Catalan Numbers

Manuel Eberl

March 17, 2025

Abstract

In this work, we define the Catalan numbers C_n and prove several equivalent definitions (including some closed-form formulae). We also show one of their applications (counting the number of binary trees of size n), prove the asymptotic growth approximation $C_n \sim \frac{4^n}{\sqrt{\pi} n^{1.5}}$, and provide reasonably efficient executable code to compute them.

The derivation of the closed-form formulae uses algebraic manipulations of the ordinary generating function of the Catalan numbers, and the asymptotic approximation is then done using generalised binomial coefficients and the Gamma function. Thanks to these highly non-elementary mathematical tools, the proofs are very short and simple.

Contents

1 Catalan numbers

theory *Catalan-Auxiliary-Integral*
imports *HOL−Analysis.Analysis HOL−Real-Asymp.Real-Asymp*
begin

1.1 Auxiliary integral

First, we will prove the integral

$$\int_0^4 \sqrt{\frac{4-x}{x}}\,\mathrm{d}x = 2\pi$$

which occurs in the proof for the integral formula for the Catalan numbers.

context
begin

We prove the integral by explicitly constructing the indefinite integral.

lemma *catalan-aux-integral*:
((λ*x*::*real*. *sqrt* ((*4* − *x*) / *x*)) *has-integral 2* ∗ *pi*) {*0*..*4*}
proof −
define *F* **where** *F* ≡ λ*x*. *sqrt* (*4*/*x* − *1*) ∗ *x* − *2* ∗ *arctan* ((*sqrt* (*4*/*x* − *1*) ∗ (*x* − *2*)) / (*x* − *4*))
— The nice part of the indefinite integral. The endpoints are removable singularities.

define *G* **where** *G* ≡ λ*x*. *if x* = *4 then pi else if x* = *0 then* −*pi else F x*
— The actual indefinite integral including the endpoints.

— We now prove that the indefinite integral indeed tends to *pi* resp. − *pi* at the edges of the integration interval.
have (*F* ⟶ −*pi*) (*at-right 0*)
unfolding *F-def* **by** *real-asymp*

hence *G-0*: (*G* ⟶ −*pi*) (*at-right 0*) **unfolding** *G-def*
by (*rule Lim-transform-eventually*) (*auto intro*!: *eventually-at-rightI*[*of 0 1*])

have (*F* ⟶ *pi*) (*at-left 4*)
unfolding *F-def* **by** *real-asymp*
hence *G-4*: (*G* ⟶ *pi*) (*at-left 4*) **unfolding** *G-def*
by (*rule Lim-transform-eventually*) (*auto intro*!: *eventually-at-leftI*[*of 1*])

— The derivative of *G* is indeed the integrand in the interior of the integration interval.
have *deriv-G*: (*G has-field-derivative sqrt* ((*4* − *x*) / *x*)) (*at x*) **if** *x*: *x* ∈ {*0*<..<*4*} **for** *x*
proof −
from *x* **have** *eventually* (λ*y*. *y* ∈ {*0*<..<*4*}) (*nhds x*)

by (intro eventually-nhds-in-open) simp-all
hence eq: eventually (λx. F x = G x) (nhds x) **by** eventually-elim (simp add: G-def)

define T **where** T ≡ λx::real. 4 / (sqrt (4/x − 1) * (x − 4) * x^2)
have *: ((λx. (sqrt (4/x − 1) * (x − 2)) / (x − 4)) has-field-derivative T x) (at x)
by (insert x, rule derivative-eq-intros refl | simp)+
(simp add: divide-simps T-def, simp add: field-simps power2-eq-square)
have ((λx. 2 * arctan ((sqrt (4/x − 1) * (x − 2)) / (x − 4))) has-field-derivative

2 * T x / (1 + (sqrt (4 / x − 1) * (x − 2) / (x − 4))^2)) (at x)
by (rule * derivative-eq-intros refl | simp)+ (simp add: field-simps)
also from x **have** (sqrt (4 / x − 1) * (x − 2) / (x − 4))^2 = (4/x − 1) * (x−2)^2 / (x − 4)^2
by (simp add: power-mult-distrib power-divide)
also from x **have** 1 + ... = −4 / ((x − 4) * x)
by (simp add: divide-simps power2-eq-square mult-ac) (simp add: algebra-simps)?
also from x **have** 2 * T x / ... = − (2 / (x * sqrt (4 / x − 1)))
by (simp add: T-def power2-eq-square)
finally have *: ((λx. 2 * arctan (sqrt (4 / x − 1) * (x − 2) / (x − 4))) has-real-derivative
− (2 / (x * sqrt (4 / x − 1)))) (at x) **.**
have (F has-field-derivative sqrt (4 / x − 1)) (at x) **unfolding** F-def
by (insert x, (rule * derivative-eq-intros refl | simp)+) (simp add: field-simps)
thus ?thesis **by** (subst (asm) DERIV-cong-ev[OF refl eq refl]) (insert x, simp add: field-simps)
qed

