

Landau Symbols

Manuel Eberl

February 6, 2026

Contents

1	Sorting and grouping factors	1
2	Decision procedure for real functions	4
2.1	Eventual non-negativity/non-zerosness	4
2.2	Rewriting Landau symbols	8
2.3	Preliminary facts	9
2.4	Decision procedure	10
2.5	Reification	16
3	Simplification procedures	20
3.1	Simplification under Landau symbols	20
3.2	Simproc setup	21
3.3	Tests	22
3.3.1	Product simplification tests	22
3.3.2	Real product decision procure tests	22
3.3.3	Sum cancelling tests	23

1 Sorting and grouping factors

```
theory Group-Sort
imports Main HOL-Library.Multiset
begin
```

For the reification of products of powers of primitive functions such as $\lambda x. x * (\ln x)^2$ into a canonical form, we need to be able to sort the factors according to the growth of the primitive function it contains and merge terms with the same function by adding their exponents. The following locale defines such an operation in a general setting; we can then instantiate it for our setting.

The locale takes as parameters a key function f that sends list elements into a linear ordering that determines the sorting order, a *merge* function to merge to equivalent (w.r.t. f) elements into one, and a list reduction

function g that reduces a list to a single value. This function must be invariant w.r.t. the order of list elements and be compatible with merging of equivalent elements. In our case, this list reduction function will be the product of all list elements.

```

locale groupsort =
  fixes f :: 'a  $\Rightarrow$  ('b::linorder)
  fixes merge :: 'a  $\Rightarrow$  'a  $\Rightarrow$  'a
  fixes g :: 'a list  $\Rightarrow$  'c
  assumes f-merge: f x = f y  $\Longrightarrow$  f (merge x y) = f x
  assumes g-cong: mset xs = mset ys  $\Longrightarrow$  g xs = g ys
  assumes g-merge: f x = f y  $\Longrightarrow$  g [x,y] = g [merge x y]
  assumes g-append-cong: g xs1 = g xs2  $\Longrightarrow$  g ys1 = g ys2  $\Longrightarrow$  g (xs1 @ ys1) =
g (xs2 @ ys2)
begin

```

```

context
begin

```

```

private function part-aux ::

```

```

  'b  $\Rightarrow$  'a list  $\Rightarrow$  ('a list)  $\times$  ('a list)  $\times$  ('a list)  $\Rightarrow$  ('a list)  $\times$  ('a list)  $\times$  ('a list)

```

```

where

```

```

  part-aux p [] (ls, eq, gs) = (ls, eq, gs)
| f x < p  $\Longrightarrow$  part-aux p (x#xs) (ls, eq, gs) = part-aux p xs (x#ls, eq, gs)
| f x > p  $\Longrightarrow$  part-aux p (x#xs) (ls, eq, gs) = part-aux p xs (ls, eq, x#gs)
| f x = p  $\Longrightarrow$  part-aux p (x#xs) (ls, eq, gs) = part-aux p xs (ls, eq@[x], gs)
<proof>

```

```

termination <proof> lemma groupsort-locale: groupsort f merge g <proof> lem-
mas part-aux-induct = part-aux.induct[split-format (complete), OF groupsort-locale]

```

```

private definition part where part p xs = part-aux (f p) xs ([], [p], [])

```

```

private lemma part:

```

```

  part p xs = (rev (filter ( $\lambda$ x. f x < f p) xs),
  p # filter ( $\lambda$ x. f x = f p) xs, rev (filter ( $\lambda$ x. f x > f p) xs))
<proof> function sort :: 'a list  $\Rightarrow$  'a list where
  sort [] = []
| sort (x#xs) = (case part x xs of (ls, eq, gs)  $\Rightarrow$  sort ls @ eq @ sort gs)
<proof>

```

```

termination <proof> lemma filter-mset-union:

```

```

  assumes  $\bigwedge$ x. x  $\in$  # A  $\Longrightarrow$  P x  $\Longrightarrow$  Q x  $\Longrightarrow$  False
  shows filter-mset P A + filter-mset Q A = filter-mset ( $\lambda$ x. P x  $\vee$  Q x) A (is ?lhs
= ?rhs)

```

```

  <proof> lemma multiset-of-sort: mset (sort xs) = mset xs

```

```

  <proof> lemma g-sort: g (sort xs) = g xs

```

```

  <proof> lemma set-sort: set (sort xs) = set xs

```

```

  <proof> lemma sorted-all-equal: ( $\bigwedge$ x. x  $\in$  set xs  $\Longrightarrow$  x = y)  $\Longrightarrow$  sorted xs

