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 mathématiques 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*⟩

lemma *coeff-minus*: *coeff p m* − *coeff q m* = *coeff* (*p*−*q*) *m*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *nth0-0*: *i* ≥ *length xs* ⟹ *xs* $!_0$ *i* = *0*
⟨*proof*⟩

lemma *nth0-Cons*: (*x* # *xs'*) $!_0$ *i* = (*case i of 0* ⇒ *x* | *Suc i'* ⇒ *xs'* $!_0$ *i'*)
⟨*proof*⟩

lemma *nth0-Cons-0* [*simp*, *code*]: (*x* # *xs*) $!_0$ *0* = *x*
⟨*proof*⟩

lemma *nth0-Cons-Suc* [*simp*, *code*]: (*x* # *xs*) $!_0$ (*Suc n*) = *xs* $!_0$ *n*
⟨*proof*⟩

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

lemma *nth0-inj*: *length xs* = *length ys* ⟹ ($!_0$) *xs* = ($!_0$) *ys* ⟹ *xs* = *ys*
⟨*proof*⟩

lemma *nth0-sum-list*: *sum-list i* = (∑ *v* | *i* $!_0$ *v* ≠ *0*. *i* $!_0$ *v*)
⟨*proof*⟩

lemma *lookup-Abs-poly-mapping-nth0*[*simp*]:

lookup (*Abs-poly-mapping* (($!_0$) *xs*)) = ($!_0$) *xs*
⟨*proof*⟩

lemma *Abs-poly-mapping-nth0-single*[*simp*]:
Abs-poly-mapping (($!_0$) [*x*]) = *Poly-Mapping.single 0 x*
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*⟩

1.1.2 The constant polynomial

lemma *Const-zero*: *Const 0* = *0*
⟨*proof*⟩
lemma *Const-one*: *Const 1* = *1*
⟨*proof*⟩
lemma *Const-add*: *Const a* + *Const b* = *Const* (*a* + *b*)
⟨*proof*⟩
lemma *Const-mult*: *Const a* ∗ *Const b* = *Const* (*a* ∗ *b*)
⟨*proof*⟩
lemma *Const-power*: *Const c* ^ *i* = *Const* (*c* ^ *i*)
⟨*proof*⟩
lemma *Const-sub*: *Const a* − *Const b* = *Const* (*a* − *b*)

⟨*proof*⟩

lemma *Const-when*: *Const* (*a when P*) = (*Const a when P*)
⟨*proof*⟩

lemma *coeff-Const-zero*: *MPoly-Type.coeff* (*Const c*) *0* = *c*
⟨*proof*⟩

lemma *Const-sum-Any*: *Const* (*Sum-any f*) = *Sum-any* (*Const* ∘ *f*)
⟨*proof*⟩

lemma *Const-numeral*: *Const* (*numeral x*) = *numeral x*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *of-nat-Const*: *of-nat n* = *Const* (*int n*)
⟨*proof*⟩

lemma *of-int-Const*: *of-int x* = *Const x*
⟨*proof*⟩

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*⟩

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*⟩

lemma *nat-sum-distrib*:
fixes *f*::*nat*⇒*int*
assumes *S-fin*: *finite S* **and** *nonneg*: ⋀*i*. *i*∈*S* ⟹ *f i* ≥ *0*
shows *nat* (∑*i*∈*S*. *f i*) = (∑*i*∈*S*. *nat* (*f i*))
⟨*proof*⟩

1.1.4 Insertion

named-theorems *insertion-simps Lemmas about insertion*

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

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

lemma *insertion-Var*[*simp*, *insertion-simps*]: *insertion f* (*Var n*) = *f n*
⟨*proof*⟩

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

lemma *insertion-numeral*[*simp*, *insertion-simps*]: *insertion f* (*numeral n*) = *numeral n*
⟨*proof*⟩

lemma *Sum-any-neg*:
fixes *f* :: - ⇒ ′*a*::*ring-1*
shows *Sum-any* (λ*a*. − *f a*) = − *Sum-any* (λ*a*. *f a*)
⟨*proof*⟩

lemma *insertion-neg*[*simp*, *insertion-simps*]:
fixes *f* :: - ⇒ ′*a*::*comm-ring-1*
shows *insertion f* (− *p*) = − *insertion f p*
⟨*proof*⟩

lemma *insertion-diff*[*simp*, *insertion-simps*]:
fixes *f* :: - ⇒ ′*a*::*comm-ring-1*
shows *insertion f* (*p* − *q*) = *insertion f p* − *insertion f q*
⟨*proof*⟩

lemma *insertion-of-int*[*simp*, *insertion-simps*]:
fixes *f*::*nat* ⇒ *int* **and** *c*::*int*
shows *insertion f* (*of-int c*) = *c*
⟨*proof*⟩

lemma *insertion-of-nat*[*simp*, *insertion-simps*]:
fixes *f*::*nat* ⇒ *int* **and** *n*::*nat*
shows *insertion f* (*of-nat n*) = *int n*
⟨*proof*⟩

lemma *insertion-pow*[*simp*, *insertion-simps*]: *insertion f* (*P*^*n*) = (*insertion f P*)^*n*
⟨*proof*⟩

lemma *insertion-sum*[*simp*, *insertion-simps*]:

finite S $\Longrightarrow$ *insertion f* ($\sum$ *i*∈*S*. *P i*) = ($\sum$ *i*∈*S*. *insertion f* (*P i*))
⟨*proof*⟩

lemma *insertion-prod*[*simp*, *insertion-simps*]:
finite S $\Longrightarrow$ *insertion f* ($\prod$ *i*∈*S*. *P i*) = ($\prod$ *i*∈*S*. *insertion f* (*P i*))
⟨*proof*⟩

lemma *insertion-monom*[*simp*, *insertion-simps*]:
insertion f (*monom m a*) = *a* ∗ ($\prod$ *x*. *f x* ˆ *lookup m x*)
⟨*proof*⟩

lemma *insertion-of-int-times* : *insertion f* (*of-int n* ∗ *P*) = *n* ∗ *insertion f P*
⟨*proof*⟩

lemma *pow-positive*:
fixes *a* :: *'a*::*idom*
assumes *a* ≠ *0*
assumes *n* > *0*
shows *a* ˆ *n* ≠ *0*
⟨*proof*⟩

One more typeclasses

instance *mpoly* :: (*semiring-no-zero-divisors*) *semiring-no-zero-divisors*
⟨*proof*⟩

end
theory *Degree*
imports *Notation*
begin

1.2 Degree of a given variable

lemma *degree-Const* [*simp*]: *degree* (*Const x*) *v* = *0*
⟨*proof*⟩

lemma *degree-Var* [*simp*]:
degree ((*Var v*):: *'a*::*comm-semiring-1 mpoly*) *v'* = (*if v* = *v'* *then 1 else 0*)
⟨*proof*⟩

lemma *degree-neg*:
fixes *P* :: *'a*::*ab-group-add mpoly*
shows *degree* (− *P*) = *degree P*
⟨*proof*⟩

lemma *degree-add*:
fixes *P Q* :: *'a*::*ab-group-add mpoly*
shows *degree* (*P* + *Q*) *v* ≤ *max* (*degree P v*) (*degree Q v*)

⟨*proof*⟩

lemma *degree-add′*:
fixes *P Q* :: *′a::ab-group-add mpoly*
shows *degree* (*P* + *Q*) *v* ≤ *degree P v* + *degree Q v*
⟨*proof*⟩

lemma *degree-add-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*)
⟨*proof*⟩

lemma *degree-diff*:
fixes *P Q* :: *′a::ab-group-add mpoly*
shows *degree* (*P* − *Q*) *v* ≤ *max* (*degree P v*) (*degree Q v*)
⟨*proof*⟩

lemma *degree-diff′*:
fixes *P Q* :: *′a::ab-group-add mpoly*
shows *degree* (*P* − *Q*) *v* ≤ *degree P v* + *degree Q v*
⟨*proof*⟩

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*)
⟨*proof*⟩

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* ((λ*i*. *degree* (*P i*) *v*) ‘ *S*))
⟨*proof*⟩

lemma *degree-mult*: *degree* (*P* ∗ *Q*) *v* ≤ *degree P v* + *degree Q v*
⟨*proof*⟩

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*⟩

lemma *degree-pow*: *degree* (*P* ^ *n*) *v* ≤ *n* ∗ *degree P v*
⟨*proof*⟩

lemma *degree-pow-positive*:
fixes *P* :: *′a::idom mpoly*

assumes *n* > *0*
shows *degree* (*P* ^ *n*) *v* = *n* ∗ *degree P v*
⟨*proof*⟩

lemma *degree-prod*:
assumes *S-fin*: *finite S*
shows *degree* (*prod P S*) *v* ≤ *sum* (λ*i*. *degree* (*P i*) *v*) *S*
⟨*proof*⟩

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*⟩

lemma *in-vars-non-zero-degree*: *v* ∈ *vars P* ⟷ *degree P v* ≠ *0*
⟨*proof*⟩

lemma *vars-non-zero-degree*: *vars P* = {*v*. *degree P v* ≠ *0*}
⟨*proof*⟩

lemma *vars-Const* [*simp*]: *vars* (*Const x*) = {}
⟨*proof*⟩

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

lemma *vars-Var* [*simp*]: *vars* ((*Var v*) :: (*'a::zero-neq-one*) *mpoly*) = {*v*}
⟨*proof*⟩

lemma *vars-neg*:
fixes *P* :: *'a::ab-group-add mpoly*
shows *vars* (− *P*) = *vars P*
⟨*proof*⟩

lemma *vars-add-different-degree*:
fixes *P* :: *'a::ab-group-add mpoly*
assumes ∀ *u* ∈ *vars P* ∩ *vars Q*. *degree P u* ≠ *degree Q u*
shows *vars* (*P* + *Q*) = *vars P* ∪ *vars Q*
⟨*proof*⟩

lemma *vars-diff*:
fixes *P* :: *'a::ab-group-add mpoly*
shows *vars* (*P* − *Q*) ⊆ *vars P* ∪ *vars Q*

⟨*proof*⟩

lemma *vars-diff-different-degree*:
fixes P :: $'a$::*ab-group-add mpoly*
assumes $\forall u \in vars\ P \cap vars\ Q.\ degree\ P\ u \neq degree\ Q\ u$
shows $vars\ (P - Q) = vars\ P \cup vars\ Q$
⟨*proof*⟩

lemma *vars-mult-non-zero*:
fixes $P\ Q$ :: $'a$::*idom mpoly*
shows $P \neq 0 \Longrightarrow Q \neq 0 \Longrightarrow vars\ (P * Q) = vars\ P \cup vars\ Q$
⟨*proof*⟩

lemma *vars-pow*: $vars\ (P \hat{}\ n) \subseteq vars\ P$
⟨*proof*⟩

lemma *vars-pow-positive*:
fixes P :: $'a$::*idom mpoly*
assumes $n > 0$
shows $vars\ (P\ \hat{}\ n) = vars\ P$
⟨*proof*⟩

lemma *vars-prod*:
fixes S :: $'a\ set$
and f :: - ⇒ (- :: *zero-neq-one*) *mpoly*
shows $vars\ (prod\ f\ S) \subseteq \bigcup\ (vars\ `\ f\ `\ S)$
⟨*proof*⟩

lemma *vars-empty*:
assumes $vars\ P = \{\}$
shows $\exists c.\ P = Const\ c$
⟨*proof*⟩

1.3.1 Maximum variable index

definition *max-vars* **where** $max\text{-}vars\ P \equiv Max\ (insert\ 0\ (vars\ P))$

lemma *after-max-vars*:
$lookup\ (mapping\text{-}of\ P)\ m \neq 0 \Longrightarrow \forall v \geq max\text{-}vars\ P + 1.\ lookup\ m\ v = 0$
⟨*proof*⟩

lemma *max-vars-of-nonempty*: $vars\ P \neq \{\} \Longrightarrow max\text{-}vars\ P = Max\ (vars\ P)$
⟨*proof*⟩

1.3.2 Simplification rules for maximum variable index

lemma *max-vars-Const* [*simp*]: $max\text{-}vars\ (Const\ x) = 0$
⟨*proof*⟩

lemma *max-vars-one* [*simp*]: $max\text{-}vars\ 1 = 0$

⟨*proof*⟩

lemma *max-vars-Var* [*simp*]: *max-vars* ((*Var v*) :: ('*a*::*zero-neq-one*) *mpoly*) = *v*
⟨*proof*⟩

lemma *max-vars-add*: *max-vars* (*P* + *Q*) ≤ *max* (*max-vars P*) (*max-vars Q*)
⟨*proof*⟩

lemma *max-vars-neg*: *max-vars* (− *P*) = *max-vars P*
⟨*proof*⟩

lemma *max-vars-diff*:
fixes *P* :: '*a*::*ab-group-add mpoly*
shows *max-vars* (*P* − *Q*) ≤ *max* (*max-vars P*) (*max-vars Q*)
⟨*proof*⟩

lemma *max-vars-diff'*:
fixes *P* :: *int mpoly*
shows *max-vars* (*P* − *Q*) ≤ *max* (*max-vars P*) (*max-vars Q*)
⟨*proof*⟩

lemma *max-vars-pow*: *max-vars* (*P* ^ *n*) ≤ *max-vars P*
⟨*proof*⟩

lemma *max-vars-pow-positive*:
fixes *P* :: '*a*::*idom mpoly*
assumes *n* > *0*
shows *max-vars* (*P* ^ *n*) = *max-vars P*
⟨*proof*⟩

lemma *max-vars-mult*: *max-vars* (*P* ∗ *Q*) ≤ *max* (*max-vars P*) (*max-vars Q*)
⟨*proof*⟩

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*
  ⟨*proof*⟩

lemma *total-degree-Var* [*simp*]:
  total-degree ((*Var v*):: ′*a*::*comm-semiring-1 mpoly*) = *1*
  ⟨*proof*⟩

lemma *total-degree-zero-degree-zero*:
  assumes *total-degree P* = *0*
  shows *degree P v* = *0*
⟨*proof*⟩

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

lemma *total-degree-neg*[*total-degree-simps*]:
  fixes *P* :: ′*a*::*ab-group-add mpoly*
  shows *total-degree* (− *P*) = *total-degree P*
  ⟨*proof*⟩

lemma *total-degree-add*[*total-degree-simps*]:
  shows *total-degree* (*P* + *Q*) ≤ *max* (*total-degree P*) (*total-degree Q*)
⟨*proof*⟩

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*⟩

lemma *total-degree-diff*[*total-degree-simps*]:
  fixes *P* :: ′*a*::*ab-group-add mpoly*
  shows *total-degree* (*P* − *Q*) ≤ *max* (*total-degree P*) (*total-degree Q*)
  ⟨*proof*⟩

lemma *total-degree-diff-different-total-degree*:
  fixes *P* :: ′*a*::*ab-group-add mpoly*

assumes *total-degree P* $\neq$ *total-degree Q*
shows *total-degree* (*P* − *Q*) = *max* (*total-degree P*) (*total-degree Q*)
⟨*proof*⟩

lemma *total-degree-mult*[*total-degree-simps*]:
total-degree (*P* ∗ *Q*) ≤ *total-degree P* + *total-degree Q*
⟨*proof*⟩

lemma *total-degree-mult-non-zero*:
fixes *P Q* :: ′*a*::*idom mpoly*
assumes *P* ≠ *0* *Q* ≠ *0*
shows *total-degree* (*P* ∗ *Q*) = *total-degree P* + *total-degree Q*
⟨*proof*⟩

lemma *total-degree-pow*: *total-degree* (*P* ^ *n*) ≤ *n* ∗ *total-degree P*
⟨*proof*⟩

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*⟩

lemma *total-degree-sum*:
fixes *P* :: ′*a* ⇒ ′*b*::*comm-monoid-add mpoly*
assumes *S-fin*: *finite S*
shows *total-degree* (*sum P S*) ≤ *Max* (*insert 0* ((λ*i*. *total-degree* (*P i*)) ‘ *S*))
⟨*proof*⟩

lemma *total-degree-prod*:
assumes *S-fin*: *finite S*
shows *total-degree* (*prod P S*) ≤ *sum* (λ*i*. *total-degree* (*P i*)) *S*
⟨*proof*⟩

lemma *Max-function-mono*:
fixes *f g* :: ′*a* ⇒ *nat*
assumes *finite A*
assumes *A* ≠ {}
assumes ∀ *a*∈*A*. *f a* ≤ *g a*
shows *Max* (*f* ‘ *A*) ≤ *Max* (*g* ‘ *A*)
⟨*proof*⟩

lemma *degree-total-degree-bound*:
degree P v ≤ *total-degree P*
⟨*proof*⟩

lemma *total-degree-bound*:
total-degree P ≤ *sum* (*degree P*) (*vars P*)
⟨*proof*⟩

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

1.5 Explicit expansions

lemma *mpoly-keys-subset*: $keys\ (mapping\text{-}of\ P) \subseteq Abs\text{-}poly\text{-}mapping\ `\ (!_0)\ `$
$\{i.\ length\ i = max\text{-}vars\ P + 1 \wedge sum\text{-}list\ i \leq total\text{-}degree\ P\}$
⟨*proof*⟩

lemma *monom-single*: $monom\ (Poly\text{-}Mapping.single\ v\ p)\ a = Const\ a * (Var\ v)$
$\hat{}\ p$
⟨*proof*⟩

lemma *coeff-monom* :
$coeff\ (monom\ m\ a)\ m' = (if\ m{=}m'\ then\ a\ else\ 0)$
⟨*proof*⟩

lemma *monom-eq-var*:
$monom\ (Abs\text{-}poly\text{-}mapping\ (\lambda v'.\ (Suc\ 0)\ when\ v{=}v'))\ 1 = MPoly\ (Var_0\ v)$
⟨*proof*⟩

lemma *monom-eq-power-var*:
$monom\ (Abs\text{-}poly\text{-}mapping\ (\lambda v'.\ n\ when\ v = v'))\ 1 = MPoly\ (Var_0\ v)\hat{}n$
⟨*proof*⟩

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*⟩

lemma *finite-sum-mpoly-commute*:
$finite\ S \Longrightarrow (\sum m{\in}S.\ MPoly\ (f\ m)) = MPoly\ (\sum m{\in}S.\ f\ m)$
⟨*proof*⟩

lemma *finite-prod-mpoly-commute*:
$finite\ S \Longrightarrow (\prod m{\in}S.\ MPoly\ (f\ m)) = MPoly\ (\prod m{\in}S.\ f\ m)$
⟨*proof*⟩

lemma *power-mpoly-commute*: $MPoly\ a\ \hat{}\ p = MPoly\ (a\ \hat{}\ p)$
⟨*proof*⟩

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*⟩

lemma *coeff-sum-monom*:
assumes *finite* $\{m.\ f\ m \neq 0\}$
shows *coeff* (*Sum-any* ($\lambda m.$ *monom m* (*f m*))) = *f*
⟨*proof*⟩