— It is now obvious that G is continuous over the entire integration interval.
have cont-G: continuous-on {0..4} G **unfolding** continuous-on
proof safe
fix x :: real **assume** x ∈ {0..4}
then consider x = 0 | x = 4 | x ∈ {0<..<4} **by** force
thus (G ⟶ G x) (at x within {0..4})
proof cases
assume x = 0
have *: at (0::real) within {0..4} ≤ at-right 0 **unfolding** at-within-def
by (rule inf-mono) auto
from G-0 **have** (G ⟶ −pi) (at x within {0..4})
by (rule filterlim-mono) (simp-all add: * ‹x = 0›)
also have −pi = G x **by** (simp add: G-def ‹x = 0›)
finally show ?thesis **.**
next
assume x = 4
have *: at (4::real) within {0..4} ≤ at-left 4 **unfolding** at-within-def
by (rule inf-mono) auto
from G-4 **have** (G ⟶ pi) (at x within {0..4})

```
      by (rule filterlim-mono) (simp-all add: * ‹x = 4›)
    also have pi = G x by (simp add: G-def ‹x = 4›)
    finally show ?thesis .
  next
    assume x ∈ {0<..<4}
    from deriv-G[OF this] have isCont G x by (rule DERIV-isCont)
   thus ?thesis unfolding isCont-def by (rule filterlim-mono) (auto simp: at-le)
  qed
qed

— Finally, we can apply the Fundamental Theorem of Calculus.
have ((λx. sqrt ((4 − x) / x)) has-integral G 4 − G 0) {0..4}
  using cont-G deriv-G
  by (intro fundamental-theorem-of-calculus-interior)
     (auto simp: has-real-derivative-iff-has-vector-derivative)
also have G 4 − G 0 = 2 ∗ pi by (simp add: G-def)
finally show ?thesis .
qed

end

end


theory Catalan-Numbers
imports
  Complex-Main
  Catalan-Auxiliary-Integral
  HOL−Analysis.Analysis
  HOL−Computational-Algebra.Formal-Power-Series
  HOL−Library.Landau-Symbols
  Landau-Symbols.Landau-More
begin
```

1.2 Other auxiliary lemmas

```
lemma mult-eq-imp-eq-div:
  assumes a ∗ b = c (a :: 'a :: semidom-divide) ≠ 0
  shows   b = c div a
  by (simp add: assms(2) assms(1) [symmetric])

lemma Gamma-minus-one-half-real:
  Gamma (−(1/2) :: real) = − 2 ∗ sqrt pi
  using rGamma-plus1[of −1/2 :: real]
  by (simp add: rGamma-inverse-Gamma divide-simps Gamma-one-half-real split:
if-split-asm)

lemma gbinomial-asymptotic':
  assumes z ∉ ℕ
```