```

```

  <proof> lemma sorted-sort: sorted (map f (sort xs))

```

```

  <proof> fun group where

```

```

  group [] = []

```

| $group (x\#xs) = (case\ partition\ (\lambda y. f\ y = f\ x)\ xs\ of\ (xs',\ xs'') \Rightarrow$
 $fold\ merge\ xs'\ x\ \# group\ xs'')$

private lemma *f-fold-merge*: $(\bigwedge y. y \in set\ xs \Rightarrow f\ y = f\ x) \Rightarrow f\ (fold\ merge\ xs\ x) = f\ x$

<proof> **lemma** *f-group*: $x \in set\ (group\ xs) \Rightarrow \exists x' \in set\ xs. f\ x = f\ x'$

<proof> **lemma** *sorted-group*: $sorted\ (map\ f\ xs) \Rightarrow sorted\ (map\ f\ (group\ xs))$

<proof> **lemma** *distinct-group*: $distinct\ (map\ f\ (group\ xs))$

<proof> **lemma** *g-fold-same*:

assumes $\bigwedge z. z \in set\ xs \Rightarrow f\ z = f\ x$

shows $g\ (fold\ merge\ xs\ x\ \# ys) = g\ (x\#xs@ys)$

<proof> **lemma** *g-group*: $g\ (group\ xs) = g\ xs$

<proof>

function *group-part-aux* ::

$'b \Rightarrow 'a\ list \Rightarrow ('a\ list) \times 'a \times ('a\ list) \Rightarrow ('a\ list) \times 'a \times ('a\ list)$

where

$group-part-aux\ p\ []\ (ls,\ eq,\ gs) = (ls,\ eq,\ gs)$

| $f\ x < p \Rightarrow group-part-aux\ p\ (x\#xs)\ (ls,\ eq,\ gs) = group-part-aux\ p\ xs\ (x\#ls,\ eq,\ gs)$

| $f\ x > p \Rightarrow group-part-aux\ p\ (x\#xs)\ (ls,\ eq,\ gs) = group-part-aux\ p\ xs\ (ls,\ eq,\ x\#gs)$

| $f\ x = p \Rightarrow group-part-aux\ p\ (x\#xs)\ (ls,\ eq,\ gs) = group-part-aux\ p\ xs\ (ls,\ merge\ x\ eq,\ gs)$

<proof>

termination *<proof>* **lemmas** *group-part-aux-induct* =

group-part-aux.induct[split-format (complete), OF groupsort-locale]

definition *group-part* **where** $group-part\ p\ xs = group-part-aux\ (f\ p)\ xs\ ([],\ p,\ [])$

private lemma *group-part*:

$group-part\ p\ xs = (rev\ (filter\ (\lambda x. f\ x < f\ p)\ xs),$

$fold\ merge\ (filter\ (\lambda x. f\ x = f\ p)\ xs)\ p,\ rev\ (filter\ (\lambda x. f\ x > f\ p)\ xs))$

<proof>

function *group-sort* :: $'a\ list \Rightarrow 'a\ list$ **where**

$group-sort\ [] = []$

| $group-sort\ (x\#xs) = (case\ group-part\ x\ xs\ of\ (ls,\ eq,\ gs) \Rightarrow group-sort\ ls\ @\ eq\ \# group-sort\ gs)$

<proof>

termination *<proof>* **lemma** *group-append*:

assumes $\bigwedge x\ y. x \in set\ xs \Rightarrow y \in set\ ys \Rightarrow f\ x \neq f\ y$

shows $group\ (xs\ @\ ys) = group\ xs\ @\ group\ ys$

<proof> **lemma** *group-empty-iff [simp]*: $group\ xs = [] \longleftrightarrow xs = []$

<proof>

lemma *group-sort-correct*: $group-sort\ xs = group\ (sort\ xs)$

<proof>

lemma *sorted-group-sort: sorted (map f (group-sort xs))*
<proof>

lemma *distinct-group-sort: distinct (map f (group-sort xs))*
<proof>

lemma *g-group-sort: g (group-sort xs) = g xs*
<proof>

lemmas [*simp del*] = *group-sort.simps group-part-aux.simps*

end

end

end

2 Decision procedure for real functions

theory *Landau-Real-Products*

imports

Main

HOL-Library.Function-Algebras

HOL-Library.Set-Algebras

HOL-Library.Landau-Symbols

Group-Sort

begin

2.1 Eventual non-negativity/non-zeroness

For certain transformations of Landau symbols, it is required that the functions involved are eventually non-negative or non-zero. In the following, we set up a system to guide the simplifier to discharge these requirements during simplification at least in obvious cases.