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*⟩

lemma *cst-poly-times-monom*: *MPoly* ($Const_0$ (*a*::('*a*::*semiring-1*))) $*$ *monom m b* = *monom m* ($a{*}b$)
⟨*proof*⟩

lemma *cst-poly-times-monom-one*: *MPoly* ($Const_0$ (*a*::('*a*::*semiring-1*))) $*$ *monom m 1* = *monom m a*
⟨*proof*⟩

lemma *poly-eq-sum-monom*: *P* = *Sum-any* ($\lambda m.$ *monom m* (*coeff P m*))
⟨*proof*⟩

lemma *poly-eq-sum-monom-alt*: *P* = ($\sum m \in$(*keys* (*mapping-of P*)). *monom m* (*coeff P m*))
⟨*proof*⟩

lemma *coeff-sum*:
fixes *P*::- $\Rightarrow$ '*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*⟩

lemma *coeff-var-power-le*:
$j \le i \Longrightarrow$ *MPoly-Type.coeff* (*Var v* ^ *j* $*$ *P*) (*Poly-Mapping.single v i*)
= *MPoly-Type.coeff P* (*Poly-Mapping.single v* ($i - j$))
⟨*proof*⟩

lemma *coeff-var-power-eq*: *MPoly-Type.coeff* (*Var v* ^ *i*) (*Poly-Mapping.single v i*) = *1*
⟨*proof*⟩

lemma *coeff-const*: $i > 0 \Longrightarrow$ *MPoly-Type.coeff* (*Const c*) (*Poly-Mapping.single v i*) = *0*
⟨*proof*⟩

lemma *mpoly-univariate-expansion*:
fixes *P*::'*a*::*comm-semiring-1 mpoly* **and** *v*::*nat*
assumes *univariate*: *vars P* $\subseteq \{v\}$

shows $P = Sum\text{-}any\ (\lambda i.\ monom\ (Poly\text{-}Mapping.single\ v\ i)\ (coeff\ P\ (Poly\text{-}Mapping.single\ v\ i)))$
⟨*proof*⟩

lemma *term-expansion-lemma-1*: $i \neq [] \Longrightarrow$
$Poly\text{-}Mapping.single\ (Abs\text{-}poly\text{-}mapping\ ((!_0)\ i))\ (1 :: 'a::comm\text{-}semiring\text{-}1) =$
$(\prod s = 0..(length\ i - 1).\ Var_0\ s\ \hat{}\ (i\ !\ s))$
⟨*proof*⟩

lemma *term-expansion-lemma-2*: $i \neq [] \Longrightarrow$
$monom\ (Abs\text{-}poly\text{-}mapping\ ((!_0)\ i))\ c =$
$MPoly\ (Const_0\ c) * MPoly\ (\prod s = 0..length\ i - 1.\ Var_0\ s\ \hat{}\ (i\ !\ s))$
⟨*proof*⟩

lemma *term-expansion*: $i \neq [] \Longrightarrow$
$monom\ (Abs\text{-}poly\text{-}mapping\ ((!_0)\ i))\ c =$
$Const\ c * (\prod s = 0..length\ i - 1.\ Var\ s\ \hat{}\ (i\ !\ s))$
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *sample-prefix-cong*:
$(\forall x<n.\ f\ x = g\ x) \Longrightarrow sample\text{-}prefix\ n\ f = sample\text{-}prefix\ n\ g$
⟨*proof*⟩

lemma *sample-prefix-inv-nth0*: $(\forall i\geq n.\ f\ i = 0) \Longrightarrow f = (!_0)\ (sample\text{-}prefix\ n\ f)$
⟨*proof*⟩

lemma *sample-prefix-inj*:
$inj\text{-}on\ (\lambda f.\ sample\text{-}prefix\ n\ f)\ \{f.\ \forall i\geq n.\ (f\ i :: 'a::zero) = 0\}$
⟨*proof*⟩

lemma *lookup-nth0-total-degree*:
$lookup\ (mapping\text{-}of\ P)\ (Abs\text{-}poly\text{-}mapping\ ((!_0)\ i)) \neq 0 \Longrightarrow$
$sum\text{-}list\ i \leq total\text{-}degree\ P$
⟨*proof*⟩

lemma *prod-monom*: $finite\ S \Longrightarrow prod\ (\lambda s.\ monom\ (x\ s)\ (a\ s))\ S = monom\ (sum\ x\ S)\ (prod\ a\ S)$
⟨*proof*⟩

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

⟨*proof*⟩

lemma *monom-expansion*:
  shows *monom x c* = *Const c* ∗ (∏ *s*∈*keys x*. *Var s* ^ (*lookup x s*))
  ⟨*proof*⟩

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*) ∗ (∏ *s* = *0*..*max-vars P*. *Var s* ^ (*lookup x s*))
⟨*proof*⟩

lemma *mpoly-multivariate-expansion′*:
  fixes *P*::′*a*::{*ring-no-zero-divisors*,*comm-semiring-1*} *mpoly*
  shows *P* = (∑ *m*∈*keys* (*mapping-of P*).
    Const (*coeff P m*) ∗ (∏ *s* = *0*..*max-vars P*. (*Var s*) ^ (*lookup m s*)))
  ⟨*proof*⟩

lemma *mpoly-multivariate-expansion*:
  fixes *P*::′*a*::*comm-semiring-1 mpoly*
  shows *P* = (∑ *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*⟩

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* ≡ (λ*d*. *coeff P* (*Poly-Mapping*.*single v d*))
  shows *P* = (∑ *d* = *0*..*q*. *Const* (*coeff-P d*) ∗ (*Var v*) ^ *d*)
  ⟨*proof*⟩

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* ⇒ ′*a*::*comm-semiring-1 mpoly*) ⇒ (*nat* ⇒$_0$ *nat*) ⇒ ′*a mpoly*
  where *poly-subst-monom f t* = (∏ *x*. (*f x*) ^ (*lookup t x*))

definition *poly-subst* :: (*nat* ⇒ *'a*::*comm-semiring-1 mpoly*) ⇒ *'a mpoly* ⇒ *'a mpoly*
where *poly-subst f p* = *Sum-any* (λ*t*. (*Const* (*coeff p t*)) ∗ (*poly-subst-monom f t*))

definition *poly-subst-list* **where** *poly-subst-list* ≡ *poly-subst* ∘ ($!_0$)
abbreviation *insertion-list* **where** *insertion-list* ≡ *insertion* ∘ ($!_0$)

lemma *poly-subst-monom-alt*: *poly-subst-monom f t* = (∏ *x* ∈ *keys t*. (*f x*) ^ (*lookup t x*))
⟨*proof*⟩

lemma *poly-subst-alt*: *poly-subst f p* = (∑ *t* ∈ *keys* (*mapping-of p*). (*Const* (*coeff p t*)) ∗ (*poly-subst-monom f t*))
⟨*proof*⟩

lemma *poly-subst-list-id*:
fixes *p* :: *'a*::{*comm-semiring-1*,*ring-no-zero-divisors*} *mpoly*
assumes *k* ≥ *max-vars p*
shows *poly-subst-list* (*map Var* [*0*..<*Suc k*]) *p* = *p*
⟨*proof*⟩

lemma *insertion-poly-subst-monom*:
insertion g (*poly-subst-monom f t*) = (∏ *x*. (*insertion g* (*f x*)) ^ (*lookup t x*))
⟨*proof*⟩

lemma *insertion-poly-subst*:
insertion g (*poly-subst f p*) = *insertion* ((*insertion g*) ∘ *f*) *p*
⟨*proof*⟩

lemma *insertion-nth0*: *insertion f* (*l* $!_0$ *x*) = (*map* (*insertion f*) *l*) $!_0$ *x*
⟨*proof*⟩

lemma *poly-subst-monom-zero* [*simp*]:
poly-subst-monom f 0 = *1*
⟨*proof*⟩

lemma *poly-subst-monom-single* [*simp*]:
poly-subst-monom f (*Poly-Mapping.single v 1*) = *f v*
⟨*proof*⟩

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
⟨*proof*⟩

lemma *poly-subst-zero* [*simp*]:
poly-subst f 0 = *0*
⟨*proof*⟩

lemma *poly-subst-one* [*simp*]:
poly-subst f 1 = *1*
⟨*proof*⟩

lemma *poly-subst-Var* [*simp*]:
poly-subst f (*Var v*) = *f v*
⟨*proof*⟩

lemma *poly-subst-Const* [*simp*]:
poly-subst f (*Const c*) = (*Const c*)
⟨*proof*⟩

lemma *poly-subst-numeral*[*simp*]:
poly-subst f (*numeral n*) = (*numeral n*)
⟨*proof*⟩

lemma *poly-subst-add* [*simp*]:
poly-subst f (*P* + *Q*) = *poly-subst f P* + *poly-subst f Q*
⟨*proof*⟩

lemma *poly-subst-uminus* [*simp*]:
poly-subst f (− *P*) = − *poly-subst f P*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *poly-subst-sum*:
poly-subst f (*sum P A*) = *sum* (*poly-subst f* ∘ *P*) *A*
⟨*proof*⟩

lemma *poly-subst-mult* [*simp*]:
poly-subst f (*P* ∗ *Q*) = *poly-subst f P* ∗ *poly-subst f Q*
⟨*proof*⟩

lemma *poly-subst-prod*:
poly-subst f (*prod P A*) = *prod* (*poly-subst f* ∘ *P*) *A*
⟨*proof*⟩

lemma *poly-subst-monom-id*: *poly-subst-monom* (*Var*) *t* = *monom t 1*
⟨*proof*⟩

lemma *poly-subst-id*: *poly-subst* (*Var*) *p* = *p*
⟨*proof*⟩

Vars of substitutions

lemma *vars-poly-subst-monom*: *vars* (*poly-subst-monom f t*) ⊆ ⋃ (*vars* ‘ (*f* ‘ *keys* *t*))
⟨*proof*⟩

lemma *vars-poly-subst-monom′*: *vars* (*poly-subst-monom* ($(!_0)$ *ls*) *t*) ⊆ ⋃ (*vars* ‘ *set ls*)
⟨*proof*⟩

lemma *vars-poly-subst-list*: *vars* (*poly-subst-list ls p*) ⊆ ⋃ (*vars* ‘ *set ls*)
⟨*proof*⟩

lemma *vars-poly-subst-monom-bounded*:
∀ *v*∈(*keys t*). *v* ≤ *bound* ⟹ *vars* (*poly-subst-monom* ($(!_0)$ *ls*) *t*) ⊆ ⋃ (*vars* ‘ *set* (*take* (*Suc bound*) *ls*))
⟨*proof*⟩

lemma *aux0*: *max-vars p* ≤ *bound* ⟹ *m* ∈ *keys* (*mapping-of p*) ⟹ ∀ *v*∈(*keys* *m*). *v* ≤ *bound*
⟨*proof*⟩

lemma *vars-poly-subst-list-bounded*:
assumes *max-vars p* ≤ *bound*
shows *vars* (*poly-subst-list ls p*) ⊆ ⋃ (*vars* ‘ *set* (*take* (*Suc bound*) *ls*))
⟨*proof*⟩

lemma *vars-poly-subst*:
vars (*poly-subst f p*) ⊆ (⋃ *t* ∈ *vars p*. *vars* (*f t*))
⟨*proof*⟩

lemma *max-vars-poly-subst-list-general*:
shows *max-vars* (*poly-subst-list ls p*) ≤ *Max* (*max-vars* ‘ *set ls*)
⟨*proof*⟩

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*⟩

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*
⟨*proof*⟩

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*
$\leq$ ($\sum$ *t* | *v*$\in$*vars* (*f t*). *degree-monom m t* $*$ *degree* (*f t*) *v*)
⟨*proof*⟩

lemma *degree-poly-subst*:
fixes *p* :: *'a::comm-ring-1 mpoly*
fixes *f* :: *nat* $\Rightarrow$ *'a mpoly*
assumes *finite* {*k*. *f k* $\neq$ *0*}
shows *degree* (*poly-subst f p*) *v* $\leq$ ($\sum$ *t* | *v* $\in$ *vars* (*f t*). *degree p t* $*$ *degree* (*f t*) *v*)
⟨*proof*⟩

lemma *degree-poly-subst'*:
fixes *p* :: *'a::comm-ring-1 mpoly*
fixes *f* :: *nat* $\Rightarrow$ *'a mpoly*
assumes *finite* {*k*. *f k* $\neq$ *0*}
shows *degree* (*poly-subst f p*) *v* $\leq$ ($\sum$ *t*$\in$*vars p*. *degree p t* $*$ *degree* (*f t*) *v*)
⟨*proof*⟩

lemma *degree-poly-subst-list*:
fixes *p* :: *'a::comm-ring-1 mpoly*
shows *degree* (*poly-subst-list ls p*) *v* $\leq$ ($\sum$ *t* | *v* $\in$ *vars* (*ls* $!_0$ *t*). *degree p t* $*$ *degree* (*ls* $!_0$ *t*) *v*)
⟨*proof*⟩

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*)
⟨*proof*⟩

Total degree of substitions

lemma *deg-imp-tot-deg-zero*: ($\forall$ *v* $\in$ *vars P*. *degree P v* = *0*) $\Longrightarrow$ *total-degree P* = *0*
⟨*proof*⟩

lemma *total-degree-poly-subst-monom*:
fixes *f*
defines *degree-monom* $\equiv$ (λ*m t*. (*lookup m*) *t*)
shows *total-degree* (*poly-subst-monom f m*)
$\leq$ ($\sum$ *t*$\in$*keys m*. *degree-monom m t* $*$ *total-degree* (*f t*))
⟨*proof*⟩

lemma *total-degree-poly-subst*:
shows *total-degree* (*poly-subst f p*) $\leq$ ($\sum$ *t*$\in$*vars p*. *degree p t* $*$ *total-degree* (*f t*))
⟨*proof*⟩

lemma *total-degree-poly-subst-list*:
fixes *p* :: *'a::comm-ring-1 mpoly*

shows *total-degree* (*poly-subst-list ls p*) ≤ (∑ *t*∈*vars p*. *degree p t* ∗ *total-degree* (*ls* $!_0$ *t*))
⟨*proof*⟩

lemma *total-degree-poly-subst-list′*:
fixes *p* :: ′*a*::*comm-ring-1 mpoly*
assumes *max-vars p* ≤ *length ls*
shows *total-degree* (*poly-subst-list ls p*) ≤ (∑ *t*≤*length ls*. *degree p t* ∗ *total-degree* (*ls* $!_0$ *t*))
⟨*proof*⟩

lemma *total-degree-poly-subst-list′′*:
fixes *p* :: ′*a*::*comm-ring-1 mpoly*
assumes ∀ *i* ≤ *length ls*. *card* (*vars* (*ls* ! *i*)) ≤ *1*
shows *total-degree* (*poly-subst-list ls p*) ≤
length ls ∗ (∑ *t* ≤ *length ls*. *degree p t* ∗ *total-degree* (*ls* $!_0$ *t*))
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *of-int-mpoly-zero* [*of-int-mpoly-simps*]:
of-int-mpoly 0 = *0*
⟨*proof*⟩

lemma *eq-onp-intF*:
fixes $\mathcal{F}$ :: *int mpoly*
shows *eq-onp* (λ*f*. *finite* {*x*. *f x* ≠ *0*}) (λ*x*. *of-int* (*lookup* (*mapping-of* $\mathcal{F}$) *x*))
(λ*x*. *of-int* (*lookup* (*mapping-of* $\mathcal{F}$) *x*))
⟨*proof*⟩

lemma *of-int-mpoly-one* [*of-int-mpoly-simps*]:
of-int-mpoly 1 = *1*
⟨*proof*⟩

lemma *of-int-mpoly-Var* [*of-int-mpoly-simps*]:
of-int-mpoly (*Var n*) = *Var n*
⟨*proof*⟩

lemma *of-int-mpoly-Const* [*of-int-mpoly-simps*]:
of-int-mpoly (*Const c*) = *Const* (*of-int c*)
⟨*proof*⟩

lemma *of-int-mpoly-numeral* [*of-int-mpoly-simps*]:
of-int-mpoly (*numeral n*) = *numeral n*
⟨*proof*⟩

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*⟩

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*⟩

lemma *of-int-mpoly-diff* [*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*⟩

lemma *of-int-mpoly-mult* [*of-int-mpoly-simps*]:
fixes $\mathcal{F}$ $\mathcal{G}$ :: *int mpoly*
shows (*of-int-mpoly* ($\mathcal{F}$ ∗ $\mathcal{G}$) :: (′*a*::*ring-1*) *mpoly*) = *of-int-mpoly* $\mathcal{F}$ ∗ *of-int-mpoly* $\mathcal{G}$
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

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*⟩

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

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*
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *of-int-general-Const*: (*of-int x* :: (*'a*::*ring-1*) *mpoly*) = *of-int-mpoly* (*Const x*)
⟨*proof*⟩

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

lemma *vars-of-int-mpoly*:
vars (*of-int-mpoly P* :: (*'a* :: {*comm-semiring-1*, *ring-1*}) *mpoly*) ⊆ *vars P*
⟨*proof*⟩

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*⟩

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*⟩

corollary *bit-set-finite*: *finite* (*bit-set n*)
⟨*proof*⟩

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*⟩

corollary *count-bits-def-sum*: *n* < *2*^*k* ⟹ *count-bits n* = (∑ *i*<*k*. *n* ¡ *i*)
⟨*proof*⟩

corollary *count-bits-bounded*: *n* < *2*^*k* ⟹ *count-bits n* ≤ *k*
⟨*proof*⟩

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

lemma *carry-set-subset*: *carry-set a b* $\subseteq$ {..*max a b*}
⟨*proof*⟩

corollary *carry-set-finite*: *finite* (*carry-set a b*)
⟨*proof*⟩

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

lemma *carry-set-subset-variant*: *a* + *b* < *2^k* $\Longrightarrow$ *carry-set a b* $\subseteq$ {..<*k*}
⟨*proof*⟩

corollary *count-carries-def-sum*: *a* + *b* < *2^k* $\Longrightarrow$ *count-carries a b* = ($\sum$ *i*<*k*. *bin-carry a b i*)
⟨*proof*⟩

