

Chebyshev Polynomials

Manuel Eberl

April 18, 2024

Abstract

The multiple-angle formulas for \cos and \sin state that for any natural number n , the values of $\cos nx$ and $\sin nx$ can be expressed in terms of $\cos x$ and $\sin x$. To be more precise, there are polynomials T_n and U_n such that $\cos nx = T_n(\cos x)$ and $\sin nx = U_n(\cos x) \sin x$. These are called the *Chebyshev polynomials of the first and second kind*, respectively.

This entry contains a definition of these two families of polynomials in Isabelle/HOL along with some of their most important properties. In particular, it is shown that T_n and U_n are *orthogonal* families of polynomials.

Moreover, we show the well-known result that for any monic polynomial p of degree $n > 0$, it holds that $\sup_{x \in [-1, 1]} |p(x)| \geq 2^{n-1}$, and that this inequality is sharp since equality holds with $p = 2^{1-n} T_n$. This has important consequences in the theory of function interpolation, since it implies that the roots of T_n (also called the *Chebyshev nodes*) are exceptionally well-suited as interpolation nodes.

Contents

1	Parametricity of polynomial operations	3
2	Missing Library Material	6
3	Chebyshev Polynomials	7
3.1	Definition	7
3.2	Relation to trigonometric functions	11
3.3	Relation to hyperbolic functions	13
3.4	Roots	13
3.5	Generating functions	15
3.6	Optimality with respect to the ∞ -norm	16
3.7	Some basic equations	17
3.8	Signs of the coefficients	24
3.9	Orthogonality and integrals	25
3.10	Clenshaw's algorithm	27

1 Parametricity of polynomial operations

```
theory Polynomial_Transfer
  imports "HOL-Computational_Algebra.Polynomial"
begin

definition rel_poly :: "('a  $\Rightarrow$  'b  $\Rightarrow$  bool)  $\Rightarrow$  'a :: zero poly  $\Rightarrow$  'b ::
zero poly  $\Rightarrow$  bool" where
  "rel_poly R p q  $\longleftrightarrow$  rel_fun (=) R (coeff p) (coeff q)"

lemma left_unique_rel_poly [transfer_rule]: "left_unique R  $\Longrightarrow$  left_unique
(rel_poly R)"
  <proof>

lemma right_unique_rel_poly [transfer_rule]: "right_unique R  $\Longrightarrow$  right_unique
(rel_poly R)"
  <proof>

lemma bi_unique_rel_poly [transfer_rule]: "bi_unique R  $\Longrightarrow$  bi_unique
(rel_poly R)"
  <proof>

lemma rel_poly_swap: "rel_poly R x y  $\longleftrightarrow$  rel_poly ( $\lambda$ y x. R x y) y x"
  <proof>

lemma coeff_transfer [transfer_rule]:
  "rel_fun (rel_poly R) (rel_fun (=) R) coeff coeff"
  <proof>

lemma map_poly_transfer:
  assumes "rel_fun R S f g" "f 0 = 0" "g 0 = 0"
  shows "rel_fun (rel_poly R) (rel_poly S) (map_poly f) (map_poly g)"
  <proof>

lemma map_poly_transfer':
  assumes "rel_fun R S f g" "rel_poly R p q" "f 0 = 0" "g 0 = 0"
  shows "rel_poly S (map_poly f p) (map_poly g q)"
  <proof>

lemma rel_poly_id: "p = q  $\Longrightarrow$  rel_poly (=) p q"
  <proof>

lemma left_total_rel_poly [transfer_rule]:
  assumes "left_total R" "right_unique R" "R 0 0"
  shows "left_total (rel_poly R)"
  <proof>
```

```

lemma right_total_rel_poly [transfer_rule]:
  assumes "right_total R" "left_unique R" "R 0 0"
  shows "right_total (rel_poly R)"
  <proof>

lemma bi_total_rel_poly [transfer_rule]:
  assumes "bi_total R" "bi_unique R" "R 0 0"
  shows "bi_total (rel_poly R)"
  <proof>

lemma zero_poly_transfer [transfer_rule]: "R 0 0  $\implies$  rel_poly R 0 0"
  <proof>

lemma one_poly_transfer [transfer_rule]: "R 0 0  $\implies$  R 1 1  $\implies$  rel_poly
R 1 1"
  <proof>

lemma pCons_transfer [transfer_rule]:
  "rel_fun R (rel_fun (rel_poly R) (rel_poly R)) pCons pCons"
  <proof>

lemma plus_poly_transfer [transfer_rule]:
  "rel_fun R (rel_fun R R) (+) (+)  $\implies$ 
  rel_fun (rel_poly R) (rel_fun (rel_poly R) (rel_poly R)) (+) (+)"
  <proof>

lemma minus_poly_transfer [transfer_rule]:
  "rel_fun R (rel_fun R R) (-) (-)  $\implies$ 
  rel_fun (rel_poly R) (rel_fun (rel_poly R) (rel_poly R)) (-) (-)"
  <proof>

lemma uminus_poly_transfer [transfer_rule]:
  "rel_fun R R uminus uminus  $\implies$  rel_fun (rel_poly R) (rel_poly R) uminus
uminus"
  <proof>

lemma smult_transfer [transfer_rule]:
  "rel_fun R (rel_fun R R) (*) (*)  $\implies$ 
  rel_fun R (rel_fun (rel_poly R) (rel_poly R)) smult smult"
  <proof>

lemma monom_transfer [transfer_rule]:
  "R 0 0  $\implies$  rel_fun R (rel_fun (=) (rel_poly R)) monom monom"
  <proof>

lemma pderiv_transfer [transfer_rule]:
  assumes "R 0 0" "rel_fun R (rel_fun R R) (+) (+)"
  shows "rel_fun (rel_poly R) (rel_poly R) pderiv pderiv"
  <proof>

```

```

lemma If_transfer':
  assumes "P = P'" "P  $\implies$  R x x'" " $\neg$ P  $\implies$  R y y'"
  shows "R (if P then x else y) (if P' then x' else y)"
  <proof>

lemma nth_transfer:
  assumes "list_all2 R xs ys" "i = j" "i < length xs"
  shows "R (xs ! i) (ys ! j)"
  <proof>

lemma Poly_transfer [transfer_rule]:
  assumes [transfer_rule]: "R 0 0" "bi_unique R"
  shows "rel_fun (list_all2 R) (rel_poly R) Poly Poly"
  <proof>

lemma poly_of_list_transfer [transfer_rule]:
  assumes [transfer_rule]: "R 0 0" "bi_unique R"
  shows "rel_fun (list_all2 R) (rel_poly R) poly_of_list poly_of_list"
  <proof>

lemma degree_transfer [transfer_rule]:
  assumes [transfer_rule]: "R 0 0" "bi_unique R"
  shows "rel_fun (rel_poly R) (=) degree degree"
  <proof>

lemma coeffs_transfer [transfer_rule]:
  assumes [transfer_rule]: "R 0 0" "bi_unique R"
  shows "rel_fun (rel_poly R) (list_all2 R) coeffs coeffs"
  <proof>

lemma times_poly_transfer [transfer_rule]:
  assumes [transfer_rule]: "rel_fun R (rel_fun R R) (+) (+)"
  "rel_fun R (rel_fun R R) (*) (*)" "R 0 0" "bi_unique
R"
  shows "rel_fun (rel_poly R) (rel_fun (rel_poly R) (rel_poly R)) (*)
(*)"
  <proof>

lemma dvd_poly_transfer [transfer_rule]:
  assumes [transfer_rule]: "rel_fun R (rel_fun R R) (+) (+)"
  "rel_fun R (rel_fun R R) (*) (*)" "R 0 0" "bi_unique
R" "bi_total R"
  shows "rel_fun (rel_poly R) (rel_fun (rel_poly R) (=)) (dvd) (dvd)"
  <proof>

lemma poly_transfer [transfer_rule]:
  assumes [transfer_rule]: "rel_fun R (rel_fun R R) (+) (+)"

