The Lambert W Function on the Reals

Manuel Eberl

October 13, 2025

Abstract

The Lambert W function is a multi-valued function defined as the inverse function of $x \mapsto xe^x$. Besides numerous applications in combinatorics, physics, and engineering, it also frequently occurs when solving equations containing both e^x and x, or both x and $\log x$.

This article provides a definition of the two real-valued branches $W_0(x)$ and $W_{-1}(x)$ and proves various properties such as basic identities and inequalities, monotonicity, differentiability, asymptotic expansions, and the MacLaurin series of $W_0(x)$ at $x = 0$.

Contents

1 The Lambert W Function on the reals

theory *Lambert-W*
imports
  Complex-Main
  HOL−Library.FuncSet
  HOL−Real-Asymp.Real-Asymp
begin

1.1 Properties of the function $x \mapsto xe^x$

lemma *exp-times-self-gt*:
  assumes $x \neq -1$
  shows $x * exp\ x > -exp\ (-1::real)$
proof −
  define f **where** $f = (\lambda x::real.\ x * exp\ x)$
  define f' **where** $f' = (\lambda x::real.\ (x + 1) * exp\ x)$
  have (f *has-field-derivative* $f'\ x$) (*at* x) **for** x
    by (*auto simp*: *f-def f′-def intro*!: *derivative-eq-intros simp*: *algebra-simps*)
  define $l\ r$ **where** $l = min\ x\ (-1)$ **and** $r = max\ x\ (-1)$

  have $\exists z.\ z > l \wedge z < r \wedge f\ r - f\ l = (r - l) * f'\ z$
    unfolding *f-def f′-def l-def r-def* **using** *assms*
    by (*intro MVT2*) (*auto intro*!: *derivative-eq-intros simp*: *algebra-simps*)
  then obtain z **where** z: $z \in \{l<..<r\}\ f\ r - f\ l = (r - l) * f'\ z$
    by *auto*
  from z **have** $f\ x = f\ (-1) + (x + 1) * f'\ z$
    using *assms* **by** (*cases* $x \geq -1$) (*auto simp*: *l-def r-def max-def min-def algebra-simps*)
  moreover have $sgn\ ((x + 1) * f'\ z) = 1$
    using *z assms*
   by (*cases* x (-1) :: *real rule*: *linorder-cases*; *cases* z (-1) :: *real rule*: *linorder-cases*)
     (*auto simp*: *f′-def sgn-mult l-def r-def*)
  hence $(x + 1) * f'\ z > 0$ **using** *sgn-greater* **by** *fastforce*
  ultimately show *?thesis* **by** (*simp add*: *f-def*)
qed

lemma *exp-times-self-ge*: $x * exp\ x \geq -exp\ (-1::real)$
  using *exp-times-self-gt*[*of* x] **by** (*cases* $x = -1$) *auto*

lemma *exp-times-self-strict-mono*:
  assumes $x \geq -1\ x < (y :: real)$
  shows $x * exp\ x < y * exp\ y$
  using *assms*(*2*)
proof (*rule DERIV-pos-imp-increasing-open*)
  fix t **assume** t: $x < t\ t < y$
  have (($\lambda x.\ x * exp\ x$) *has-real-derivative* $(t + 1) * exp\ t$) (*at* t)
    by (*auto intro*!: *derivative-eq-intros simp*: *algebra-simps*)
  moreover have $(t + 1) * exp\ t > 0$

using *t assms* **by** (*intro mult-pos-pos*) *auto*
ultimately show $\exists\, y.$ (($\lambda a.$ $a * exp\ a$) *has-real-derivative* y) (*at t*) $\wedge$ $0 < y$ **by** *blast*
qed (*auto intro*!: *continuous-intros*)

lemma *exp-times-self-strict-antimono*:
assumes $y \leq -1$ $x < (y :: real)$
shows $x * exp\ x > y * exp\ y$
proof −
have $-x * exp\ x < -y * exp\ y$
using *assms*(2)
proof (*rule DERIV-pos-imp-increasing-open*)
fix *t* **assume** *t*: $x < t$ $t < y$
have (($\lambda x.$ $-x * exp\ x$) *has-real-derivative* $(-(t + 1)) * exp\ t$) (*at t*)
by (*auto intro*!: *derivative-eq-intros simp*: *algebra-simps*)
moreover have $(-(t + 1)) * exp\ t > 0$
using *t assms* **by** (*intro mult-pos-pos*) *auto*
ultimately show $\exists\, y.$ (($\lambda a.$ $-a * exp\ a$) *has-real-derivative* y) (*at t*) $\wedge$ $0 < y$ **by** *blast*
qed (*auto intro*!: *continuous-intros*)
thus *?thesis* **by** *simp*
qed

lemma *exp-times-self-mono*:
assumes $x \geq -1$ $x \leq (y :: real)$
shows $x * exp\ x \leq y * exp\ y$
using *exp-times-self-strict-mono*[*of x y*] *assms* **by** (*cases* $x = y$) *auto*

lemma *exp-times-self-antimono*:
assumes $y \leq -1$ $x \leq (y :: real)$
shows $x * exp\ x \geq y * exp\ y$
using *exp-times-self-strict-antimono*[*of y x*] *assms* **by** (*cases* $x = y$) *auto*

lemma *exp-times-self-inj*: *inj-on* (λx::*real*. $x * exp\ x$) $\{-1..\}$
proof
fix *x y* :: *real*
assume $x \in \{-1..\}$ $y \in \{-1..\}$ $x * exp\ x = y * exp\ y$
thus $x = y$
using *exp-times-self-strict-mono*[*of x y*] *exp-times-self-strict-mono*[*of y x*]
by (*cases x y rule*: *linorder-cases*) *auto*
qed

lemma *exp-times-self-inj'*: *inj-on* (λx::*real*. $x * exp\ x$) $\{..-1\}$
proof
fix *x y* :: *real*
assume $x \in \{..-1\}$ $y \in \{..-1\}$ $x * exp\ x = y * exp\ y$
thus $x = y$
using *exp-times-self-strict-antimono*[*of x y*] *exp-times-self-strict-antimono*[*of y x*]

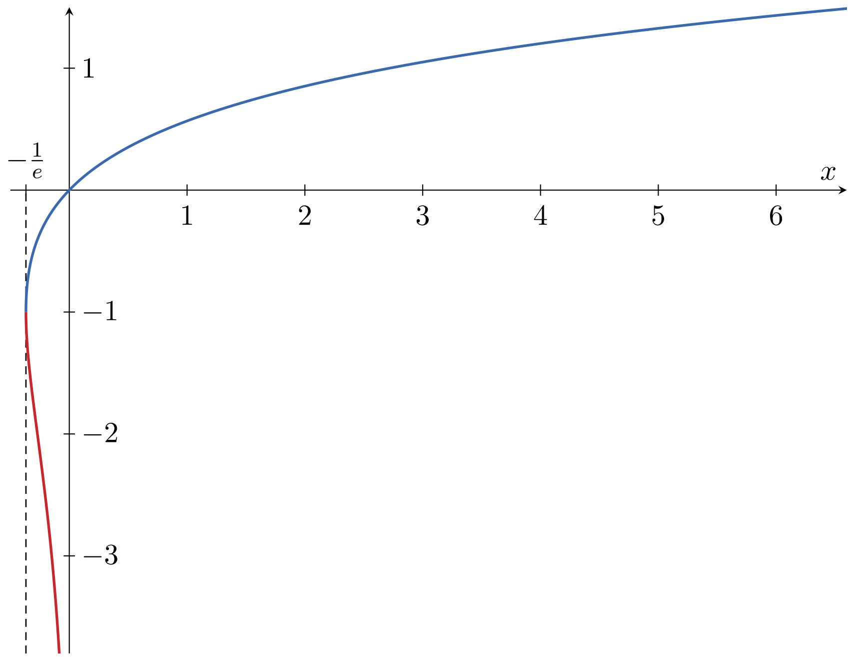

Figure 1: The two real branches of the Lambert W function: W_0 (blue) and W_{-1} (red).

by (*cases x y rule*: *linorder-cases*) *auto*
qed

1.2 Definition

The following are the two branches $W_0(x)$ and $W_{-1}(x)$ of the Lambert W function on the real numbers. These are the inverse functions of the function $x \mapsto xe^x$, i. e. we have $W(x)e^{W(x)} = x$ for both branches wherever they are defined. The two branches meet at the point $x = -\frac{1}{e}$.
$W_0(x)$ is the principal branch, whose domain is $[-\frac{1}{e}; \infty)$ and whose range is $[-1; \infty)$. $W_{-1}(x)$ has the domain $[-\frac{1}{e}; 0)$ and the range $(-\infty; -1]$. Figure 1 shows plots of these two branches for illustration.

definition *Lambert-W* :: *real* ⇒ *real* **where**
Lambert-W x = (*if x* < −*exp*(−*1*) *then* −*1* *else* (*THE w. w* ≥ −*1* ∧ *w* ∗ *exp w* = *x*))

definition *Lambert-W′* :: *real* ⇒ *real* **where**
Lambert-W′ x = (*if x* ∈ {−*exp*(−*1*)..<*0*} *then* (*THE w. w* ≤ −*1* ∧ *w* ∗ *exp w* = *x*) *else* −*1*)

lemma *Lambert-W-ex1*:

assumes $(x::real) \geq -exp\ (-1)$
shows $\exists!w.\ w \geq -1 \wedge w * exp\ w = x$
proof (*rule ex-ex1I*)
have *filterlim* ($\lambda w::real.\ w * exp\ w$) *at-top at-top*
by *real-asymp*
hence *eventually* ($\lambda w.\ w * exp\ w \geq x$) *at-top*
by (*auto simp*: *filterlim-at-top*)
hence *eventually* ($\lambda w.\ w \geq 0 \wedge w * exp\ w \geq x$) *at-top*
by (*intro eventually-conj eventually-ge-at-top*)
then obtain w' **where** w': $w' * exp\ w' \geq x\ w' \geq 0$
by (*auto simp*: *eventually-at-top-linorder*)
from w' *assms* **have** $\exists\, w.\ -1 \leq w \wedge w \leq w' \wedge w * exp\ w = x$
by (*intro IVT′ continuous-intros*) *auto*
thus $\exists\, w.\ w \geq -1 \wedge w * exp\ w = x$ **by** *blast*
next
fix $w\ w'$ :: *real*
assume ww': $w \geq -1 \wedge w * exp\ w = x\ w' \geq -1 \wedge w' * exp\ w' = x$
hence $w * exp\ w = w' * exp\ w'$ **by** *simp*
thus $w = w'$
using *exp-times-self-strict-mono*[*of* $w\ w'$] *exp-times-self-strict-mono*[*of* $w'\ w$] ww'
by (*cases* $w\ w'$ *rule*: *linorder-cases*) *auto*
qed

lemma *Lambert-W′-ex1*:
assumes $(x::real) \in \{-exp\ (-1)..<0\}$
shows $\exists!w.\ w \leq -1 \wedge w * exp\ w = x$
proof (*rule ex-ex1I*)
have *eventually* ($\lambda w.\ x \leq w * exp\ w$) *at-bot*
using *assms* **by** *real-asymp*
hence *eventually* ($\lambda w.\ w \leq -1 \wedge w * exp\ w \geq x$) *at-bot*
by (*intro eventually-conj eventually-le-at-bot*)
then obtain w' **where** w': $w' * exp\ w' \geq x\ w' \leq -1$
by (*auto simp*: *eventually-at-bot-linorder*)

from w' *assms* **have** $\exists\, w.\ w' \leq w \wedge w \leq -1 \wedge w * exp\ w = x$
by (*intro IVT2′ continuous-intros*) *auto*
thus $\exists\, w.\ w \leq -1 \wedge w * exp\ w = x$ **by** *blast*
next
fix $w\ w'$ :: *real*
assume ww': $w \leq -1 \wedge w * exp\ w = x\ w' \leq -1 \wedge w' * exp\ w' = x$
hence $w * exp\ w = w' * exp\ w'$ **by** *simp*
thus $w = w'$
using *exp-times-self-strict-antimono*[*of* $w\ w'$] *exp-times-self-strict-antimono*[*of* $w'\ w$] ww'
by (*cases* $w\ w'$ *rule*: *linorder-cases*) *auto*
qed

lemma *Lambert-W-times-exp-self*:

assumes *x* ≥ −*exp* (−*1*)
shows *Lambert-W x* ∗ *exp* (*Lambert-W x*) = *x*
using *theI'*[*OF Lambert-W-ex1*[*OF assms*]] *assms* **by** (*auto simp*: *Lambert-W-def*)

lemma *Lambert-W-times-exp-self'*:
assumes *x* ≥ −*exp* (−*1*)
shows *exp* (*Lambert-W x*) ∗ *Lambert-W x* = *x*
using *Lambert-W-times-exp-self*[*of x*] *assms* **by** (*simp add*: *mult-ac*)

lemma *Lambert-W'-times-exp-self*:
assumes *x* ∈ {−*exp* (−*1*)..<*0*}
shows *Lambert-W' x* ∗ *exp* (*Lambert-W' x*) = *x*
using *theI'*[*OF Lambert-W'-ex1*[*OF assms*]] *assms* **by** (*auto simp*: *Lambert-W'-def*)

lemma *Lambert-W'-times-exp-self'*:
assumes *x* ∈ {−*exp* (−*1*)..<*0*}
shows *exp* (*Lambert-W' x*) ∗ *Lambert-W' x* = *x*
using *Lambert-W'-times-exp-self*[*of x*] *assms* **by** (*simp add*: *mult-ac*)

lemma *Lambert-W-ge*: *Lambert-W x* ≥ −*1*
using *theI'*[*OF Lambert-W-ex1*[*of x*]] **by** (*auto simp*: *Lambert-W-def*)

lemma *Lambert-W'-le*: *Lambert-W' x* ≤ −*1*
using *theI'*[*OF Lambert-W'-ex1*[*of x*]] **by** (*auto simp*: *Lambert-W'-def*)

lemma *Lambert-W-eqI*:
assumes *w* ≥ −*1 w* ∗ *exp w* = *x*
shows *Lambert-W x* = *w*
proof −
from *assms exp-times-self-ge*[*of w*] **have** *x* ≥ −*exp* (−*1*)
by (*cases x* ≥ −*exp* (−*1*)) *auto*
from *Lambert-W-ex1*[*OF this*] *Lambert-W-times-exp-self*[*OF this*] *Lambert-W-ge*[*of x*] *assms*
show *?thesis* **by** *metis*
qed

lemma *Lambert-W'-eqI*:
assumes *w* ≤ −*1 w* ∗ *exp w* = *x*
shows *Lambert-W' x* = *w*
proof −
from *assms exp-times-self-ge*[*of w*] **have** *x* ≥ −*exp* (−*1*)
by (*cases x* ≥ −*exp* (−*1*)) *auto*
moreover from *assms* **have** *w* ∗ *exp w* < *0*
by (*intro mult-neg-pos*) *auto*
ultimately have *x* ∈ {−*exp* (−*1*)..<*0*}
using *assms* **by** *auto*

from *Lambert-W'-ex1*[*OF this*(*1*)] *Lambert-W'-times-exp-self*[*OF this*(*1*)] *Lambert-W'-le assms*

show *?thesis* **by** *metis*
qed

$W_0(x)$ and $W_{-1}(x)$ together fully cover all solutions of $we^w = x$:

lemma *exp-times-self-eqD*:
assumes $w * exp\ w = x$
shows $x \geq -exp\ (-1)$ **and** $w = Lambert\text{-}W\ x \lor x < 0 \land w = Lambert\text{-}W'\ x$
proof −
from *assms* **show** $x \geq -exp\ (-1)$
using *exp-times-self-ge*[*of w*] **by** *auto*
show $w = Lambert\text{-}W\ x \lor x < 0 \land w = Lambert\text{-}W'\ x$
proof (*cases* $w \geq -1$)
case *True*
hence $Lambert\text{-}W\ x = w$
using *assms* **by** (*intro Lambert-W-eqI*) *auto*
thus *?thesis* **by** *auto*
next
case *False*
from *False* **have** $w * exp\ w < 0$
by (*intro mult-neg-pos*) *auto*
from *False* **have** $Lambert\text{-}W'\ x = w$
using *assms* **by** (*intro Lambert-W'-eqI*) *auto*
thus *?thesis* **using** *assms* ‹$w * exp\ w < 0$› **by** *auto*
qed
qed

theorem *exp-times-self-eq-iff*:
$w * exp\ w = x \longleftrightarrow x \geq -exp\ (-1) \land (w = Lambert\text{-}W\ x \lor x < 0 \land w = Lambert\text{-}W'\ x)$
using *exp-times-self-eqD*[*of w x*]
by (*auto simp*: *Lambert-W-times-exp-self Lambert-W'-times-exp-self*)

lemma *Lambert-W-exp-times-self* [*simp*]: $x \geq -1 \Longrightarrow Lambert\text{-}W\ (x * exp\ x) = x$
by (*rule Lambert-W-eqI*) *auto*

lemma *Lambert-W-exp-times-self'* [*simp*]: $x \geq -1 \Longrightarrow Lambert\text{-}W\ (exp\ x * x) = x$
by (*rule Lambert-W-eqI*) *auto*

lemma *Lambert-W'-exp-times-self* [*simp*]: $x \leq -1 \Longrightarrow Lambert\text{-}W'\ (x * exp\ x) = x$
by (*rule Lambert-W'-eqI*) *auto*

lemma *Lambert-W'-exp-times-self'* [*simp*]: $x \leq -1 \Longrightarrow Lambert\text{-}W'\ (exp\ x * x) = x$
by (*rule Lambert-W'-eqI*) *auto*

lemma *Lambert-W-times-ln-self*:

assumes $x \geq exp\ (-1)$
shows $Lambert\text{-}W\ (x * ln\ x) = ln\ x$
proof −
have $0 < exp\ (-1 :: real)$
by *simp*
also note ‹... $\leq x$›
finally have $x > 0$ **.**
from *assms* **have** $ln\ (exp\ (-1)) \leq ln\ x$
using ‹$x > 0$› **by** (*subst ln-le-cancel-iff*) *auto*
hence $Lambert\text{-}W\ (exp\ (ln\ x) * ln\ x) = ln\ x$
by (*subst Lambert-W-exp-times-self'*) *auto*
thus *?thesis* **using** ‹$x > 0$› **by** *simp*
qed

lemma *Lambert-W-times-ln-self'*:
assumes $x \geq exp\ (-1)$
shows $Lambert\text{-}W\ (ln\ x * x) = ln\ x$
using *Lambert-W-times-ln-self*[*OF assms*] **by** (*simp add*: *mult.commute*)

lemma *Lambert-W-eq-minus-exp-minus1* [*simp*]: $Lambert\text{-}W\ (-exp\ (-1)) = -1$
by (*rule Lambert-W-eqI*) *auto*

lemma *Lambert-W'-eq-minus-exp-minus1* [*simp*]: $Lambert\text{-}W'\ (-exp\ (-1)) = -1$
by (*rule Lambert-W'-eqI*) *auto*

lemma *Lambert-W-0* [*simp*]: $Lambert\text{-}W\ 0 = 0$
by (*rule Lambert-W-eqI*) *auto*

1.3 Monotonicity properties

lemma *Lambert-W-strict-mono*:
assumes $x \geq -exp(-1)$ $x < y$
shows $Lambert\text{-}W\ x < Lambert\text{-}W\ y$
proof (*rule ccontr*)
assume $\neg(Lambert\text{-}W\ x < Lambert\text{-}W\ y)$
hence $Lambert\text{-}W\ x * exp\ (Lambert\text{-}W\ x) \geq Lambert\text{-}W\ y * exp\ (Lambert\text{-}W\ y)$
by (*intro exp-times-self-mono*) (*auto simp*: *Lambert-W-ge*)
hence $x \geq y$
using *assms* **by** (*simp add*: *Lambert-W-times-exp-self*)
with *assms* **show** *False* **by** *simp*
qed

lemma *Lambert-W-mono*:
assumes $x \geq -exp(-1)$ $x \leq y$
shows $Lambert\text{-}W\ x \leq Lambert\text{-}W\ y$
using *Lambert-W-strict-mono*[*of x y*] *assms* **by** (*cases* $x = y$) *auto*

lemma *Lambert-W-eq-iff* [*simp*]:
$x \geq -exp(-1) \Longrightarrow y \geq -exp(-1) \Longrightarrow Lambert\text{-}W\ x = Lambert\text{-}W\ y \longleftrightarrow x = y$

using *Lambert-W-strict-mono*[*of x y*] *Lambert-W-strict-mono*[*of y x*]
by (*cases x y rule*: *linorder-cases*) *auto*

lemma *Lambert-W-le-iff* [*simp*]:
$x \geq -exp(-1) \Longrightarrow y \geq -exp(-1) \Longrightarrow Lambert\text{-}W\ x \leq Lambert\text{-}W\ y \longleftrightarrow x \leq y$
using *Lambert-W-strict-mono*[*of x y*] *Lambert-W-strict-mono*[*of y x*]
by (*cases x y rule*: *linorder-cases*) *auto*

lemma *Lambert-W-less-iff* [*simp*]:
$x \geq -exp(-1) \Longrightarrow y \geq -exp(-1) \Longrightarrow Lambert\text{-}W\ x < Lambert\text{-}W\ y \longleftrightarrow x < y$
using *Lambert-W-strict-mono*[*of x y*] *Lambert-W-strict-mono*[*of y x*]
by (*cases x y rule*: *linorder-cases*) *auto*

lemma *Lambert-W-le-minus-one*:
assumes $x \leq -exp(-1)$
shows $Lambert\text{-}W\ x = -1$
proof (*cases* $x = -exp(-1)$)
case *False*
thus *?thesis* **using** *assms*
by (*auto simp*: *Lambert-W-def*)
qed *auto*

lemma *Lambert-W-pos-iff* [*simp*]: $Lambert\text{-}W\ x > 0 \longleftrightarrow x > 0$
proof (*cases* $x \geq -exp\ (-1)$)
case *True*
thus *?thesis*
using *Lambert-W-less-iff*[*of 0 x*] **by** (*simp del*: *Lambert-W-less-iff*)
next
case *False*
hence $x < -\ exp(-1)$ **by** *auto*
also have $\ldots \leq 0$ **by** *simp*
finally show *?thesis* **using** *False*
by (*auto simp*: *Lambert-W-le-minus-one*)
qed

lemma *Lambert-W-eq-0-iff* [*simp*]: $Lambert\text{-}W\ x = 0 \longleftrightarrow x = 0$
using *Lambert-W-eq-iff*[*of x 0*]
by (*cases* $x \geq -exp\ (-1)$) (*auto simp*: *Lambert-W-le-minus-one simp del*: *Lambert-W-eq-iff*)

lemma *Lambert-W-nonneg-iff* [*simp*]: $Lambert\text{-}W\ x \geq 0 \longleftrightarrow x \geq 0$
using *Lambert-W-pos-iff*[*of x*]
by (*cases* $x = 0$) (*auto simp del*: *Lambert-W-pos-iff*)

lemma *Lambert-W-neg-iff* [*simp*]: $Lambert\text{-}W\ x < 0 \longleftrightarrow x < 0$
using *Lambert-W-nonneg-iff*[*of x*] **by** (*auto simp del*: *Lambert-W-nonneg-iff*)

lemma *Lambert-W-nonpos-iff* [*simp*]: $Lambert\text{-}W\ x \leq 0 \longleftrightarrow x \leq 0$
using *Lambert-W-pos-iff*[*of x*] **by** (*auto simp del*: *Lambert-W-pos-iff*)

```
lemma Lambert-W-geI:
  assumes y * exp y ≤ x
  shows   Lambert-W x ≥ y
proof (cases y ≥ −1)
  case False
  hence y ≤ −1 by simp
  also have −1 ≤ Lambert-W x by (rule Lambert-W-ge)
  finally show ?thesis .
next
  case True
  have Lambert-W x ≥ Lambert-W (y * exp y)
    using assms exp-times-self-ge[of y] by (intro Lambert-W-mono) auto
  thus ?thesis using assms True by simp
qed

lemma Lambert-W-gtI:
  assumes y * exp y < x
  shows   Lambert-W x > y
proof (cases y ≥ −1)
  case False
  hence y < −1 by simp
  also have −1 ≤ Lambert-W x by (rule Lambert-W-ge)
  finally show ?thesis .
next
  case True
  have Lambert-W x > Lambert-W (y * exp y)
    using assms exp-times-self-ge[of y] by (intro Lambert-W-strict-mono) auto
  thus ?thesis using assms True by simp
qed

lemma Lambert-W-leI:
  assumes y * exp y ≥ x y ≥ −1 x ≥ −exp (−1)
  shows   Lambert-W x ≤ y
proof −
  have Lambert-W x ≤ Lambert-W (y * exp y)
    using assms exp-times-self-ge[of y] by (intro Lambert-W-mono) auto
  thus ?thesis using assms by simp
qed

lemma Lambert-W-lessI:
  assumes y * exp y > x y ≥ −1 x ≥ −exp (−1)
  shows   Lambert-W x < y
proof −
  have Lambert-W x < Lambert-W (y * exp y)
    using assms exp-times-self-ge[of y] by (intro Lambert-W-strict-mono) auto
  thus ?thesis using assms by simp
qed
```