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

lemma *bit-set-0*[*simp*]: *bit-set 0* = {}
⟨*proof*⟩

corollary *count-bits-0*[*simp*]: *count-bits 0* = *0*
⟨*proof*⟩

lemma *bit-set-1*[*simp*]: *bit-set 1* = {*0*}
⟨*proof*⟩

corollary *count-bits-1*[*simp*]: *count-bits 1* = *1*
⟨*proof*⟩

lemma *carry-set-n0*[*simp*]: *carry-set n 0* = {}
⟨*proof*⟩
lemma *carry-set-0n*[*simp*]: *carry-set 0 n* = {}
⟨*proof*⟩

corollary *count-carries-n0*[*simp*]: *count-carries n 0* = *0*
⟨*proof*⟩
corollary *count-carries-0n*[*simp*]: *count-carries 0 n* = *0*
⟨*proof*⟩

lemma *carry-set-sym*: *carry-set a b* = *carry-set b a*
⟨*proof*⟩

corollary *count-carries-sym*: *count-carries a b* = *count-carries b a*
⟨*proof*⟩

lemma *aux-geometric-sum*:
(*x*::*nat*) > *1* $\Longrightarrow$ (*x*−*1*) ∗ ($\sum$ *i*<*n*. *x^i*) = *x^n* − *1*
⟨*proof*⟩

lemma *aux-digit-sum-bound*:

assumes $1 < (b::nat)$ **and** $\forall i<q.\ f\ i < b$
shows $(\sum i<q.\ f\ i * b\hat{\ }i) < b\hat{\ }q$
⟨*proof*⟩

lemma *carry-set-same*[*simp*]: *carry-set a a* = *Suc* ' *bit-set a*
(**is** *?A* = *?B*)
⟨*proof*⟩

corollary *count-carries-same*[*simp*]: *count-carries a a* = *count-bits a*
⟨*proof*⟩

lemma *bit-set-pow2*[*simp*]: *bit-set* $(2\hat{\ }k) = \{k\}$
⟨*proof*⟩

corollary *count-bits-pow2*[*simp*]: *count-bits* $(2\hat{\ }k) = 1$
⟨*proof*⟩

lemma *bit-set-block-ones*[*simp*]: *bit-set* $(2\hat{\ }k - 1) = \{..<k\}$
⟨*proof*⟩

corollary *count-bits-block-ones*[*simp*]: *count-bits* $(2\hat{\ }k-1) = k$
⟨*proof*⟩

The binary complement of a number with k bits

lemma *nth-bit-complement*:
$a < 2\hat{\ }k \Longrightarrow (2\hat{\ }k-1-a)$ ¡ i = (*if* $i < k$ *then* $1 - (a$ ¡ $i)$ *else* 0)
⟨*proof*⟩

lemma *bit-set-complement*:
$a < 2\hat{\ }k \Longrightarrow$ *bit-set* $(2\hat{\ }k-1-a) = \{..<k\}$ − *bit-set a*
⟨*proof*⟩

corollary *count-bits-complement*:
$a < 2\hat{\ }k \Longrightarrow$ *count-bits* $(2\hat{\ }k-1-a) = k$ − *count-bits a*
⟨*proof*⟩

lemma *carry-set-pow2-block-ones*[*simp*]: *carry-set* $(2\hat{\ }k)$ $(2\hat{\ }k-1) = \{\}$
⟨*proof*⟩

corollary *count-carries-pow2-block-ones*[*simp*]: *count-carries* $(2\hat{\ }k)$ $(2\hat{\ }k-1) = 0$
⟨*proof*⟩

lemma *bit-set-add-shift*:
$a < 2\hat{\ }k \Longrightarrow$ *bit-set* $(a + b * 2\hat{\ }k)$ = *bit-set a* ∪ ((+) *k*) ' *bit-set b*
(**is** - $\Longrightarrow$ *?A* = *?B*)
⟨*proof*⟩

corollary *count-bits-add-shift*:
$a < 2\hat{\ }k \Longrightarrow$ *count-bits* $(a + b * 2\hat{\ }k)$ = *count-bits a* + *count-bits b*

⟨*proof*⟩

corollary *count-bits-even*[*simp*]: *count-bits* (*2*∗*n*) = *count-bits n*
⟨*proof*⟩

corollary *count-bits-odd*[*simp*]: *count-bits* (*2*∗*n*+*1*) = *1* + *count-bits n*
⟨*proof*⟩

lemma *count-bits-digitwise*:
assumes *1* ≤ *k* **and** ∀ *i*<*q*. *f i* < *2*ˆ*k*
shows *count-bits* (∑ *i*<*q*. *f i* ∗ (*2*ˆ*k*)ˆ*i*) = (∑ *i*<*q*. *count-bits* (*f i*))
⟨*proof*⟩

lemma *count-carries-count-bits*:
count-bits (*a*+*b*) + *count-carries a b* = *count-bits a* + *count-bits b*
⟨*proof*⟩

corollary *count-bits-add-le*: *count-bits* (*a*+*b*) ≤ *count-bits a* + *count-bits b*
⟨*proof*⟩

{@*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*)
⟨*proof*⟩

lemma *count-bits-sum-le*:
assumes *S-fin*: *finite S*
shows *count-bits* (*sum f S*) ≤ (∑ *i*∈*S*. *count-bits* (*f i*))
⟨*proof*⟩

lemma *aux1-carry-set-add-shift*:
a < *2*ˆ*k* ⟹ *c* < *2*ˆ*k* ⟹ *i* ≤ *k* ⟹ *bin-carry* (*a*+*b*∗*2*ˆ*k*) (*c*+*d*∗*2*ˆ*k*) *i* = *bin-carry a c i*
⟨*proof*⟩

lemma *aux2-carry-set-add-shift*:
a < *2*ˆ*k* ⟹ *c* < *2*ˆ*k* ⟹ *k* ≤ *i* ⟹ *bin-carry* (*a*+*b*∗*2*ˆ*k*) (*c*+*d*∗*2*ˆ*k*) *i* ≥ *bin-carry b d* (*i*−*k*)
⟨*proof*⟩

lemma *aux3-carry-set-add-shift*:
a + *c* < *2*ˆ*k* ⟹ *k* ≤ *i* ⟹ *bin-carry* (*a*+*b*∗*2*ˆ*k*) (*c*+*d*∗*2*ˆ*k*) *i* = *bin-carry b d* (*i*−*k*)
⟨*proof*⟩

lemma *carry-set-add-shift*:
a < *2*ˆ*k* ⟹ *c* < *2*ˆ*k* ⟹ *carry-set a c* ∪ ((+) *k*) ‘ *carry-set b d* ⊆ *carry-set* (*a*+*b*∗*2*ˆ*k*) (*c*+*d*∗*2*ˆ*k*)
(**is** - ⟹ - ⟹ *?A1* ∪ *?A2* ⊆ *?B*)

⟨*proof*⟩

corollary *count-carries-add-shift*:
  $a < 2\hat{}k \Longrightarrow c < 2\hat{}k \Longrightarrow$ *count-carries* $(a + b*2\hat{}k)$ $(c + d*2\hat{}k)$
  $\geq$ *count-carries* a c + *count-carries* b d
⟨*proof*⟩

lemma *carry-set-add-shift-no-overflow*:
  $a + c < 2\hat{}k \Longrightarrow$ *carry-set* $(a+b*2\hat{}k)$ $(c+d*2\hat{}k)$ = *carry-set* a c $\cup$ $((+)\ k)$ ‘ *carry-set* b d
  (**is** - $\Longrightarrow$ $?A = ?B$)
⟨*proof*⟩

corollary *count-carries-add-shift-no-overflow*:
  $a + c < 2\hat{}k \Longrightarrow$ *count-carries* $(a + b*2\hat{}k)$ $(c + d*2\hat{}k)$ = *count-carries* a c + *count-carries* b d
⟨*proof*⟩

corollary *count-carries-even-even*: *count-carries* $(2*a)$ $(2*b)$ = *count-carries* a b
  ⟨*proof*⟩

lemma *count-carries-digitwise*:
  assumes $1 \leq k$ **and** $\forall i<q.\ f\ i < 2\hat{}k \wedge g\ i < 2\hat{}k$
  shows *count-carries* $(\sum i<q.\ f\ i * (2\hat{}k)\hat{}i)$ $(\sum i<q.\ g\ i * (2\hat{}k)\hat{}i) \geq$
    $(\sum i<q.$ *count-carries* $(f\ i)$ $(g\ i))$
⟨*proof*⟩

corollary *count-carries-digitwise-specific*:
  assumes $1 \leq k$ **and** $\forall i<q.\ f\ i < 2\hat{}k \wedge g\ i < 2\hat{}k$
  shows $i < q \Longrightarrow$ *count-carries* $(\sum i<q.\ f\ i * (2\hat{}k)\hat{}i)$ $(\sum i<q.\ g\ i * (2\hat{}k)\hat{}i)$
$\geq$
    count-carries $(f\ i)$ $(g\ i)$
⟨*proof*⟩

lemma *count-carries-digitwise-no-overflow*:
  assumes $k \geq 1$ **and** $\forall i<q.\ f\ i + g\ i < 2\hat{}k$
  shows *count-carries* $(\sum i<q.\ f\ i * (2\hat{}k)\hat{}i)$ $(\sum i<q.\ g\ i * (2\hat{}k)\hat{}i) =$
    $(\sum i<q.$ *count-carries* $(f\ i)$ $(g\ i))$
⟨*proof*⟩

lemma *carry-set-empty-iff*:
  carry-set a b = {} $\longleftrightarrow$ $(\forall i.\ a$ ¡ $i + b$ ¡ $i \leq 1)$
⟨*proof*⟩

corollary *count-carries-zero-iff*:
  count-carries a b = 0 $\longleftrightarrow$ $(\forall i.\ a$ ¡ $i + b$ ¡ $i \leq 1)$
  ⟨*proof*⟩

lemma *no-carry-no-overflow*:

assumes $a < 2\hat{}k$ **and** $b < 2\hat{}k$ **and** *count-carries a b* $= 0$
shows $a + b < 2\hat{}k$
⟨*proof*⟩

lemma *count-carries-divisibility-pow2*: *count-carries* $(2\hat{}k-1)$ $x = 0 \longleftrightarrow 2\hat{}k$ *dvd* x
⟨*proof*⟩

lemma *nth-digit-gen-power-series-general*:
assumes $1 < b$ **and** $\forall k \leq q.\ f\ k < b$
shows *nth-digit* $(\sum k=0..q.\ f\ k * b\hat{}k)$ $i\ b$ = (*if* $i \leq q$ *then* $f\ i$ *else* 0)
(**is** *nth-digit ?X* - - = -)
⟨*proof*⟩

lemma *aux-count-bits-multiplicity*:
count-bits (*Suc x*) + *multiplicity* 2 (*Suc x*) = *count-bits x* + 1
⟨*proof*⟩

lemma *count-bits-multiplicity*:
count-bits x = *multiplicity* 2 (2∗*x choose x*)
⟨*proof*⟩

corollary *count-bits-divibility-binomial*:
$2\hat{}k$ *dvd* (2∗*x choose x*) ⟷ $k \leq$ *count-bits x*
⟨*proof*⟩

end
theory *Utils*
imports *Main*
begin

definition *is-square*::*int*⇒*bool* **where**
is-square n = (∃ *k*. $n = k\hat{}2$)

definition *is-power2*::*int*⇒*bool* **where**
is-power2 x ≡ (∃ *n*::*nat*. $x = 2\hat{}n$)

lemma *is-power2-ge1*: *is-power2 x* ⟹ $1 \leq x$
⟨*proof*⟩

lemma *is-power2-mult*[*simp*]: *is-power2 x* ⟹ *is-power2 y* ⟹ *is-power2* (*x* ∗ *y*)
⟨*proof*⟩

lemma *is-power2-pow*[*simp*]: *is-power2 x* ⟹ *is-power2* ($x\hat{}n$)
⟨*proof*⟩

lemma *is-power2-1*[*simp*]: *is-power2 1*

⟨*proof*⟩
lemma *is-power2-2*[*simp*]: *is-power2 2*
⟨*proof*⟩
lemma *is-power2-4*[*simp*]: *is-power2 4*
⟨*proof*⟩

lemma *is-power2-div2*: $is\text{-}power2\ x \Longrightarrow 2 \leq x \Longrightarrow is\text{-}power2\ (x\ div\ 2)$
⟨*proof*⟩

lemma *digit-repr-lt*:
fixes $q :: nat$
fixes $b :: int$
assumes $b > 1$
assumes $\forall k.\ f\ k < b$
shows $(\sum k = 0..q.\ f\ k * b\ \hat{}\ k) < b\hat{}(Suc\ q)$
⟨*proof*⟩

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*⟩

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*⟩

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 $\delta\ \nu\ \mathcal{B}\ b\ C$ :: *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**
$m\ j = (\mathit{if}\ j \in n\ `\ \{1..\nu\}\ \mathit{then}\ \mathcal{B} - b\ \mathit{else}\ \mathcal{B} - 1)$

$mask(b, \mathcal{B}, \delta, \nu)$

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

lemma *b-ge-1*: $1 \leq b$
  ⟨*proof*⟩

lemma *b-dvd-*$\mathcal{B}$: b *dvd* $\mathcal{B}$
  ⟨*proof*⟩

lemma *n-inj-on*: *inj-on* $n\ A$
  ⟨*proof*⟩

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*⟩

lemma *direct-tau-zero*:

assumes *z-bound*: ∀ *i*. *z i* < *b*
and *g-code*: *g* = (∑ *i*=*1*..*ν*. *z i* ∗ *B* ^(*n i*))
shows *τ g M* = *0*
⟨*proof*⟩

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*⟩

lemma *masking-lemma*:
(∃ *z*. (∀ *i*. *z i* < *b*) ∧ *g* = (∑ *i*=*1*..*ν*. *z i* ∗ *B* ^(*n i*))) ⟷ (*g* < *C* ∧ *τ g M* = *0*)
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *mfact-Cons*[*simp*]: *mfact* (*i* # *is*) = *fact i* ∗ *mfact is*
⟨*proof*⟩

lemma *mfact'-equiv*: *mfact'* = *mfact* ⟨*proof*⟩

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*
⟨*proof*⟩

lemma *mpow-any-Nil*[*simp*]: *mpow xs* [] = *1*
⟨*proof*⟩

lemma *mpow-Cons*[*simp*]: *mpow* (*x* # *xs*) (*n* # *ns*) = (*x* ^ *n*) ∗ (*mpow xs ns*)
⟨*proof*⟩

lemma *mpow′-equiv*: *mpow′* = *mpow* ⟨*proof*⟩

lemma *multinomial′-dvd*: *mfact ks dvd fact* (*sum-list ks*)
⟨*proof*⟩

lemma *mchoose-dvd*: *sum-list ks* ≤ *n* ⟹
mfact ks ∗ *fact* (*n* − *sum-list ks*) *dvd fact n*
⟨*proof*⟩

lemma *mchoose-le*:
sum-list ks ≤ *n* ⟹ *mfact ks* ∗ *fact* (*n* − *sum-list ks*) ≤ *fact n*
⟨*proof*⟩

The multinomial coefficient.

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

lemma *multinomial′-Nil*[*simp*]: *multinomial′* [] = *1*
⟨*proof*⟩

lemma *multinomial′-Cons*[*simp*]: *multinomial′* (*k* # *ks*) =
((*k* + *sum-list ks*) *choose k*) ∗ *multinomial′ ks*
⟨*proof*⟩

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*⟩

The proof is by induction on *xs*, and using the standard binomial theorem $((?a + ?b)^{?n} = (\sum k \le ?n.\ of\text{-}nat\ (?n\ choose\ k) * ?a^k * ?b^{?n - k}))$ 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*⟩

This version of the multinomial theorem is also useful.

corollary *multinomial-ring-alt*:
  fixes *xs* :: *'a*::*comm-semiring-1 list*
  shows $(1 + \textit{sum-list xs})\hat{}n = (\sum \textit{ks} \mid \textit{length ks} = \textit{length xs} \wedge \textit{sum-list ks} \leq n.$
    $\textit{of-nat } (n \textit{ mchoose ks}) * \textit{mpow xs ks})$
⟨*proof*⟩

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** $\delta = \textit{total-degree } P$
definition ν::*nat* **where** $\nu = \textit{max-vars } P$
definition *b*::*nat* **where** $b \equiv \mathcal{B} \textit{ div } 2$
definition *n*::*nat* ⇒ *nat* **where** $n\ j = (\delta+1)\hat{}j$

definition *X*::*int mpoly* **where** $X = \textit{Var } 0$

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

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

We will often use this set as indices of sums.

definition δ*-tuples* :: (*nat list*) *set* **where**
  $\delta\textit{-tuples} \equiv \{i.\ \textit{length } i = \nu + 1 \wedge \textit{sum-list } i \leq \delta\}$

definition *P-coeff* :: *nat list* ⇒ *int* **where**
  $\textit{P-coeff } i \equiv \textit{coeff } P\ (\textit{Abs-poly-mapping } ((!_0)\ i))$

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*ˆ(*D-exponent i*)

This is really a univariate polynomial

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

Definition of the constant K

definition *R*::*int mpoly* **where** *R* ≡ (*1*+*c*)ˆ*δ* ∗ *D*
definition *S*::*nat* **where** *S* ≡ (∑ *i*≤(*2*∗*δ*+*1*) ∗ *n ν*. *b* ∗ *ℬ*ˆ*i*)
definition *K*::*int* **where** *K* ≡ *insertion 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*: *δ* > *0*
and *L-pos*: *L* > *0*
and *L-lower-bound*: *L* ≥ *max-coeff P*
and *len-z*: *length z* = *ν*+*1*
and *ℬ-even*: *2 dvd ℬ*
and *ℬ-lower-bound*: *ℬ* > *2* ∗ *fact δ* ∗ (*nat L*) ∗ (*1* + *sum-list z*)ˆ*δ*
begin

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

lemma *ℬ-ge-1*[*simp*]: *ℬ* ≥ *1*
⟨*proof*⟩

lemma *ℬ-gt-2*[*simp*]: *ℬ* > *2*
⟨*proof*⟩

Also for convenience.

lemma *B-ge-2*[*simp*]: $\mathcal{B} \geq 2$
⟨*proof*⟩

lemma *b-def-reverse*: $2 * b = \mathcal{B}$ ⟨*proof*⟩

lemma *b-ge-1*[*simp*]: $b \geq 1$
⟨*proof*⟩

lemma *n-ge-1*[*simp*]: $n\ j \geq 1$
⟨*proof*⟩

lemma *L-lower-bound-specialize*: $L \geq abs\ (P\text{-}coeff\ i)$
⟨*proof*⟩

lemma *δ-tuples-finite*[*simp*]: *finite δ-tuples*
⟨*proof*⟩

lemma *P-z-insertion*: $insertion\ z\text{-}assign\ P = (\sum i{\in}\delta\text{-}tuples.\ P\text{-}coeff\ i * mpow\ z\ i)$
⟨*proof*⟩

This is essentially an instance of the multinomial theorem.