```

```

"rel_fun R (rel_fun R R) (*) (*)" "R 0 0" "bi_unique
R"
shows "rel_fun (rel_poly R) (rel_fun R R) poly poly"
⟨proof⟩

lemma pcompose_transfer [transfer_rule]:
  assumes [transfer_rule]: "rel_fun R (rel_fun R R) (+) (+)"
    "rel_fun R (rel_fun R R) (*) (*)" "R 0 0" "bi_unique
R"
  shows "rel_fun (rel_poly R) (rel_fun (rel_poly R) (rel_poly R)) pcompose
pcompose"
⟨proof⟩

lemma order_0_right: "order x 0 = Least (λ_. False)"
⟨proof⟩

lemma order_poly_transfer [transfer_rule]:
  assumes [transfer_rule]:
    "rel_fun R (rel_fun R R) (+) (+)" "rel_fun R (rel_fun R R) (*) (*)"
    "rel_fun R R uminus uminus"
    "R 0 0" "R 1 1" "bi_unique R" "bi_total R" "R x y" "rel_poly R p q"
  shows "order x p = order y q"
⟨proof⟩

end

```

2 Missing Library Material

```

theory Chebyshev_Polynomials_Library
  imports "HOL-Computational_Algebra.Polynomial" "HOL-Library.FuncSet"
begin

```

The following two lemmas give a full characterisation of the *filter* function: The list *filter* *P* *xs* is the only list *ys* for which there exists a strictly increasing function $f : \{0, \dots, |ys| - 1\} \rightarrow \{0, \dots, |xs| - 1\}$ such that:

- $ys_i = xs_{f(i)}$
- $P(xs_i) \iff \exists j < n. f(j) = i$, i.e. the range of *f* are precisely the indices of the elements of *xs* that satisfy *P*.

```

lemma filterE:
  fixes P :: "'a ⇒ bool" and xs :: "'a list"
  assumes "length (filter P xs) = n"
  obtains f :: "nat ⇒ nat" where
    "strict_mono_on {..n} f"
    "∧i. i < n ⇒ f i < length xs"

```

```

    "∧i. i < n ⇒ filter P xs ! i = xs ! f i"
    "∧i. i < length xs ⇒ P (xs ! i) ↔ (∃j. j < n ∧ f j = i)"
    ⟨proof⟩

```

The following lemma shows the uniqueness of the above property. It is very useful for finding a “closed form” for `filter P xs` in some concrete situation. For example, if we know that exactly every other element of `xs` satisfies `P`, we can use it to prove that `filter P xs = map ((* 2) [0..`

```

lemma filter_eqI:
  fixes f :: "nat ⇒ nat" and xs ys :: "'a list"
  defines "n ≡ length ys"
  assumes "strict_mono_on {.. $n$ } f"
  assumes "∧i. i < n ⇒ f i < length xs"
  assumes "∧i. i < n ⇒ ys ! i = xs ! f i"
  assumes "∧i. i < length xs ⇒ P (xs ! i) ↔ (∃j. j < n ∧ f j = i)"
  shows "filter P xs = ys"
  ⟨proof⟩

```

```

lemma filter_eq_iff:
  "filter P xs = ys ↔
    (∃f. strict_mono_on {.. $\text{length ys}$ } f ∧
      (∀i<length ys. f i < length xs ∧ ys ! i = xs ! f i) ∧
      (∀i<length xs. P (xs ! i) ↔ (∃j. j < length ys ∧ f j = i)))"
  (is "?lhs = ?rhs")
  ⟨proof⟩

```

end

3 Chebyshev Polynomials

```

theory Chebyshev_Polynomials
imports
  "HOL-Analysis.Analysis"
  "HOL-Real_Asymp.Real_Asymp"
  "HOL-Computational_Algebra.Formal_Laurent_Series"
  "Polynomial_Interpolation.Ring_Hom_Poly"
  "Descartes_Sign_Rule.Descartes_Sign_Rule"
  Polynomial_Transfer
  Chebyshev_Polynomials_Library
begin

```

3.1 Definition

We choose the recursive definition of T_n and U_n and do some setup to define both of them at once.

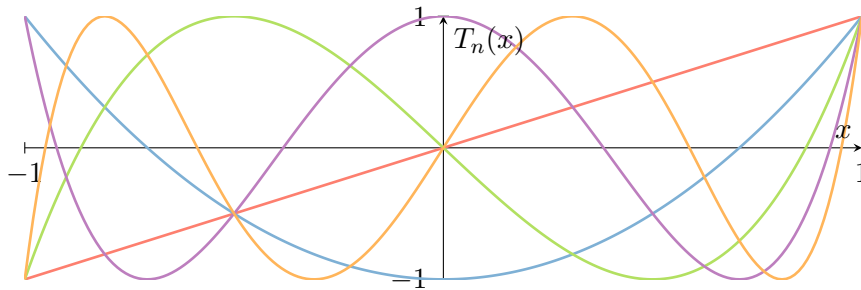


Figure 1: Some of the Chebyshev polynomials of the first kind, T_1 to T_5 .