definition *eventually-nonzero* $F f \longleftrightarrow \text{eventually } (\lambda x. (f x :: - :: \text{real-normed-field}) \neq 0) F$

definition *eventually-nonneg* $F f \longleftrightarrow \text{eventually } (\lambda x. (f x :: - :: \text{linordered-field}) \geq 0) F$

named-theorems *eventually-nonzero-simps*

lemmas [*eventually-nonzero-simps*] =
eventually-nonzero-def [symmetric] eventually-nonneg-def [symmetric]

lemma *eventually-nonzeroD: eventually-nonzero F f \implies eventually $(\lambda x. f x \neq 0)$ F*

<proof>

lemma *eventually-nonzero-const* [*eventually-nonzero-simps*]:
eventually-nonzero F ($\lambda x. c$) $\longleftrightarrow F = \text{bot} \vee c \neq 0$
<proof>

lemma *eventually-nonzero-inverse* [*eventually-nonzero-simps*]:
eventually-nonzero F ($\lambda x. \text{inverse } (f x)$) $\longleftrightarrow \text{eventually-nonzero } F f$
<proof>

lemma *eventually-nonzero-mult* [*eventually-nonzero-simps*]:
eventually-nonzero F ($\lambda x. f x * g x$) $\longleftrightarrow \text{eventually-nonzero } F f \wedge \text{eventually-nonzero } F g$
<proof>

lemma *eventually-nonzero-pow* [*eventually-nonzero-simps*]:
eventually-nonzero F ($\lambda x. f x ^ n$) $\longleftrightarrow n = 0 \vee \text{eventually-nonzero } F f$
<proof>

lemma *eventually-nonzero-divide* [*eventually-nonzero-simps*]:
eventually-nonzero F ($\lambda x. f x / g x$) $\longleftrightarrow \text{eventually-nonzero } F f \wedge \text{eventually-nonzero } F g$
<proof>

lemma *eventually-nonzero-ident-at-top-linorder* [*eventually-nonzero-simps*]:
eventually-nonzero at-top ($\lambda x. a$) $\longleftrightarrow a \neq 0$
<proof>

lemma *eventually-nonzero-ident-nhds* [*eventually-nonzero-simps*]:
eventually-nonzero (nhds a) ($\lambda x. x$) $\longleftrightarrow a \neq 0$
<proof>

lemma *eventually-nonzero-ident-at-within* [*eventually-nonzero-simps*]:
eventually-nonzero (at a within A) ($\lambda x. x$)
<proof>

lemma *eventually-nonzero-ln-at-top* [*eventually-nonzero-simps*]:
eventually-nonzero at-top ($\lambda x. \ln x$)
<proof>

lemma *eventually-nonzero-ln-const-at-top* [*eventually-nonzero-simps*]:
 $b > 0 \implies \text{eventually-nonzero at-top } (\lambda x. \ln (b * x))$
<proof>

lemma *eventually-nonzero-ln-const'-at-top* [*eventually-nonzero-simps*]:
 $b > 0 \implies \text{eventually-nonzero at-top } (\lambda x. \ln (x * b))$
<proof>

lemma *eventually-nonzero-powr-at-top* [*eventually-nonzero-simps*]:
eventually-nonzero at-top ($\lambda x::\text{real. } f \ x \ \text{powr } p$) \longleftrightarrow *eventually-nonzero at-top* f
 $\langle \text{proof} \rangle$

lemma *eventually-nonneg-const* [*eventually-nonzero-simps*]:
eventually-nonneg F ($\lambda-. \ c$) \longleftrightarrow $F = \text{bot} \vee c \geq 0$
 $\langle \text{proof} \rangle$

lemma *eventually-nonneg-inverse* [*eventually-nonzero-simps*]:
eventually-nonneg F ($\lambda x. \ \text{inverse } (f \ x)$) \longleftrightarrow *eventually-nonneg* $F \ f$
 $\langle \text{proof} \rangle$