shows $(\lambda n.\ z\ gchoose\ (n + k)) \sim[at\text{-}top]$
$(\lambda n.\ (-1)\hat{}(n{+}k)\ /\ (Gamma\ (-z) * of\text{-}nat\ n\ powr\ (z + 1)) :: real)$
proof −
from *assms* **have** [*simp*]: $Gamma\ (-z) \neq 0$
by (*simp-all add*: *Gamma-eq-zero-iff uminus-in-nonpos-Ints-iff*)
have *filterlim* $(\lambda n.\ n + k)$ *at-top at-top*
by (*intro filterlim-subseq strict-mono-add*)
from *asymp-equivI′-const*[*OF gbinomial-asymptotic*[*of z*]] *assms*
have $(\lambda n.\ z\ gchoose\ n) \sim[at\text{-}top]\ (\lambda n.\ (-1)\hat{}n\ /\ (Gamma\ (-z) * exp\ ((z{+}1) * ln\ (real\ n))))$
by (*simp add*: *Gamma-eq-zero-iff uminus-in-nonpos-Ints-iff field-simps*)
also have *eventually* $(\lambda n.\ exp\ ((z{+}1) * ln\ (real\ n)) = real\ n\ powr\ (z{+}1))$ *at-top*
using *eventually-gt-at-top*[*of 0*] **by** *eventually-elim* (*simp add*: *powr-def*)
finally have $(\lambda x.\ z\ gchoose\ (x + k)) \sim[at\text{-}top]$
$(\lambda x.\ (-\ 1)\ \hat{}\ (x + k)\ /\ (Gamma\ (-\ z) * real\ (x + k)\ powr\ (z + 1)))$
by (*rule asymp-equiv-compose′*) (*simp add*: *filterlim-subseq strict-mono-add*)
also have $(\lambda x.\ real\ x + real\ k) \sim[at\text{-}top]\ real$
by (*subst asymp-equiv-add-right*) *auto*
hence $(\lambda x.\ real\ (x + k)\ powr\ (z + 1)) \sim[at\text{-}top]\ (\lambda x.\ real\ x\ powr\ (z + 1))$
by (*intro asymp-equiv-powr-real*) *auto*
finally show *?thesis* **by** − (*simp-all add*: *asymp-equiv-intros*)
qed

1.3 Definition

We define Catalan numbers by their well-known recursive definition. We shall later derive a few more equivalent definitions from this one.

fun *catalan* :: *nat* ⇒ *nat* **where**
catalan 0 = *1*
| *catalan* (*Suc n*) = $(\sum i{\le}n.\ catalan\ i * catalan\ (n - i))$

The easiest proof of the more profound properties of the Catalan numbers (such as their closed-form equation and their asymptotic growth) uses their ordinary generating function (OGF). This proof is almost mechanical in the sense that it does not require 'guessing' the closed form; one can read it directly from the generating function.

We therefore define the OGF of the Catalan numbers ($\sum_{n=0}^{\infty} C_n z^n$ in standard mathematical notation):

definition *fps-catalan* = *Abs-fps* (*of-nat* ∘ *catalan*)

lemma *fps-catalan-nth* [*simp*]: *fps-nth fps-catalan n* = *of-nat* (*catalan n*)
by (*simp add*: *fps-catalan-def*)

Given their recursive definition, it is easy to see that the OGF of the Catalan numbers satisfies the following recursive equation:

lemma *fps-catalan-recurrence*:
fps-catalan = *1* + *fps-X* ∗ *fps-catalan*^*2*
proof (*rule fps-ext*)
fix *n* :: *nat*
show *fps-nth fps-catalan n* = *fps-nth* (*1* + *fps-X* ∗ *fps-catalan*^*2*) *n*
by (*cases n*) (*simp-all add*: *fps-square-nth catalan-Suc*)
qed

We can now easily solve this equation for *fps-catalan*: if we denote the unknown OGF as $F(z)$, we get $F(z) = \frac{1}{2}(1 - \sqrt{1-4z})$.

Note that we do not actually use the square root as defined on real or complex numbers. Any $(1 + cz)^\alpha$ can be expressed using the formal power series whose coefficients are the generalised binomial coefficients, and thus we can do all of these transformations in a purely algebraic way: $\sqrt{1-4z} = (1+z)^{\frac{1}{2}} \circ (-4z)$ (where $\circ$ denotes composition of formal power series) and $(1+z)^\alpha$ has the well-known expansion $\sum_{n=0}^{\infty} \binom{\alpha}{n} z^n$.

lemma *fps-catalan-fps-binomial*:
fps-catalan = (*1*/*2* ∗ (*1* − (*fps-binomial* (*1*/*2*) *oo* (−*4*∗*fps-X*)))) / *fps-X*
proof (*rule mult-eq-imp-eq-div*)
let *?F* = *fps-catalan* :: *'a fps*
have *fps-X* ∗ (*1* + *fps-X* ∗ *?F*^*2*) = *fps-X* ∗ *?F* **by** (*simp only*: *fps-catalan-recurrence* [*symmetric*])
hence (*1* / *2* − *fps-X* ∗ *?F*)^2 = − *fps-X* + *1* / *4*
by (*simp add*: *algebra-simps power2-eq-square fps-numeral-simps*)
also have ... = (*1*/*2* ∗ (*fps-binomial* (*1*/*2*) *oo* (−*4*∗*fps-X*)))^*2*
by (*simp add*: *power-mult-distrib div-power fps-binomial-1 fps-binomial-power fps-compose-power fps-compose-add-distrib ring-distribs*)
finally have *1*/*2* − *fps-X* ∗ *?F* = *1*/*2* ∗ (*fps-binomial* (*1*/*2*) *oo* (−*4*∗*fps-X*))
by (*rule fps-power-eqD*) *simp-all*
thus *fps-X*∗*?F* = *1*/*2* ∗ (*1* − (*fps-binomial* (*1*/*2*) *oo* (−*4*∗*fps-X*))) **by** *algebra*
qed *simp-all*