lemma *c-delta-expansion*:
$(1{+}c)\,\hat{}\,\delta = (\sum i{\in}\delta\text{-}tuples.\ of\text{-}nat\ ((\delta\ mchoose\ i) * mpow\ z\ i) * X\hat{}(\sum s{\le}\nu.\ i!s * n\ s))$
⟨*proof*⟩

lemma *n-νp1-ge-sum*: $i{\in}\delta\text{-}tuples \Longrightarrow n\ (\nu{+}1) \geq (\sum s{\le}\nu.\ i!s * n\ s)$
⟨*proof*⟩

lemma *D-exponent-inj*: *inj-on D-exponent δ-tuples*
⟨*proof*⟩

definition *R-exponent* :: *nat list* ⇒ *nat list* ⇒ *nat* **where**
$R\text{-}exponent\ i\ j = D\text{-}exponent\ j + (\sum s{\le}\nu.\ i!s * n\ s)$

lemma *R-expansion*:
$R = (\sum i{\in}\delta\text{-}tuples.\ (\sum j{\in}\delta\text{-}tuples.$
$of\text{-}int\ ((\delta\ mchoose\ i) * mfact\ j * fact\ (\delta - sum\text{-}list\ j) * mpow\ z\ i * P\text{-}coeff\ j) *$
$X\hat{}(R\text{-}exponent\ i\ j)))$
⟨*proof*⟩

lemma *c-degree-bound*: $degree\ c\ 0 \leq n\ \nu$
⟨*proof*⟩

lemma *D-degree-bound*: $degree\ D\ 0 \leq n\ (\nu{+}1)$
⟨*proof*⟩

lemma *R-degree-bound*: $degree\ R\ 0 \leq (2{*}\delta{+}1) * n\ \nu$
⟨*proof*⟩

lemma *R-univariate*: *vars R* ⊆ {*0*}
⟨*proof*⟩

lemma *R-sum-e*: *R* = (∑ *i*≤(*2*∗δ+*1*) ∗ *n* ν. (*of-int* (*e i*)) ∗ *X*ˆ*i*)
⟨*proof*⟩

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*⟩

This is a key step of the proof.

lemma *e-n-ν1-expression*: *e* (*n* (ν+*1*)) = *fact* δ ∗ *insertion z-assign P*
⟨*proof*⟩

lemma *abs-int*[*simp*]: *abs* (*int x*) = *int x*
⟨*proof*⟩

This is a key step of the proof.

lemma *e-upper-bound*:
abs (*e p*) ≤ *fact* δ ∗ *L* ∗ (*1* + *sum-list z*)ˆδ
⟨*proof*⟩

lemma *e-b-bound*: **shows** *0* < *e j* + *b* **and** *e j* + *b* < 𝓑
⟨*proof*⟩

This is a key step of the proof.

lemma *K-expression*: *K* = (∑ *i*≤(*2*∗δ+*1*) ∗ *n* ν. (*e i* + *int b*) ∗ 𝓑ˆ*i*)
⟨*proof*⟩

lemma δ-*n-positive*: *0* < (*2*∗δ+*1*) ∗ *n* ν
⟨*proof*⟩

lemma *K-lower-bound*: *K* > *int* 𝓑ˆ((*2*∗δ+*1*) ∗ *n* ν)
⟨*proof*⟩

corollary *K-gt-0*: *K* > *0*
⟨*proof*⟩

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 *B-power2*: *is-power2* (*int* $\mathcal{B}$)
begin

lemma *b-power2*: *is-power2* (*int b*)
  ⟨*proof*⟩

lemma *K-upper-bound*: $K < int\ \mathcal{B}\hat{}((2{*}\delta{+}1) * n\ \nu + 1)$
⟨*proof*⟩

lemma *direct-implication*:
  insertion z-assign $P = 0 \Longrightarrow \tau\ (nat\ K)\ ((b{-}1) * \mathcal{B}\hat{}(n\ (\nu{+}1))) = 0$
⟨*proof*⟩

lemma *reverse-implication*:
  $\tau\ (nat\ K)\ ((b{-}1) * \mathcal{B}\hat{}(n\ (\nu{+}1))) = 0 \Longrightarrow$ *insertion z-assign* $P = 0$
⟨*proof*⟩

lemma *tau-rewrite*:
  $\tau\ (2 * nat\ K)\ ((\mathcal{B}{-}2) * \mathcal{B}\hat{}(n\ (\nu{+}1))) = \tau\ (nat\ K)\ ((b{-}1) * \mathcal{B}\hat{}(n\ (\nu{+}1)))$
  ⟨*proof*⟩

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 $K < int\ \mathcal{B}\hat{}((2{*}\delta{+}1) * n\ \nu + 1)$
  ⟨*proof*⟩

end

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

definition *is-nonnegative* :: $(nat \Rightarrow int) \Rightarrow bool$ **where**
  is-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 $A\ P =$

(∀ *a*. (*a* ∈ *A*) ⟷ (∃ *f*. *insertion* (*f*(*0* := *int a*)) *P* = *0*))

definition *is-diophantine-over-N* :: *nat set* ⇒ *bool* **where**
is-diophantine-over-N A = (∃ *P*.
(∀ *a*. (*a* ∈ *A*) ⟷ (∃ *f*. *insertion* (*f*(*0* := *int a*)) *P* = *0* ∧ *is-nonnegative f*)))

definition *is-diophantine-over-N-with* :: *nat set* ⇒ *int mpoly* ⇒ *bool* **where**
is-diophantine-over-N-with A P =
(∀ *a*. (*a* ∈ *A*) ⟷ (∃ *f*. *insertion* (*f*(*0* := *int a*)) *P* = *0* ∧ *is-nonnegative f*))

lemma *is-diophantine-finite-vars*:
assumes *is-diophantine-over-N-with A P*
shows *a* ∈ *A* ⟷ (∃ *f*. *insertion* (*f*(*0* := *int a*)) *P* = *0* ∧ *is-nonnegative f* ∧ (∀ *i* > *max-vars P*. *f i* = *0*))
⟨*proof*⟩

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*))) ⟨*proof*⟩

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*
⟨*proof*⟩

lemma *total-degree-env-zero*[*simp*]: *total-degree-env f 0* = *0*
⟨*proof*⟩

lemma *total-degree-env-one*[*simp*]: *total-degree-env f 1* = *0*
⟨*proof*⟩

lemma *total-degree-env-Const*[*simp*]: *total-degree-env f* (*Const c*) = *0*
⟨*proof*⟩

lemma *total-degree-env-Const-le*: *total-degree-env f* (*Const c*) ≤ *0*
⟨*proof*⟩

lemma *total-degree-env-Var*[*simp*]:
total-degree-env f (*Var i*) = *f i*
⟨*proof*⟩

lemma *total-degree-env-Var-le*: *total-degree-env f* (*Var i*) ≤ *f i*
⟨*proof*⟩

lemma *total-degree-env-neg*: *total-degree-env f* (−*P*) = *total-degree-env f P*
⟨*proof*⟩

lemma *total-degree-env-mult*: *total-degree-env f* (*P* ∗ *Q*) ≤ *total-degree-env f P* + *total-degree-env f Q*
⟨*proof*⟩

lemma *total-degree-env-pow*: *total-degree-env f* (*P* ^ *n*) ≤ *n* ∗ *total-degree-env f P*
⟨*proof*⟩

lemma *total-degree-env-add*: *total-degree-env f* (*P* + *Q*) ≤ *max* (*total-degree-env f P*) (*total-degree-env f Q*)
⟨*proof*⟩

lemma *total-degree-env-diff*:
fixes *P* :: ′*a*::{*ab-group-add*,*zero-neq-one*} *mpoly*
shows *total-degree-env f* (*P* − *Q*) ≤ *max* (*total-degree-env f P*) (*total-degree-env f Q*)
⟨*proof*⟩

lemma *total-degree-env-sum*:
fixes *P* :: ′*a* ⇒ ′*b*::{*ab-group-add*,*zero-neq-one*} *mpoly*
assumes *S-fin*: *finite S*
shows *total-degree-env ctxt* (*sum P S*) ≤ *Max* (*insert 0* ((λ*i*. *total-degree-env ctxt* (*P i*)) ‘ *S*))
⟨*proof*⟩

lemma *total-degree-env-prod*:
assumes *S-fin*: *finite S*
shows *total-degree-env ctxt* (*prod P S*) ≤ *sum* (λ*i*. *total-degree-env ctxt* (*P i*)) *S*
⟨*proof*⟩

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*)
≤ (∑ *t*∈*keys m*. *degree-monom m t* ∗ *total-degree-env ctxt* (*f t*))
⟨*proof*⟩

lemma *total-degree-env-poly-subst-list*:
fixes *p* :: ′*a*::*comm-ring-1 mpoly*
shows *total-degree-env ctxt* (*poly-subst-list fs p*)
≤ *total-degree-env* (λ*m*. *total-degree-env ctxt* (*fs* $!_0$ *m*)) *p*

⟨*proof*⟩

lemma *total-degree-poly-subst-list-env*:
fixes p :: *'a::comm-ring-1 mpoly*
shows *total-degree* (*poly-subst-list fs p*)
$\leq$ *total-degree-env* (λm. *total-degree* (*fs* $!_0$ *m*)) *p*
⟨*proof*⟩

lemma *total-degree-env-Var-list-bound*: *total-degree-env* (λ-. *1*) ((*map Var ls*) $!_0$ *i*) $\leq$ *1*
⟨*proof*⟩

lemma *total-degree-env-Var-list*:
total-degree-env (λ-. *1*) ((*map Var ls*) $!_0$ *i*) = (*if i* < *length ls then 1 else 0*)
⟨*proof*⟩

lemma *total-degree-map-Var*:
total-degree ((*map Var ls*) $!_0$ *j* :: *'a::comm-semiring-1 mpoly*) $\leq$ *1*
⟨*proof*⟩

lemma *total-degree-map-Var-int*:
total-degree ((*map Var ls*) $!_0$ *j* :: *int mpoly*) $\leq$ *Suc 0*
⟨*proof*⟩

lemma *total-degree-env-mono3-map-Var*:
($\bigwedge i$. *env i* $\leq$ *1*) $\Longrightarrow$ *total-degree-env env* ((*map Var ls*) $!_0$ *j*) $\leq$ *1*
⟨*proof*⟩

lemma *total-degree-env-reduce*: *i* < *length ls*
$\Longrightarrow$ *total-degree-env env* ((*ls* @ *xs*) $!_0$ *i*) = *total-degree-env env* (*ls* $!_0$ *i*)
⟨*proof*⟩

lemma *total-degree-env-mono*:
fixes *P* :: *int mpoly*
assumes $\forall$ *i*$\leq$*max-vars P*. *env1 i* $\leq$ *env2 i*
shows *total-degree-env env1 P* $\leq$ *total-degree-env env2 P*
⟨*proof*⟩

lemma *total-degree-env-mono2*:
fixes *P* :: *int mpoly*
shows *total-degree P* $\leq$ *rhs1* $\Longrightarrow$ ($\bigwedge i$. *i* $\leq$ *max-vars P* $\Longrightarrow$ *env i* $\leq$ *1*) $\Longrightarrow$ *rhs1* = *rhs2*
$\Longrightarrow$ *total-degree-env env P* $\leq$ *rhs2*

⟨*proof*⟩

lemma *total-degree-env-mono3-bounded*:
fixes *ls* :: *int mpoly list*
shows $j \leq bound \Longrightarrow (\bigwedge i.\ i \leq bound \Longrightarrow env\ i \leq 1) \Longrightarrow max\text{-}vars\ (ls\ !_0\ j) \leq bound$
$\Longrightarrow total\text{-}degree\ (ls\ !_0\ j) \leq Suc\ 0 \Longrightarrow total\text{-}degree\text{-}env\ env\ (ls\ !_0\ j) \leq Suc\ 0$
⟨*proof*⟩

lemma *total-degree-env-mono3*:
$(\bigwedge i.\ env\ i \leq 1) \Longrightarrow total\text{-}degree\ (ls\ !_0\ j) \leq 1$
$\Longrightarrow total\text{-}degree\text{-}env\ env\ (ls\ !_0\ j) \leq 1$
⟨*proof*⟩

lemma *total-degree-env-mono3'*:
$(\bigwedge i.\ env\ i \leq Suc\ 0) \Longrightarrow total\text{-}degree\ (ls\ !_0\ j) \leq Suc\ 0$
$\Longrightarrow total\text{-}degree\text{-}env\ env\ (ls\ !_0\ j) \leq Suc\ 0$
⟨*proof*⟩

lemma *total-degree-env-mono4*:
$(\bigwedge i.\ env\ i \leq 1) \Longrightarrow total\text{-}degree\text{-}env\ (\lambda\text{-}.\ 1)\ (ls\ !_0\ j) \leq 1$
$\Longrightarrow total\text{-}degree\text{-}env\ env\ (ls\ !_0\ j) \leq 1$
⟨*proof*⟩

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**
$fresh\text{-}var\ P = (if\ vars\ P = \{\}\ then\ 1\ else\ max\text{-}vars\ P + 1)$

definition *suitable-for-coding* :: *int mpoly* ⇒ *int mpoly* **where**
$suitable\text{-}for\text{-}coding\ P \equiv P^2 + (Var\ (fresh\text{-}var\ P) - 1)^2$

lemma *suitable-for-coding-degree-vars*:
shows $degree\ (suitable\text{-}for\text{-}coding\ P)\ (fresh\text{-}var\ P) > 0$
$vars\ (suitable\text{-}for\text{-}coding\ P) = insert\ (fresh\text{-}var\ P)\ (vars\ P)$
⟨*proof*⟩

lemma *suitable-for-coding-coeff0*:
fixes *P*
defines $n \equiv max\text{-}vars\ (suitable\text{-}for\text{-}coding\ P)$
defines $m0 \equiv Abs\text{-}poly\text{-}mapping\ ((!_0)\ (replicate\ (n+1)\ 0))$
shows $coeff\ (suitable\text{-}for\text{-}coding\ P)\ m0 > 0$
⟨*proof*⟩

lemma *suitable-for-coding-max-vars*:
  assumes *vars P* ≠ {}
  shows *max-vars* (*suitable-for-coding P*) = *max-vars P* + *1*
⟨*proof*⟩

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*⟩

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*⟩

lemma *suitable-for-coding-total-degree*:
  shows *total-degree* (*suitable-for-coding P*) > *0*
  ⟨*proof*⟩

lemma *suitable-for-coding-total-degree-bound*:
  assumes *total-degree P* > *0*
  shows *total-degree* (*suitable-for-coding P*) ≤ *2* ∗ *total-degree P*
  ⟨*proof*⟩

end
theory *Poly-Degree*
  imports *More-More-MPoly-Type Total-Degree-Env Poly-Extract*
  keywords *poly-degree* :: *thy-defn* **and** |
begin

⟨*ML*⟩

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 $b \geq 2$
shows $(\forall k{\leq}q.\ f\ k < b) \Longrightarrow (\sum k = 0..q.\ f\ k * b\ \hat{}\ k) < b\hat{}(Suc\ q)$
⟨*proof*⟩

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** $\delta \equiv$ *total-degree P*
definition ν :: *nat* **where** $\nu \equiv$ *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** $n\ i \equiv (\delta{+}1)\hat{}i$
definition δ*-tuples* :: (*nat list*) *set* **where**
δ*-tuples* ≡ $\{i.\ length\ i = \nu + 1 \wedge sum\text{-}list\ i \leq \delta\}$

lemma δ*-tuples-finite*[*simp*]: *finite* δ*-tuples*
⟨*proof*⟩

abbreviation σ **where** $\sigma \equiv$ *count-bits*
abbreviation τ **where** $\tau \equiv$ *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** $\mathcal{L} \equiv nat\ (\sum i{\in}\delta\text{-}tuples.\ abs\ (P\text{-}coeff\ i))$

We have to use Inf instead of Min to define r because the set is infinite

definition r :: *nat* **where** $r \equiv Inf\ \{y.\ 4\hat{}y > 2 * fact\ \delta * \mathcal{L} * (\nu + 3)\hat{}\delta\}$
definition β :: *nat* **where** $\beta \equiv 4\hat{}r$
definition γ :: *nat* **where** $\gamma \equiv \beta\hat{}(n\ \nu)$

definition α :: *nat* **where** $\alpha \equiv \delta * n\ \nu + 1$

lemma *α-gt-0*: $\alpha > 0$ ⟨*proof*⟩
lemma *γ-gt-0*: $\gamma > 0$ ⟨*proof*⟩

definition *b* :: *int* **where**
$b \equiv 1 + 3*(2*a + 1) * f$
definition $\mathcal{B}$ :: *int* **where**
$\mathcal{B} \equiv (\textit{of-nat}\ \beta) * b \hat{}\delta$
definition *N0* :: *int* **where**
$N0 \equiv \mathcal{B}\hat{}((\delta+1)\hat{}\nu + 1)$
definition *N1* :: *int* **where**
$N1 \equiv 4 * \mathcal{B}\hat{}((2*\delta+1) * (\delta+1)\hat{}\nu + 1)$
definition *N* :: *int* **where**
$N \equiv N0 * N1$
definition *c* :: *int* ⇒ *int* **where**
$c\ g \equiv 1 + a*\mathcal{B} + g$

poly-extract *b*
poly-degree *b-poly*

poly-extract $\mathcal{B}$
poly-degree $\mathcal{B}$*-poly*

poly-extract *N0*
poly-extract *N1*
poly-extract *N*
poly-extract *c*
poly-degree *c-poly*

The M polynomial.

definition *m* :: *nat* ⇒ *int* **where**
$m\ j \equiv (\textit{if}\ j \in n\ \text{`}\ \{1..\nu\}\ \textit{then}\ \mathcal{B} - b\ \textit{else}\ \mathcal{B} - 1)$
definition *M* :: *int* **where**
$M \equiv (\sum j{=}0..n\ \nu.\ m\ j * \mathcal{B}\hat{}j)$

definition *m-poly* :: *nat* ⇒ *int mpoly* **where**
$\textit{m-poly}\ j = (\textit{if}\ j \in n\ \text{`}\ \{1..\nu\}\ \textit{then}\ \mathcal{B}\textit{-poly} - \textit{b-poly}\ \textit{else}\ \mathcal{B}\textit{-poly} - 1)$

definition *M-poly* :: *int mpoly* **where**
$\textit{M-poly} \equiv (\sum\ j{=}0..n\ \nu.\ \textit{m-poly}\ j * \mathcal{B}\textit{-poly}\ \hat{}\ j)$

lemma *m-correct*: *insertion fn* (*m-poly j*) = *coding-variables.m P* (*fn 0*) (*fn* (*Suc 0*)) *j*
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *M-correct*:
insertion fn (*coding-variables.M-poly P*) = *coding-variables.M P* (*fn 0*) (*fn 1*)
⟨*proof*⟩

lemma *m-poly-degree-correct*:
shows $\delta > 0 \Longrightarrow$ *total-degree* (*m-poly j*) $\leq 2{*}\delta$
⟨*proof*⟩

lemma *M-poly-degree-correct*:
assumes *asm*: $\delta > 0$
shows *total-degree M-poly* $\leq (1+(\delta+1)\hat{}\nu) * 2{*}\delta$
⟨*proof*⟩

definition *D-exponent* :: *nat list* ⇒ *nat* **where**
D-exponent i ≡ *n* $(\nu+1) - (\sum s\leq\nu.\ i!s * n\ s)$
definition *D-precoeff* :: *nat list* ⇒ *int* **where**
D-precoeff i ≡ *int* (*mfact i* ∗ *fact* (δ − *sum-list i*))
definition *D* :: *int* **where**
D ≡ $\sum$ *i*∈δ*-tuples. of-int* (*D-precoeff i* ∗ *P-coeff i*) ∗ $\mathcal{B}$^(*D-exponent i*)

definition *D-poly* :: *int mpoly* **where**
D-poly ≡ $\sum$ *i*∈δ*-tuples. Const* (*D-precoeff i* ∗ *P-coeff i*) ∗ $\mathcal{B}$*-poly* ^ (*D-exponent i*)

lemma *D-correct*:
insertion fn (*coding-variables.D-poly P*) = *coding-variables.D P* (*fn 0*) (*fn 1*)
⟨*proof*⟩

lemma *D-poly-degree-env-correct*:
shows *total-degree-env fv D-poly* ≤ *n* $(\nu+1)$ ∗ *total-degree-env fv* $\mathcal{B}$*-poly*
⟨*proof*⟩

lemma *D-poly-degree-correct*: *total-degree* (*coding-variables.D-poly P*) $\leq (\delta+1)\hat{}(\nu+1) * (2{*}\delta)$
⟨*proof*⟩

definition *K* :: *int* ⇒ *int* **where**
K g ≡ (*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 ≡ *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*)
⟨*proof*⟩