lemma *eventually-nonneg-add* [*eventually-nonzero-simps*]:
assumes *eventually-nonneg* $F \ f$ *eventually-nonneg* $F \ g$
shows *eventually-nonneg* F ($\lambda x. \ f \ x + g \ x$)
 $\langle \text{proof} \rangle$

lemma *eventually-nonneg-mult* [*eventually-nonzero-simps*]:
assumes *eventually-nonneg* $F \ f$ *eventually-nonneg* $F \ g$
shows *eventually-nonneg* F ($\lambda x. \ f \ x * g \ x$)
 $\langle \text{proof} \rangle$

lemma *eventually-nonneg-mult'* [*eventually-nonzero-simps*]:
assumes *eventually-nonneg* F ($\lambda x. \ -f \ x$) *eventually-nonneg* F ($\lambda x. \ -g \ x$)
shows *eventually-nonneg* F ($\lambda x. \ f \ x * g \ x$)
 $\langle \text{proof} \rangle$

lemma *eventually-nonneg-divide* [*eventually-nonzero-simps*]:
assumes *eventually-nonneg* $F \ f$ *eventually-nonneg* $F \ g$
shows *eventually-nonneg* F ($\lambda x. \ f \ x / g \ x$)
 $\langle \text{proof} \rangle$

lemma *eventually-nonneg-divide'* [*eventually-nonzero-simps*]:
assumes *eventually-nonneg* F ($\lambda x. \ -f \ x$) *eventually-nonneg* F ($\lambda x. \ -g \ x$)
shows *eventually-nonneg* F ($\lambda x. \ f \ x / g \ x$)
 $\langle \text{proof} \rangle$

lemma *eventually-nonneg-ident-at-top* [*eventually-nonzero-simps*]:
eventually-nonneg at-top ($\lambda x. \ x$) $\langle \text{proof} \rangle$

lemma *eventually-nonneg-ident-nhds* [*eventually-nonzero-simps*]:
fixes $a :: 'a :: \{\text{linorder-topology, linordered-field}\}$
shows $a > 0 \implies$ *eventually-nonneg* (*nhds* a) ($\lambda x. \ x$) $\langle \text{proof} \rangle$

lemma *eventually-nonneg-ident-at-within* [*eventually-nonzero-simps*]:
fixes $a :: 'a :: \{\text{linorder-topology, linordered-field}\}$
shows $a > 0 \implies$ *eventually-nonneg* (*at* a *within* A) ($\lambda x. \ x$)

<proof>

lemma *eventually-nonneg-pow* [*eventually-nonzero-simps*]:
eventually-nonneg F f \implies *eventually-nonneg F* ($\lambda x. f x \wedge n$)
<proof>

lemma *eventually-nonneg-powr* [*eventually-nonzero-simps*]:
eventually-nonneg F ($\lambda x. f x \text{ powr } y :: \text{real}$) *<proof>*

lemma *eventually-nonneg-ln-at-top* [*eventually-nonzero-simps*]:
eventually-nonneg at-top ($\lambda x. \ln x :: \text{real}$)
<proof>

lemma *eventually-nonneg-ln-const* [*eventually-nonzero-simps*]:
 $b > 0 \implies$ *eventually-nonneg at-top* ($\lambda x. \ln (b*x) :: \text{real}$)
<proof>

lemma *eventually-nonneg-ln-const'* [*eventually-nonzero-simps*]:
 $b > 0 \implies$ *eventually-nonneg at-top* ($\lambda x. \ln (x*b) :: \text{real}$)
<proof>

lemma *eventually-nonzero-bigtheta'*:
 $f \in \Theta[F](g) \implies$ *eventually-nonzero F f* \longleftrightarrow *eventually-nonzero F g*
<proof>

lemma *eventually-nonneg-at-top*:
assumes *filterlim f at-top F*
shows *eventually-nonneg F f*
<proof>

lemma *eventually-nonzero-at-top*:
assumes *filterlim* ($f :: 'a \Rightarrow 'b :: \{\text{linordered-field, real-normed-field}\}$) *at-top F*
shows *eventually-nonzero F f*
<proof>

lemma *eventually-nonneg-at-top-ASSUMPTION* [*eventually-nonzero-simps*]:
ASSUMPTION (*filterlim f at-top F*) \implies *eventually-nonneg F f*
<proof>

lemma *eventually-nonzero-at-top-ASSUMPTION* [*eventually-nonzero-simps*]:
ASSUMPTION (*filterlim f (at-top :: 'a :: \{\text{linordered-field, real-normed-field}\}*
filter) F) \implies
eventually-nonzero F f
<proof>