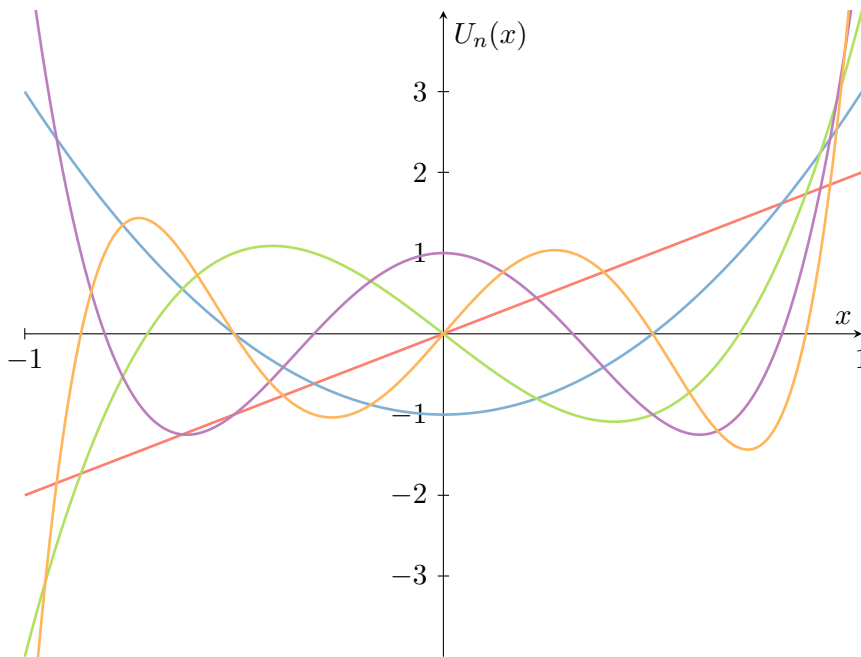


Figure 2: Some of the Chebyshev polynomials of the second kind, U_1 to U_5 .


```

locale gen_cheb_poly =
  fixes c :: "'a :: comm_ring_1"
begin

fun f :: "nat  $\Rightarrow$  'a  $\Rightarrow$  'a" where
  "f 0 x = 1"
| "f (Suc 0) x = c * x"
| "f (Suc (Suc n)) x = 2 * x * f (Suc n) x - f n x"

fun P :: "nat  $\Rightarrow$  ('a :: comm_ring_1) poly" where
  "P 0 = 1"
| "P (Suc 0) = [:0, c:]"
| "P (Suc (Suc n)) = [:0, 2:] * P (Suc n) - P n"

lemma eval [simp]: "poly (P n) x = f n x"
  <proof>

lemma eval_0:
  "f n 0 = (if odd n then 0 else (-1) ^ (n div 2))"
  <proof>

lemma eval_1 [simp]:
  "f n 1 = of_nat n * (c - 1) + 1"
  <proof>

lemma uminus [simp]: "f n (-x) = (-1) ^ n * f n x"
  <proof>

lemma pcompose_minus: "pcompose (P n) (monom (-1) 1) = (-1) ^ n * P n"
  <proof>

lemma degree_le: "degree (P n)  $\leq$  n"
  <proof>

lemma lead_coeff:
  "coeff (P n) n = (if n = 0 then 1 else c * 2 ^ (n - 1))"
  <proof>

lemma degree_eq:
  "c * 2 ^ (n - 1)  $\neq$  0  $\implies$  degree (P n :: 'a poly) = n"
  <proof>

lemmas [simp del] = f.simps(3) P.simps(3)

end

```

The two related constants *Cheb_poly* and *cheb_poly* denote the n -th Chebyshev polynomial of the first kind T_n and its interpretation as a function. We make the definition polymorphic so that it works on every commutative

ring; however, many results will only hold for rings (or even only fields) of characteristic 0.

definition `cheb_poly` :: "nat \Rightarrow 'a :: comm_ring_1 \Rightarrow 'a" where
`"cheb_poly = gen_cheb_poly.f 1"`

definition `Cheb_poly` :: "nat \Rightarrow 'a :: comm_ring_1 poly" where
`"Cheb_poly = gen_cheb_poly.P 1"`

interpretation `cheb_poly`: `gen_cheb_poly 1`
 rewrites "`gen_cheb_poly.f 1 \equiv cheb_poly`" and "`gen_cheb_poly.P 1 = Cheb_poly`"
 and " $\bigwedge x :: 'a. 1 * x = x$ "
 and " $\bigwedge n. \text{of_nat } n * (1 - 1 :: 'a) + 1 = 1$ "
 \langle proof \rangle

lemmas `cheb_poly_simps` [`code`] = `cheb_poly.f_simps`
lemmas `Cheb_poly_simps` [`code`] = `cheb_poly.P_simps`

lemma `Cheb_poly_of_int`: "`of_int_poly (Cheb_poly n) = Cheb_poly n`"
 \langle proof \rangle

lemma `degree_Cheb_poly` [`simp`]:
`"degree (Cheb_poly n :: 'a :: {idom, ring_char_0} poly) = n"`
 \langle proof \rangle

lemma `lead_coeff_Cheb_poly` [`simp`]:
`"lead_coeff (Cheb_poly n :: 'a :: {idom, ring_char_0} poly) = 2 ^ (n-1)"`
 \langle proof \rangle

lemma `Cheb_poly_nonzero` [`simp`]: "`Cheb_poly n \neq 0`"
 \langle proof \rangle

lemma `continuous_cheb_poly` [`continuous_intros`]:
 fixes `f :: "'b :: topological_space \Rightarrow 'a :: {real_normed_algebra_1, comm_ring_1}"`
 shows "`continuous_on A f \implies continuous_on A ($\lambda x. \text{cheb_poly } n (f x)$)`"
 \langle proof \rangle

Similarly, we introduce two constants for U_n .

definition `cheb_poly'` :: "nat \Rightarrow 'a :: comm_ring_1 \Rightarrow 'a" where
`"cheb_poly' = gen_cheb_poly.f 2"`

definition `Cheb_poly'` :: "nat \Rightarrow 'a :: comm_ring_1 poly" where
`"Cheb_poly' = gen_cheb_poly.P 2"`

interpretation `cheb_poly'`: `gen_cheb_poly 2`
 rewrites "`gen_cheb_poly.f 2 \equiv cheb_poly'`" and "`gen_cheb_poly.P 2 = Cheb_poly'`"
 and " $\bigwedge n. \text{of_nat } n * (2 - 1 :: 'a) + 1 = \text{of_nat } (\text{Suc } n)$ "
 \langle proof \rangle

```

lemmas cheb_poly'_simps [code] = cheb_poly'.f.simps
lemmas Cheb_poly'_simps [code] = cheb_poly'.P.simps

lemma Cheb_poly'_of_int: "of_int_poly (Cheb_poly' n) = Cheb_poly' n"
  ⟨proof⟩

lemma degree_Cheb_poly' [simp]:
  "degree (Cheb_poly' n :: 'a :: {idom, ring_char_0} poly) = n"
  ⟨proof⟩

lemma lead_coeff_Cheb_poly' [simp]:
  "lead_coeff (Cheb_poly' n :: 'a :: {idom, ring_char_0} poly) = 2 ^ n"
  ⟨proof⟩

lemma Cheb_poly_nonzero' [simp]: "Cheb_poly' n ≠ (0 :: 'a :: {comm_ring_1,
ring_char_0} poly)"
  ⟨proof⟩

lemma continuous_cheb_poly' [continuous_intros]:
  fixes f :: "'b :: topological_space ⇒ 'a :: {real_normed_algebra_1,
comm_ring_1}"
  shows "continuous_on A f ⇒ continuous_on A (λx. cheb_poly' n (f x))"
  ⟨proof⟩

```

3.2 Relation to trigonometric functions

Consider the multiple angle formulas for the cosine function:

$$\begin{aligned}
\cos 1x &= \cos x \\
\cos 2x &= 1 + 2 \cos^2 x \\
\cos 3x &= -3 \cos x + 4 \cos^3 x \\
\cos 4x &= 1 - 8 \cos^2 x + 8 \cos^4 x
\end{aligned}$$

It seems that for any $n \in \mathbb{N}$, we can write $\cos(nx)$ as a sum of powers $\cos^i x$ for $0 \leq i \leq n$, i.e. as a polynomial in $\cos x$ of degree n . It turns out that this polynomial is exactly T_n . This can also serve as an alternative, trigonometric definition of T_n .

