, 1975.

[6] C. Sternagel, R. Thiemann, A. Maletzky, F. Immler, F. Haftmann, A. Lochbihler, and A. Bentkamp. Executable multivariate polynomials. *Archive of Formal Proofs*, 2010. https://isa-afp.org/entries/Polynomials.html, Formal proof development.

[7] Z.-W. Sun. Reduction of unknowns in Diophantine representations. *Science in China Series A*, 35(3):257–269, 1992.

[8] Z.-W. Sun. Further results on Hilbert's Tenth Problem. *Science China Math*, 64(2):281–306, 2021.