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*⟩

definition *T* **where** *T* ≡ *M* + ($\mathcal{B}$−*2*) ∗ $\mathcal{B}$ ^((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*⟩

corollary *x-series-bound*:
assumes *0* < δ
assumes *x* ∈ *δ-tuples*
shows ($\sum$ *s*≤ν. *x* ! *s* ∗ *Suc* δ ^ *s*) ≤ (δ+*1*) ^(ν+*1*)
⟨*proof*⟩

lemma *D-exponent-injective*:
assumes *0* < δ
shows *inj-on D-exponent δ-tuples*
⟨*proof*⟩

corollary *D-exponent-injective′*: $0 < \delta \Longrightarrow$ *inj-on D-exponent* (δ*-tuples* $-$ $\{x\}$)
⟨*proof*⟩

lemma *D-precoeff-bound*:
assumes $0 <$ *sum-list i* **and** *sum-list i* $\leq \delta$
shows |*D-precoeff i*| $\leq$ *fact* δ
⟨*proof*⟩

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*⟩

lemma $\mathcal{L}$*-pos*:
assumes $\delta > 0$
shows $\mathcal{L} > 0$
⟨*proof*⟩

lemma $\mathcal{L}$*-ge-P-coeff*: *i*$\in\delta$*-tuples* $\Longrightarrow$ *abs* (*P-coeff i*) $\leq$ *int* $\mathcal{L}$ **for** *i*
⟨*proof*⟩

lemma $\mathcal{L}$*-ge-max-coeff*:
assumes $\delta > 0$
shows *max-coeff P* $\leq$ *int* $\mathcal{L}$
⟨*proof*⟩

lemma β*-lower-bound*: $\beta > 2 *$ *fact* $\delta * \mathcal{L} * (\nu+3)$ ^δ
⟨*proof*⟩

corollary *r-pos*:
assumes $\delta > 0$
shows $r > 0$
⟨*proof*⟩

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*⟩

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*: $a \geq 2 \wedge k \geq 0$
  shows ∃ (*s*::*nat*). $a\hat{}s \geq b \wedge s \geq k$
⟨*proof*⟩

lemma *little-fermat*:
  fixes *a m* :: *int*
  assumes *gcd*: *coprime a m* **and** *posit*: $a \geq 1 \wedge m \geq 2$
  shows ∃ *k l*. $a\hat{}k-1 = m*l \wedge k \geq 1$
⟨*proof*⟩

lemma *lemma-2-2*:
  fixes *a Z* :: *int*
  assumes *posit*:$Z > 0 \wedge a \geq 0$
  shows ∃ *f r*. $f \geq Z \wedge 1 + 3*(2*a+1)*f = 4\hat{}r$
⟨*proof*⟩

lemma *lemma-2-2-useful*:
  fixes *a min-b* :: *int*
  assumes *min-b* $\geq 0 \wedge a \geq 0$
  defines $b \equiv \lambda f.\ 1 + 3 * (2*a + 1) * f$
  shows ∃*f*. $f > 0 \wedge$ *is-square* (*b f*) ∧ *is-power2* (*b f*) ∧ *b f* > *min-b*
⟨*proof*⟩

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* (ν+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*⟩

lemma (**in** *coding-variables*) *lower-bounds*:
  fixes g :: *int*
  assumes $a \geq 0$
  assumes $f > 0$
  assumes $g \geq 0$
  assumes $\delta > 0$
  assumes *p0*: *P-coeff* (*replicate* (ν+*1*) *0*) > *0*
  shows $X\ g \geq 3 * b$ **and** $Y \geq 2\hat{}8$ **and** $Y \geq b$
⟨*proof*⟩

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** $\forall\, i \in \{0..q\}.\ f\ i < b$
  shows ($\sum i$=*0*..*q*. $f\ i * b\hat{}i$) < $b\hat{}$(*Suc q*)
⟨*proof*⟩

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* $\mathcal{B}$
  ⟨*proof*⟩
lemma *N0-power2*: *is-power2 N0*
  ⟨*proof*⟩
lemma *N1-power2*: *is-power2 N1*
  ⟨*proof*⟩
lemma *N-power2*: *is-power2 N*
  ⟨*proof*⟩
lemma *Ng-power2*: *is-power2 N*
  ⟨*proof*⟩

lemma $\mathcal{B}$*-ge-2*: $\mathcal{B} \geq 2$
  ⟨*proof*⟩

lemma $\mathcal{B}$*-even*: *2 dvd* $\mathcal{B}$

⟨*proof*⟩

lemma *b-le-ℬ*: $b \leq \mathcal{B}$
⟨*proof*⟩

lemma *M-bound*: $0 \leq M \wedge M < N0$
⟨*proof*⟩

context
fixes g::*int*
assumes *g-lower-bound*: $0 \leq g$
and *g-upper-bound*: $g < 2 * b * \mathcal{B}\hat{}(n\ \nu)$
begin

lemma *g-lt-N0*: $g < N0$
⟨*proof*⟩

lemma *c-bound*: $abs\ (c\ g) < 3 * b * \mathcal{B}\hat{}(n\ \nu)$
⟨*proof*⟩

lemma *D-bound*: $abs\ D \leq fact\ \delta * \mathcal{L} * \mathcal{B}\hat{}(n\ (\nu+1))$
⟨*proof*⟩

lemma *c-δ-D-bound*: $2 * abs\ ((c\ g)\hat{}\delta * D) \leq \mathcal{B}\hat{}((2*\delta+1)*n\ \nu + 1)$
⟨*proof*⟩

lemma *K-bound*: $0 \leq K\ g \wedge 2 * K\ g < 3 * \mathcal{B}\hat{}((2*\delta+1)*n\ \nu + 1)$
⟨*proof*⟩

technical condition 2.7, first part

lemma *T-bound*: $0 \leq T \wedge T < N$
⟨*proof*⟩

technical condition 2.7, second part

lemma *S-bound*: $0 \leq S\ g \wedge S\ g < N$
⟨*proof*⟩

Technical condition 2.8

lemma *tau-S-T-decomp*: $\tau\ (nat\ (S\ g))\ (nat\ (T)) =$
$\tau\ (nat\ g)\ (nat\ (M)) + \tau\ (nat\ (2 * K\ g))\ (nat\ ((\mathcal{B} - 2) * \mathcal{B}\hat{}(n\ (\nu+1))))$
⟨*proof*⟩

end

Helper lemmas for masking

lemma *n-masking-lemma*[*simp*]:
assumes *masking-lemma* $\delta\ \nu\ (nat\ \mathcal{B})\ (nat\ b)\ C$
shows *masking-lemma.n* $\delta = n$

⟨proof⟩

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⟩

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⟩

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*: $g = (\sum i{=}1..\nu.\ z\ i * \mathcal{B}\hat{}(n\ i))$
      and *z-bound*: $\forall i.\ 0 \le z\ i \land z\ i < b$
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*
  ⟨proof⟩

lemma *z-list-nth-tail*: $1 \le i \Longrightarrow i \le \nu \Longrightarrow$ *z-list*!*i* = *nat* (*z i*)
⟨proof⟩

lemma *length-z-list*[*simp*]: *length z-list* = ν+*1*
  ⟨proof⟩

lemma δ*-lemma-1-8*[*simp*]: *Lemma-1-8-Defs.*δ *P* = δ
  ⟨proof⟩

lemma ν*-lemma-1-8*[*simp*]: *Lemma-1-8-Defs.*ν *P* = ν
  ⟨proof⟩

lemma *n-lemma-1-8*[*simp*]: *Lemma-1-8-Defs.n P* = *n*
  ⟨proof⟩

lemma *insertion-z-assign*[*simp*]:
  insertion (*Lemma-1-8-Defs.z-assign z-list*) *P* = *insertion* (*z*(*0* := *a*)) *P*
⟨proof⟩

lemma *S-lemma-1-8*[*simp*]:
int (*Lemma-1-8-Defs.S P* (*nat* $\mathcal{B}$)) = ($\sum$ *i*=*0*..(*2*∗δ+*1*)∗*n* ν. *int* (β *div 2*) ∗ *b*^δ ∗ $\mathcal{B}$^*i*)
⟨*proof*⟩

lemma *c-lemma-1-8*[*simp*]:
insertion (*Lemma-1-8-Defs.*$\mathcal{B}$*-assign* (*nat* $\mathcal{B}$)) (*Lemma-1-8-Defs.c P z-list*) = *a* ∗ $\mathcal{B}$ + *g*
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *D-lemma-1-8*[*simp*]:
insertion (*Lemma-1-8-Defs.*$\mathcal{B}$*-assign* (*nat* $\mathcal{B}$)) (*Lemma-1-8-Defs.D P*) = *D*
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *R-lemma-1-8*[*simp*]:
insertion (*Lemma-1-8-Defs.*$\mathcal{B}$*-assign* (*nat* $\mathcal{B}$)) (*Lemma-1-8-Defs.R P z-list*) = (*c g*)^δ ∗ *D*
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *K-lemma-1-8*[*simp*]:
Lemma-1-8-Defs.K P (*nat* $\mathcal{B}$) *z-list* = *K g*
⟨*proof*⟩

lemma *lemma-1-8-helper*:
shows *insertion* (*z*(*0* := *a*)) *P* = *0* ⟷ τ (*nat* (*2* ∗ *K g*)) (*nat* (($\mathcal{B}$ − *2*) ∗ $\mathcal{B}$^(*n* (ν+*1*)))) = *0*
and *K g* > $\mathcal{B}$^((*2*∗δ+*1*) ∗ *n* ν)
and *K g* < $\mathcal{B}$^((*2*∗δ+*1*) ∗ *n* ν + *1*)
⟨*proof*⟩

end

lemma *aux-sum-bound-reindex-n*:
0 ≤ (*x*::*int*) ⟹ ($\sum$ *i*=*0*..*q*. *x*^(*n i*)) ≤ ($\sum$ *i*=*0*..*n q*. *x*^*i*)
⟨*proof*⟩

lemma *coding-theorem-direct*:
statement1-strong y ⟹ (∃ *g*. *statement2-strong g*)
⟨*proof*⟩

lemma *coding-theorem-reverse*:
statement2-weak g ⟹ (∃ *y*. *statement1-weak y*)
⟨*proof*⟩

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 $\exists z.\ (z\ 0 = a) \wedge (\forall i.\ 0 \leq z\ i \wedge z\ i < b) \wedge insertion\ z\ P = 0$
⟨*proof*⟩

end

end
theory *Lucas-Sequences*
imports *Main HOL.Parity*
begin

5 Lucas Sequences

fun ψ :: *int* ⇒ *nat* ⇒ *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* ⇒ *nat* ⇒ *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*]:
$P\ 0 \Longrightarrow P\ 1 \Longrightarrow (\bigwedge n.\ P\ (n + 1) \Longrightarrow P\ n \Longrightarrow P\ (n + 2)) \Longrightarrow P\ (n{::}nat)$
⟨*proof*⟩

theorem *ψ-induct-strict* [*consumes 0*, *case-names 0 1 2 sucsuc*]:
$P\ 0 \Longrightarrow P\ 1 \Longrightarrow P\ 2 \Longrightarrow (\bigwedge n.\ n{>}0 \Longrightarrow P\ (n + 1) \Longrightarrow P\ n \Longrightarrow P\ (n + 2))$
$\Longrightarrow P\ (n{::}nat)$
⟨*proof*⟩

lemma *lem0*: $n{\geq}2 \Longrightarrow \exists m.\ n = Suc\ (Suc\ m)$
⟨*proof*⟩

lemma *ψ-reverse*:
assumes $n{\geq}1$
shows $\psi\ A\ (n{-}1) = A * (\psi\ A\ n) - (\psi\ A\ (n{+}1))$
⟨*proof*⟩

Strict monotonicity

lemma *lucas-strict-monotonicity*: $A{>}1 \Longrightarrow \psi\ A\ (Suc\ n) > \psi\ A\ n \wedge \psi\ A\ (Suc\ n)$
> 0
⟨*proof*⟩

lemma *lucas-monotone1*:
fixes *A*

assumes $A>1$
shows $n\geq 2 \longrightarrow \psi\ A\ n \geq A$
⟨*proof*⟩

lemma *lucas-monotone2*:
fixes $A\ n\ m$
assumes $A>1$
shows $\psi\ A\ n \leq \psi\ A\ (n+m)$
⟨*proof*⟩

lemma *lucas-monotone3*:
fixes $A\ n$
assumes $A > 1$
shows $\psi\ A\ n \geq int\ n$
⟨*proof*⟩

lemma *lucas-monotone4*:
fixes $A\ n\ m$
assumes $A>1$ **and** $n \leq m$
shows $\psi\ A\ n \leq \psi\ A\ m$
⟨*proof*⟩

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*⟩

lemma *lucas-exp-growth-le*:
fixes A::*int* **and** n::*nat*
assumes $A>1$
shows $\psi\ A\ (Suc\ (Suc\ n)) \leq A\,\hat{}(n+1)$
⟨*proof*⟩

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*⟩

lemma *lucas-symmetry-A*:
fixes A::*int* **and** n::*nat*
assumes $A \geq 2$
shows $(\psi\ A\ n) = (if\ (odd\ n)\ then\ \psi\ (-A)\ n\ else - \psi\ (-A)\ n)$
⟨*proof*⟩

lemma *lucas-symmetry-A2*: $-\psi\ A\ n = (-1::int)\hat{}n * \psi\ (-A)\ n$
⟨*proof*⟩

lemma *lucas-symmetry-A-abs*: **assumes** *abs* $A > 1$ **shows** *abs* $(\psi\ A\ n) = \psi$ (*abs* $A)\ n$
⟨*proof*⟩

lemma *lucas-A-eq-2*:
fixes $n::nat$
shows $(\psi\ 2\ n) = (int\ n)$
⟨*proof*⟩

lemma *lucas-periodic-modN*:
fixes $N::int$
assumes $N > 0$
shows $\exists\ T{\geq}1.\ \forall\ n.\ (\psi\ A\ (T + n))\ mod\ N = (\psi\ A\ n)\ mod\ N$
⟨*proof*⟩

lemma *lucas-modN*:
fixes $N::int$
assumes $N{>}0$
shows $\forall\ n.\ \exists\ k{\geq}n.\ \psi\ A\ k\ mod\ N = 0$
⟨*proof*⟩

lemma *lucas-parity*:
fixes $A::int$ **and** $B::nat$
assumes *even* A
shows *even* $(\psi\ A\ B) =$ *even* B
⟨*proof*⟩

corollary *lucas-parity2*:
fixes $A::int$ **and** $B::nat$
assumes *even* A
shows *even* $(\psi\ A\ B - int\ B)$
⟨*proof*⟩

lemma *lucas-monotone-A*:
assumes $1{<}A\ A{\leq}A'$
shows $\psi\ A\ n \leq \psi\ A'\ n$
⟨*proof*⟩

lemma *lucas-congruence*:

fixes *A::int* **and** *B::int* **and** *n::int*
assumes *n=n* ∧ *A mod n* = *B mod n*
shows (*ψ A m*) *mod n* = (*ψ B m*) *mod n*
⟨*proof*⟩

corollary *lucas-congruence2*:
fixes *α::int* **and** *m::nat*
shows *ψ α m mod* (*α* − *2*) = *int m mod* (*α* − *2*)
⟨*proof*⟩

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*]: ⌊*real-of-int x*⌋ = *x*
⟨*proof*⟩

lemma *floor-of-real-of-int-sub2*[*real-of-int*]: ⌊*x* − *real-of-int y*⌋ = ⌊*x*⌋ − *y*
⟨*proof*⟩

lemma *floor-of-real-of-int-mult*[*real-of-int*]: ⌊*real-of-int x* ∗ *real-of-int y*⌋ = *x* ∗ *y*
⟨*proof*⟩

lemma *real-of-int-inequality*: *X* ≤ *Y* ⟷ *real-of-int X* ≤ *real-of-int Y* ⟨*proof*⟩
lemma *real-of-int-strict-inequality*: *X* < *Y* ⟷ *real-of-int X* < *real-of-int Y*
⟨*proof*⟩

lemma *evenX2k*:
fixes *X::int*
assumes *evenX:even X*
shows ∃ *k*. *X* = *2*∗*k*
⟨*proof*⟩

lemma *distrib-add-diff*:
fixes *a b c d::real*
shows (*a*+*b*)∗(*c*−*d*) = *a*∗*c* − *a*∗*d* + *b*∗*c* − *b*∗*d*
⟨*proof*⟩

lemma *floor-even*:
fixes *X::int*
assumes *Xeven*: *even X*
shows *real-of-int* ⌊(*real-of-int X*)/*2*⌋ = (*real-of-int X*)/*2*

⟨proof⟩

lemma *even-to-mod2*:
fixes X Y::*int*
assumes *even* X = *even* Y
shows X *mod* 2 = Y *mod* 2
⟨proof⟩

lemma *oddA-to-mod*:
fixes X Y A::*int*
assumes *odd* A
shows $A\hat{}2$ *mod* 4 = 1
⟨proof⟩

lemma *sol-non-zero*:
fixes X Y A::*int*
assumes *sol*: $X\hat{}2 - (A\hat{}2-4)*Y\hat{}2 = 4$ **and** *Alarge*: $A\hat{}2>4$
shows $X+sqrt(A\hat{}2-4)*Y \neq 0$
⟨proof⟩

lemma *conj-inversion*:
fixes X::*int* **and** Y::*int* **and** A::*int*
assumes *A4*:$A\hat{}2 > 4$ **and** *sol*:$X\hat{}2 - (A\hat{}2 - 4)*Y\hat{}2 = 4$
shows $1/2*(X-sqrt(A\hat{}2-4)*Y) = 2*inverse(X+sqrt(A\hat{}2-4)*Y)$
⟨proof⟩

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⟩

lemma *group-structure-evenXi*:
fixes X Y A::*int*
assumes *sol*:$(X\hat{}2 - (A\hat{}2-4)*Y\hat{}2 = 4)$ **and** *even* A
shows *even* X
⟨proof⟩

lemma *XimodYi*:
fixes X Y A::*int*
assumes *A4*:$A\hat{}2 > 4$ **and** *sol*:$(X\hat{}2 - (A\hat{}2-4)*Y\hat{}2 = 4)$ **and** *odd* A
shows X *mod* 2 = Y *mod* 2
⟨proof⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