Proving it is a simple induction:

```

lemma cheb_poly_cos [simp]:
  fixes x :: "'a :: {banach, real_normed_field}"
  shows "cheb_poly n (cos x) = cos (of_nat n * x)"
  ⟨proof⟩

```

If we look at the multiple angular formulae for the sine function, we see a

similar pattern:

$$\begin{aligned}\sin 1x &= \sin x \\ \sin 2x &= 2 \sin x \cos x \\ \sin 3x &= \sin x(-1 + 4 \cos^2 x) \\ \sin 4x &= \sin x(-4 \cos x + 8 \cos^3 x)\end{aligned}$$

It seems that $\sin nx / \sin x$ can be expressed as a polynomial in $\cos x$ of degree $n - 1$. This polynomial turns out to be exactly U_{n-1} .

lemma *cheb_poly'_cos*:

```
fixes x :: "'a :: {banach, real_normed_field}"
shows "cheb_poly' n (cos x) * sin x = sin (of_nat (n+1) * x)"
⟨proof⟩
```

lemma *cheb_poly_conv_cos*:

```
assumes "|x::real| ≤ 1"
shows "cheb_poly n x = cos (n * arccos x)"
⟨proof⟩
```

lemma *cheb_poly'_cos'*:

```
fixes x :: "'a :: {real_normed_field, banach}"
shows "sin x ≠ 0 ⇒ cheb_poly' n (cos x) = sin (of_nat (n+1) * x)
/ sin x"
⟨proof⟩
```

lemma *cheb_poly'_conv_cos*:

```
assumes "|x::real| < 1"
shows "cheb_poly' n x = sin (real (n+1) * arccos x) / sqrt (1 - x^2)"
⟨proof⟩
```

lemma *cos_multiple*:

```
fixes x :: "'a :: {banach, real_normed_field}"
shows "cos (numeral n * x) = poly (Cheb_poly (numeral n)) (cos x)"
⟨proof⟩
```

lemma *sin_multiple*:

```
fixes x :: "'a :: {banach, real_normed_field}"
shows "sin (numeral n * x) = sin x * poly (Cheb_poly' (pred_numeral
n)) (cos x)"
⟨proof⟩
```

Example application: quadruple-angle formulas for sin and cos:

lemma *cos_quadruple*:

```
fixes x :: "'a :: {banach, real_normed_field}"
shows "cos (4 * x) = 8 * cos x ^ 4 - 8 * cos x ^ 2 + 1"
```

<proof>

lemma *sin_quadruple*:

fixes *x* :: "'a :: {banach, real_normed_field}"

shows "sin (4 * x) = sin x * (8 * cos x ^ 3 - 4 * cos x)"

<proof>

3.3 Relation to hyperbolic functions

lemma *cheb_poly_cosh [simp]*:

fixes *x* :: "'a :: {banach, real_normed_field}"

shows "cheb_poly n (cosh x) = cosh (of_nat n * x)"

<proof>

lemma *cheb_poly'_cosh*:

fixes *x* :: "'a :: {real_normed_field, banach}"

shows "cheb_poly' n (cosh x) * sinh x = sinh (of_nat (n+1) * x)"

<proof>

lemma *cheb_poly_conv_cosh*:

assumes "(x :: real) ≥ 1"

shows "cheb_poly n x = cosh (n * arcosh x)"

<proof>

lemma *cheb_poly'_cosh'*:

fixes *x* :: "'a :: {real_normed_field, banach}"

shows "sinh x ≠ 0 ⇒ cheb_poly' n (cosh x) = sinh (of_nat (n+1) * x) / sinh x"

<proof>

lemma *cheb_poly'_conv_cosh*:

assumes "x > (1 :: real)"

shows "cheb_poly' n x = sinh (real (n+1) * arcosh x) / sqrt (x² - 1)"

<proof>

3.4 Roots

T_n has n distinct real roots, namely:

$$x_k = \cos \left(\frac{2k+1}{2n} \pi \right)$$

These are called the *Chebyshev nodes* of degree n .

definition *cheb_node* :: "nat ⇒ nat ⇒ real" **where**

"cheb_node n k = cos (real (2*k+1) / real (2*n) * pi)"

lemma *cheb_poly_cheb_node [simp]*:

assumes "k < n"

shows "cheb_poly n (cheb_node n k) = 0"
 ⟨proof⟩

lemma strict_antimono_cheb_node: "monotone_on {.. n } (<) (>) (cheb_node n)"
 ⟨proof⟩

lemma cheb_node_pos_iff:
assumes $k < n$
shows "cheb_node n k > 0 \longleftrightarrow $k < n \text{ div } 2$ "
 ⟨proof⟩

lemma cheb_poly_roots_bij_betw:
 "bij_betw (cheb_node n) {.. n } { x . cheb_poly n x = 0}"
 ⟨proof⟩

lemma card_cheb_poly_roots: "card { $x :: \text{real}$. cheb_poly n x = 0} = n "
 ⟨proof⟩

It is easy to see that all the Chebyshev nodes have order 1 as roots of T_n .

lemma order_Cheb_poly_cheb_node [simp]:
assumes $k < n$
shows "order (cheb_node n k) (Cheb_poly n) = 1"
 ⟨proof⟩

lemma order_Cheb_poly [simp]:
assumes "poly (Cheb_poly n) ($x :: \text{real}$) = 0"
shows "order x (Cheb_poly n) = 1"
 ⟨proof⟩

This also means that T_n is square-free. We only show this for the case where we view T_n as a real polynomial, but this also holds in every other reasonable ring since \mathbb{R} is a splitting field of T_n (as we have just shown). However, we chose not to do this here.

lemma rsquarefree_Cheb_poly_real: "rsquarefree (Cheb_poly n :: real poly)"
 ⟨proof⟩

Similarly, the n distinct real roots of U_n are:

$$y_i = \cos\left(\frac{k+1}{n+1}\pi\right)$$

definition cheb_node' :: "nat \Rightarrow nat \Rightarrow real" where
 "cheb_node' n k = cos (real (k+1) / real (n+1) * pi)"

lemma cheb_poly'_cheb_node' [simp]:
assumes $k < n$
shows "cheb_poly' n (cheb_node' n k) = 0"

<proof>

lemma *strict_antimono_cheb_node'*: "monotone_on {..*n*} (<) (>) (*cheb_node' n*)"

<proof>

lemma *cheb_node'_pos_iff*:

assumes *k*: "*k* < *n*"

shows "*cheb_node' n k* > 0 \longleftrightarrow *k* < *n* div 2"

<proof>

lemma *cheb_poly'_roots_bij_betw*:

"*bij_betw* (*cheb_node' n*) {..*n*} {*x*. *cheb_poly' n x* = 0}"

<proof>

lemma *card_cheb_poly'_roots*: "card {*x*::real. *cheb_poly' n x* = 0} = *n*"

<proof>

lemma *order_Cheb_poly'_cheb_node'* [*simp*]:

assumes "*k* < *n*"

shows "*order* (*cheb_node' n k*) (*Cheb_poly' n*) = 1"

<proof>

lemma *order_Cheb_poly'* [*simp*]:

assumes "*poly* (*Cheb_poly' n*) (*x* :: real) = 0"

shows "*order x* (*Cheb_poly' n*) = 1"

<proof>

lemma *rsquarefree_Cheb_poly'_real*: "*rsquarefree* (*Cheb_poly' n* :: real *poly*)"

<proof>

3.5 Generating functions

T_n and U_n have the following rational generating functions:

$$\sum_{n=0}^{\infty} T_n(x)t^n = \frac{1-tx}{1-2tx+t^2} \quad \sum_{n=0}^{\infty} U_n(x)t^n = \frac{1}{1-2tx+t^2}$$

This is a simple consequence of the linear recurrence equations they satisfy (which we used as their definitions).