lemma *Lambert-W′-strict-antimono*:
  assumes $-exp\ (-1) \leq x\ x < y\ y < 0$
  shows *Lambert-W′ x > Lambert-W′ y*
proof (*rule ccontr*)
  assume ¬(*Lambert-W′ x > Lambert-W′ y*)
  hence *Lambert-W′ x ∗ exp* (*Lambert-W′ x*) ≥ *Lambert-W′ y ∗ exp* (*Lambert-W′ y*)
    using *assms* **by** (*intro exp-times-self-antimono Lambert-W′-le*) *auto*
  hence $x \geq y$
    using *assms* **by** (*simp add*: *Lambert-W′-times-exp-self*)
  with *assms* **show** *False* **by** *simp*
qed

lemma *Lambert-W′-antimono*:
  assumes $x \geq -exp(-1)\ x \leq y\ y < 0$
  shows *Lambert-W′ x ≥ Lambert-W′ y*
  using *Lambert-W′-strict-antimono*[*of x y*] *assms* **by** (*cases x = y*) *auto*

lemma *Lambert-W′-eq-iff* [*simp*]:
  $x \in \{-exp(-1)..<0\} \Longrightarrow y \in \{-exp(-1)..<0\} \Longrightarrow$ *Lambert-W′ x = Lambert-W′ y* $\longleftrightarrow x = y$
  using *Lambert-W′-strict-antimono*[*of x y*] *Lambert-W′-strict-antimono*[*of y x*]
  by (*cases x y rule*: *linorder-cases*) *auto*

lemma *Lambert-W′-le-iff* [*simp*]:
  $x \in \{-exp(-1)..<0\} \Longrightarrow y \in \{-exp(-1)..<0\} \Longrightarrow$ *Lambert-W′ x ≤ Lambert-W′ y* $\longleftrightarrow x \geq y$
  using *Lambert-W′-strict-antimono*[*of x y*] *Lambert-W′-strict-antimono*[*of y x*]
  by (*cases x y rule*: *linorder-cases*) *auto*

lemma *Lambert-W′-less-iff* [*simp*]:
  $x \in \{-exp(-1)..<0\} \Longrightarrow y \in \{-exp(-1)..<0\} \Longrightarrow$ *Lambert-W′ x < Lambert-W′ y* $\longleftrightarrow x > y$
  using *Lambert-W′-strict-antimono*[*of x y*] *Lambert-W′-strict-antimono*[*of y x*]
  by (*cases x y rule*: *linorder-cases*) *auto*

lemma *Lambert-W′-le-minus-one*:
  assumes $x \leq -exp(-1)$
  shows *Lambert-W′ x* $= -1$
proof (*cases* $x = -exp(-1)$)
  case *False*
  thus *?thesis* **using** *assms*
    by (*auto simp*: *Lambert-W′-def*)
qed *auto*

lemma *Lambert-W′-ge-zero*: $x \geq 0 \Longrightarrow$ *Lambert-W′ x* $= -1$
  by (*simp add*: *Lambert-W′-def*)

```
lemma Lambert-W'-neg: Lambert-W' x < 0
  by (rule le-less-trans[OF Lambert-W'-le]) auto

lemma Lambert-W'-nz [simp]: Lambert-W' x ≠ 0
  using Lambert-W'-neg[of x] by simp

lemma Lambert-W'-geI:
  assumes y * exp y ≥ x y ≤ −1 x ≥ −exp(−1)
  shows   Lambert-W' x ≥ y
proof −
  from assms have y * exp y < 0
    by (intro mult-neg-pos) auto
  hence Lambert-W' x ≥ Lambert-W' (y * exp y)
    using assms exp-times-self-ge[of y] by (intro Lambert-W'-antimono) auto
  thus ?thesis using assms by simp
qed

lemma Lambert-W'-gtI:
  assumes y * exp y > x y ≤ −1 x ≥ −exp(−1)
  shows   Lambert-W' x ≥ y
proof −
  from assms have y * exp y < 0
    by (intro mult-neg-pos) auto
  hence Lambert-W' x > Lambert-W' (y * exp y)
   using assms exp-times-self-ge[of y] by (intro Lambert-W'-strict-antimono) auto
  thus ?thesis using assms by simp
qed

lemma Lambert-W'-leI:
  assumes y * exp y ≤ x x < 0
  shows   Lambert-W' x ≤ y
proof (cases y ≤ −1)
  case True
  have Lambert-W' x ≤ Lambert-W' (y * exp y)
    using assms exp-times-self-ge[of y] by (intro Lambert-W'-antimono) auto
  thus ?thesis using assms True by simp
next
  case False
  have Lambert-W' x ≤ −1
    by (rule Lambert-W'-le)
  also have ... < y
    using False by simp
  finally show ?thesis by simp
qed

lemma Lambert-W'-lessI:
  assumes y * exp y < x x < 0
  shows   Lambert-W' x < y
```

```
proof (cases y ≤ −1)
  case True
  have Lambert-W′ x < Lambert-W′ (y ∗ exp y)
   using assms exp-times-self-ge[of y] by (intro Lambert-W′-strict-antimono) auto
  thus ?thesis using assms True by simp
next
  case False
  have Lambert-W′ x ≤ −1
    by (rule Lambert-W′-le)
  also have . . . < y
    using False by simp
  finally show ?thesis by simp
qed


lemma bij-betw-exp-times-self-atLeastAtMost:
  fixes a b :: real
  assumes a ≥ −1 a ≤ b
  shows   bij-betw (λx. x ∗ exp x) {a..b} {a ∗ exp a..b ∗ exp b}
  unfolding bij-betw-def
proof
  show inj-on (λx. x ∗ exp x) {a..b}
    by (rule inj-on-subset[OF exp-times-self-inj]) (use assms in auto)
next
  show (λx. x ∗ exp x) ‘ {a..b} = {a ∗ exp a..b ∗ exp b}
  proof safe
    fix x assume x ∈ {a..b}
    thus x ∗ exp x ∈ {a ∗ exp a..b ∗ exp b}
      using assms by (auto intro!: exp-times-self-mono)
  next
    fix x assume x: x ∈ {a ∗ exp a..b ∗ exp b}
    have (−1) ∗ exp (−1) ≤ a ∗ exp a
      using assms by (intro exp-times-self-mono) auto
    also have . . . ≤ x using x by simp
    finally have x ≥ −exp (−1) by simp

    have Lambert-W x ∈ {a..b}
        using x ‹x ≥ −exp (−1)› assms by (auto intro!: Lambert-W-geI Lambert-W-leI)
    moreover have Lambert-W x ∗ exp (Lambert-W x) = x
      using ‹x ≥ −exp (−1)› by (simp add: Lambert-W-times-exp-self)
    ultimately show x ∈ (λx. x ∗ exp x) ‘ {a..b}
      unfolding image-iff by metis
  qed
qed

lemma bij-betw-exp-times-self-atLeastAtMost′:
  fixes a b :: real
  assumes a ≤ b b ≤ −1
```

```
  shows   bij-betw (λx. x ∗ exp x) {a..b} {b ∗ exp b..a ∗ exp a}
  unfolding bij-betw-def
proof
  show inj-on (λx. x ∗ exp x) {a..b}
    by (rule inj-on-subset[OF exp-times-self-inj']) (use assms in auto)
next
  show (λx. x ∗ exp x) ‘ {a..b} = {b ∗ exp b..a ∗ exp a}
  proof safe
    fix x assume x ∈ {a..b}
    thus x ∗ exp x ∈ {b ∗ exp b..a ∗ exp a}
      using assms by (auto intro!: exp-times-self-antimono)
  next
    fix x assume x: x ∈ {b ∗ exp b..a ∗ exp a}
    from assms have a ∗ exp a < 0
      by (intro mult-neg-pos) auto
    with x have x < 0 by auto
    have (−1) ∗ exp (−1) ≤ b ∗ exp b
      using assms by (intro exp-times-self-antimono) auto
    also have . . . ≤ x using x by simp
    finally have x ≥ −exp (−1) by simp

    have Lambert-W' x ∈ {a..b}
      using x ‹x ≥ −exp (−1)› ‹x < 0› assms
      by (auto intro!: Lambert-W'-geI Lambert-W'-leI)
    moreover have Lambert-W' x ∗ exp (Lambert-W' x) = x
      using ‹x ≥ −exp (−1)› ‹x < 0› by (auto simp: Lambert-W'-times-exp-self)
    ultimately show x ∈ (λx. x ∗ exp x) ‘ {a..b}
      unfolding image-iff by metis
  qed
qed

lemma bij-betw-exp-times-self-atLeast:
  fixes a :: real
  assumes a ≥ −1
  shows   bij-betw (λx. x ∗ exp x) {a..} {a ∗ exp a..}
  unfolding bij-betw-def
proof
  show inj-on (λx. x ∗ exp x) {a..}
    by (rule inj-on-subset[OF exp-times-self-inj]) (use assms in auto)
next
  show (λx. x ∗ exp x) ‘ {a..} = {a ∗ exp a..}
  proof safe
    fix x assume x ≥ a
    thus x ∗ exp x ≥ a ∗ exp a
      using assms by (auto intro!: exp-times-self-mono)
  next
    fix x assume x: x ≥ a ∗ exp a
    have (−1) ∗ exp (−1) ≤ a ∗ exp a
      using assms by (intro exp-times-self-mono) auto
```

also have ... $\leq x$ **using** x **by** *simp*
finally have $x \geq -exp\ (-1)$ **by** *simp*

have *Lambert-W x* $\in$ {*a*..}
using x ‹$x \geq -exp\ (-1)$› *assms* **by** (*auto intro*!: *Lambert-W-geI Lambert-W-leI*)
moreover have *Lambert-W x* $*$ *exp* (*Lambert-W x*) = *x*
using ‹$x \geq -exp\ (-1)$› **by** (*simp add*: *Lambert-W-times-exp-self*)
ultimately show $x \in (\lambda x.\ x * exp\ x)$ ‘ {*a*..}
unfolding *image-iff* **by** *metis*
qed
qed