1.4 Closed-form formulae and more recurrences

We can now read a closed-form formula for the Catalan numbers directly from the generating function $\frac{1}{2z}(1 - (1+z)^{\frac{1}{2}} \circ (-4z))$.

theorem *catalan-closed-form-gbinomial*:
real (*catalan n*) = *2* ∗ (− *4*) ^ *n* ∗ (*1*/*2 gchoose Suc n*)
proof −
have (*catalan n* :: *real*) = *fps-nth fps-catalan n* **by** *simp*
also have ... = *2* ∗ (− *4*) ^ *n* ∗ (*1*/*2 gchoose Suc n*)
by (*subst fps-catalan-fps-binomial*)
(*simp add*: *fps-div-fps-X-nth numeral-fps-const fps-compose-linear*)
finally show *?thesis* **.**
qed

This closed-form formula can easily be rewritten to the form $C_n = \frac{1}{n+1}\binom{2n}{n}$, which contains only 'normal' binomial coefficients and not the generalised ones:

lemma *catalan-closed-form-aux*:
  catalan n ∗ Suc n = (2∗n) choose n
proof −
  have *real ((2∗n) choose n) = fact (2∗n) / (fact n)^2*
    by (*simp add*: *binomial-fact power2-eq-square*)
  also have (*fact (2∗n) :: real) = 4^n ∗ pochhammer (1 / 2) n ∗ fact n*
    by (*simp add*: *fact-double power-mult*)
  also have ... / (*fact n*)^2 / *real* (*n+1*) = *real* (*catalan n*)
    by (*simp add*: *catalan-closed-form-gbinomial gbinomial-pochhammer pochhammer-rec*
        field-simps power2-eq-square power-mult-distrib [*symmetric*] *del*: *of-nat-Suc*)
  finally have *real* (*catalan n ∗ Suc n*) = *real* ((*2∗n*) *choose n*) **by** (*simp add*: *field-simps*)
  thus *?thesis* **by** (*simp only*: *of-nat-eq-iff*)
qed

theorem *of-nat-catalan-closed-form*:
  of-nat (*catalan n*) = (*of-nat* ((*2∗n*) *choose n*) / *of-nat* (*Suc n*) :: *'a* :: *field-char-0*)
proof −
  have *of-nat* (*catalan n ∗ Suc n*) = *of-nat* ((*2∗n*) *choose n*)
    by (*subst catalan-closed-form-aux*) (*rule refl*)
  also have *of-nat* (*catalan n ∗ Suc n*) = *of-nat* (*catalan n*) ∗ *of-nat* (*Suc n*)
    by (*simp only*: *of-nat-mult*)
  finally show *?thesis* **by** (*simp add*: *divide-simps del*: *of-nat-Suc*)
qed

theorem *catalan-closed-form*:
  catalan n = ((*2∗n*) *choose n*) *div Suc n*
  by (*subst catalan-closed-form-aux* [*symmetric*]) (*simp del*: *mult-Suc-right*)

The following is another nice closed-form formula for the Catalan numbers, which directly follows from the previous one:

corollary *catalan-closed-form'*:
  catalan n = ((*2∗n*) *choose n*) − ((*2∗n*) *choose* (*Suc n*))
proof (*cases n*)
  case (*Suc m*)
  have *real* ((*2∗n*) *choose n*) − *real* ((*2∗n*) *choose* (*Suc n*)) =
        fact (*2∗m+2*) / (*fact* (*m+1*))^2 − *fact* (*2∗m+2*) / (*real* (*m+2*) ∗ *fact* (*m+1*) ∗ *fact m*)
    by (*subst* (*1 2*) *binomial-fact*) (*simp-all add*: *Suc power2-eq-square*)
  also have ... = *fact* (*2∗m+2*) / ((*fact* (*m+1*))^2 ∗ *real* (*m+2*))
    by (*simp add*: *divide-simps power2-eq-square*) (*simp-all add*: *algebra-simps*)
  also have ... = *real* (*catalan n*)
    by (*subst of-nat-catalan-closed-form*, *subst binomial-fact*) (*simp-all add*: *Suc power2-eq-square*)
  finally show *?thesis* **by** *linarith*
qed *simp-all*