Due to some limitations coming from the type class structure, we cannot currently write this down nicely as an equation, but the following form is almost as good.

theorem *Abs_fps_Cheb_poly*:

fixes *F X T* :: "real *fps fps*"

defines "*X* \equiv *fps_const fps_X*" and "*T* \equiv *fps_X*"

defines "*F* \equiv *Abs_fps* (*fps_of_poly* \circ *Cheb_poly*)"

shows $"F * (1 - 2 * T * X + T^2) = 1 - T * X"$
 ⟨proof⟩

theorem *Abs_fps_Cheb_poly'*:
fixes $F X T :: "real\ fps\ fps"$
defines $"X \equiv fps_const\ fps_X"$ and $"T \equiv fps_X"$
defines $"F \equiv Abs_fps\ (fps_of_poly \circ Cheb_poly)'"$
shows $"F * (1 - 2 * T * X + T^2) = 1"$
 ⟨proof⟩

3.6 Optimality with respect to the ∞ -norm

We now turn towards a property of T_n that explains why they are interesting for interpolating smooth functions. If $f : [0, 1] \rightarrow \mathbb{R}$ is a smooth function on the unit interval, the approximation error attained when interpolating f with a polynomial P of degree n at the interpolation points x_1, \dots, x_n is

$$\frac{f^{(n)}(\xi)}{n!} \prod_{i=1}^n (x - x_i) .$$

Therefore, it makes sense to choose the interpolation points such that $\prod_{i=1}^n (x - x_i)$ is minimal.

We will show below results that imply that this product cannot be smaller than 2^{1-n} , and it is easy to see that if we choose x_i to be the Chebyshev nodes then the product becomes exactly 2^{1-n} and thus optimal.

Our first result is now the following: The ∞ -norm of a monic polynomial of degree n on the unit interval $[-1, 1]$ is at least 2^{1-n} . This gives us a kind of lower bound on the “oscillation” of polynomials: a monic polynomial of degree n cannot stay closer than 2^{1-n} to 0 at every point of the unit interval.

lemma *Sup_abs_poly_bound_aux*:

fixes $p :: "real\ poly"$
assumes $"lead_coeff\ p = 1"$
shows $"\exists x \in \{-1..1\}. |poly\ p\ x| \geq 1 / 2 \wedge (degree\ p - 1)"$
 ⟨proof⟩

lemma *Sup_abs_poly_bound_unit_ivl*:

fixes $p :: "real\ poly"$
shows $"(SUP\ x \in \{-1..1\}. |poly\ p\ x|) \geq |lead_coeff\ p| / 2 \wedge (degree\ p - 1)"$
 ⟨proof⟩

Using an appropriate change of variables, we obtain the following bound in the most general form for a non-constant polynomial $P(x)$ on some non-empty interval $[a, b]$:

$$\sup_{x \in [a, b]} |P(x)| \geq 2 \cdot lc(p) \cdot \left(\frac{b - a}{4} \right)^{\deg(p)}$$

where $\text{lc}(p)$ denotes the leading coefficient of p .

theorem *Sup_abs_poly_bound*:

```

  fixes p :: "real poly"
  assumes "a < b" and "degree p > 0"
  shows "(SUP x∈{a..b}. |poly p x|) ≥ 2 * |lead_coeff p| * ((b - a)
/ 4) ^ degree p"
⟨proof⟩

```

If we scale T_n with a factor of 2^{1-n} , it exactly attains the lower bound we just derived. The Chebyshev polynomials of the first kind are, in that sense, the polynomials that stay closest to 0 within the unit interval.

With some more work (that we will not do), one can see that T_n is in fact the *only* polynomial that attains this minimal deviation (see e.g. Corollary 3.4B in Mason & Handscomb [1]). This fact, however, requires proving the Equioscillation Theorem, which is not so easy and beyond the scope of this entry.

lemma *abs_cheb_poly_le_1*:

```

  assumes "(x :: real) ∈ {-1..1}"
  shows "|cheb_poly n x| ≤ 1"
⟨proof⟩

```

theorem *Sup_abs_poly_bound_sharp*:

```

  fixes n :: nat and p :: "real poly"
  defines "p ≡ smult (1 / 2 ^ (n - 1)) (Cheb_poly n)"
  shows "degree p = n" and "lead_coeff p = 1"
  and "(SUP x∈{-1..1}. |poly p x|) = 1 / 2 ^ (n - 1)"
⟨proof⟩

```

A related fact: among all the real polynomials of degree n whose absolute value is bounded by 1 within the unit interval, T_n is the one that grows fastest *outside* the unit interval.

theorem *cheb_poly_fastest_growth*:

```

  fixes p :: "real poly"
  defines "n ≡ degree p"
  assumes p_bounded: "∧x. |x| ≤ 1 ⇒ |poly p x| ≤ 1"
  assumes x: "x ∉ {-1<.. $1$ }"
  shows "|cheb_poly n x| ≥ |poly p x|"
⟨proof⟩

```

3.7 Some basic equations

We first set up a mechanism to allow us to prove facts about Chebyshev polynomials on any ring with characteristic 0 by proving them for Chebyshev polynomials over \mathbb{R} .

definition *rel_ring_int* :: "'a :: ring_1 ⇒ 'b :: ring_1 ⇒ bool" where
rel_ring_int x y $\longleftrightarrow (\exists n::int. x = \text{of_int } n \wedge y = \text{of_int } n)$

```

lemma rel_ring_int_0: "rel_ring_int 0 0"
  ⟨proof⟩

lemma rel_ring_int_1: "rel_ring_int 1 1"
  ⟨proof⟩

lemma rel_ring_int_add:
  "rel_fun rel_ring_int (rel_fun rel_ring_int rel_ring_int) (+) (+)"
  ⟨proof⟩

lemma rel_ring_int_mult:
  "rel_fun rel_ring_int (rel_fun rel_ring_int rel_ring_int) (*) (*)"
  ⟨proof⟩

lemma rel_ring_int_minus:
  "rel_fun rel_ring_int (rel_fun rel_ring_int rel_ring_int) (-) (-)"
  ⟨proof⟩

lemma rel_ring_int_uminus:
  "rel_fun rel_ring_int rel_ring_int uminus uminus"
  ⟨proof⟩

lemma sgn_of_int: "sgn (of_int n :: 'a :: linordered_idom) = of_int (sgn
n)"
  ⟨proof⟩

lemma rel_ring_int_sgn:
  "rel_fun rel_ring_int (rel_ring_int :: 'a :: linordered_idom ⇒ 'b ::
linordered_idom ⇒ bool) sgn sgn"
  ⟨proof⟩

lemma bi_unique_rel_ring_int:
  "bi_unique (rel_ring_int :: 'a :: ring_char_0 ⇒ 'b :: ring_char_0 ⇒
bool)"
  ⟨proof⟩

lemmas rel_ring_int_transfer =
  rel_ring_int_0 rel_ring_int_1 rel_ring_int_add rel_ring_int_mult rel_ring_int_minus
  rel_ring_int_uminus bi_unique_rel_ring_int

lemma rel_poly_rel_ring_int:
  "rel_poly rel_ring_int p q ⟷ (∃ r. p = of_int_poly r ∧ q = of_int_poly
r)"
  ⟨proof⟩

lemma Cheb_poly_transfer:
  "rel_fun (=) (rel_poly rel_ring_int) Cheb_poly Cheb_poly"
  ⟨proof⟩

```

```

lemma Cheb_poly'_transfer:
  "rel_fun (=) (rel_poly rel_ring_int) Cheb_poly' Cheb_poly'"
  <proof>

```

context

```

  fixes T :: "'a :: {idom, ring_char_0} itself"
  notes [transfer_rule] = rel_ring_int_transfer [where ?'a = real and
?'b = 'a]
                                     Cheb_poly_transfer[where ?'a = real and ?'b
= 'a]
                                     Cheb_poly'_transfer[where ?'a = real and ?'b
= 'a]
                                     transfer_rule_of_nat transfer_rule_numeral
begin