1.4 Basic identities and bounds

lemma *Lambert-W-2-ln-2* [*simp*]: *Lambert-W* (*2* $*$ *ln 2*) = *ln 2*
proof −
have $-1 \leq$ (*0* :: *real*)
by *simp*
also have ... $\leq$ *ln 2*
by *simp*
finally have $-1 \leq$ (*ln 2* :: *real*) **.**
thus *?thesis*
by (*intro Lambert-W-eqI*) *auto*
qed

lemma *Lambert-W-exp-1* [*simp*]: *Lambert-W* (*exp 1*) = *1*
by (*rule Lambert-W-eqI*) *auto*

lemma *Lambert-W-neg-ln-over-self*:
assumes $x \in$ {*exp* (−*1*)..*exp 1*}
shows *Lambert-W* (−*ln x* / *x*) = −*ln x*
proof −
have *0* < (*exp* (−*1*) :: *real*)
by *simp*
also have ... $\leq x$
using *assms* **by** *simp*
finally have $x > 0$ **.**
from ‹$x > 0$› *assms* **have** *ln x* $\leq$ *ln* (*exp 1*)
by (*subst ln-le-cancel-iff*) *auto*
also have *ln* (*exp 1*) = (*1* :: *real*)
by *simp*
finally have *ln x* $\leq$ *1* **.**
show *?thesis*
using *assms* ‹$x > 0$› ‹*ln x* $\leq$ *1*›
by (*intro Lambert-W-eqI*) (*auto simp*: *exp-minus field-simps*)
qed

lemma *Lambert-W'-neg-ln-over-self*:

assumes $x \geq exp\ 1$
shows $Lambert\text{-}W'\ (-ln\ x\ /\ x) = -ln\ x$
proof (*rule Lambert-W′-eqI*)
have $0 < (exp\ 1 :: real)$
by *simp*
also have $\ldots \leq x$
by *fact*
finally have $x > 0$ **.**
from *assms* ‹$x > 0$› **have** $ln\ x \geq ln\ (exp\ 1)$
by (*subst ln-le-cancel-iff*) *auto*
thus $-ln\ x \leq -1$ **by** *simp*
show $-ln\ x * exp\ (-ln\ x) = -ln\ x\ /\ x$
using ‹$x > 0$› **by** (*simp add*: *field-simps exp-minus*)
qed

lemma *exp-Lambert-W*: $x \geq -exp\ (-1) \Longrightarrow x \neq 0 \Longrightarrow exp\ (Lambert\text{-}W\ x) = x\ /\ Lambert\text{-}W\ x$
using *Lambert-W-times-exp-self*[*of x*] **by** (*auto simp add*: *divide-simps mult-ac*)

lemma *exp-Lambert-W′*: $x \in \{-exp\ (-1)..<0\} \Longrightarrow exp\ (Lambert\text{-}W'\ x) = x\ /\ Lambert\text{-}W'\ x$
using *Lambert-W′-times-exp-self*[*of x*] **by** (*auto simp add*: *divide-simps mult-ac*)

lemma *ln-Lambert-W*:
assumes $x > 0$
shows $ln\ (Lambert\text{-}W\ x) = ln\ x - Lambert\text{-}W\ x$
proof −
have $-exp\ (-1) \leq (0 :: real)$
by *simp*
also have $\ldots < x$ **by** *fact*
finally have *x*: $x > -exp(-1)$ **.**

have $exp\ (ln\ (Lambert\text{-}W\ x)) = exp\ (ln\ x - Lambert\text{-}W\ x)$
using *assms x* **by** (*subst exp-diff*) (*auto simp*: *exp-Lambert-W*)
thus *?thesis* **by** (*subst* (*asm*) *exp-inj-iff*)
qed

lemma *ln-minus-Lambert-W′*:
assumes $x \in \{-exp\ (-1)..<0\}$
shows $ln\ (-Lambert\text{-}W'\ x) = ln\ (-x) - Lambert\text{-}W'\ x$
proof −
have $exp\ (ln\ (-x) - Lambert\text{-}W'\ x) = -Lambert\text{-}W'\ x$
using *assms* **by** (*simp add*: *exp-diff exp-Lambert-W′*)
also have $\ldots = exp\ (ln\ (-Lambert\text{-}W'\ x))$
using *Lambert-W′-neg*[*of x*] **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma *Lambert-W-plus-Lambert-W-eq*:

assumes $x > 0$ $y > 0$
shows *Lambert-W x + Lambert-W y = Lambert-W (x ∗ y ∗ (1 / Lambert-W x + 1 / Lambert-W y))*
proof (*rule sym*, *rule Lambert-W-eqI*)
have $x > -exp(-1)$ $y > -exp\ (-1)$
by (*rule less-trans*[*OF - assms*(*1*)] *less-trans*[*OF - assms*(*2*)], *simp*)+
with *assms* **show** (*Lambert-W x + Lambert-W y*) ∗ *exp* (*Lambert-W x + Lambert-W y*) =
x ∗ y ∗ (*1 / Lambert-W x + 1 / Lambert-W y*)
by (*auto simp*: *field-simps exp-add exp-Lambert-W*)
have $-1 \leq (0 :: real)$
by *simp*
also from *assms* **have** ... ≤ *Lambert-W x + Lambert-W y*
by (*intro add-nonneg-nonneg*) *auto*
finally show ... ≥ -1 **.**
qed

lemma *Lambert-W′-plus-Lambert-W′-eq*:
assumes $x \in \{-exp(-1)..<0\}$ $y \in \{-exp(-1)..<0\}$
shows *Lambert-W′ x + Lambert-W′ y = Lambert-W′ (x ∗ y ∗ (1 / Lambert-W′ x + 1 / Lambert-W′ y))*
proof (*rule sym*, *rule Lambert-W′-eqI*)
from *assms* **show** (*Lambert-W′ x + Lambert-W′ y*) ∗ *exp* (*Lambert-W′ x + Lambert-W′ y*) =
x ∗ y ∗ (*1 / Lambert-W′ x + 1 / Lambert-W′ y*)
by (*auto simp*: *field-simps exp-add exp-Lambert-W′*)
have *Lambert-W′ x + Lambert-W′ y* $\leq -1 + -1$
by (*intro add-mono Lambert-W′-le*)
also have ... ≤ -1 **by** *simp*
finally show *Lambert-W′ x + Lambert-W′ y* ≤ -1 **.**
qed

lemma *Lambert-W-gt-ln-minus-ln-ln*:
assumes $x > exp\ 1$
shows *Lambert-W x* $> ln\ x - ln\ (ln\ x)$
proof (*rule Lambert-W-gtI*)
have $x > 1$
by (*rule less-trans*[*OF - assms*]) *auto*
have $ln\ x > ln\ (exp\ 1)$
by (*subst ln-less-cancel-iff*) (*use* ‹$x > 1$› *assms* **in** *auto*)
thus $(ln\ x - ln\ (ln\ x)) * exp\ (ln\ x - ln\ (ln\ x)) < x$
using *assms* ‹$x > 1$› **by** (*simp add*: *exp-diff field-simps*)
qed

lemma *Lambert-W-less-ln*:
assumes $x > exp\ 1$
shows *Lambert-W x* $< ln\ x$
proof (*rule Lambert-W-lessI*)
have $x > 0$

by (*rule less-trans*[*OF* - *assms*]) *auto*
have *ln x* > *ln* (*exp 1*)
by (*subst ln-less-cancel-iff*) (*use* ‹*x* > *0*› *assms* **in** *auto*)
thus *x* < *ln x* ∗ *exp* (*ln x*)
using ‹*x* > *0*› **by** *simp*
show *ln x* ≥ −*1*
by (*rule less-imp-le*[*OF le-less-trans*[*OF* - ‹*ln x* > -›]]) *auto*
show *x* ≥ −*exp* (−*1*)
by (*rule less-imp-le*[*OF le-less-trans*[*OF* - ‹*x* > *0*›]]) *auto*
qed

1.5 Limits, continuity, and differentiability

lemma *filterlim-Lambert-W-at-top* [*tendsto-intros*]: *filterlim Lambert-W at-top at-top*
unfolding *filterlim-at-top*
proof
fix *C* :: *real*
have *eventually* (λ*x*. *x* ≥ *C* ∗ *exp C*) *at-top*
by (*rule eventually-ge-at-top*)
thus *eventually* (λ*x*. *Lambert-W x* ≥ *C*) *at-top*
proof *eventually-elim*
case (*elim x*)
thus *?case*
by (*intro Lambert-W-geI*) *auto*
qed
qed

lemma *filterlim-Lambert-W-at-left-0* [*tendsto-intros*]:
filterlim Lambert-W′ at-bot (*at-left 0*)
unfolding *filterlim-at-bot*
proof
fix *C* :: *real*
define *C′* **where** *C′* = *min C* (−*1*)
have *C′* < *0* *C′* ≤ *C*
by (*simp-all add*: *C′-def*)
have *C′* ∗ *exp C′* < *0*
using ‹*C′* < *0*› **by** (*intro mult-neg-pos*) *auto*
hence *eventually* (λ*x*. *x* ≥ *C′* ∗ *exp C′*) (*at-left 0*)
by *real-asymp*
moreover have *eventually* (λ*x*::*real*. *x* < *0*) (*at-left 0*)
by *real-asymp*
ultimately show *eventually* (λ*x*. *Lambert-W′ x* ≤ *C*) (*at-left 0*)
proof *eventually-elim*
case (*elim x*)
hence *Lambert-W′ x* ≤ *C′*
by (*intro Lambert-W′-leI*) *auto*
also have … ≤ *C* **by** *fact*
finally show *?case* **.**
qed

```
qed

lemma continuous-on-Lambert-W [continuous-intros]: continuous-on {−exp (−1)..}
Lambert-W
proof −
  have ∗: continuous-on {−exp (−1)..b ∗ exp b} Lambert-W if b ≥ 0 for b
  proof −
    have continuous-on ((λx. x ∗ exp x) ‘ {−1..b}) Lambert-W
      by (rule continuous-on-inv) (auto intro!: continuous-intros)
    also have (λx. x ∗ exp x) ‘ {−1..b} = {−exp (−1)..b ∗ exp b}
      using bij-betw-exp-times-self-atLeastAtMost[of −1 b] ‹b ≥ 0›
      by (simp add: bij-betw-def)
    finally show ?thesis .
  qed

  have continuous (at x) Lambert-W if x ≥ 0 for x
  proof −
    have x: −exp (−1) < x
      by (rule less-le-trans[OF - that]) auto

    define b where b = Lambert-W x + 1
    have b ≥ 0
      using Lambert-W-ge[of x] by (simp add: b-def)
    have x = Lambert-W x ∗ exp (Lambert-W x)
      using that x by (subst Lambert-W-times-exp-self) auto
    also have ... < b ∗ exp b
      by (intro exp-times-self-strict-mono) (auto simp: b-def Lambert-W-ge)
    finally have b ∗ exp b > x .
    have continuous-on {−exp(−1)<..<b ∗ exp b} Lambert-W
      by (rule continuous-on-subset[OF ∗[of b]]) (use ‹b ≥ 0› in auto)
    moreover have x ∈ {−exp(−1)<..<b ∗ exp b}
      using ‹b ∗ exp b > x› x by auto
    ultimately show continuous (at x) Lambert-W
      by (subst (asm) continuous-on-eq-continuous-at) auto
  qed
  hence continuous-on {0..} Lambert-W
    by (intro continuous-at-imp-continuous-on) auto
  moreover have continuous-on {−exp (−1)..0} Lambert-W
    using ∗[of 0] by simp
  ultimately have continuous-on ({−exp (−1)..0} ∪ {0..}) Lambert-W
    by (intro continuous-on-closed-Un) auto
  also have {−exp (−1)..0} ∪ {0..} = {−exp (−1::real)..}
    using order.trans[of −exp (−1)::real 0] by auto
  finally show ?thesis .
qed

lemma continuous-on-Lambert-W-alt [continuous-intros]:
  assumes continuous-on A f ⋀x. x ∈ A ⟹ f x ≥ −exp (−1)
  shows   continuous-on A (λx. Lambert-W (f x))
```

```
  using continuous-on-compose2[OF continuous-on-Lambert-W assms(1)] assms
by auto

lemma continuous-on-Lambert-W' [continuous-intros]: continuous-on {−exp (−1)..<0}
Lambert-W'
proof −
 have ∗: continuous-on {−exp (−1)..−b ∗ exp (−b)} Lambert-W' if b ≥ 1 for b
 proof −
  have continuous-on ((λx. x ∗ exp x) ‘ {−b..−1}) Lambert-W'
   by (intro continuous-on-inv ballI) (auto intro!: continuous-intros)
  also have (λx. x ∗ exp x) ‘ {−b..−1} = {−exp (−1)..−b ∗ exp (−b)}
   using bij-betw-exp-times-self-atLeastAtMost'[of −b −1] that
   by (simp add: bij-betw-def)
  finally show ?thesis .
 qed

 have continuous (at x) Lambert-W' if x > −exp (−1) x < 0 for x
 proof −
  define b where b = Lambert-W x + 1
  have eventually (λb. −b ∗ exp (−b) > x) at-top
   using that by real-asymp
  hence eventually (λb. b ≥ 1 ∧ −b ∗ exp (−b) > x) at-top
   by (intro eventually-conj eventually-ge-at-top)
  then obtain b where b: b ≥ 1 −b ∗ exp (−b) > x
   by (auto simp: eventually-at-top-linorder)

  have continuous-on {−exp(−1)<..<−b ∗ exp (−b)} Lambert-W'
   by (rule continuous-on-subset[OF ∗[of b]]) (use ‹b ≥ 1› in auto)
  moreover have x ∈ {−exp(−1)<..<−b ∗ exp (−b)}
   using b that by auto
  ultimately show continuous (at x) Lambert-W'
   by (subst (asm) continuous-on-eq-continuous-at) auto
 qed
 hence ∗∗: continuous-on {−exp (−1)<..<0} Lambert-W'
  by (intro continuous-at-imp-continuous-on) auto

 show ?thesis
  unfolding continuous-on-def
 proof
  fix x :: real assume x: x ∈ {−exp(−1)..<0}
  show (Lambert-W' ⟶ Lambert-W' x) (at x within {−exp(−1)..<0})
  proof (cases x = −exp(−1))
   case False
   hence isCont Lambert-W' x
    using x ∗∗ by (auto simp: continuous-on-eq-continuous-at)
   thus ?thesis
    using continuous-at filterlim-within-subset by blast
  next
   case True
```

```
      define a :: real where a = −2 ∗ exp (−2)
      have a: a > −exp (−1)
        using exp-times-self-strict-antimono[of −1 −2] by (auto simp: a-def)
      from True have x ∈ {−exp (−1)..<a}
        using a by (auto simp: a-def)
      have continuous-on {−exp (−1)..<a} Lambert-W′
        unfolding a-def by (rule continuous-on-subset[OF ∗[of 2]]) auto
      hence (Lambert-W′ ⟶ Lambert-W′ x) (at x within {−exp (−1)..<a})
        using ‹x ∈ {−exp (−1)..<a}› by (auto simp: continuous-on-def)
      also have at x within {−exp (−1)..<a} = at-right x
        using a by (intro at-within-nhd[of - {..<a}]) (auto simp: True)
      also have . . . = at x within {−exp (−1)..<0}
        using a by (intro at-within-nhd[of - {..<0}]) (auto simp: True)
      finally show ?thesis .
    qed
  qed
qed

lemma continuous-on-Lambert-W′-alt [continuous-intros]:
  assumes continuous-on A f ⋀x. x ∈ A ⟹ f x ∈ {−exp (−1)..<0}
  shows   continuous-on A (λx. Lambert-W′ (f x))
  using continuous-on-compose2[OF continuous-on-Lambert-W′ assms(1)] assms
  by (auto simp: subset-iff)


lemma tendsto-Lambert-W-1:
  assumes (f ⟶ L) F eventually (λx. f x ≥ −exp (−1)) F
  shows   ((λx. Lambert-W (f x)) ⟶ Lambert-W L) F
proof (cases F = bot)
  case [simp]: False
  from tendsto-lowerbound[OF assms] have L ≥ −exp (−1) by simp
  thus ?thesis
    using continuous-on-tendsto-compose[OF continuous-on-Lambert-W assms(1)]
assms(2) by simp
qed auto

lemma tendsto-Lambert-W-2:
  assumes (f ⟶ L) F L > −exp (−1)
  shows   ((λx. Lambert-W (f x)) ⟶ Lambert-W L) F
  using order-tendstoD(1)[OF assms] assms
  by (intro tendsto-Lambert-W-1) (auto elim: eventually-mono)

lemma tendsto-Lambert-W [tendsto-intros]:
  assumes (f ⟶ L) F eventually (λx. f x ≥ −exp (−1)) F ∨ L > −exp (−1)
  shows   ((λx. Lambert-W (f x)) ⟶ Lambert-W L) F
  using assms(2)
proof
  assume L > −exp (−1)
  from order-tendstoD(1)[OF assms(1) this] assms(1) show ?thesis
```