We can now easily show that the Catalan numbers also satisfy another, simpler recurrence, namely $C_{n+1} = \frac{2(2n+1)}{n+2} C_n$. We will later use this to prove code equations to compute the Catalan numbers more efficiently.

lemma *catalan-Suc-aux*:
  $(n + 2) * catalan\ (Suc\ n) = 2 * (2 * n + 1) * catalan\ n$
proof −
  have *real* (*catalan* (*Suc n*)) ∗ *real* (*n* + *2*) = *real* (*catalan n*) ∗ *2* ∗ *real* (*2* ∗ *n* + *1*)
  proof (*cases n*)
    case (*Suc n*)
    thus *?thesis*
      by (*subst* (*1 2*) *of-nat-catalan-closed-form*, *subst* (*1 2*) *binomial-fact*)
        (*simp-all add*: *divide-simps*)
  qed *simp-all*
  hence *real* ((*n* + *2*) ∗ *catalan* (*Suc n*)) = *real* (*2* ∗ (*2* ∗ *n* + *1*) ∗ *catalan n*)
    by (*simp only*: *mult-ac of-nat-mult*)
  thus *?thesis* **by** (*simp only*: *of-nat-eq-iff*)
qed

theorem *of-nat-catalan-Suc′*:
  of-nat (*catalan* (*Suc n*)) =
    (*of-nat* (*2*∗(*2*∗*n*+*1*)) / *of-nat* (*n*+*2*) ∗ *of-nat* (*catalan n*) :: ′*a* :: *field-char-0*)
proof −
  have (*of-nat* (*2*∗(*2*∗*n*+*1*)) / *of-nat* (*n*+*2*) ∗ *of-nat* (*catalan n*) :: ′*a*) =
        of-nat (*2*∗(*2*∗*n* + *1*) ∗ *catalan n*) / *of-nat* (*n*+*2*)
    by (*simp add*: *divide-simps mult-ac del*: *mult-Suc mult-Suc-right*)
  also note *catalan-Suc-aux*[*of n*, *symmetric*]
  also have *of-nat* ((*n* + *2*) ∗ *catalan* (*Suc n*)) / *of-nat* (*n* + *2*) = (*of-nat* (*catalan* (*Suc n*)) :: ′*a*)
    by (*simp del*: *of-nat-Suc mult-Suc-right mult-Suc*)
  finally show *?thesis* **..**
qed

theorem *catalan-Suc′*:
  catalan (*Suc n*) = (*catalan n* ∗ (*2*∗(*2*∗*n*+*1*))) *div* (*n*+*2*)
proof −
  from *catalan-Suc-aux*[*of n*] **have** *catalan n* ∗ (*2*∗(*2*∗*n*+*1*)) = *catalan* (*Suc n*) ∗ (*n*+*2*)
    by (*simp add*: *algebra-simps*)
  also have . . . *div* (*n*+*2*) = *catalan* (*Suc n*) **by** (*simp del*: *mult-Suc mult-Suc-right*)
  finally show *?thesis* **..**
qed

1.5 Integral formula

The recursive formula we have just proven allows us to derive an integral formula for the Catalan numbers. The proof was adapted from a textbook proof by Steven Roman. [1]

context
begin

private definition *I* :: *nat* ⇒ *real* **where**
I n = *integral* {*0..4*} (λ*x*. *x powr* (*of-nat n* − *1/2*) ∗ *sqrt* (*4* − *x*))

private lemma *has-integral-I0*: ((λ*x*. *x powr* (−(*1/2*)) ∗ *sqrt* (*4* − *x*)) *has-integral* *2*∗*pi*) {*0..4*}
proof −
have ⋀*x*. *x*∈{*0..4*}−{} ⟹ *x powr* (−(*1/2*)) ∗ *sqrt* (*4* − *x*) = *sqrt* ((*4* − *x*) / *x*)
by (*auto simp*: *powr-minus field-simps powr-half-sqrt real-sqrt-divide*)
thus *?thesis* **by** (*rule has-integral-spike*[*OF negligible-empty - catalan-aux-integral*])
qed