```

The following rule allows us to prove an equality of real polynomials $P = Q$ by proving that $P(\cos x) = Q(\cos x)$ for all $x \in (0, \alpha)$ for some $\alpha > 0$. This holds because there are infinitely many such $\cos x$, but $P - Q$, being a polynomial, can only have finitely many roots if $P \neq 0$.

```

lemma Cheb_poly_equalities_aux:
  fixes p q :: "real poly"
  assumes "a > 0"
  assumes "\x. x \in {0 <..<a} \implies poly p (cos x) = poly q (cos x)"
  shows "p = q"
  <proof>

```

First, we show that $T_n(x) = nU_{n-1}(x)$:

```

lemma pderiv_Cheb_poly: "pderiv (Cheb_poly n) = of_nat n * (Cheb_poly'
(n - 1) :: 'a poly)"
  <proof>

```

Next, we show that:

$$U'_n(x) = \frac{1}{x^2 - 1}((n + 1)T_{n+1}(x) - xU_n(x))$$

```

lemma pderiv_Cheb_poly':
  "pderiv (Cheb_poly' n) * [:-1, 0, 1 :: 'a:] =
  of_nat (n+1) * Cheb_poly (n+1) - [:0,1:] * Cheb_poly' n"
  <proof>

```

Next, we have $T_n(x) = \frac{1}{2}(U_n(x) - U_{n-2}(x))$.

```

lemma Cheb_poly_rec:
  assumes n: "n \ge 2"
  shows "2 * Cheb_poly n = Cheb_poly' n - (Cheb_poly' (n - 2) :: 'a poly)"
  <proof>

```

```

lemma cheb_poly_rec:
  assumes n: "n ≥ 2"
  shows "2 * cheb_poly n x = cheb_poly' n x - cheb_poly' (n - 2) (x::'a)"
  ⟨proof⟩

```

Next, we have $U_n(x) = xU_{n-1}(x) + T_n(x)$.

```

lemma Cheb_poly'_rec:
  assumes n: "n > 0"
  shows "Cheb_poly' n = [:0,1::'a:] * Cheb_poly' (n - 1) + Cheb_poly
n"
  ⟨proof⟩

```

```

lemma cheb_poly'_rec:
  assumes n: "n > 0"
  shows "cheb_poly' n x = x * cheb_poly' (n-1) x + cheb_poly n (x::'a)"
  ⟨proof⟩

```

Next, $T_n(x) = xT_{n-1}(x) + (x^2 - 1)U_{n-2}(x)$.

```

lemma Cheb_poly_rec':
  assumes n: "n ≥ 2"
  shows "Cheb_poly n = [:0,1::'a:] * Cheb_poly (n-1) + [-1,0,1:] * Cheb_poly'
(n-2)"
  ⟨proof⟩

```

```

lemma cheb_poly_rec':
  assumes n: "n ≥ 2"
  shows "cheb_poly n x = x * cheb_poly (n-1) x + (x^2 - 1) * cheb_poly'
(n-2) (x::'a)"
  ⟨proof⟩

```

T_n and U_{-1} are a solution to a Pell-like equation on polynomials:

$$T_n(x)^2 + (1 - x^2)U_{n-1}(x)^2 = 1$$

```

lemma Cheb_poly_Pell:
  assumes n: "n > 0"
  shows "Cheb_poly n ^ 2 + [:1, 0, -1::'a:] * Cheb_poly' (n - 1) ^ 2
= 1"
  ⟨proof⟩

```

```

lemma cheb_poly_Pell:
  assumes n: "n > 0"
  shows "cheb_poly n x ^ 2 + (1 - x^2) * cheb_poly' (n-1) x ^ 2 = (1 ::
'a)"
  ⟨proof⟩

```

The following Turán-style equation also holds:

$$T_{n+1}(x)^2 - T_{n+2}(x)T_n(x) = 1 - x^2$$

lemma *Cheb_poly_Turan*:
 "Cheb_poly (n+1) ^ 2 - Cheb_poly (n+2) * Cheb_poly n = [:1,0,-1::'a:]"
 ⟨proof⟩

lemma *cheb_poly_Turan*:
 "cheb_poly (n+1) x ^ 2 - cheb_poly (n+2) x * cheb_poly n x = (1 - x
 ^ 2)::'a)"
 ⟨proof⟩

And, the analogous one for U_n :

$$U_{n+1}(x)^2 - U_{n+2}(x)U_n(x) = 1$$

lemma *Cheb_poly'_Turan*:
 "Cheb_poly' (n+1) ^ 2 - Cheb_poly' (n+2) * Cheb_poly' n = (1 :: 'a
 poly)"
 ⟨proof⟩

lemma *cheb_poly'_Turan*:
 "cheb_poly' (n+1) x ^ 2 - cheb_poly' (n+2) x * cheb_poly' n x = (1
 :: 'a)"
 ⟨proof⟩

There is also a nice formula for the product of two Chebyshev polynomials of the first kind:

$$T_m(x)T_n(x) = \frac{1}{2}(T_{m+n}(x) + T_{m-n}(x))$$

lemma *Cheb_poly_prod*:
 assumes "n ≤ m"
 shows "2 * Cheb_poly m * Cheb_poly n = Cheb_poly (m + n) + (Cheb_poly
 (m - n) :: 'a poly)"
 ⟨proof⟩

lemma *cheb_poly_prod'*:
 assumes "n ≤ m"
 shows "2 * cheb_poly m x * cheb_poly n x = cheb_poly (m + n) x + cheb_poly
 (m - n) (x :: 'a)"
 ⟨proof⟩

In particular, this leads to a divide-and-conquer-style recurrence relation for T_n for even and odd n :

$$\begin{aligned} T_{2n}(x) &= 2T_n(x)^2 - 1 \\ T_{2n+1}(x) &= 2T_n(x)T_{n+1}(x) - x \end{aligned}$$

lemma *Cheb_poly_even*:
 "Cheb_poly (2 * n) = 2 * Cheb_poly n ^ 2 - (1 :: 'a poly)"

<proof>

lemma *cheb_poly_even*:

"*cheb_poly* (2 * n) x = 2 * *cheb_poly* n x ^ 2 - (1 :: 'a)"
<proof>

lemma *Cheb_poly_odd*:

"*Cheb_poly* (2 * n + 1) = 2 * *Cheb_poly* n * *Cheb_poly* (Suc n) - [:0,1::'a:]"
<proof>

lemma *cheb_poly_odd*:

"*cheb_poly* (2 * n + 1) x = 2 * *cheb_poly* n x * *cheb_poly* (Suc n) x -
(x :: 'a)"
<proof>

Remarkably, we also have the following formula for the composition of two Chebyshev polynomials of the first kind:

$$T_{mn}(x) = T_m(T_n(x))$$

theorem *Cheb_poly_mult*:

"(*Cheb_poly* (m * n) :: 'a poly) = *pcompose* (*Cheb_poly* m) (*Cheb_poly* n)"
<proof>

corollary *cheb_poly_mult*: "*cheb_poly* m (*cheb_poly* n x) = *cheb_poly* (m * n) (x :: 'a)"

<proof>

For the Chebyshev polynomials of the second kind, the following more complicated relationship holds:

$$U_{mn-1}(x) = U_{m-1}(T_n(x)) \cdot U_{n-1}(x)$$

theorem *Cheb_poly'_mult*:

assumes "m > 0" "n > 0"

shows "*Cheb_poly'* (m * n - 1) :: 'a poly) =
pcompose (*Cheb_poly'* (m-1)) (*Cheb_poly* n) * *Cheb_poly'* (n-1)"