by (*intro tendsto-Lambert-W-1*) (*auto elim*: *eventually-mono*)
qed (*use tendsto-Lambert-W-1*[*OF assms*(*1*)] **in** *auto*)

lemma *tendsto-Lambert-W′-1*:
assumes ($f \longrightarrow L$) *F* *eventually* ($\lambda x.\ f\ x \geq -exp\ (-1)$) *F* $L < 0$
shows (($\lambda x.$ *Lambert-W′* ($f\ x$)) $\longrightarrow$ *Lambert-W′* L) *F*
proof (*cases* $F = bot$)
case [*simp*]: *False*
from *tendsto-lowerbound*[*OF assms*(*1*,*2*)] **have** *L-ge*: $L \geq -exp\ (-1)$ **by** *simp*
from *order-tendstoD*(*2*)[*OF assms*(*1*,*3*)] **have** *ev*: *eventually* ($\lambda x.\ f\ x < 0$) *F*
by *auto*
with *assms*(*2*) **have** *eventually* ($\lambda x.\ f\ x \in \{-exp\ (-1)..<0\}$) *F*
by *eventually-elim auto*
thus *?thesis* **using** *L-ge assms*(*3*)
by (*intro continuous-on-tendsto-compose*[*OF continuous-on-Lambert-W′ assms*(*1*)])
auto
qed *auto*

lemma *tendsto-Lambert-W′-2*:
assumes ($f \longrightarrow L$) *F* $L > -exp\ (-1)$ $L < 0$
shows (($\lambda x.$ *Lambert-W′* ($f\ x$)) $\longrightarrow$ *Lambert-W′* L) *F*
using *order-tendstoD*(*1*)[*OF assms*(*1*,*2*)] *assms*
by (*intro tendsto-Lambert-W′-1*) (*auto elim*: *eventually-mono*)

lemma *tendsto-Lambert-W′* [*tendsto-intros*]:
assumes ($f \longrightarrow L$) *F* *eventually* ($\lambda x.\ f\ x \geq -exp\ (-1)$) *F* $\vee$ $L > -exp\ (-1)$
$L < 0$
shows (($\lambda x.$ *Lambert-W′* ($f\ x$)) $\longrightarrow$ *Lambert-W′* L) *F*
using *assms*(*2*)
proof
assume $L > -exp\ (-1)$
from *order-tendstoD*(*1*)[*OF assms*(*1*) *this*] *assms*(*1*,*3*) **show** *?thesis*
by (*intro tendsto-Lambert-W′-1*) (*auto elim*: *eventually-mono*)
qed (*use tendsto-Lambert-W′-1*[*OF assms*(*1*) - *assms*(*3*)] **in** *auto*)

lemma *continuous-Lambert-W* [*continuous-intros*]:
assumes *continuous F f* f (*Lim F* ($\lambda x.\ x$)) $> -exp\ (-1)$ $\vee$ *eventually* ($\lambda x.\ f\ x$
$\geq -exp\ (-1)$) *F*
shows *continuous F* ($\lambda x.$ *Lambert-W* ($f\ x$))
using *assms* **unfolding** *continuous-def* **by** (*intro tendsto-Lambert-W*) *auto*

lemma *continuous-Lambert-W′* [*continuous-intros*]:
assumes *continuous F f* f (*Lim F* ($\lambda x.\ x$)) $> -exp\ (-1)$ $\vee$ *eventually* ($\lambda x.\ f\ x$
$\geq -exp\ (-1)$) *F*
f (*Lim F* ($\lambda x.\ x$)) < 0
shows *continuous F* ($\lambda x.$ *Lambert-W′* ($f\ x$))
using *assms* **unfolding** *continuous-def* **by** (*intro tendsto-Lambert-W′*) *auto*

```
lemma has-field-derivative-Lambert-W [derivative-intros]:
  assumes x: x > −exp (−1)
  shows   (Lambert-W has-real-derivative inverse (x + exp (Lambert-W x))) (at x
within A)
proof −
  write Lambert-W (‹W›)
  from x have W x > W (−exp (−1))
    by (subst Lambert-W-less-iff) auto
  hence W x > −1 by simp

  note [derivative-intros] = DERIV-inverse-function[where g = Lambert-W]
  have ((λx. x ∗ exp x) has-real-derivative (1 + W x) ∗ exp (W x)) (at (W x))
    by (auto intro!: derivative-eq-intros simp: algebra-simps)
  hence (W has-real-derivative inverse ((1 + W x) ∗ exp (W x))) (at x)
    by (rule DERIV-inverse-function[where a = −exp (−1) and b = x + 1])
       (use x ‹W x > −1› in ‹auto simp: Lambert-W-times-exp-self Lim-ident-at
                          intro!: continuous-intros›)
  also have (1 + W x) ∗ exp (W x) = x + exp (W x)
    using x by (simp add: algebra-simps Lambert-W-times-exp-self)
  finally show ?thesis by (rule has-field-derivative-at-within)
qed

lemma has-field-derivative-Lambert-W-gen [derivative-intros]:
  assumes (f has-real-derivative f′) (at x within A) f x > −exp (−1)
  shows   ((λx. Lambert-W (f x)) has-real-derivative
            (f′ / (f x + exp (Lambert-W (f x))))) (at x within A)
 using DERIV-chain2[OF has-field-derivative-Lambert-W[OF assms(2)] assms(1)]
  by (simp add: field-simps)

lemma has-field-derivative-Lambert-W′ [derivative-intros]:
  assumes x: x ∈ {−exp (−1)<..<0}
  shows   (Lambert-W′ has-real-derivative inverse (x + exp (Lambert-W′ x))) (at
x within A)
proof −
  write Lambert-W′ (‹W›)
  from x have W x < W (−exp (−1))
    by (subst Lambert-W′-less-iff) auto
  hence W x < −1 by simp

  note [derivative-intros] = DERIV-inverse-function[where g = Lambert-W]
  have ((λx. x ∗ exp x) has-real-derivative (1 + W x) ∗ exp (W x)) (at (W x))
    by (auto intro!: derivative-eq-intros simp: algebra-simps)
  hence (W has-real-derivative inverse ((1 + W x) ∗ exp (W x))) (at x)
    by (rule DERIV-inverse-function[where a = −exp (−1) and b = 0])
       (use x ‹W x < −1› in ‹auto simp: Lambert-W′-times-exp-self Lim-ident-at
                            intro!: continuous-intros›)
  also have (1 + W x) ∗ exp (W x) = x + exp (W x)
    using x by (simp add: algebra-simps Lambert-W′-times-exp-self)
```

```
  finally show ?thesis by (rule has-field-derivative-at-within)
qed

lemma has-field-derivative-Lambert-W′-gen [derivative-intros]:
  assumes (f has-real-derivative f′) (at x within A) f x ∈ {−exp (−1)<..<0}
  shows   ((λx. Lambert-W′ (f x)) has-real-derivative
            (f′ / (f x + exp (Lambert-W′ (f x))))) (at x within A)
 using DERIV-chain2[OF has-field-derivative-Lambert-W′[OF assms(2)] assms(1)]
  by (simp add: field-simps)
```

1.6 Asymptotic expansion

Lastly, we prove some more detailed asymptotic expansions of W and W' at their singularities. First, we show that:

$$W(x) = \log x - \log\log x + o(\log\log x) \qquad \text{for } x \to \infty$$
$$W'(x) = \log(-x) - \log(-\log(-x)) + o(\log(-\log(-x))) \qquad \text{for } x \to 0^-$$

```
theorem Lambert-W-asymp-equiv-at-top:
  (λx. Lambert-W x − ln x) ∼[at-top] (λx. −ln (ln x))
proof −
  have (λx. Lambert-W x − ln x) ∼[at-top] (λx. (−1) ∗ ln (ln x))
  proof (rule asymp-equiv-sandwich′)
    fix c′ :: real assume c′: c′ ∈ {−2<..<−1}
    have eventually (λx. (ln x + c′ ∗ ln (ln x)) ∗ exp (ln x + c′ ∗ ln (ln x)) ≤ x)
at-top
         eventually (λx. ln x + c′ ∗ ln (ln x) ≥ −1) at-top
      using c′ by real-asymp+
    thus eventually (λx. Lambert-W x − ln x ≥ c′ ∗ ln (ln x)) at-top
    proof eventually-elim
      case (elim x)
      hence Lambert-W x ≥ ln x + c′ ∗ ln (ln x)
        by (intro Lambert-W-geI)
      thus ?case by simp
    qed
  next
    fix c′ :: real assume c′: c′ ∈ {−1<..<0}
    have eventually (λx. (ln x + c′ ∗ ln (ln x)) ∗ exp (ln x + c′ ∗ ln (ln x)) ≥ x)
at-top
         eventually (λx. ln x + c′ ∗ ln (ln x) ≥ −1) at-top
      using c′ by real-asymp+
    thus eventually (λx. Lambert-W x − ln x ≤ c′ ∗ ln (ln x)) at-top
      using eventually-ge-at-top[of −exp (−1)]
    proof eventually-elim
      case (elim x)
      hence Lambert-W x ≤ ln x + c′ ∗ ln (ln x)
        by (intro Lambert-W-leI)
      thus ?case by simp
    qed
```

```
  qed auto
  thus ?thesis by simp
qed

lemma Lambert-W-asymp-equiv-at-top′ [asymp-equiv-intros]:
  Lambert-W ∼[at-top] ln
proof −
  have (λx. Lambert-W x − ln x) ∈ Θ(λx. −ln (ln x))
    by (intro asymp-equiv-imp-bigtheta Lambert-W-asymp-equiv-at-top)
  also have (λx::real. −ln (ln x)) ∈ o(ln)
    by real-asymp
  finally show ?thesis by (simp add: asymp-equiv-altdef)
qed

theorem Lambert-W′-asymp-equiv-at-left-0:
  (λx. Lambert-W′ x − ln (−x)) ∼[at-left 0] (λx. −ln (−ln (−x)))
proof −
  have (λx. Lambert-W′ x − ln (−x)) ∼[at-left 0] (λx. (−1) ∗ ln (−ln (−x)))
  proof (rule asymp-equiv-sandwich′)
    fix c′ :: real assume c′: c′ ∈ {−2<..<−1}
    have eventually (λx. x ≤ (ln (−x) + c′ ∗ ln (−ln (−x))) ∗ exp (ln (−x) + c′
∗ ln (−ln (−x)))) (at-left 0)
         eventually (λx::real. ln (−x) + c′ ∗ ln (−ln (−x)) ≤ −1) (at-left 0)
         eventually (λx::real. −exp (−1) ≤ x) (at-left 0)
      using c′ by real-asymp+
    thus eventually (λx. Lambert-W′ x − ln (−x) ≥ c′ ∗ ln (−ln (−x))) (at-left 0)
    proof eventually-elim
      case (elim x)
      hence Lambert-W′ x ≥ ln (−x) + c′ ∗ ln (−ln (−x))
        by (intro Lambert-W′-geI)
      thus ?case by simp
    qed
  next
    fix c′ :: real assume c′: c′ ∈ {−1<..<0}
    have eventually (λx. x ≥ (ln (−x) + c′ ∗ ln (−ln (−x))) ∗ exp (ln (−x) + c′
∗ ln (−ln (−x)))) (at-left 0)
      using c′ by real-asymp
    moreover have eventually (λx::real. x < 0) (at-left 0)
      by (auto simp: eventually-at intro: exI[of - 1])
     ultimately show eventually (λx. Lambert-W′ x − ln (−x) ≤ c′ ∗ ln (−ln
(−x))) (at-left 0)
    proof eventually-elim
      case (elim x)
      hence Lambert-W′ x ≤ ln (−x) + c′ ∗ ln (−ln (−x))
        by (intro Lambert-W′-leI)
      thus ?case by simp
    qed
  qed auto
  thus ?thesis by simp
```

qed

lemma *Lambert-W′-asymp-equiv′-at-left-0* [*asymp-equiv-intros*]:
  Lambert-W′ ∼[*at-left 0*] (λ*x*. *ln* (−*x*))
proof −
  have (λ*x*. *Lambert-W′ x* − *ln* (−*x*)) ∈ Θ[*at-left 0*](λ*x*. −*ln* (−*ln* (−*x*)))
    by (*intro asymp-equiv-imp-bigtheta Lambert-W′-asymp-equiv-at-left-0*)
  also have (λ*x*::*real*. −*ln* (−*ln* (−*x*))) ∈ *o*[*at-left 0*](λ*x*. *ln* (−*x*))
    by *real-asymp*
  finally show *?thesis* **by** (*simp add*: *asymp-equiv-altdef*)
qed

Next, we look at the branching point $a := \frac{1}{e}$. Here, the asymptotic behaviour is as follows:

$$W(x) = -1 + \sqrt{2e}(x-a)^{\frac{1}{2}} - \tfrac{2}{3}e(x-a) + o(x-a) \qquad \text{for} x \to a^+$$
$$W'(x) = -1 - \sqrt{2e}(x-a)^{\frac{1}{2}} - \tfrac{2}{3}e(x-a) + o(x-a) \qquad \text{for} x \to a^+$$

lemma *sqrt-sqrt-mult*:
  assumes *x* ≥ (*0* :: *real*)
  shows   *sqrt x* ∗ (*sqrt x* ∗ *y*) = *x* ∗ *y*
  using *assms* **by** (*subst mult.assoc* [*symmetric*]) *auto*

theorem *Lambert-W-asymp-equiv-at-right-minus-exp-minus1*:
  defines *e* ≡ *exp 1*
  defines *a* ≡ −*exp* (−*1*)
  defines *C1* ≡ *sqrt* (*2* ∗ *exp 1*)
  defines *f* ≡ (λ*x*. −*1* + *C1* ∗ *sqrt* (*x* − *a*))
  shows   (λ*x*. *Lambert-W x* − *f x*) ∼[*at-right a*] (λ*x*. −*2*/*3* ∗ *e* ∗ (*x* − *a*))
proof −
  define *C* :: *real* ⇒ *real* **where** *C* = (λ*c*. *sqrt* (*2*/*e*)/*3* ∗ (*2*∗*e*+*3*∗*c*))
  have *asymp-equiv*: (λ*x*. (*f x* + *c* ∗ (*x* − *a*)) ∗ *exp* (*f x* + *c* ∗ (*x* − *a*)) − *x*)
                    ∼[*at-right a*] (λ*x*. *C c* ∗ (*x* − *a*) *powr* (*3*/*2*)) **if** *c* ≠ −*2*/*3* ∗ *e*
for *c*
  proof −
    from *that* **have** *C c* ≠ *0*
      by (*auto simp*: *C-def e-def*)
    have (λ*x*. (*f x* + *c* ∗ (*x* − *a*)) ∗ *exp* (*f x* + *c* ∗ (*x* − *a*)) − *x* − *C c* ∗ (*x* − *a*)
powr (*3*/*2*))
          ∈ *o*[*at-right a*](λ*x*. (*x* − *a*) *powr* (*3*/*2*))
      unfolding *f-def a-def C-def C1-def e-def*
      by (*real-asymp simp*: *field-simps real-sqrt-mult real-sqrt-divide sqrt-sqrt-mult*
                    exp-minus simp flip: *sqrt-def*)
    thus *?thesis*
      using ‹*C c* ≠ *0*› **by** (*intro smallo-imp-asymp-equiv*) *auto*
  qed

  show *?thesis*
  proof (*rule asymp-equiv-sandwich′*)

fix c' :: *real* **assume** c': $c' \in \{-e<..<-2/3*e\}$
hence *neq*: $c' \neq -2/3 * e$ **by** *auto*
from c' **have** *neg*: $C\ c' < 0$ **unfolding** *C-def* **by** (*auto intro*!: *mult-pos-neg*)
hence *eventually* ($\lambda x.\ C\ c' * (x - a)$ *powr* $(3\ /\ 2) < 0$) (*at-right a*)
by *real-asymp*
hence *eventually* ($\lambda x.\ (f\ x + c' * (x - a)) * exp\ (f\ x + c' * (x - a)) - x < 0$) (*at-right a*)
using *asymp-equiv-eventually-neg-iff*[*OF asymp-equiv*[*OF neq*]]
by *eventually-elim* (*use neg* **in** *auto*)
thus *eventually* ($\lambda x.$ *Lambert-W* $x - f\ x \geq c' * (x - a)$) (*at-right a*)
proof *eventually-elim*
case (*elim x*)
hence *Lambert-W* $x \geq f\ x + c' * (x - a)$
by (*intro Lambert-W-geI*) *auto*
thus *?case* **by** *simp*
qed
next
fix c' :: *real* **assume** c': $c' \in \{-2/3*e<..<0\}$
hence *neq*: $c' \neq -2/3 * e$ **by** *auto*
from c' **have** *pos*: $C\ c' > 0$ **unfolding** *C-def* **by** *auto*
hence *eventually* ($\lambda x.\ C\ c' * (x - a)$ *powr* $(3\ /\ 2) > 0$) (*at-right a*)
by *real-asymp*
hence *eventually* ($\lambda x.\ (f\ x + c' * (x - a)) * exp\ (f\ x + c' * (x - a)) - x > 0$) (*at-right a*)
using *asymp-equiv-eventually-pos-iff*[*OF asymp-equiv*[*OF neq*]]
by *eventually-elim* (*use pos* **in** *auto*)
moreover have *eventually* ($\lambda x.\ -1 \leq f\ x + c' * (x - a)$) (*at-right a*)
eventually ($\lambda x.\ x > a$) (*at-right a*)
unfolding *a-def f-def C1-def* c' **by** *real-asymp+*
ultimately show *eventually* ($\lambda x.$ *Lambert-W* $x - f\ x \leq c' * (x - a)$) (*at-right a*)
proof *eventually-elim*
case (*elim x*)
hence *Lambert-W* $x \leq f\ x + c' * (x - a)$
by (*intro Lambert-W-leI*) (*auto simp*: *a-def*)
thus *?case* **by** *simp*
qed
qed (*auto simp*: *e-def*)
qed

theorem *Lambert-W'-asymp-equiv-at-right-minus-exp-minus1*:
defines $e \equiv exp\ 1$
defines $a \equiv -exp\ (-1)$
defines $C1 \equiv sqrt\ (2 * exp\ 1)$
defines $f \equiv (\lambda x.\ -1 - C1 * sqrt\ (x - a))$
shows ($\lambda x.$ *Lambert-W'* $x - f\ x$) $\sim$[*at-right a*] ($\lambda x.\ -2/3 * e * (x - a)$)
proof $-$
define C :: *real* $\Rightarrow$ *real* **where** $C = (\lambda c.\ -sqrt\ (2/e)/3 * (2*e+3*c))$

have *asymp-equiv*: $(\lambda x.\ (f\ x + c * (x - a)) * exp\ (f\ x + c * (x - a)) - x)$
$\sim$[*at-right a*] $(\lambda x.\ C\ c * (x - a)\ powr\ (3/2))$ **if** $c \neq -2/3 * e$
for *c*
proof –
from *that* **have** $C\ c \neq 0$
by (*auto simp*: *C-def e-def*)
have $(\lambda x.\ (f\ x + c * (x - a)) * exp\ (f\ x + c * (x - a)) - x - C\ c * (x - a)\ powr\ (3/2))$
$\in o$[*at-right a*]$(\lambda x.\ (x - a)\ powr\ (3/2))$
unfolding *f-def a-def C-def C1-def e-def*
by (*real-asymp simp*: *field-simps real-sqrt-mult real-sqrt-divide sqrt-sqrt-mult exp-minus simp flip*: *sqrt-def*)
thus *?thesis*
using ‹$C\ c \neq 0$› **by** (*intro smallo-imp-asymp-equiv*) *auto*
qed

show *?thesis*
proof (*rule asymp-equiv-sandwich'*)
fix c' :: *real* **assume** c': $c' \in \{-e{<}..{<}-2/3*e\}$
hence *neq*: $c' \neq -2/3 * e$ **by** *auto*
from c' **have** *pos*: $C\ c' > 0$ **unfolding** *C-def* **by** (*auto intro!*: *mult-pos-neg*)
hence *eventually* $(\lambda x.\ C\ c' * (x - a)\ powr\ (3\ /\ 2) > 0)$ (*at-right a*)
by *real-asymp*
hence *eventually* $(\lambda x.\ (f\ x + c' * (x - a)) * exp\ (f\ x + c' * (x - a)) - x > 0)$ (*at-right a*)
using *asymp-equiv-eventually-pos-iff*[*OF asymp-equiv*[*OF neq*]]
by *eventually-elim* (*use pos* **in** *auto*)
moreover have *eventually* $(\lambda x.\ x > a)$ (*at-right a*)
eventually $(\lambda x.\ f\ x + c' * (x - a) \leq -1)$ (*at-right a*)
unfolding *a-def f-def C1-def c'* **by** *real-asymp+*
ultimately show *eventually* $(\lambda x.\ Lambert\text{-}W'\ x - f\ x \geq c' * (x - a))$ (*at-right a*)
proof *eventually-elim*
case (*elim x*)
hence $Lambert\text{-}W'\ x \geq f\ x + c' * (x - a)$
by (*intro Lambert-W'-geI*) (*auto simp*: *a-def*)
thus *?case* **by** *simp*
qed
next
fix c' :: *real* **assume** c': $c' \in \{-2/3*e{<}..{<}0\}$
hence *neq*: $c' \neq -2/3 * e$ **by** *auto*
from c' **have** *neg*: $C\ c' < 0$ **unfolding** *C-def* **by** *auto*
hence *eventually* $(\lambda x.\ C\ c' * (x - a)\ powr\ (3\ /\ 2) < 0)$ (*at-right a*)
by *real-asymp*
hence *eventually* $(\lambda x.\ (f\ x + c' * (x - a)) * exp\ (f\ x + c' * (x - a)) - x < 0)$ (*at-right a*)
using *asymp-equiv-eventually-neg-iff*[*OF asymp-equiv*[*OF neq*]]
by *eventually-elim* (*use neg* **in** *auto*)
moreover have *eventually* $(\lambda x.\ x < 0)$ (*at-right a*)

```
    unfolding a-def by real-asymp
  ultimately show eventually (λx. Lambert-W' x − f x ≤ c' ∗ (x − a)) (at-right
a)
   proof eventually-elim
    case (elim x)
    hence Lambert-W' x ≤ f x + c' ∗ (x − a)
     by (intro Lambert-W'-leI) auto
    thus ?case by simp
   qed
 qed (auto simp: e-def)
qed
```