5.2.4 Finite generation of solutions

lemma *finite-generation-nat*:
  fixes *X Y A*::*int* **and** *n*::*nat*
  assumes *sol*: *X^2 − (A^2−4)∗Y^2 = 4* **and** *Alarge*: *A^2 > 4*
 shows ∃ *X3*. ∃ *Y3*. *2∗(((X + sqrt(A^2−4)∗Y)/2)^n) = X3 + sqrt(A^2−4)∗Y3*
∧
    X3^2 − (A^2−4)∗Y3^2 = 4 (**is** *?P n*)
⟨*proof*⟩

lemma *finite-generation*:

fixes X Y A::*int* **and** n::*nat*
assumes *sol*: $X\hat{}2 - (A\hat{}2-4)*Y\hat{}2 = 4$ **and** $A\hat{}2>4$
shows $\exists X1. \exists Y1. 2*(inverse((X + sqrt(A\hat{}2-4)*Y)/2)\hat{}n) = X1 + sqrt(A\hat{}2-4)*Y1$ $\wedge$
$X1\hat{}2 - (A\hat{}2-4)*Y1\hat{}2 = 4$
⟨*proof*⟩

lemma *real-arch-power*:
fixes x::*real* **and** y::*real*
assumes *x1*: $x>1$ **and** *y1*: $y \geq 1$
shows $\exists n. x\hat{}n \leq y \wedge y < x\hat{}(n+1)$
⟨*proof*⟩

lemma *finite-gen-all-sol*:
fixes X::*int* **and** Y::*int* **and** A::*int*
defines $rho \equiv |A| + sqrt(A\hat{}2-4)$
and $Z \equiv X + sqrt(A\hat{}2-4)*Y$ **and** $D \equiv A\hat{}2-4$
assumes *Alarge*: $A\hat{}2>4$ **and** *XYsol*: $X\hat{}2-(A\hat{}2-4)*Y\hat{}2 = 4$
shows $\exists n. Z \in \{2*(rho/2)\hat{}n, -2*(rho/2)\hat{}n, 2*inverse(rho/2)\hat{}n, -2*inverse(rho/2)\hat{}n\}$ $\wedge$
$Y \in \{1/sqrt(D)*((rho/2)\hat{}n - inverse(rho/2)\hat{}n), -1/sqrt(D)*((rho/2)\hat{}n - inverse(rho/2)\hat{}n)\}$
⟨*proof*⟩

5.2.5 Link between Pell equation and Lucas sequences

lemma *link-to-lucas*:
fixes A::*int* **and** n::*nat*
assumes *A4*: $A\hat{}2 > 4$
shows $inverse(sqrt(A\hat{}2-4))*((1/2*((real\text{-}of\text{-}int\ A)+sqrt(A\hat{}2-4)))\hat{}n - (2*inverse((real\text{-}of\text{-}int\ A)+sqrt(A\hat{}2-4)))\hat{}(n)) = \psi\ A\ n$
⟨*proof*⟩

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*⟩

lemma *lucas-pell-sublemmaAmin2*:
fixes Y::*int*
shows $\exists m. Y = \psi\ (-2)\ m \vee Y = - \psi\ (-2)\ m$
⟨*proof*⟩

lemma *lucas-pell-sublemmaA0*:
fixes Y::*int*
assumes *assm*: $\exists k. (-4)*Y\hat{}2 + 4 = k\hat{}2$
shows $\exists m. Y = \psi\ 0\ m \vee Y = - \psi\ 0\ m$
⟨*proof*⟩

lemma *lucas-pell-sublemmaA1*:
 fixes $Y::int$
 assumes *assm*: $\exists k.\ (1\hat{}2-4)*Y\hat{}2 + 4 = k\hat{}2$
 shows $\exists m.\ Y = \psi\ 1\ m \lor Y = -\ \psi\ 1\ m$
⟨*proof*⟩

lemma *lucas-pell-sublemmaAmin1*:
 fixes $Y::int$
 assumes *assm*: $\exists k.\ ((-1)\hat{}2-4)*Y\hat{}2 + 4 = k\hat{}2$
 shows $\exists m.\ Y = \psi\ (-1)\ m \lor Y = -\ \psi\ (-1)\ m$
⟨*proof*⟩

5.2.7 The main equivalence

lemma *lucas-pell-part1*:
 fixes $A\ Y::int$
 shows $(\exists k.\ (A\hat{}2-4)*Y\hat{}2 + 4 = k\hat{}2) \Longrightarrow (\exists m.\ Y = \psi\ A\ m \lor Y = -\ \psi\ A\ m)$
⟨*proof*⟩

lemma *lucas-pell-part3*:
 fixes $A::int$ **and** $m::nat$
 shows $(A\hat{}2-4)*(\psi\ A\ m)\hat{}2+4 = (\chi\ A\ m)\hat{}2$
⟨*proof*⟩

lemma *lucas-pell-part2*:
 fixes $A\ X::int$
 shows $(\exists m.\ X = \psi\ A\ m \lor X = -\ \psi\ A\ m) \Longrightarrow (\exists k.\ (A\hat{}2-4)*X\hat{}2 + 4 = k\hat{}2)$
⟨*proof*⟩

lemma *lucas-pell-nat*:
 fixes $A\ Y :: int$
 shows $(\exists k.\ (A\hat{}2-4)*Y\hat{}2 + 4 = k\hat{}2) = (\exists m.\ Y = \psi\ A\ m \lor Y = -\ \psi\ A\ m)$
  and $(A\hat{}2-4)*(\psi\ A\ m)\hat{}2 + 4 = (\chi\ A\ m)\hat{}2$
 ⟨*proof*⟩

corollary *lucas-pell-corollary*:
 fixes $A::int$ **and** $X::int$
 shows *is-square* $((A\hat{}2-1)*X\hat{}2+1) = (\exists m.\ X = \psi\ (2*A)\ m \lor X = -\ \psi\ (2*A)\ m)$
⟨*proof*⟩

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 $(3 * 2\hat{}B * \psi\ A\ B)\ mod\ (2*A-5) = (2 * (2\hat{}(2*B) - 1))\ mod\ (2*A - 5)$
⟨*proof*⟩

A few lemmas helping variable changes in sums

lemma *translation-var-0-to-1*:
fixes f::*nat* ⇒ *int* **and** n::*nat*
shows $(\sum i=0..n.\ f\ (i+1)) = (\sum i=1..n+1.\ (f\ i))$
⟨*proof*⟩

lemma *chang-var2*:
fixes f::*nat* ⇒ *nat* ⇒ *int* **and** n::*nat*
shows $(\sum i=0..n.\ f\ (i+1)\ (n-i)) = (\sum i=1..n+1.\ f\ i\ (n+1-i))$
⟨*proof*⟩

lemma *chang-var3*:
fixes f::*nat* ⇒ *nat* ⇒ *int* **and** n::*nat*
assumes $n \geq 1$
shows $(\sum i=0..n-1.\ f\ (i+1)\ (n-i)) = (\sum i=1..n.\ f\ i\ (n+1-i))$
⟨*proof*⟩

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* $U\ V\ a\ b = U\hat{}(2*a)*V\hat{}(2*b)$

lemma *lucas-exponential-diophantine*:
fixes A::*int* **and** B::*nat* **and** U::*int* **and** V::*int*
assumes $B>0$
shows $(U*V)\hat{}(B-1) * \psi\ A\ B\ mod\ (U\hat{}2 - A*U*V + V\hat{}2)$
$= (\sum r=0..(B-1).\ (U\hat{}(2*r))*(V\hat{}(2*(B-1-r))))\ mod\ (U\hat{}2 - A*U*V + V\hat{}2)$
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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 ≥ 2*int X* **and** *V ≥ 1* **and** *X ≥ 1*
shows *abs (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*⟩

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 ≥ 2*int X* **and** *V ≥ 1* **and** *X ≥ 1*
shows *−2*int X*(V+1)^(2*X)*ψ (U^2*V) (X+1)*
*≤ abs U*V*(V^X * ψ (U*(V+1)) (2*X +1) − (V+1)^(2*X) * ψ (U^2*V) (X +1))*
*∧ abs U*V*(V^X * ψ (U*(V+1)) (2*X +1) − (V+1)^(2*X) * ψ (U^2*V) (X +1))*

$\leq$ *2∗int X∗(V+1)^(2∗X)∗ψ (U^2∗V) (X+1)*
⟨*proof*⟩

lemma *lemma-4-4-cor-abs*:
fixes *U::int* **and** *V::int* **and** *X::nat*
assumes *abs U ≥ 2∗int X* **and** *V ≥ 1* **and** *X ≥ 1*
shows *abs (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*⟩

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 ≥ 2∗int X* **and** *V ≥ 1* **and** *X ≥ 1*
defines *ϱ ≡ (real-of-int (V+1)^(2∗X))/(real-of-int V^X)*
shows *abs (ψ (U∗(V+1)) (2∗X +1)/ ψ (U^2∗V) (X +1) − ϱ) ≤ 2∗int X∗ϱ / (U∗V)*
⟨*proof*⟩

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 ≥ 2∗int X* **and** *V ≥ 1* **and** *X ≥ 1*
assumes *ϱ ≡ (real-of-int (V+1)^(2∗X))/(real-of-int V^X)*
shows *abs (ψ (U∗(V+1)) (2∗X +1)/ ψ (U^2∗V) (X+1) − ϱ) ≤ 2∗int X∗ϱ / (abs U∗V)*
⟨*proof*⟩

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^2 − 4) ∗ C^2 + 4

definition *E-f::int ⇒ int ⇒ int ⇒ int* **where**
E-f C D x = C^2 ∗ D ∗ x

definition *F-f:: int ⇒ int ⇒ int* **where**
F-f A E = 4 ∗ (A^2 − 4) ∗ E^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*)ˆ*2* ∗ *E*ˆ*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*ˆ*2* − *1*) ∗ *H*ˆ*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*⟩

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 ∀ *m*. ∃ *n*. ∀ *x*. *x*≥*n* ⟶ *y-num x* ≥ *m* ∗ *y-den x* ∧ *y-den x* > *0*
⟨*proof*⟩

lemma *mod-mult*:
  fixes *a*::*int* **and** *b*::*int* **and** *c*::*int* **and** *d*::*int*
  assumes *a mod c = b mod c* ∧ *a mod d = b mod d* ∧ *coprime c d*
  shows *a mod* (*c*∗*d*) = *b mod* (*c*∗*d*)
⟨*proof*⟩

lemma *lemma-4-2-y-int*:
  fixes *A B x*
  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 *y-den x dvd y-num x*
⟨*proof*⟩

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*⟩

lemma *lemma-4-2-cor*:
  fixes *A B*
  defines *C* ≡ *ψ A* (*nat B*)
  assumes *evA*: *even A A*≥*4 B*≥*3*
  shows ∀ *n*. ∃ *x*≥*n*. ∃ *y*≥*n*. *is-square* ((*D-f A C*) ∗ (*F-ACx A C x*) ∗ (*I-ABCxy A B C x y*))
⟨*proof*⟩

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*]:

$P\ 0 \Longrightarrow P\ (Suc\ 0) \Longrightarrow P\ (Suc\ (Suc\ 0)) \Longrightarrow (\bigwedge n.\ (\ (\forall i{\leq}Suc\ (Suc\ n).\ P\ i) \Longrightarrow P\ (Suc\ (Suc\ (Suc\ n)))\ ))$
$\Longrightarrow P\ (n{::}nat)$
⟨*proof*⟩

lemma *sun-lemma2*: $\bigwedge n\ k\ r.\ \psi\ A\ (k{*}n{+}r) =$
$(\sum i{\leq}n.\ int\ (n\ choose\ i) * (\psi\ A\ (Suc\ k) - A{*}\psi\ A\ k)\hat{\ }(n{-}i){*}(\psi\ A\ k)\hat{\ }i{*}\psi\ A\ (r{+}i))$
⟨*proof*⟩

lemma *lucas-consec-coprime*: *coprime* (ψ *A k*) (ψ *A* (*Suc k*))
⟨*proof*⟩

lemma *eq-mod-power*: **fixes** *a*::*int* **and** *b*::*int* **assumes** *a mod n* = *b mod n* **shows** *a*ˆ*k mod n* = *b*ˆ*k mod n*
⟨*proof*⟩

lemma *euclids-lemma*: (*coprime* (*a*::*int*) *b*) ∧ *a dvd* (*b*∗*c*) ⟶ *a dvd c*
⟨*proof*⟩

lemma *coprime-power*: **fixes** *a*::*int* **and** *b*::*int* **assumes** *coprime a b* **shows** *coprime* (*a*ˆ*k*) *b*
⟨*proof*⟩

lemma *dvd-remove-psi*:
fixes *A*::*int* **and** *k*::*nat* **and** *m*::*nat*
assumes (ψ *A k*) *dvd* (ψ *A m*) **and** *A*ˆ*2*−*4*≥*0* **and** *k*>*0*
shows (*int k*) *dvd* (*int m*)
⟨*proof*⟩

lemma *sun-lemma7*:
fixes *A*::*int* **and** *k*::*nat* **and** *m*::*nat*
assumes *A*ˆ*2*−*4*≥*0* **and** (ψ *A k*)ˆ*2 dvd* ψ *A m* **and** *k*>*0*
shows ψ *A k dvd* (*int m*)
⟨*proof*⟩

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* ⇒ *int* ⇒ *int* **where**
ψ-*int A n* = (−*1*)ˆ(*if n* ≥ *0 then 0 else 1*) ∗ ψ *A* (*nat* (*abs n*))
definition χ-*int*::*int* ⇒ *int* ⇒ *int* **where**
χ-*int A n* = χ *A* (*nat* (*abs n*))

lemma ψ-*int-eq*: ψ-*int A n* = (*if n* ≥ *0 then 1 else* −*1*) ∗ ψ *A* (*nat* (*abs n*))

⟨proof⟩

lemma *ψ-int-ind*:
fixes *A::int* **and** *n::int*
shows ψ*-int A (n+2) = A∗ψ-int A (n+1) − ψ-int A n*
⟨proof⟩

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⟩

lemma *ψ-int-odd*:
fixes *A::int* **and** *n::int*
shows *ψ-int A (−n) = −ψ-int A n*
⟨proof⟩

lemma *χ-int-even*:
fixes *A::int* **and** *n::int*
shows *χ-int A (−n) = χ-int A n*
⟨proof⟩

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⟩

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⟩

lemma *lucas-solves-pell*:
fixes *A :: int*
shows *(A^2−4)∗(ψ-int A m)^2 + 4 = (χ-int A m)^2*
⟨proof⟩

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⟩

corollary *technical-lemma2-part2*:

fixes *r*::*int* **and** *A*::*int* **and** *n*::*int* **and** *q*::*int* **and** *k*::*int*
assumes $n \neq 0$ **and** *χ-int A n* = *2*∗k*
shows *ψ-int A (4∗n∗q+r) mod k = ψ-int A r mod k*
⟨*proof*⟩

corollary *technical-cor3*:
fixes *r*::*int* **and** *A*::*int* **and** *n*::*int* **and** *k*::*int*
assumes $n \neq 0$ **and** *χ-int A n* = *2*∗k*
shows *ψ-int A (2∗n+r) mod k = (−ψ-int A r) mod k*
⟨*proof*⟩

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*⟩

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** *α^2 = A^2−4* **and** *α > 0*
defines *βp ≡ (A+α) / 2* **and** *βm ≡ (A−α) / 2*
shows *real-of-int (ψ A n) = (βp^n−βm^n) / α ∧*
real-of-int (χ A n) = βp^n+βm^n
⟨*proof*⟩

lemma *χ-is-Bigger-sqrt5ψ*: *A > 2 ⟹ χ A n > sqrt 5∗ψ A n*
⟨*proof*⟩

lemma *χ-is-Bigger-2ψ*: *A > 2 ⟹ χ A n > 2∗ψ A n*
⟨*proof*⟩

lemma *ψ-ineq-opti*:
fixes *A*::*int* **and** *n*::*nat*
assumes *A > 2*
shows *5∗ ψ A n < 2∗ψ A (n+1)*
⟨*proof*⟩

lemma *ψ-doubles*:
fixes *A*::*int* **and** *n*::*nat*
assumes *A > 2*
shows *2∗ψ A n < ψ A (n+1)*
⟨*proof*⟩

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*⟩

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*⟩

lemma *mod-pos*:
fixes *k*::*int* **and** *n*::*int*
assumes *n* > *0*
shows *0* ≤ *k mod n* ∧ *k mod n* < *n*
⟨*proof*⟩

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*⟩

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*)
⟹ (*s mod* (*4*∗*n*) = *t mod* (*4*∗*n*) ∨ (*s*+*t*) *mod* (*4*∗*n*) = (*2*∗*n*) *mod* (*4*∗*n*))
⟨*proof*⟩

lemma (**in** *bridge-variables*) *sun-lemma24*:
fixes *A*::*int* **and** *B*::*int* **and** *C*::*int* **and** *x*::*int* **and** *y*::*int*
assumes *abs A* ≥ *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*)
∧ *is-square* (*F-ACx A C x*) ∧ *is-square* (*I-ABCxy A B C x y*))
⟨*proof*⟩

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 $(\exists k.\ (A\hat{}2-4)*X\hat{}2 + 4 = k\hat{}2) \Longrightarrow (\exists m.\ X = \psi\text{-}int\ A\ m)$
⟨*proof*⟩

lemma *lucas-modN-int*:
  fixes *A*::*int* **and** *B*::*int* **and** *n*::*int*
  assumes $A\ mod\ n = B\ mod\ n$
  shows $(\psi\text{-}int\ A\ m)\ mod\ n = (\psi\text{-}int\ B\ m)\ mod\ n$
⟨*proof*⟩

lemma *div-mod*: $(n::int)\ dvd\ m \Longrightarrow k\ mod\ m = l\ mod\ m \Longrightarrow k\ mod\ n = l\ mod\ n$
⟨*proof*⟩

lemma *ψ-int-minusA*: $\psi\text{-}int\ (-X)\ n = (-1)\hat{}(nat\ (abs\ n)\ +1)*\psi\text{-}int\ X\ n$ **for** *n* *X*
⟨*proof*⟩

lemma *eq-ψ-int*: $abs\ X > 1 \Longrightarrow abs\ (\psi\text{-}int\ X\ n) = \psi\ (abs\ X)\ (nat\ (abs\ n))$ **for** *X n*
  ⟨*proof*⟩

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** $\chi\text{-}int\ (abs\ A)\ n = 2*k$
  shows $(\psi\text{-}int\ A\ s\ mod\ k = \psi\text{-}int\ A\ t\ mod\ k)$
    $\Longrightarrow (s\ mod\ (2*n) = t\ mod\ (2*n) \lor (s+t)\ mod\ (2*n) = 0\ mod\ (2*n))$
⟨*proof*⟩