private lemma *integrable-I*:
(λ*x*. *x powr* (*of-nat n* − *1/2*) ∗ *sqrt* (*4* − *x*)) *integrable-on* {*0..4*}
proof (*cases n* = *0*)
case *True*
with *has-integral-I0* **show** *?thesis* **by** (*simp add*: *has-integral-integrable*)
next
case *False*
thus *?thesis* **by** (*intro integrable-continuous-real continuous-on-mult continuous-on-powr′*)
(*auto intro*!: *continuous-intros*)
qed

private lemma *I-Suc*: *I* (*Suc n*) = *real* (*2* ∗ (*2*∗*n* + *1*)) / *real* (*n* + *2*) ∗ *I n*
proof −
define *u′ u v v′*
where *u′* = (λ*x*. *sqrt* (*4* − *x* :: *real*))
and *u* = (λ*x*. −*2/3* ∗ (*4* − *x*) *powr* (*3/2* :: *real*))
and *v* = (λ*x*. *x powr* (*real n* + *1/2*))
and *v′* = (λ*x*. (*real n* + *1/2*) ∗ *x powr* (*real n* − *1/2*))
define *c* **where** *c* = −*2/3* ∗ (*real n* + *1/2*)
define *i* **where** *i* = (λ*n x*. *x powr* (*real n* − *1/2*) ∗ *sqrt* (*4* − *x*) :: *real*)

have *I* (*Suc n*) = *integral* {*0..4*} (λ*x*. *u′ x* ∗ *v x*)
unfolding *I-def* **by** (*simp add*: *algebra-simps u′-def v-def*)
have ((λ*x*. *u′ x* ∗ *v x*) *has-integral* − *c* ∗ (*4* ∗ *I n* − *I* (*Suc n*))) {*0..4*}
proof (*rule integration-by-parts-interior*[*OF bounded-bilinear-mult*])
show *continuous-on* {*0..4*} *u* **unfolding** *u-def*
by (*intro continuous-on-powr′ continuous-on-mult*) (*auto intro*!: *continuous-intros*)
show *continuous-on* {*0..4*} *v* **unfolding** *v-def*
by (*intro continuous-on-powr′ continuous-on-mult*) (*auto intro*!: *continuous-intros*)
fix *x* :: *real* **assume** *x*: *x* ∈ {*0*<..<*4*}
from *x* **show** (*u has-vector-derivative u′ x*) (*at x*)

unfolding *has-real-derivative-iff-has-vector-derivative* [*symmetric*] *u-def u′-def*
by (*auto intro*!: *derivative-eq-intros simp*: *field-simps powr-half-sqrt*)
from *x* **show** (*v has-vector-derivative v′ x*) (*at x*)
unfolding *has-real-derivative-iff-has-vector-derivative* [*symmetric*] *v-def v′-def*
by (*auto intro*!: *derivative-eq-intros simp*: *field-simps*)
next
show ((λ*x*. *u x* ∗ *v′ x*) *has-integral u 4* ∗ *v 4* − *u 0* ∗ *v 0* − − *c* ∗ (*4* ∗ *I n* − *I* (*Suc n*))) {*0*..*4*}
proof (*rule has-integral-spike*; (*intro ballI*)*?*)
fix *x* :: *real* **assume** *x*: *x* ∈ {*0*..*4*}−{*0*}
have *u x* ∗ *v′ x* = *c* ∗ ((*4* − *x*) *powr* (*1* + *1*/*2*) ∗ *x powr* (*real n* − *1*/*2*))
by (*simp add*: *u-def v′-def c-def*)
also from *x* **have** (*4* − *x*) *powr* (*1* + *1*/*2*) = (*4* − *x*) ∗ *sqrt* (*4* − *x*)
by (*subst powr-add*) (*simp-all add*: *powr-half-sqrt*)
also have . . . ∗ *x powr* (*real n* − *1*/*2*) = *4* ∗ *sqrt* (*4* − *x*) ∗ *x powr* (*real n* − *1*/*2*) −
sqrt (*4* − *x*) ∗ *x powr* (*real n* − *1*/*2* + *1*)
by (*subst powr-add*) (*insert x*, *simp add*: *field-simps*)
also have *real n* − *1*/*2* + *1* = *real* (*Suc n*) − *1*/*2* **by** *simp*
finally show *u x* ∗ *v′ x* = *c* ∗ (*4* ∗ *i n x* − *i* (*Suc n*) *x*) **by** (*simp add*: *i-def*)
next
have ((λ*x*. *c* ∗ (*4* ∗ *i n x* − *i* (*Suc n*) *x*)) *has-integral c* ∗ (*4* ∗ *I n* − *I* (*Suc n*))) {*0*..*4*}
unfolding *i-def I-def*
by (*intro has-integral-mult-right has-integral-diff integrable-integral integrable-I*)
thus ((λ*x*. *c* ∗ (*4* ∗ *i n x* − *i* (*Suc n*) *x*)) *has-integral u 4* ∗ *v 4* − *u 0* ∗ *v 0* − −
c ∗ (*4* ∗ *I n* − *I* (*Suc n*))) {*0*..*4*} **by** (*simp add*: *u-def v-def*)
qed *simp-all*
qed *simp-all*
also have (λ*x*. *u′ x* ∗ *v x*) = *i* (*Suc n*)
by (*rule ext*) (*simp add*: *u′-def v-def i-def algebra-simps*)
finally have *I* (*Suc n*) = − *c* ∗ (*4* ∗ *I n* − *I* (*Suc n*)) **unfolding** *I-def i-def* **by** *blast*
hence (*1* − *c*) ∗ *I* (*Suc n*) = −*4* ∗ *c* ∗ *I n* **by** *algebra*
hence *I* (*Suc n*) = (−*4* ∗ *c*) / (*1* − *c*) ∗ *I n* **by** (*simp add*: *field-simps c-def*)
also have (−*4* ∗ *c*) / (*1* − *c*) = *real* (*2*∗(*2*∗*n* + *1*)) / *real* (*n* + *2*)
by (*simp add*: *c-def field-simps*)
finally show *?thesis* **.**
qed