Lastly, just for fun, we derive a slightly more accurate expansion of $W_0(x)$ for $x \to \infty$:

```
theorem Lambert-W-asymp-equiv-at-top'':
 (λx. Lambert-W x − ln x + ln (ln x)) ∼[at-top] (λx. ln (ln x) / ln x)
proof −
 have (λx. Lambert-W x − ln x + ln (ln x)) ∼[at-top] (λx. 1 ∗ (ln (ln x) / ln x))
 proof (rule asymp-equiv-sandwich')
   fix c' :: real assume c': c' ∈ {0<..<1}
   define a where a = (λx::real. ln x − ln (ln x) + c' ∗ (ln (ln x) / ln x))
   have eventually (λx. a x ∗ exp (a x) ≤ x) at-top
    using c' unfolding a-def by real-asymp+
   thus eventually (λx. Lambert-W x − ln x + ln (ln x) ≥ c' ∗ (ln (ln x) / ln x))
at-top
   proof eventually-elim
    case (elim x)
    hence Lambert-W x ≥ a x
     by (intro Lambert-W-geI)
    thus ?case by (simp add: a-def)
   qed
 next
   fix c' :: real assume c': c' ∈ {1<..<2}
   define a where a = (λx::real. ln x − ln (ln x) + c' ∗ (ln (ln x) / ln x))
   have eventually (λx. a x ∗ exp (a x) ≥ x) at-top
      eventually (λx. a x ≥ −1) at-top
    using c' unfolding a-def by real-asymp+
   thus eventually (λx. Lambert-W x − ln x + ln (ln x) ≤ c' ∗ (ln (ln x) / ln x))
at-top
    using eventually-ge-at-top[of −exp (−1)]
   proof eventually-elim
    case (elim x)
    hence Lambert-W x ≤ a x
     by (intro Lambert-W-leI)
    thus ?case by (simp add: a-def)
   qed
 qed auto
 thus ?thesis by simp
qed
```

end

theory *Lambert-W-MacLaurin-Series*
imports
HOL−Computational-Algebra.Formal-Power-Series
Bernoulli.Bernoulli-FPS
Stirling-Formula.Stirling-Formula
Lambert-W
begin

1.7 The MacLaurin series of $W_0(x)$ at $x = 0$

In this section, we derive the MacLaurin series of $W_0(x)$ as a formal power series at $x = 0$ and prove that its radius of convergenge is e^{-1}.

We do not actually show that this series evaluates to 1 since Isabelle's library does not contain the required theorems about convergence of the composition of two power series yet. If it did, however, this last remaining step would be trivial since we did all the real work here.

lemma *Stirling-Suc-n-n*: *Stirling* (*Suc n*) *n* = (*Suc n choose 2*)
by (*induction n*) (*auto simp*: *choose-two*)

lemma *Stirling-n-n-minus-1*: *n* > *0* ⟹ *Stirling n* (*n* − *1*) = (*n choose 2*)
using *Stirling-Suc-n-n*[*of n* − *1*] **by** (*cases n*) *auto*

The following defines the power series $W(X)$ as the formal inverse of the formal power series Xe^X:

definition *fps-Lambert-W* :: *real fps* **where**
fps-Lambert-W = *fps-inv* (*fps-X* ∗ *fps-exp 1*)

The formal composition of $W(X)$ and Xe^X is, in fact, the identity (in both directions).

lemma *fps-compose-Lambert-W*: *fps-compose fps-Lambert-W* (*fps-X* ∗ *fps-exp 1*) = *fps-X*
unfolding *fps-Lambert-W-def* **by** (*rule fps-inv*) *auto*

lemma *fps-compose-Lambert-W'*: *fps-compose* (*fps-X* ∗ *fps-exp 1*) *fps-Lambert-W* = *fps-X*
unfolding *fps-Lambert-W-def* **by** (*rule fps-inv-right*) *auto*

We have $W(0) = 0$, which shows that $W(X)$ indeed represents the branch W_0.

lemma *fps-nth-Lambert-W-0* [*simp*]: *fps-nth fps-Lambert-W 0* = *0*
by (*simp add*: *fps-Lambert-W-def fps-inv-def*)

lemma *fps-nth-Lambert-W-1* [*simp*]: *fps-nth fps-Lambert-W 1* = *1*
by (*simp add*: *fps-Lambert-W-def fps-inv-def*)

All the equalities that hold for the analytic Lambert W function in a neighbourhood of 0 also hold formally for the formal power series, e.g. $W(X) = Xe^{-W(X)}$:

lemma *fps-Lambert-W-over-X*:
fps-Lambert-W = *fps-X* ∗ *fps-compose* (*fps-exp* (−*1*)) *fps-Lambert-W*
proof −
have *fps-nth* (*fps-exp 1 oo fps-Lambert-W*) *0* = *1*
by *simp*
hence *nz*: *fps-exp 1 oo fps-Lambert-W* ≠ *0*
by *force*
have *fps-Lambert-W* ∗ *fps-compose* (*fps-exp 1*) *fps-Lambert-W* =
fps-compose (*fps-X* ∗ *fps-exp 1*) *fps-Lambert-W*
by (*simp add*: *fps-compose-mult-distrib*)
also have ... = *fps-X* ∗ *fps-compose 1 fps-Lambert-W*
by (*simp add*: *fps-compose-Lambert-W′*)
also have *1* = *fps-exp* (−*1*) ∗ *fps-exp* (*1* :: *real*)
by (*simp flip*: *fps-exp-add-mult*)
also have *fps-X* ∗ *fps-compose* ... *fps-Lambert-W* =
fps-X ∗ *fps-compose* (*fps-exp* (−*1*)) *fps-Lambert-W* ∗
fps-compose (*fps-exp 1*) *fps-Lambert-W*
by (*simp add*: *fps-compose-mult-distrib mult-ac*)
finally show *?thesis*
using *nz* **by** *simp*
qed

We now derive the closed-form expression

$$W(X) = \sum_{n=1}^{\infty} \frac{(-n)^{n-1}}{n!} X^n \ .$$

lemma *fps-nth-Lambert-W*: *fps-nth fps-Lambert-W n* = (*if n* = *0 then 0 else* ((−*n*)^(*n*−*1*) / *fact n*))
proof −
define *F* :: *real fps* **where** *F* = *fps-X* ∗ *fps-exp 1*
have *fps-nth-eq*: *fps-nth F n* = *1* / *fact* (*n* − *1*) **if** *n* > *0* **for** *n*
using *that* **unfolding** *F-def* **by** *simp*
have *F-power*: *F* ^ *n* = *fps-X* ^ *n* ∗ *fps-exp* (*of-nat n*) **for** *n*
by (*simp add*: *F-def power-mult-distrib fps-exp-power-mult*)

have *fps-nth* (*fps-inv F*) *n* = (*if n* = *0 then 0 else* ((−*n*)^(*n*−*1*) / *fact n*)) **for** *n*
proof (*induction n rule*: *less-induct*)
case (*less n*)
consider *n* = *0* | *n* = *1* | *n* > *1* **by** *force*
thus *?case*
proof *cases*
case *3*
hence *fps-nth* (*fps-inv F*) *n* = −(∑ *i*=*0*..*n*−*1*. *fps-nth* (*fps-inv F*) *i* ∗ *fps-nth* (*F* ^ *i*) *n*)

```
    (is - = −?S) by (cases n) (auto simp: fps-inv-def F-def)
  also have ?S = (∑ i=1..<n. fps-nth (fps-inv F) i ∗ fps-nth (F ^ i) n)
    using less[of 1] 3 by (intro sum.mono-neutral-right) (auto simp: not-le)
  also have . . . = (−1) ^ (n+1) / fact n ∗
                  (∑ i=1..<n. ((−1)^(n − i) ∗ real (n choose i) ∗ real i ^ (n −
1)))
    unfolding sum-divide-distrib sum-distrib-left
  proof (intro sum.cong, goal-cases)
    case (2 i)
    hence fps-nth (fps-inv F) i ∗ fps-nth (F ^ i) n =
            (−1) ^ (i − 1) ∗ real (i ^ (i − 1) ∗ i ^ (n − i)) ∗
            (fact n / (fact i ∗ fact (n − i)) / fact n)
        using less.IH[of i] by (simp add: F-power less fps-X-power-mult-nth
power-minus′)
    also have (fact n / (fact i ∗ fact (n − i))) = real (n choose i)
      using 2 by (subst binomial-fact) auto
    also have i ^ (i − 1) ∗ i ^ (n − i) = i ^ (n − 1)
      using 2 by (subst power-add [symmetric]) auto
    also have (−1) ^ (i − 1) = ((−1) ^ (n+1) ∗ (−1)^(n−i) :: real)
    using 2 by (subst power-add [symmetric]) (auto simp: minus-one-power-iff)
    finally show ?case by simp
  qed auto
  also have (∑ i=1..<n. ((−1)^(n − i) ∗ real (n choose i) ∗ real i ^ (n −
1))) =
             (∑ i∈{..n}−{n}. ((−1)^(n − i) ∗ real (n choose i) ∗ real i ^ (n −
1)))
    using 3 by (intro sum.mono-neutral-left) auto
  also have . . . = (∑ i≤n. ((−1)^(n − i) ∗ real (n choose i) ∗ real i ^ (n −
1))) −
                 real n ^ (n − 1)
    by (subst (2) sum.remove[of - n]) auto
  also have (∑ i≤n. ((−1)^(n − i) ∗ real (n choose i) ∗ real i ^ (n − 1))) =
               real (Stirling (n − 1) n) ∗ fact n
    by (subst Stirling-closed-form) auto
  also have Stirling (n − 1) n = 0
    using 3 by (subst Stirling-less) auto
  finally have fps-nth (fps-inv F) n = −((−1) ^ n ∗ real n ^ (n − 1) / fact n)
    by simp
  also have . . . = (−real n) ^ (n − 1) / fact n
    using 3 by (subst power-minus) (auto simp: minus-one-power-iff)
  finally show ?thesis
    using 3 by simp
 qed (auto simp: fps-inv-def F-def)
qed
thus ?thesis by (simp add: F-def fps-Lambert-W-def)
qed
```