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** $x \neq 0$
and *is-square* $(D\text{-}f\ A\ C * F\text{-}ACx\ A\ C\ x * I\text{-}ABCxy\ A\ B\ C\ x\ y)$
shows $C = \psi\text{-}int\ A\ B$
⟨*proof*⟩

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 $L :: int \Rightarrow int \Rightarrow int$
where $L\ l\ Y = l * Y$
definition $U :: int \Rightarrow int \Rightarrow int \Rightarrow int$
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* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *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 $a \geq 3$
shows $(a-1)$ ^$(n+2) \geq 3 + int\ n + (a-2)$ ^2
⟨*proof*⟩

lemma *change-sum*:
fixes $f::int \Rightarrow int$ **and** *n::nat*
shows $(\sum i \leq n.\ (f\ (int\ i))) = sum\ (\lambda i.\ f\ i)\ (set[0..int\ n])$
⟨*proof*⟩

lemma *chang-var1*:
fixes $f::int \Rightarrow int$ **and** *n::nat*
shows $sum\ (\lambda i.\ f\ (i+1))\ (set[0..int\ n]) = sum\ (\lambda i.\ f\ i)\ (set[1..int\ (Suc\ n)])$
⟨*proof*⟩

lemma *chang-var*:
fixes $f::int \Rightarrow int$ **and** *n::nat* **and** *m::nat*
shows $sum\ (\lambda i.\ f\ i)\ (set[int\ n..int\ (n+m)]) = sum\ (\lambda i.\ f\ (int\ (n+m) - i))$ $(set[0..int\ m])$
⟨*proof*⟩

6.2 Proof of implication *(1)*⟹*(3)*

context *bridge-variables*
begin

lemma *theorem-II-1-3*:
assumes *b-def1*:$(b::int) \geq 0$ **and** *Y-def*:$(Y::int) \geq b \wedge Y \geq 2$^$8$ **and** *X-def*:$(X::int) \geq 3*b$
and *g-def*:$(g::int) \geq 1$
shows $(statement1\ b\ Y\ X) \Longrightarrow (statement2a\ b\ Y\ X\ g)$
⟨*proof*⟩

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 $n \geq p$ **and** $a \neq 0$
shows a^n / a^$p = a$^$(n-p)$
⟨*proof*⟩

lemma *inv-decr*:

fixes *a*::*real* **and** *b*::*real*
assumes $a \geq b$ **and** $b > 0$
shows $1/a \leq 1/b$
⟨*proof*⟩

lemma *div-pow*:
fixes *a*::*real* **and** *n*::*nat* **and** *m*::*nat*
assumes $m<n$ **and** $a \neq 0$
shows $a\hat{}n/a\hat{}m = a*a\hat{}(n-m-1)$
⟨*proof*⟩

lemma *power-majoration*:
fixes *a*::*real* **and** *n*::*nat*
assumes $0 < a$ **and** $a \leq 1$
shows $(1+a)\hat{}n \leq 1 + (2\hat{}n-1)*a$
⟨*proof*⟩

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*⟩

lemma *lucas-modN-int*:
fixes α::*int* **and** *m*::*int*
shows ψ-*int* α *m mod* $(\alpha - 2)$ = *m mod* $(\alpha - 2)$
⟨*proof*⟩

6.3 Proof of implication (*2*)⟹(*1*)

lemma (**in** *bridge-variables*) *theorem-II-2-1*:
assumes *b-def*:(*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 (*statement2 b Y X g*)⟹(*statement1 b Y X*)
⟨*proof*⟩

6.4 Proof of implication (*2a*)⟹(*2*)

lemma (**in** *bridge-variables*) *theorem-II-3-2*:
assumes *b-def*:(*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 (*statement2a b Y X g*)⟹(*statement2 b Y X g*)
⟨*proof*⟩

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*
  ⟨*proof*⟩

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*))
  ⟨*proof*⟩

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*⟩

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*)
⟨*proof*⟩

lemma *square-sqrt*: (*cpx-sqrt n*)^*2* = *n*
⟨*proof*⟩

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*⟩

lemma *Zero-in-field*: *0* ∈ *field l*
⟨*proof*⟩

lemma *field-incr*: *field l* ⊆ *field* (*a*#*l*)
⟨*proof*⟩

lemma *int-in-field*: ⋀*x*::*int*. *of-int x* ∈ *field l*
⟨*proof*⟩

lemma *field-add-mult*: ⋀*x y*. *x* ∈ *field l* ∧ *y* ∈ *field l* ⟹ (*x* + *y* ∈ *field l* ∧ *x*∗*y* ∈ *field l*)
⟨*proof*⟩

lemma *field-square*: *x* ∈ *field l* ⟹ *x*^*2* ∈ *field l* ⟨*proof*⟩

lemma *field-opp*: *x* ∈ *field l* ⟹ −*x* ∈ *field l*
⟨*proof*⟩

lemma *field-inv-div*: ⋀*x y*. *x* ∈ *field l* ∧ *y* ∈ *field l* ⟹ (*1*/*x* ∈ *field l* ∧ *y*/*x* ∈ *field l*)
⟨*proof*⟩

lemma *field-sum*: (∀ *x*∈*S*. *f x*∈*field l*) ⟹ *finite S* ⟹ (∑ *x*∈*S*. *f x*) ∈ *field l*
⟨*proof*⟩

lemma *field-comm*: *field* (*a*#*b*#*l*) = *field* (*b*#*a*#*l*)
⟨*proof*⟩

lemma *field-idempot1*: *field* (*a*#*a*#*l*) = *field* (*a*#*l*)
⟨*proof*⟩

lemma *field-idempot*: *a* ∈ *set l* ⟹ *field* (*a*#*l*) = *field l*
⟨*proof*⟩

lemma *disjoint-field-extensions-no-prime-roots*:

fixes *p-l::int list*
shows $\bigwedge$(*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*⟩

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*}

⟨*proof*⟩

lemma *field-prod*: *field* ((*a*∗*b*)#*l*) ⊆ *field* (*a*#*b*#*l*)
⟨*proof*⟩

lemma *field-add-in*: *cpx-sqrt a* ∈ *field l* ⟹ *field* (*a*#*l*) = *field l*
⟨*proof*⟩

lemma *field-add-0*: *field* (*0*#*l*) = *field l*
⟨*proof*⟩

lemma *field-remove-zeros*: ∃ *l′::int list*. *set l′* = *set l* − {*0*} ∧ *field l′* = *field l*
⟨*proof*⟩

lemma *sqrt-in-field*: *x* ∈ *set l* ⟹ *cpx-sqrt x* ∈ *field l*
⟨*proof*⟩

lemma *field-incr2*: *field l* ⊆ *field m* ⟹ *field* (*a*#*l*) ⊆ *field* (*a*#*m*)
⟨*proof*⟩

lemma *min-set-induct*: (*P::int set* ⇒ *bool*) {}
⟹ ($\bigwedge$*X*. $\bigwedge$*x*. *finite X* ⟹ *x* = *Min* (*X*∪{*x*}) ⟹ *P X* ⟹ *P* (*X*∪{*x*})) ⟹ ($\bigwedge$*X*. *finite X* ⟹ *P X*)
⟨*proof*⟩

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*)
⟨*proof*⟩

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*⟩

corollary *sorting-set3*:
  fixes *B*::*int set* **and** *C*::*int set*
  assumes ∀ *b*∈*B*. ∀ *c*∈*C*. *b* < *c* **and** *finite B* **and** *finite C*
  shows *sorted-list-of-set* (*B*∪*C*) = (*sorted-list-of-set B*)@(*sorted-list-of-set C*)
  ⟨*proof*⟩

lemma *min-mset-induct*: (*P*::*int multiset* ⇒ *bool*) {#}
  ⟹ (⋀*X*. ⋀*x*. *x* = *Min-mset* (*X* + {#*x*#}) ⟹ *P X* ⟹ *P* (*X* + {#*x*#}))
⟹ (⋀*X*. *P X*)
⟨*proof*⟩

lemma *field-prod2*: *field* ((*prod-mset A*)#*l*) ⊆ *field* ((*sorted-list-of-set* (*set-mset A*))@*l*)
⟨*proof*⟩

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*⟩

lemma *field-n-in-field-prime2*:
  fixes *n*::*int* **and** *l*::*int list*
  shows *field* (*n*#*l*) ⊆ *field* ((−*1*)#(*prime-list n*)@*l*)
⟨*proof*⟩

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*⟩

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*⟩

lemma *sqrt-int-smaller*:
  fixes *a*::*int* **assumes** *a*≥*0*
  shows *sqrt a* ≤ *a*
⟨*proof*⟩

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* ≡ *Var 0*

definition $\mathcal{A}_1$ :: *int mpoly* **where** $\mathcal{A}_1$ ≡ *Var 1*
definition $\mathcal{A}_2$ :: *int mpoly* **where** $\mathcal{A}_2$ ≡ *Var 2*
definition $\mathcal{A}_3$ :: *int mpoly* **where** $\mathcal{A}_3$ ≡ *Var 3*
definition $\mathcal{X}_3$ :: *int mpoly* **where** $\mathcal{X}_3$ ≡ *Const 1* + $\mathcal{A}_1$ ^*2* + $\mathcal{A}_2$ ^*2* + $\mathcal{A}_3$ ^*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×'a×'a ⇒ 'a* **where** *fst3* (*a*, *b*, *c*) = *a*
fun *snd3*:: *'a×'a×'a ⇒ 'a* **where** *snd3* (*a*, *b*, *c*) = *b*
fun *trd3*:: *'a×'a×'a ⇒ 'a* **where** *trd3* (*a*, *b*, *c*) = *c*
fun *fun3*:: *'a×'a×'a⇒nat⇒'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* ⟨*proof*⟩
lemma *fun3-2-eq-snd3*: *fun3 a 2* = *snd3 a* ⟨*proof*⟩
lemma *fun3-3-eq-trd3*: *fun3 a 3* = *trd3 a*
⟨*proof*⟩

lemmas *fun3-def* = *fun3-1-eq-fst3 fun3-2-eq-snd3 fun3-3-eq-trd3*

definition *J3* :: *int mpoly* **where**
J3 = ((*x*^*2*+$\mathcal{A}_1$+$\mathcal{A}_2$∗$\mathcal{X}_3$ ^*2*−$\mathcal{A}_3$∗$\mathcal{X}_3$ ^*4*) ^*2* + *4*∗*x*^*2*∗$\mathcal{A}_1$ − *4*∗*x*^*2*∗$\mathcal{A}_2$∗$\mathcal{X}_3$ ^*2* − *4*∗$\mathcal{A}_1$∗$\mathcal{A}_2$∗$\mathcal{X}_3$ ^*2*) ^*2*
− $\mathcal{A}_1$ ∗ ((*4*∗*x*∗(*x*^*2*+$\mathcal{A}_1$+$\mathcal{A}_2$∗$\mathcal{X}_3$ ^*2* − $\mathcal{A}_3$∗$\mathcal{X}_3$ ^*4*) − *8*∗*x*∗$\mathcal{A}_2$∗$\mathcal{X}_3$ ^*2*)) ^*2*

definition *r* **where** *r* = *MPoly-Type.degree J3 0*

lemma *J3-vars*: *vars J3* ⊆ {*0*, *1*, *2*, *3*}
⟨*proof*⟩

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^2))
⟨*proof*⟩

lemma *J3-cancels-iff*:
  fixes *f::nat⇒int*
  defines $X \equiv 1 + (f\ 1)\hat{}2 + (f\ 2)\hat{}2 + (f\ 3)\hat{}2$
  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*⟩

lemma *J3-zeros-bound*:
  fixes *A1 A2 A3*
  defines $X \equiv 1 + A1\hat{}2 + A2\hat{}2 + A3\hat{}2$
  defines $I \equiv X\hat{}3$
  shows *(∀x. insertion ((λ-. 0)(0:=x, 1:=A1, 2:=A2, 3:=A3)) J3 = 0 ⟶ x > −I)*
⟨*proof*⟩

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*⟩

lemma *sqrt-is-int-iff-square*:
$(\exists\, k{::}int.\ cpx\text{-}sqrt\ n = of\text{-}int\ k) \longleftrightarrow (\exists\, a.\ n = a\hat{\ }2)$
⟨*proof*⟩

abbreviation *divd* (**infixl** *divd 70*) **where** (*divd*) ≡ *Rings.divide-class.divide*

lemma *square-odd-mult-prime*:
assumes $b \geq 0$
shows $\neg is\text{-}square\ b \Longrightarrow \exists\, p.\ prime\ p \wedge odd(multiplicity\ p\ b)$
⟨*proof*⟩

lemma *square-even-multiplicity*: $prime\ p \wedge is\text{-}square\ a \Longrightarrow even\ (multiplicity\ p\ a)$
⟨*proof*⟩

J3 correctly encodes the three squares

lemma *J3-encodes-three-squares*:
fixes *a1*::*int* **and** *a2*::*int* **and** *a3*::*int*
defines $f \equiv (\lambda y.\ (\lambda\text{-}.\ 0)(0{:=}y,\ 1{:=}a1,\ 2{:=}a2,\ 3{:=}a3))$
shows $(is\text{-}square\ a1 \wedge is\text{-}square\ a2 \wedge is\text{-}square\ a3) \longleftrightarrow (\exists\, y{::}int.\ insertion\ (f\ y)\ J3 = 0)$
⟨*proof*⟩

end

end
theory *Pi-Relations*
imports *J3-Relations*
begin

7.4 The Π polynomial

lemma *finite-when*: $finite\ \{x.\ (f\ x\ when\ x = c) \neq 0\}$
⟨*proof*⟩

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** $\mathcal{V} = \mathcal{T}\hat{\ }2 + \mathcal{S}\hat{\ }2 * \mathrm{n}$

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* (λk. (*degree* $\mathcal{F}$ *v when k* = *v*))) = *1*
begin

definition *coeff-F* **where**
  coeff-F d = *coeff* $\mathcal{F}$ (*Abs-poly-mapping* (λk. (*d when k* = *v*)))
definition *q* :: *nat* **where**
  q = *degree* $\mathcal{F}$ *v*
definition *G* :: *int mpoly* **where**
  G = ($\sum$ *i*=*0*..*q*. *of-int* (*coeff-F i*) ∗ ($\mathcal{Y}$−$\mathcal{I}$−$\mathcal{T}$ ^*2*) ^*i*)
definition *G-sub* :: *nat*⇒*int mpoly* **where**
  G-sub j = ($\sum$ *i*=*j*..*q*. *of-int* (*coeff-F i*) ∗ *of-nat* (*i choose j*) ∗ (−$\mathcal{I}$−$\mathcal{T}$ ^*2*) ^(*i*−*j*))
definition *H* :: *int mpoly* **where**
  H = ($\sum$ *j*=*0*..*q*. *G-sub j* ∗ $\mathfrak{Z}$ ^*j* ∗ $\mathcal{J}$ ^(*q*−*j*))
definition Π :: *int mpoly* **where**
  Π = ($\sum$ *j*=*0*..*q*. *G-sub j* ∗ ($\mathcal{S}$ ^*2*∗n + $\mathcal{T}$ ^*2*) ^*j* ∗ ($\mathcal{S}$ ^*2*∗(*2*∗$\mathcal{R}$−*1*)) ^(*q*−*j*))

lemma *eval-F*:
  insertion f $\mathcal{F}$ = ($\sum$ *d* = *0*..*q*. *coeff-F d* ∗ (*f v* ^ *d*))
⟨*proof*⟩

lemma *triangular-sum*: ($\sum$ *k*=*0*..(*n*::*nat*). ($\sum$ *j*=*0*..*k*. *f j k*)) = ($\sum$ *j*=*0*..(*n*::*nat*). ($\sum$ *k*=*j*..*n*. *f j k*))
⟨*proof*⟩

lemma *G-expr*:
   G = ($\sum$ *j*=*0*..*q*. $\mathcal{Y}$ ^*j* ∗ ($\sum$ *i*=*j*..*q*. *of-int* (*coeff-F i*) ∗ *of-nat* (*i choose j*) ∗ (−$\mathcal{I}$−$\mathcal{T}$ ^*2*) ^(*i*−*j*)))
⟨*proof*⟩

lemma *G-in-Y*: *G* = ($\sum$ *j*=*0*..*q*. $\mathcal{Y}$ ^*j* ∗ *G-sub j*)
  ⟨*proof*⟩

lemma *Gq-eq-1*: *G-sub q* = *1*
⟨*proof*⟩

lemma *lemma-J-div-Z* :
  ($\sum$ *j'*=*0*..*q*. *insertion f* (*G-sub j'*) ∗ *z*^*j'* ∗ *j*^(*q*−*j'*)) = *0* ⟹ *j dvd z* **for** *f z j*
⟨*proof*⟩

lemma *lemma-pos*:
  assumes $(\sum j'=0..q.\ insertion\ f\ (G\text{-}sub\ j') * z\hat{}j' * j\hat{}(q-j')) = 0$
  $j\neq 0$ $z = z' * j$
  shows $insertion\ (\lambda\text{-}.\ z' - f\ 7 - (f\ 5)\hat{}2)\ \mathcal{F} = 0$
⟨*proof*⟩

lemma Π-*encodes-correctly*:
  fixes *S T R I* :: *int*
  assumes $S \neq 0$
  $(\forall x.\ insertion\ (\lambda\text{-}.\ x)\ \mathcal{F} = 0 \longrightarrow x > -I)$
  and *S dvd T*
  $R > 0$
  $insertion\ (\lambda\text{-}.\ y)\ \mathcal{F} = 0$
  defines $\varphi\ n \equiv (\lambda\text{-}.\ 0)(4{:=}\ S,\ 5{:=}T,\ 6{:=}R,\ 7{:=}I,\ 11{:=}int\ n)$
  shows $\exists n :: nat.\ insertion\ (\varphi\ n)\ \Pi = 0 \wedge n = (2{*}R{-}1){*}(y + I + T\hat{}2) - (T\ div\ S)\ \hat{}\ 2$
⟨*proof*⟩

lemma Π-*encodes-correctly-2*:
  fixes *S T R I* :: *int*
  assumes $S \neq 0$
  $(\forall x.\ insertion\ (\lambda\text{-}.\ x)\ \mathcal{F} = 0 \longrightarrow x > -I)$
  defines $\varphi\ n \equiv (\lambda\text{-}.\ 0)(4{:=}\ S,\ 5{:=}T,\ 6{:=}R,\ 7{:=}I,\ 11{:=}int\ n)$
  assumes $\exists n :: nat.\ insertion\ (\varphi\ n)\ \Pi = 0$
  shows $S\ dvd\ T \wedge R > 0 \wedge (\exists x{::}int.\ (insertion\ (\lambda\text{-}.\ x)\ \mathcal{F}) = 0)$
⟨*proof*⟩