lemma *filterlim-at-top-iff-smallomega*:
fixes $f :: - \Rightarrow \text{real}$
shows *filterlim f at-top F* \longleftrightarrow $f \in \omega[F](\lambda-. 1) \wedge$ *eventually-nonneg F f*
<proof>

lemma *smallomega-1-iff*:

eventually-nonneg $F f \implies f \in \omega[F](\lambda-. 1 :: \text{real}) \longleftrightarrow \text{filterlim } f \text{ at-top } F$
 ⟨*proof*⟩

lemma *smallo-1-iff*:

eventually-nonneg $F f \implies (\lambda-. 1 :: \text{real}) \in o[F](f) \longleftrightarrow \text{filterlim } f \text{ at-top } F$
 ⟨*proof*⟩

lemma *eventually-nonneg-add1* [*eventually-nonzero-simps*]:

assumes *eventually-nonneg* $F f g \in o[F](f)$
shows *eventually-nonneg* $F (\lambda x. f x + g x :: \text{real})$
 ⟨*proof*⟩

lemma *eventually-nonneg-add2* [*eventually-nonzero-simps*]:

assumes *eventually-nonneg* $F g f \in o[F](g)$
shows *eventually-nonneg* $F (\lambda x. f x + g x :: \text{real})$
 ⟨*proof*⟩

lemma *eventually-nonneg-diff1* [*eventually-nonzero-simps*]:

assumes *eventually-nonneg* $F f g \in o[F](f)$
shows *eventually-nonneg* $F (\lambda x. f x - g x :: \text{real})$
 ⟨*proof*⟩

lemma *eventually-nonneg-diff2* [*eventually-nonzero-simps*]:

assumes *eventually-nonneg* $F (\lambda x. - g x) f \in o[F](g)$
shows *eventually-nonneg* $F (\lambda x. f x - g x :: \text{real})$
 ⟨*proof*⟩

2.2 Rewriting Landau symbols

lemma *bigheta-mult-eq*: $\Theta[F](\lambda x. f x * g x) = \Theta[F](f) * \Theta[F](g)$

⟨*proof*⟩

Since the simplifier does not currently rewriting with relations other than equality, but we want to rewrite terms like $\Theta(\lambda x. \log 2 x * x)$ to $\Theta(\lambda x. \ln x * x)$, we need to bring the term into something that contains $\Theta(\log 2)$ and $\Theta(\lambda x. x)$, which can then be rewritten individually. For this, we introduce the following constants and rewrite rules. The rules are mainly used by the simprocs, but may be useful for manual reasoning occasionally.

definition *set-mult* $A B = \{\lambda x. f x * g x \mid f g. f \in A \wedge g \in B\}$

definition *set-inverse* $A = \{\lambda x. \text{inverse } (f x) \mid f. f \in A\}$

definition *set-divide* $A B = \{\lambda x. f x / g x \mid f g. f \in A \wedge g \in B\}$

definition *set-pow* $A n = \{\lambda x. f x \wedge^n \mid f. f \in A\}$

definition *set-powr* $A y = \{\lambda x. f x \text{ powr } y \mid f. f \in A\}$

lemma *bigheta-mult-eq-set-mult*:

shows $\Theta[F](\lambda x. f x * g x) = \text{set-mult } (\Theta[F](f)) (\Theta[F](g))$
 ⟨*proof*⟩

lemma *bigheta-inverse-eq-set-inverse:*

shows $\Theta[F](\lambda x. \text{inverse } (f x)) = \text{set-inverse } (\Theta[F](f))$
<proof>

lemma *set-divide-inverse:*

set-divide $(A :: (- \Rightarrow (- :: \text{division-ring})) \text{ set}) B = \text{set-mult } A (\text{set-inverse } B)$
<proof>

lemma *bigheta-divide-eq-set-divide:*

shows $\Theta[F](\lambda x. f x / g x) = \text{set-divide } (\Theta[F](f)) (\Theta[F](g))$
<proof>

primrec *bigheta-pow where*

bigheta-pow $F A 0 = \Theta[F](\lambda-. 1)$
 $| \text{ bigheta-pow } F A (\text{Suc } n) = \text{set-mult } A (\text{bigheta-pow } F A n)$

lemma *bigheta-pow-eq-set-pow:* $\Theta[F](\lambda x. f x \wedge^n) = \text{bigheta-pow } F (\Theta[F](f)) n$
<proof>