<proof>

lemma *cheb_poly'_mult*:

assumes "m > 0" "n > 0"

shows "*cheb_poly'* (m * n - 1) (x :: 'a) =
cheb_poly' (m-1) (*cheb_poly* n x) * *cheb_poly'* (n-1) x"

<proof>

The following two lemmas tell us that

$$U'_n(1) = 2 \binom{n+2}{3} = \frac{1}{3}n(n+1)(n+2)$$

$$U'_n(-1) = (-1)^{n+1} 2 \binom{n+2}{3} = \frac{(-1)^{n+1}}{3}n(n+1)(n+2)$$

This is good to know because our formula for U'_n has a “division by zero” at ± 1 , so we cannot use it to establish these values.

lemma *poly_pderiv_Cheb_poly'_1*:

```
"3 * poly (pderiv (Cheb_poly' n) :: 'a poly) 1 = of_nat ((n + 2) * (n + 1) * n)"
⟨proof⟩
```

lemma *poly_pderiv_Cheb_poly'_neg_1*:

```
"3 * poly (pderiv (Cheb_poly' n) :: 'a poly) (-1) = (-1) ^ Suc n * of_nat ((n + 2) * (n + 1) * n)"
⟨proof⟩
```

Another alternative definition of T_n and U_n is as the solutions of the ordinary differential equations

$$(1 - x^2)T''_n - xT'_n + n^2T_n = 0$$

$$(1 - x^2)U''_n - 3xU'_n + n(n+2)U_n = 0$$

lemma *Cheb_poly_ODE*:

```
fixes n :: nat
defines "p ≡ (Cheb_poly n :: 'a poly)"
shows "[1,0,-1:] * (pderiv ^^ 2) p - [0,1:] * pderiv p + of_nat n ^ 2 * p = 0"
⟨proof⟩
```

lemma *Cheb_poly'_ODE*:

```
fixes n :: nat
defines "p ≡ (Cheb_poly' n :: 'a poly)"
shows "[1,0,-1:] * (pderiv ^^ 2) p - [0,3:] * pderiv p + of_nat (n*(n+2)) * p = 0"
⟨proof⟩
```

end

lemma *cheb_poly_prod*:

```
fixes x :: "'a :: field_char_0"
assumes "n ≤ m"
shows "cheb_poly m x * cheb_poly n x = (cheb_poly (m + n) x + cheb_poly (m - n) x) / 2"
⟨proof⟩
```

```

lemma has_field_derivative_cheb_poly [derivative_intros]:
  assumes "(f has_field_derivative f') (at x within A)"
  shows   "((λx. cheb_poly n (f x)) has_field_derivative
            (of_nat n * cheb_poly' (n- 1) (f x) * f')) (at x within
A)"
  <proof>

```

```

lemma has_field_derivative_cheb_poly' [derivative_intros]:
  "(cheb_poly' n has_field_derivative
    (if x = 1 then of_nat ((n + 2) * (n + 1) * n) / 3
     else if x = -1 then (-1)^Suc n * of_nat ((n + 2) * (n + 1) * n)
 / 3
    else (of_nat (n+1) * cheb_poly (Suc n) x - x * cheb_poly' n x) /
(x2 - 1)))
  (at x within A)" (is "(_ has_field_derivative ?f') (at _ within _)")
<proof>

```

```

lemmas has_field_derivative_cheb_poly'' [derivative_intros] =
  DERIV_chain' [OF _ has_field_derivative_cheb_poly']

```

3.8 Signs of the coefficients

Since $T_n(-x) = (-1)^n T_n(x)$ and analogously for U_n , the Chebyshev polynomials are even functions when n is even and odd functions when n is odd. Consequently, when n is even, the coefficients of X^k for any odd k are 0 and analogously when n is odd.

```

lemma coeff_Cheb_poly_eq_0:
  assumes "odd (n + k)"
  shows   "coeff (Cheb_poly n :: 'a :: {idom,ring_char_0} poly) k = 0"
  <proof>

```

```

lemma coeff_Cheb_poly'_eq_0:
  assumes "odd (n + k)"
  shows   "coeff (Cheb_poly' n :: 'a :: {idom,ring_char_0} poly) k = 0"
  <proof>

```

Next, we analyse the behaviour of the signs of the coefficients of T_n and U_n more generally and show that:

- The leading coefficient is positive.
- After that, every second coefficient is 0.
- The remaining coefficients are non-zero and their signs alternate.

In conclusion, we have

$$\operatorname{sgn}([X^k] T_n(X)) = \operatorname{sgn}([X^k] U_n(X)) = \begin{cases} 0 & \text{if } k > n \text{ or } (n+k) \text{ is odd} \\ (-1)^{\frac{n-k}{2}} & \text{otherwise} \end{cases}$$

The proof works using Descartes' rule of signs: We know that T_n and U_n have n distinct real roots and $\lfloor \frac{n}{2} \rfloor$ of them are positive. By Descartes' rule of signs, this implies that the coefficient sequences of T_n and U_n must have at least $\lfloor \frac{n}{2} \rfloor$ sign alternations. However, we also already know that every other coefficient of T_n and U_n starting with $[X^{n-1}]$ is 0, so the number of sign alternations must be *exactly* $\lfloor \frac{n}{2} \rfloor$.

```
lemma sgn_coeff_Cheb_poly_aux:
  fixes n :: nat and P :: "real poly"
  assumes "degree P = n"
  assumes "\i. odd (n + i) ==> coeff P i = 0"
  assumes "card {x. x > 0 ^ poly P x = 0} = n div 2"
  assumes "rsquarefree P"
  assumes "coeff P n > 0"
  shows "sgn (coeff P i) = (if i > n \vee odd (n + i) then 0 else (-1) ^
((n - i) div 2))"
<proof>
```

```
theorem sgn_coeff_Cheb_poly:
  "sgn (coeff (Cheb_poly n) i :: 'a :: linordered_idom) =
  (if i > n \vee odd (n + i) then 0 else (-1) ^ ((n - i) div 2))"
<proof>
```

```
theorem sgn_coeff_Cheb_poly':
  "sgn (coeff (Cheb_poly' n) i :: 'a :: linordered_idom) =
  (if i > n \vee odd (n + i) then 0 else (-1) ^ ((n - i) div 2))"
<proof>
```

3.9 Orthogonality and integrals

```
lemma cis_eq_1_iff: "cis x = 1 <==> (\exists n. x = 2 * pi * real_of_int n)"
<proof>
```

context

```
  fixes n :: nat and x :: "nat => real"
  defines "x \equiv (\lambda k. cos (real (Suc (2 * k)) / real (2 * n) * pi))"
begin
```

```
lemma cheb_poly_orthogonality_discrete_aux:
  assumes "1 \in {0<..<2*n}"
  shows "(sum k<n. cos (real 1 * real (Suc (2 * k)) / real (2 * n) * pi))
= 0"
```