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* :: $(int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int) \Rightarrow int$ (- π [*1000*] *1000*)
syntax *pi-fn* :: $(int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int) \Rightarrow int$ (λπ. - [*0*] *0*)
⟨*ML*⟩

locale *section5* = *section5-given* + *section5-variables*

```
begin

definition X_0 :: int ⇒ int ⇒ int ⇒ int ⇒ int ⇒ int ⇒ int ⇒ int where
  X_0 π = 1 + A_1^2 + A_2^2 + A_3^2
definition I_0 :: int ⇒ int ⇒ int ⇒ int ⇒ int ⇒ int ⇒ int ⇒ int where
  I_0 π = (X_0 π)^3
definition U_0 :: int ⇒ int ⇒ int ⇒ int ⇒ int ⇒ int ⇒ int ⇒ int where
  U_0 π = S^2 * (2 * R − 1)
definition V_0 :: int ⇒ int ⇒ int ⇒ int ⇒ int ⇒ int ⇒ int ⇒ int where
  V_0 π = S^2 * n + T^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 ⇒ int where
  M3 A_1 A_2 A_3 S T R n =
    (
      ((
        (
          (
            ((V_0 π) − (U_0 π) * (I_0 π) − (U_0 π) * T^2)^2
              + (U_0 π)^2 * A_1
              + (U_0 π)^2 * A_2 * (X_0 π)^2
              − (U_0 π)^2 * A_3 * (X_0 π)^4
          )
        )^2
          + 4 * (U_0 π)^2 * (V_0 π − U_0 π * I_0 π − U_0 π * T^2)^2 * A_1
          − 4 * (U_0 π)^2 * (V_0 π − U_0 π * I_0 π − U_0 π * T^2)^2 * A_2 * (X_0
π)^2
          − 4 * (U_0 π)^4 * A_1 * A_2 * (X_0 π)^2
      )^2)
        − A_1 * (U_0 π * (
          4
            * (V_0 π − U_0 π * I_0 π − U_0 π * T^2)
            * (
              ((V_0 π − U_0 π * I_0 π − U_0 π * T^2))^2
                + (U_0 π)^2 * A_1
                + (U_0 π)^2 * A_2 * (X_0 π)^2
                − (U_0 π)^2 * A_3 * (X_0 π)^4
              )
                − 8
                  * (U_0 π)^2
                  * (V_0 π − U_0 π * I_0 π − U_0 π * T^2)
                  * A_2
                  * (X_0 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*⟩

locale *matiyasevich-polynomial* = *section5*
begin

type-synonym τ=*real*

definition x' :: τ *mpoly* **where** $x' \equiv$ *of-int-mpoly* x
definition $\mathcal{A}_1{}'$ :: τ *mpoly* **where** $\mathcal{A}_1{}' \equiv$ *of-int-mpoly* $\mathcal{A}_1$
definition $\mathcal{A}_2{}'$ :: τ *mpoly* **where** $\mathcal{A}_2{}' \equiv$ *of-int-mpoly* $\mathcal{A}_2$
definition $\mathcal{A}_3{}'$ :: τ *mpoly* **where** $\mathcal{A}_3{}' \equiv$ *of-int-mpoly* $\mathcal{A}_3$
definition $\mathcal{X}_3{}'$ :: τ *mpoly* **where** $\mathcal{X}_3{}' \equiv$ *of-int-mpoly* $\mathcal{X}_3$

definition $\mathcal{S}'$ :: τ *mpoly* **where** $\mathcal{S}' \equiv$ *of-int-mpoly* $\mathcal{S}$
definition $\mathcal{T}'$ :: τ *mpoly* **where** $\mathcal{T}' \equiv$ *of-int-mpoly* $\mathcal{T}$
definition $\mathcal{R}'$ :: τ *mpoly* **where** $\mathcal{R}' \equiv$ *of-int-mpoly* $\mathcal{R}$
definition $\mathcal{I}'$ :: τ *mpoly* **where** $\mathcal{I}' \equiv$ *of-int-mpoly* $\mathcal{I}$
definition $\mathcal{Y}'$ :: τ *mpoly* **where** $\mathcal{Y}' \equiv$ *of-int-mpoly* $\mathcal{Y}$
definition $\mathcal{Z}'$ :: τ *mpoly* **where** $\mathcal{Z}' \equiv$ *of-int-mpoly* $\mathfrak{Z}$
definition $\mathcal{J}'$ :: τ *mpoly* **where** $\mathcal{J}' \equiv$ *of-int-mpoly* $\mathcal{J}$
definition n' :: τ *mpoly* **where** $\mathrm{n}' \equiv$ *of-int-mpoly* n

definition $\mathcal{U}'$ ::τ *mpoly* **where** $\mathcal{U}' =$ *of-int-mpoly* $\mathcal{U}$
definition $\mathcal{V}'$ ::τ *mpoly* **where** $\mathcal{V}' =$ *of-int-mpoly* $\mathcal{V}$
definition *J3'* :: τ *mpoly* **where** *J3'* = *of-int-mpoly* *J3*
definition ψ' :: *nat* $\Rightarrow \tau$ *mpoly* **where** $\psi' =$ *of-int-mpoly* $\circ$ $((\lambda\text{-}.\ 0)(0 := \mathcal{T}\ \hat{}2{+}\mathcal{S}\ \hat{}2{*}\mathrm{n}$
$-\mathcal{T}\ \hat{}2{*}\mathcal{U}-\ \mathcal{X}_3\ \hat{}3{*}\mathcal{U},\ 1 := \mathcal{U}\ \hat{}2{*}\mathcal{A}_1,\ 2 := \mathcal{U}\ \hat{}2{*}\mathcal{A}_2,\ 3 := \mathcal{U}\ \hat{}2{*}\mathcal{A}_3))$
definition *M3-poly'* :: τ *mpoly* **where** *M3-poly'* $\equiv$ *of-int-mpoly M3-poly*

lemma *Pi-equals-M3-rationals*:
  fixes A_1 A_2 A_3 R S T n :: *int*
  defines $X \equiv X_0\ \pi$
  defines $I \equiv I_0\ \pi$

defines $U \equiv U_0\ \pi$
defines $V \equiv V_0\ \pi$

defines $\varphi \equiv (\lambda\text{-}.\ 0)(0 := Var\ 0,\ 1 := Const\ A_1,\ 2 := Const\ A_2,\ 3 := Const\ A_3)$

defines $\varphi' \equiv$ *of-int-mpoly* $\circ\ \varphi$
defines $\mathcal{F} \equiv$ *poly-subst* φ *J3*
defines $\mathcal{F}' \equiv$ *of-int-mpoly* $\mathcal{F}$ :: τ *mpoly*
defines $q \equiv$ *MPoly-Type.degree* $\mathcal{F}$ *0*
defines $\Pi \equiv$ *pi-relations*.Π $\mathcal{F}$ *0*

defines *G-sub* $\equiv$ *pi-relations.G-sub* $\mathcal{F}$ *0*

defines *G-sub'* $\equiv$ *of-int-mpoly* $\circ$ (*pi-relations.G-sub* $\mathcal{F}$ *0*)

defines $\xi \equiv ((\lambda\text{-}.\ 0)(4 := S,\ 5 := T,\ 6{:=}R,\ 7{:=}X\hat{\ }3,\ 11{:=}n))$

defines $\xi' \equiv$ *of-int* $\circ\ \xi$

assumes $U \neq 0$
assumes $q = 8$
assumes *F-monom-over-0*: *vars* $\mathcal{F} \subseteq \{0\}$
and *F-one-0*: *MPoly-Type.coeff* $\mathcal{F}$ (*Abs-poly-mapping* (λk. (*MPoly-Type.degree* $\mathcal{F}$ *0 when* $k = 0$))) $= 1$

shows *insertion* $\xi\ \Pi = M3\ \pi$
⟨*proof*⟩

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 $S{\neq}0$

defines $\gamma' \equiv \lambda(n :: int)\ fn.\ fn\ A_1\ A_2\ A_3\ S\ T\ R\ n :: int$

shows (S *dvd* $T \wedge R > 0 \wedge$ *is-square* $A_1 \wedge$ *is-square* $A_2 \wedge$ *is-square* A_3)
$= (\exists\, n.\ n{\geq}0 \wedge (\gamma'\ n)\ M3 = 0)$ (**is** *?LHS* = *?RHS*)

⟨*proof*⟩

lemma *relation-combining*:
  assumes $S \neq 0$
  shows $(S\ dvd\ T \wedge R > 0 \wedge is\text{-}square\ A_1 \wedge is\text{-}square\ A_2 \wedge is\text{-}square\ A_3)$
  $= (\exists n \geq 0.\ M3\ A_1\ A_2\ A_3\ S\ T\ R\ n = 0)$
  ⟨*proof*⟩
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 ℕ and 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* :: $(int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int) \Rightarrow int$ (- τ [1000] 1000)
syntax *tau-fn* :: $(int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int) \Rightarrow int$ (λτ. - [0] 0)

⟨*ML*⟩

locale *combined-variables* =
  fixes *P* :: *int mpoly*
begin

Copy of the polynomials from Theorem I

definition P_1 :: *int mpoly* **where**
  $P_1 \equiv suitable\text{-}for\text{-}coding\ P$

abbreviation $\delta \equiv coding\text{-}variables.\delta\ P_1$
abbreviation $\nu \equiv coding\text{-}variables.\nu\ P_1$

definition *b* :: $int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
  where $b\ \tau = coding\text{-}variables.b\ a\ f$
definition *X* :: $int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
  where $X\ \tau = coding\text{-}variables.X\ P_1\ a\ f\ g$

definition Y :: $int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
where $Y\ \tau = coding\text{-}variables.Y\ P_1\ a\ f$

poly-extract b

definition *Y-poly* :: *int mpoly* **where**
Y-poly $\equiv$ *coding-variables.Y-poly* P_1
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*)
⟨*proof*⟩

definition *X-poly* :: *int mpoly* **where**
X-poly $\equiv$ *coding-variables.X-poly* P_1
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*)
⟨*proof*⟩

lemma *δ-gt0*: $\delta > 0$
⟨*proof*⟩

Polynomials from Theorem II

definition L :: $int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
where $L\ \tau = bridge\text{-}variables.L\ l\ (Y\ \tau)$
definition U :: $int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
where $U\ \tau = bridge\text{-}variables.U\ l\ (X\ \tau)\ (Y\ \tau)$
definition V :: $int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
where $V\ \tau = bridge\text{-}variables.V\ w\ g\ (Y\ \tau)$
definition A :: $int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
where $A\ \tau = bridge\text{-}variables.A\ l\ w\ g\ (Y\ \tau)\ (X\ \tau)$
definition C :: $int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
where $C\ \tau = bridge\text{-}variables.C\ l\ w\ h\ g\ (Y\ \tau)\ (X\ \tau)$
definition D :: $int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
where $D\ \tau = bridge\text{-}variables.D\ l\ w\ h\ g\ (Y\ \tau)\ (X\ \tau)$
definition F :: $int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
where $F\ \tau = bridge\text{-}variables.F\ l\ w\ h\ x\ g\ (Y\ \tau)\ (X\ \tau)$
definition I :: $int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$

$\Rightarrow$ *int*
where $I\ \tau = bridge\text{-}variables.I\ l\ w\ h\ x\ y\ g\ (Y\ \tau)\ (X\ \tau)$
definition W :: $int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
$\Rightarrow int$
where $W\ \tau = bridge\text{-}variables.W\ w\ (coding\text{-}variables.b\ a\ f)$
definition K :: $int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
$\Rightarrow int$
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 $M3$ :: $int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
where $M3\ A_1\ A_2\ A_3\ S\ T\ Q'\ m = insertion\text{-}list\ [A_1,\ A_2,\ A_3,\ S,\ T,\ Q',\ m]$
section5.M3-poly

poly-extract $M3$

lemma *max-vars-M3*: $max\text{-}vars\ M3\text{-}poly \leq 6$
$\langle proof \rangle$

definition μ :: $int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
$\Rightarrow int$
where $\mu\ \tau = (coding\text{-}variables.\gamma\ P_1) * (b\ \tau)\ \hat{}\ (coding\text{-}variables.\alpha\ P_1)$

poly-extract μ

definition A_1 :: $int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
$\Rightarrow int$
where $A_1\ \tau = b\ \tau$
definition A_2 :: $int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
$\Rightarrow int$
where $A_2\ \tau = (D\ \tau) * (F\ \tau) * (I\ \tau)$
definition A_3 :: $int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
$\Rightarrow int$
where $A_3\ \tau = ((U\ \tau)\ \hat{}\ 4 * (V\ \tau)^2 - 4) * (K\ \tau)^2 + 4$
definition S :: $int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
$\Rightarrow int$

where $S\ \tau = bridge\text{-}variables.S\ l\ w\ g\ (Y\ \tau)\ (X\ \tau)$
definition $T :: int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
where $T\ \tau = bridge\text{-}variables.T\ l\ w\ h\ g\ (Y\ \tau)\ (X\ \tau)\ (b\ \tau)$
definition $R :: int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
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 $m :: int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
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 $\sigma :: (int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int) \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
where $(\sigma\ fn)\ \tau = fn\ (A_1\ \tau)\ (A_2\ \tau)\ (A_3\ \tau)\ (S\ \tau)\ (T\ \tau)\ (R\ \tau)\ (m\ \tau)$

definition $Q :: int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int$
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*: $total\text{-}degree\ (coding\text{-}variables.M\text{-}poly\ P_1) \leq (1+(\delta+1)\hat{}\nu) * 2*\delta$
⟨*proof*⟩

lemma *D-poly-degree-correct*: $total\text{-}degree\ (coding\text{-}variables.D\text{-}poly\ P_1) \leq (\delta+1)\hat{}(\nu+1) * (2*\delta)$
⟨*proof*⟩

lemma *K-poly-degree-correct*: $total\text{-}degree\ (coding\text{-}variables.K\text{-}poly\ P_1)$
$\leq max\ (\delta*(1+2*\delta) + (\delta+1)\hat{}(\nu+1) * 2*\delta)\ ((1 + (2*\delta+1)*(\delta+1)\hat{}\nu) * 2*\delta)$
⟨*proof*⟩

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*⟩

poly-degree A_1*-poly*
poly-degree A_2*-poly*
poly-degree A_3*-poly*
poly-degree *S-poly*
poly-degree *T-poly*
poly-degree *R-poly*

lemma A_1*-vars*: *max-vars* A_1*-poly* ≤ *8*
⟨*proof*⟩

lemma *h0*: *max-vars* (*4*::*int mpoly*) ≤ *8*
⟨*proof*⟩

lemma *h1*: *max-vars* (*8*::*int mpoly*) ≤ *8*
⟨*proof*⟩

lemma *h2*: *max-vars* (*32*::*int mpoly*) ≤ *8*
⟨*proof*⟩

lemma A_2*-vars*: *max-vars* A_2*-poly* ≤ *8*
⟨*proof*⟩

lemma A_3*-vars*: *max-vars* A_3*-poly* ≤ *8*
⟨*proof*⟩

lemma *S-vars*: *max-vars S-poly* ≤ *8*
⟨*proof*⟩

lemma *T-vars*: *max-vars T-poly* ≤ *8*
⟨*proof*⟩

lemma *K-vars*: *max-vars K-poly* ≤ *8*
⟨*proof*⟩

lemma *L-vars*: *max-vars L-poly* ≤ *8*
⟨*proof*⟩

lemma *C-vars*: *max-vars C-poly* ≤ *8*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *R-vars*: *max-vars R-poly* ≤ *8*
⟨*proof*⟩

lemma *list-vars*: $i \leq 8 \Longrightarrow$ *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*)))))))))] $!_0$
i)
$\leq$ *8*
⟨*proof*⟩

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 $ia \leq 8 \Longrightarrow$
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)
⟨*proof*⟩

lemma *final*: $\bigwedge ia.\ ia \leq 8 \Longrightarrow$
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* (*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*
⟨*proof*⟩

poly-extract *Q*

lemma *Q-alt*: *Q* = *σ M3*
⟨*proof*⟩

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*⟩

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*⟩

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-variables.Q P*) = *0*)
(**is** - = *?Pa-zero*)
⟨*proof*⟩

theorem *theorem-III*:
fixes *A* :: *nat set* **and** *P* :: *int mpoly*
assumes *is-diophantine-over-N-with A P*
shows $a \in A = (\exists f\ g\ h\ k\ l\ w\ x\ y\ n.\ n \geq 0 \wedge$
insertion ((!) [*int a*, *f*, *g*, *h*, *k*, *l*, *w*, *x*, *y*, *n*])
(*combined-variables.Q-poly P*) = *0*)
⟨*proof*⟩

theorem *nine-unknowns-over-Z-and-N*:
fixes *P* :: *int mpoly*
shows ($\exists$ *z* :: *nat* ⇒ *int*. *is-nonnegative z* ∧
insertion (*z*(*0* := *int a*)) *P* = *0*)
= ($\exists f\ g\ h\ k\ l\ w\ x\ y\ n.\ n \geq 0 \wedge$
insertion ((!) [*int a*, *f*, *g*, *h*, *k*, *l*, *w*, *x*, *y*, *n*])
(*combined-variables.Q-poly P*) = *0*)
⟨*proof*⟩

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* ⇒ *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*) $a\ z_1\ z_2\ z_3\ z_4\ z_5\ z_6\ z_7\ z_8\ (z_9\hat{}2 + z_{10}\hat{}2 + z_{11}\hat{}2 + z_{11})$

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*⟩

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*⟩

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* $\leq \delta$
assumes *max-vars P* $\leq \nu$
shows *Q-tilde-poly-degree* $\leq \eta\ \nu\ \delta$
⟨*proof*⟩

lemma *max-vars-Q-tilde*: *max-vars Q-tilde-poly* ≤ *11*
⟨*proof*⟩

lemma *eleven-unknowns-over-Z*:
fixes *A* :: *nat set*
assumes *is-diophantine-over-N-with A P*
shows $a \in A = (\exists z_1\ z_2\ z_3\ z_4\ z_5\ z_6\ z_7\ z_8\ z_9\ z_{10}\ z_{11}.$
Q-tilde (*int a*) $z_1\ z_2\ z_3\ z_4\ z_5\ z_6\ z_7\ z_8\ z_9\ z_{10}\ z_{11} = 0)$
⟨*proof*⟩

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*)
⟨*proof*⟩

end

theory *Universal-Pairs*
imports *Eleven-Unknowns-Z*
begin

definition *universal-pair-N* ('(-, -')$_{\mathbb{N}}$ [*1000*]) **where**
universal-pair-N ν $\delta \equiv$ ($\forall$ *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* ('(-, -')$_{\mathbb{Z}}$ [*1000*]) **where**
universal-pair-Z ν $\delta \equiv$ ($\forall$ *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*⟩

theorem *universal-pair-1*:
assumes $(58, 4)_{\mathbb{N}}$
shows $(11, 1681043235226619916301182624511918527834137733707408448335539840)_{\mathbb{Z}}$
⟨*proof*⟩

theorem *universal-pair-2*:
assumes $(32, 12)_{\mathbb{N}}$
shows $(11, 950817549694171759711025515571236610412597656252821888)_{\mathbb{Z}}$
⟨*proof*⟩

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.