Next, we need a few auxiliary lemmas about summability and convergence radii that should go into Isabelle's standard library at some point:

lemma *summable-comparison-test-bigo*:
  fixes f :: *nat* ⇒ *real*
  assumes *summable* ($\lambda n.$ *norm* (g n)) $f \in O(g)$
  shows *summable* f
proof −
  from ‹$f \in O(g)$› **obtain** C **where** C: *eventually* ($\lambda x.$ *norm* (f x) $\leq C *$ *norm* (g x)) *at-top*
    by (*auto elim*: *landau-o.bigE*)
  thus *?thesis*
    by (*rule summable-comparison-test-ev*) (*insert assms*, *auto intro*: *summable-mult*)
qed

lemma *summable-comparison-test-bigo′*:
  assumes *summable* ($\lambda n.$ *norm* (g n))
  assumes ($\lambda n.$ *norm* (f n :: ′a :: *banach*)) $\in O(\lambda n.$ *norm* (g n))
  shows *summable* f
proof (*rule summable-norm-cancel*, *rule summable-comparison-test-bigo*)
  show *summable* ($\lambda n.$ *norm* (*norm* (g n)))
    using *assms* **by** *simp*
qed *fact*+

lemma *conv-radius-conv-Sup′*:
  fixes f :: *nat* ⇒ ′a :: {*banach*, *real-normed-div-algebra*}
  shows *conv-radius* f = *Sup* {r. $\forall z$. *ereal* (*norm* z) $< r \longrightarrow$ *summable* ($\lambda n.$ *norm* (f $n * z$ ^ n))}
proof (*rule Sup-eqI* [*symmetric*], *goal-cases*)
  case (*1* r)
  show *?case*
  proof (*rule conv-radius-geI-ex′*)
    fix r' :: *real* **assume** r': $r' > 0$ *ereal* $r' < r$
    show *summable* ($\lambda n.$ f $n *$ *of-real* r' ^ n)
      by (*rule summable-norm-cancel*) (*use 1* r' **in** *auto*)
  qed
next
  case (*2* r)
  from *2*[*of 0*] **have** r: $r \geq 0$ **by** *auto*
  show *?case*
  proof (*intro conv-radius-leI-ex′* r)
    fix R **assume** R: $R > 0$ *ereal* $R > r$
    with r **obtain** r' **where** [*simp*]: $r =$ *ereal* r' **by** (*cases* r) *auto*
    show ¬*summable* ($\lambda n.$ f $n *$ *of-real* R ^ n)
    proof
      assume $*$: *summable* ($\lambda n.$ f $n *$ *of-real* R ^ n)
      define R' **where** $R' = (R + r') / 2$
      from R **have** R': $R' > r'$ $R' < R$ **by** (*simp-all add*: R'-*def*)
      hence $\forall z$. *norm* $z < R' \longrightarrow$ *summable* ($\lambda n.$ *norm* (f $n * z$ ^ n))
        using *powser-insidea*[*OF* $*$] **by** *auto*
      from *2*[*of* R'] **and** *this* **have** $R' \leq r'$ **by** *auto*
      with ‹$R' > r'$› **show** *False* **by** *simp*

```
    qed
  qed
qed

lemma bigo-imp-conv-radius-ge:
  fixes f g :: nat ⇒ 'a :: {banach, real-normed-field}
  assumes f ∈ O(g)
  shows   conv-radius f ≥ conv-radius g
proof −
  have conv-radius g = Sup {r. ∀ z. ereal (norm z) < r ⟶ summable (λn. norm
(g n ∗ z ^ n))}
    by (simp add: conv-radius-conv-Sup')
  also have ... ≤ Sup {r. ∀ z. ereal (norm z) < r ⟶ summable (λn. f n ∗ z ^
n)}
  proof (rule Sup-subset-mono, safe)
    fix r :: ereal and z :: 'a
    assume g: ∀ z. ereal (norm z) < r ⟶ summable (λn. norm (g n ∗ z ^ n))
    assume z: ereal (norm z) < r
    from g z have summable (λn. norm (g n ∗ z ^ n))
      by blast
    moreover have (λn. norm (f n ∗ z ^ n)) ∈ O(λn. norm (g n ∗ z ^ n))
      unfolding landau-o.big.norm-iff by (intro landau-o.big.mult assms) auto
    ultimately show summable (λn. f n ∗ z ^ n)
      by (rule summable-comparison-test-bigo')
  qed
  also have ... = conv-radius f
    by (simp add: conv-radius-conv-Sup)
  finally show ?thesis .
qed

lemma conv-radius-cong-bigtheta:
  assumes f ∈ Θ(g)
  shows   conv-radius f = conv-radius g
  using assms
  by (intro antisym bigo-imp-conv-radius-ge) (auto simp: bigtheta-def bigomega-iff-bigo)

lemma conv-radius-eqI-smallomega-smallo:
  fixes f :: nat ⇒ 'a :: {real-normed-div-algebra, banach}
  assumes ⋀ε. ε > l ⟹ ε < inverse C ⟹ (λn. norm (f n)) ∈ ω(λn. ε ^ n)
  assumes ⋀ε. ε < u ⟹ ε > inverse C ⟹ (λn. norm (f n)) ∈ o(λn. ε ^ n)
  assumes C: C > 0 and lu: l > 0 l < inverse C u > inverse C
  shows   conv-radius f = ereal C
proof (intro antisym)
  have 0 < inverse C
    using assms by (auto simp: field-simps)
  also have ... < u
    by fact
  finally have u > 0 by simp
  show conv-radius f ≥ C
```

```
    unfolding conv-radius-altdef le-Liminf-iff
  proof safe
    fix c :: ereal assume c: c < C
    hence max c (inverse u) < ereal C
      using lu C ‹u > 0› by (auto simp: field-simps)
    from ereal-dense2[OF this] obtain c' where c': c < ereal c' inverse u < c' c'
< C
      by auto
    have inverse u > 0
      using ‹u > 0› by simp
    also have ... < c' by fact
    finally have c' > 0 .

    have ∀F x in sequentially. norm (norm (f x)) ≤ 1/2 * norm (inverse c' ^ x)
      using landau-o.smallD[OF assms(2)[of inverse c'], of 1/2] c' C lu ‹c' > 0› c
      by (simp add: field-simps)
    thus ∀F n in sequentially. c < inverse (ereal (root n (norm (f n))))
      using eventually-gt-at-top[of 0]
    proof eventually-elim
      case (elim n)
      have norm (f n) ≤ 1/2 * norm (inverse c' ^ n)
        using c' using elim by (simp add: field-simps)
      also have ... < norm (inverse c' ^ n)
        using ‹c' > 0› by simp
      finally have root n (norm (f n)) < root n (norm (inverse c' ^ n))
        using ‹n > 0› c' by (intro real-root-less-mono) auto
      also have root n (norm (inverse c' ^ n)) = inverse c'
        using ‹n > 0› ‹c' > 0› by (simp add: norm-power real-root-power)
      finally have ereal (root n (norm (f n))) < ereal (inverse c')
        by simp
      also have ... = inverse (ereal c')
        using ‹c' > 0› by auto
      finally have inverse (inverse (ereal c')) < inverse (ereal (root n (norm (f
n))))
        using c' ‹n > 0› by (intro ereal-inverse-antimono-strict) auto
      also have inverse (inverse (ereal c')) = ereal c'
        using c' by simp
      finally show ?case
        using ‹c < c'› by simp
    qed
  qed
next
  show conv-radius f ≤ C
  proof (rule ccontr)
    assume ¬(conv-radius f ≤ C)
    hence conv-radius f > C by auto
    hence min (conv-radius f) (inverse l) > ereal C
      using lu C ‹l > 0› by (auto simp: field-simps)
    from ereal-dense2[OF this] obtain c where c: C < ereal c inverse l > c c <
```

conv-radius f
  by *auto*
 hence *c > 0* **using** *lu C*
  by (*simp add*: *field-simps*)

 have $\forall_F$ *n in sequentially. ereal c < inverse* (*ereal* (*root n* (*norm* (*f n*))))
  using *less-LiminfD*[*OF c*(*3*)[*unfolded conv-radius-altdef*]] **by** *simp*
 moreover have $\forall_F$ *n in sequentially. norm* (*f n*) ≥ *2* ∗ *norm* (*inverse c* ^ *n*)
  using *landau-omega.smallD*[*OF assms*(*1*)[*of inverse c*], *of 2*] *c C* ‹*c > 0*› *lu*
  by (*simp add*: *field-simps*)
 ultimately have *eventually* (λ*n. False*) *sequentially*
  using *eventually-gt-at-top*[*of 0*]
 proof *eventually-elim*
  case (*elim n*)
  have *norm* (*inverse c* ^ *n*) < *2* ∗ *norm* (*inverse c* ^ *n*)
   using *c* ‹*n > 0*› *C* **by** *simp*
  also have . . . ≤ *norm* (*f n*)
   using *elim* **by** *simp*
  finally have *root n* (*inverse c* ^ *n*) < *root n* (*norm* (*f n*))
   using ‹*n > 0*› **by** (*intro real-root-less-mono*) *auto*
  also have *root n* (*inverse c* ^ *n*) = *inverse c*
   using ‹*n > 0*› *c C* **by** (*subst real-root-power*) *auto*
  finally have *ereal* (*inverse c*) < *ereal* (*root n* (*norm* (*f n*)))
   by *simp*
  also have *ereal* (*inverse c*) = *inverse* (*ereal c*)
   using *c C* **by** *auto*
  finally have *inverse* (*ereal* (*root n* (*norm* (*f n*)))) < *inverse* (*inverse* (*ereal c*))
   using *c C*
   by (*intro ereal-inverse-antimono-strict*) *auto*
  also have . . . = *ereal c*
   using *c C* **by** *auto*
  also have . . . < *inverse* (*ereal* (*root n* (*norm* (*f n*))))
   using *elim* **by** *simp*
  finally show *False* **.**
 qed
 thus *False* **by** *simp*
 qed
qed

Finally, we show that the radius of convergence of $W(X)$ is e^{-1} by directly computing

$$\lim_{n\to\infty} \sqrt[n]{|[X^n]\,W(X)|} = e$$

using Stirling's formula for $n!$:

lemma *fps-conv-radius-Lambert-W*: *fps-conv-radius fps-Lambert-W* = *exp* (−*1*)
proof −
 have *conv-radius* (*fps-nth fps-Lambert-W*) = *conv-radius* (λ*n. exp 1* ^ *n* ∗ *n powr* (−*3*/*2*) :: *real*)

proof (*rule conv-radius-cong-bigtheta*)
have *fps-nth fps-Lambert-W* ∈ Θ(λ*n*. (−*real n*) ^ (*n* − *1*) / *fact n*)
by (*intro bigthetaI-cong eventually-mono*[*OF eventually-gt-at-top*[*of 0*]])
(*auto simp*: *fps-nth-Lambert-W*)
also have (λ*n*. (−*real n*) ^ (*n* − *1*) / *fact n*) ∈ Θ(λ*n*. *real n* ^ (*n* − *1*) / *fact n*)
by (*subst landau-theta.norm-iff* [*symmetric*], *subst norm-divide*) *auto*
also have (λ*n*. (*real n*) ^ (*n* − *1*) / *fact n*) ∈
Θ(λ*n*. (*real n*) ^ (*n* − *1*) / (*sqrt* (*2* ∗ *pi* ∗ *real n*) ∗ (*real n* / *exp 1*) ^ *n*))
by (*intro asymp-equiv-imp-bigtheta asymp-equiv-intros fact-asymp-equiv*)
also have (λ*n*. (*real n*) ^ (*n* − *1*) / (*sqrt* (*2* ∗ *pi* ∗ *real n*) ∗ (*real n* / *exp 1*) ^ *n*)) ∈
Θ(λ*n*. *exp 1* ^ *n* ∗ *n powr* (−*3*/*2*))
by (*real-asymp simp*: *ln-inverse*)
finally show *fps-nth fps-Lambert-W* ∈ Θ(λ*n*. *exp 1* ^ *n* ∗ *n powr* (−*3*/*2*) :: *real*) **.**
qed
also have ... = *inverse* (*limsup* (λ*n*. *ereal* (*root n* (*exp 1* ^ *n* ∗ *real n powr* − (*3* / *2*)))))
by (*simp add*: *conv-radius-def*)
also have *limsup* (λ*n*. *ereal* (*root n* (*exp 1* ^ *n* ∗ *real n powr* − (*3* / *2*)))) = *exp 1*
proof (*intro lim-imp-Limsup tendsto-intros*)
— real_asymp does not support *root* for a variable basis natively, so we need to convert it to (*powr*) first.

have (λ*n*. (*exp 1* ^ *n* ∗ *real n powr* −(*3*/*2*)) *powr* (*1* / *real n*)) ⟶ *exp 1*
by *real-asymp*
also have *?this* ⟷ (λ*x*. *root x* (*exp 1* ^ *x* ∗ *real x powr* − (*3* / *2*))) ⟶ *exp 1*
by (*intro filterlim-cong eventually-mono*[*OF eventually-gt-at-top*[*of 0*]])
(*auto simp*: *root-powr-inverse*)
finally show ... **.**
qed *auto*
finally show *?thesis*
by (*simp add*: *fps-conv-radius-def exp-minus*)
qed

end

References

[1] R. M. Corless, G. H. Gonnet, D. E. G. Hare, D. J. Jeffrey, and D. E. Knuth. On the Lambert *W* function. *Advances in Computational Mathematics*, 5(1):329–359, Dec. 1996.