<proof>

For $k = 0, \dots, n-1$ let $x_k = \cos(\frac{2k+1}{2n}\pi)$ be the Chebyshev nodes of order n , i.e. the roots of T_n . Then the following discrete orthogonality relation holds for the Chebyshev polynomials of the first kind (for any $i, j < n$):

$$\sum_{k=0}^{n-1} T_i(x_k)T_j(x_k) = \begin{cases} n & \text{if } i = j = 0 \\ \frac{n}{2} & \text{if } i = j \neq 0 \\ 0 & \text{if } i \neq j \end{cases}$$

theorem *cheb_poly_orthogonality_discrete:*

fixes $i\ j :: \text{nat}$

assumes " $i < n$ " " $j < n$ "

shows " $(\sum_{k=0}^{n-1} \text{cheb_poly } i (x\ k) * \text{cheb_poly } j (x\ k)) =$
(if $i = j$ *then* *if* $i = 0$ *then* n *else* $n / 2$ *else* 0 *)"*

<proof>

A similar relation holds for the Chebyshev polynomials of the second kind:

$$\sum_{k=0}^{n-1} U_i(x_k)U_j(x_k)(1 - x_k^2) = \begin{cases} n & \text{if } i = j = n - 1 \\ \frac{n}{2} & \text{if } i = j \neq n - 1 \\ 0 & \text{if } i \neq j \end{cases}$$

theorem *cheb_poly'_orthogonality_discrete:*

fixes $i\ j :: \text{nat}$

assumes " $i < n$ " " $j < n$ "

shows " $(\sum_{k=0}^{n-1} \text{cheb_poly}'\ i (x\ k) * \text{cheb_poly}'\ j (x\ k) * (1 - x_k^2)) =$

(if $i = j$ *then* *if* $i = n - 1$ *then* n *else* $n / 2$ *else* 0 *)"*

<proof>

end

We now show the continuous orthogonality relations. For the polynomials of the first kind, the relation is:

$$\int_{-1}^1 \frac{T_m(x)T_n(x)}{\sqrt{1-x^2}} dx = \begin{cases} \pi & \text{if } m = n = 0 \\ \frac{\pi}{2} & \text{if } m = n \neq 0 \\ 0 & \text{if } m \neq n \end{cases}$$

The proof works by a change of variables $x = \cos \theta$, which converts the integral to the easier form $\int_0^\pi \cos(mt) \cos(nt) dx$, which can then be solved by a computing an indefinite integral (with appropriate case distinctions on m and n).

theorem *cheb_poly_orthogonality:*

```

fixes m n :: nat
defines "I ≡ if m = n then if m = 0 then pi else pi / 2 else 0"
shows "((λx. cheb_poly m x * cheb_poly n x / sqrt (1 - x2)) has_integral
I) {-1..1}"
⟨proof⟩

```

For the polynomials of the second kind, the relation is:

$$\int_{-1}^1 U_m(x)U_n(x)\sqrt{1-x^2} dx = \begin{cases} \frac{\pi}{2} & \text{if } m = n \\ 0 & \text{if } m \neq n \end{cases}$$

The proof works the same as before.

```

theorem cheb_poly'_orthogonality:
  fixes m n :: nat
  defines "I ≡ if m = n then pi / 2 else 0"
  shows "((λx. cheb_poly' m x * cheb_poly' n x * sqrt (1 - x2)) has_integral
I) {-1..1}"
⟨proof⟩

```

We additionally show the following property about the integral from -1 to 1:

$$\int_{-1}^1 T_n(x) dx = \frac{1 + (-1)^n}{1 - n^2}$$

```

theorem cheb_poly_integral_neg1_1:
  "(cheb_poly n has_integral ((1 + (-1)n) / (1 - n2))) {-1..1::real}"
⟨proof⟩

```

And, for the polynomials of the second kind:

$$\int_{-1}^1 U_n(x) dx = \frac{1 + (-1)^n}{n + 1}$$

```

theorem cheb_poly'_integral_neg1_1:
  "(cheb_poly' n has_integral (1 + (-1)n / (n+1)) {-1..1::real}"
⟨proof⟩

```

3.10 Clenshaw's algorithm

Clenshaw's algorithm allows us to efficiently evaluate a weighted sum of Chebyshev polynomials of the first kind, i.e.

$$\sum_{i=0}^n w_i \cdot T_i(x) .$$

This is useful when evaluating interpolations.

```

locale clenshaw =
  fixes g :: "nat  $\Rightarrow$  'a :: comm_ring_1"
  fixes a b :: "nat  $\Rightarrow$  'a"
  assumes g_rec: " $\bigwedge n. g (Suc (Suc n)) = a n * g (Suc n) + b n * g n$ "
begin

context
  fixes N :: nat and c :: "nat  $\Rightarrow$  'a"
begin

function clenshaw_aux where
  "n  $\geq$  N  $\implies$  clenshaw_aux n = 0"
| "n < N  $\implies$  clenshaw_aux n =
  c (Suc n) + a n * clenshaw_aux (n+1) + b (Suc n) * clenshaw_aux (n+2)"
  <proof>
termination <proof>

lemma clenshaw_aux_correct_aux:
  assumes "n  $\leq$  N"
  shows "g n * c n + g (Suc n) * clenshaw_aux n + b n * g n * clenshaw_aux
(Suc n) = ( $\sum_{k=n..N. c k * g k}$ )"
  <proof>

fun clenshaw_aux' where
  "clenshaw_aux' 0 acc1 acc2 = g 0 * c 0 + g 1 * acc1 + b 0 * g 0 * acc2"
| "clenshaw_aux' (Suc n) acc1 acc2 = clenshaw_aux' n (c (Suc n) + a n
* acc1 + b (Suc n) * acc2) acc1"

lemma clenshaw_aux'_correct: "clenshaw_aux' N 0 0 = ( $\sum_{k \leq N. c k * g k}$ )"
  <proof>

lemmas [simp del] = clenshaw_aux'.simps

end

lemma clenshaw_aux'_cong:
  " $(\bigwedge k. k \leq n \implies c k = c' k) \implies clenshaw\_aux' c n acc1 acc2 = clenshaw\_aux' c' n acc1 acc2$ "
  <proof>

definition clenshaw where "clenshaw N c = clenshaw_aux' c N 0 0"

theorem clenshaw_correct: "clenshaw N c = ( $\sum_{k \leq N. c k * g k}$ )"
  <proof>

end

```

```

definition cheb_eval :: "'a :: comm_ring_1 list ⇒ 'a ⇒ 'a" where
  "cheb_eval cs x = (∑ k<length cs. cs ! k * cheb_poly k x)"

interpretation cheb_poly: clenshaw "λn. cheb_poly n x" "λ_. 2 * x" "λ_.
-1"
  ⟨proof⟩

fun cheb_eval_aux where
  "cheb_eval_aux 0 cs x acc1 acc2 = hd cs + x * acc1 - acc2"
| "cheb_eval_aux (Suc n) cs x acc1 acc2 =
  cheb_eval_aux n (tl cs) x (hd cs + 2 * x * acc1 - acc2) acc1"

lemma cheb_eval_aux_altdef:
  "length cs = Suc n ⇒
  cheb_eval_aux n cs x acc1 acc2 =
  cheb_poly.clenshaw_aux' x (λk. rev cs ! k) n acc1 acc2"
  ⟨proof⟩

lemmas [simp del] = cheb_eval_aux.simps

lemma cheb_eval_code [code]:
  "cheb_eval [] x = 0"
  "cheb_eval [c] x = c"
  "cheb_eval (c1 # c2 # cs) x =
  cheb_eval_aux (Suc (length cs)) (rev (c1 # c2 # cs)) x 0 0"
  ⟨proof⟩

end

```

References

- [1] J. Mason and D. Handscomb. *Chebyshev Polynomials*. CRC Press, 2002.