private lemma *catalan-eq-I*: *real* (*catalan n*) = *I n* / (*2* ∗ *pi*)
proof (*induction n*)
case *0*
thus *?case* **using** *has-integral-I0* **by** (*simp add*: *I-def integral-unique*)
next
case (*Suc n*)
show *?case* **by** (*simp add*: *of-nat-catalan-Suc′ Suc.IH I-Suc*)

qed

theorem *catalan-integral-form*:
  $((\lambda x.\ x\ powr\ (real\ n - 1\ /\ 2) * sqrt\ (4 - x)\ /\ (2*pi))$
    has-integral real $(catalan\ n))\ \{0..4\}$
proof −
  have $((\lambda x.\ x\ powr\ (real\ n - 1\ /\ 2) * sqrt\ (4 - x) * inverse\ (2*pi))$ *has-integral*

    $I\ n * inverse\ (2 * pi))\ \{0..4\}$ **unfolding** *I-def*
   by (*intro has-integral-mult-left integrable-integral integrable-I*)
  thus *?thesis* **by** (*simp add*: *catalan-eq-I field-simps*)
qed

end

1.6 Asymptotics

Using the closed form $C_n = 2\cdot(-4)^n\binom{\frac{1}{2}}{n+1}$ and the fact that $\binom{\alpha}{n} \sim \frac{(-1)^n}{\Gamma(-\alpha)n^{\alpha+1}}$ for any $\alpha \notin \mathbb{N}$, wwe can now easily analyse the asymptotic behaviour of the Catalan numbers:

theorem *catalan-asymptotics*:
  $catalan \sim[at\text{-}top]\ (\lambda n.\ 4\ \hat{}\ n\ /\ (sqrt\ pi * n\ powr\ (3/2)))$
proof −
  have $catalan \sim[at\text{-}top]\ (\lambda n.\ 2 * (-\ 4)\ \hat{}\ n * (1/2\ gchoose\ (n+1)))$
   by (*subst catalan-closed-form-gbinomial*) *simp-all*
  also have $(\lambda n.\ 1/2\ gchoose\ (n+1)) \sim[at\text{-}top]\ (\lambda n.\ (-1)\hat{}(n+1)\ /\ (Gamma$
$(-(1/2)) * real\ n\ powr\ (1/2 + 1)))$
  using *fraction-not-in-nats*[*of 2 1*] **by** (*intro asymp-equiv-intros gbinomial-asymptotic′*)
simp-all
  also have $(\lambda n.\ 2 * (-\ 4)\ \hat{}\ n * \ldots\ n) = (\lambda n.\ 4\ \hat{}\ n\ /\ (sqrt\ pi * n\ powr\ (3/2)))$
   by (*intro ext*) (*simp add*: *Gamma-minus-one-half-real power-mult-distrib* [*symmetric*])
  finally show *?thesis* **by** − (*simp-all add*: *asymp-equiv-intros*)
qed

1.7 Relation to binary trees

It is well-known that the Catalan number C_n is the number of rooted binary trees with n internal nodes (where internal nodes are those with two children and external nodes are those with no children).

We will briefly show this here to show that the above asymptotic formula also describes the number of binary trees of a given size.

qualified datatype *tree* = *Leaf* | *Node tree tree*

qualified primrec *count-nodes* :: *tree* ⇒ *nat* **where**
  count-nodes Leaf = *0*
| *count-nodes* (*Node l r*) = *1* + *count-nodes l* + *count-nodes r*

qualified definition *trees-of-size* :: *nat* ⇒ *tree set* **where**
trees-of-size n = {*t*. *count-nodes t* = *n*}

lemma *count-nodes-eq-0-iff* [*simp*]: *count-nodes t* = *0* ⟷ *t* = *Leaf*
by (*cases t*) *simp-all*

lemma *trees-of-size-0* [*simp*]: *trees-of-size 0* = {*Leaf*}
by (*simp add*: *trees-of-size-def*)

lemma *trees-of-size-Suc*:
trees-of-size (*Suc n*) = (λ(*l*,*r*). *Node l r*) ‘ (⋃*k*≤*n*. *trees-of-size k* × *trees-of-size* (*n* − *k*))
(**is** *?lhs* = *?rhs*)
proof (*rule set-eqI*)
fix *t* **show** *t* ∈ *?lhs* ⟷ *t* ∈ *?rhs* **by** (*cases t*) (*auto simp*: *trees-of-size-def*)
qed

lemma *finite-trees-of-size* [*simp*,*intro*]: *finite* (*trees-of-size n*)
by (*induction n rule*: *catalan.induct*)
(*auto simp*: *trees-of-size-Suc intro*!: *finite-imageI finite-cartesian-product*)

lemma *trees-of-size-nonempty*: *trees-of-size n* ≠ {}
by (*induction n rule*: *catalan.induct*) (*auto simp*: *trees-of-size-Suc*)

lemma *trees-of-size-disjoint*:
assumes *m* ≠ *n*
shows *trees-of-size m* ∩ *trees-of-size n* = {}
using *assms* **by** (*auto simp*: *trees-of-size-def*)

theorem *card-trees-of-size*: *card* (*trees-of-size n*) = *catalan n*
by (*induction n rule*: *catalan.induct*)
(*simp-all add*: *catalan-Suc trees-of-size-Suc card-image inj-on-def*
trees-of-size-disjoint Times-Int-Times catalan-Suc card-UN-disjoint)

1.8 Efficient computation

We shall now prove code equations that allow more efficient computation of Catalan numbers. In order to do this, we define a tail-recursive function that uses the recurrence *catalan* (*Suc n*) = *catalan n* ∗ (*2* ∗ (*2* ∗ *n* + *1*)) *div* (*n* + *2*):

qualified function *catalan-aux* **where** [*simp del*]:
catalan-aux n k acc =
(*if k* ≥ *n then acc else catalan-aux n* (*Suc k*) ((*acc* ∗ (*2*∗(*2*∗*k*+*1*))) *div* (*k*+*2*)))
by *auto*
termination by (*relation Wellfounded.measure* (λ(*a*,*b*,-). *a* − *b*)) *simp-all*

qualified lemma *catalan-aux-simps*:

```
  k ≥ n ⟹ catalan-aux n k acc = acc
  k < n ⟹ catalan-aux n k acc = catalan-aux n (Suc k) ((acc ∗ (2∗(2∗k+1)))
div (k+2))
  by (subst catalan-aux.simps, simp)+

qualified lemma catalan-aux-correct:
  assumes k ≤ n
  shows   catalan-aux n k (catalan k) = catalan n
using assms
proof (induction n k catalan k rule: catalan-aux.induct)
  case (1 n k)
  show ?case
  proof (cases k < n)
    case True
    hence catalan-aux n k (catalan k) = catalan-aux n (Suc k) (catalan (Suc k))
      by (subst catalan-Suc') (simp-all add: catalan-aux-simps)
    with 1 True show ?thesis by (simp add: catalan-Suc')
  qed (insert 1.prems, simp-all add: catalan-aux-simps)
qed

lemma catalan-code [code]: catalan n = catalan-aux n 0 1
  using catalan-aux-correct[of 0 n] by simp

end
```

References

[1] S. Roman. *An Introduction to Catalan Numbers*. Birkhäuser Basel, 2015.