definition *bigheta-powr where*

bigheta-powr $F A y = (\text{if } y = 0 \text{ then } \{f. \exists g \in A. \text{eventually-nonneg } F g \wedge f \in \Theta[F](\lambda x. g x \text{ powr } y)\}$
else $\{f. \exists g \in A. \text{eventually-nonneg } F g \wedge (\forall x. (\text{norm } (f x)) = g x \text{ powr } y)\}$)

lemma *bigheta-powr-eq-set-powr:*

assumes *eventually-nonneg* $F f$
shows $\Theta[F](\lambda x. f x \text{ powr } (y :: \text{real})) = \text{bigheta-powr } F (\Theta[F](f)) y$
<proof>

lemmas *bigheta-factors-eq =*

bigheta-mult-eq-set-mult bigheta-inverse-eq-set-inverse bigheta-divide-eq-set-divide

bigheta-pow-eq-set-pow bigheta-powr-eq-set-powr

lemmas *landau-bigheta-congs = landau-symbols[THEN landau-symbol.cong-bigheta]*

lemma (*in* *landau-symbol*) *meta-cong-bigheta:* $\Theta[F](f) \equiv \Theta[F](g) \implies L F (f) \equiv L F (g)$
<proof>

lemmas *landau-bigheta-meta-congs = landau-symbols[THEN landau-symbol.meta-cong-bigheta]*

2.3 Preliminary facts

lemma *real-powr-at-top:*

assumes $(p :: \text{real}) > 0$

shows *filterlim* $(\lambda x. x \text{ powr } p)$ *at-top at-top*

<proof>

lemma *tendsto-ln-over-powr*:

assumes $(a::real) > 0$

shows $((\lambda x. \ln x / x \text{ powr } a) \longrightarrow 0)$ *at-top*

<proof>

lemma *tendsto-ln-powr-over-powr*:

assumes $(a::real) > 0 \ b > 0$

shows $((\lambda x. \ln x \text{ powr } a / x \text{ powr } b) \longrightarrow 0)$ *at-top*

<proof>

lemma *tendsto-ln-powr-over-powr'*:

assumes $b > 0$

shows $((\lambda x::real. \ln x \text{ powr } a / x \text{ powr } b) \longrightarrow 0)$ *at-top*

<proof>

lemma *tendsto-ln-over-ln*:

assumes $(a::real) > 0 \ c > 0$

shows $((\lambda x. \ln (a*x) / \ln (c*x)) \longrightarrow 1)$ *at-top*

<proof>

lemma *tendsto-ln-powr-over-ln-powr*:

assumes $(a::real) > 0 \ c > 0$

shows $((\lambda x. \ln (a*x) \text{ powr } d / \ln (c*x) \text{ powr } d) \longrightarrow 1)$ *at-top*

<proof>

lemma *tendsto-ln-powr-over-ln-powr'*:

$c > 0 \implies ((\lambda x::real. \ln x \text{ powr } d / \ln (c*x) \text{ powr } d) \longrightarrow 1)$ *at-top*

<proof>

lemma *tendsto-ln-powr-over-ln-powr''*:

$a > 0 \implies ((\lambda x::real. \ln (a*x) \text{ powr } d / \ln x \text{ powr } d) \longrightarrow 1)$ *at-top*

<proof>

lemma *bigheta-const-ln-powr* [simp]: $a > 0 \implies (\lambda x::real. \ln (a*x) \text{ powr } d) \in$

$\Theta(\lambda x. \ln x \text{ powr } d)$

<proof>

lemma *bigheta-const-ln-pow* [simp]: $a > 0 \implies (\lambda x::real. \ln (a*x) ^ d) \in \Theta(\lambda x.$

$\ln x ^ d)$

<proof>

lemma *bigheta-const-ln* [simp]: $a > 0 \implies (\lambda x::real. \ln (a*x)) \in \Theta(\lambda x. \ln x)$

<proof>

If there are two functions f and g where any power of g is asymptotically smaller than f , propositions like $(\lambda x. (f x)^{p1} * (g x)^{q1}) \in O(\lambda x. (f x)^{p2} * (g x)^{q2})$ can be decided just by looking at the exponents: the proposition is

true iff $p1 < p2$ or $p1 = p2 \wedge q1 \leq q2$.

The functions $\lambda x. x$, \ln , $\lambda x. \ln (\ln x)$, \dots form a chain in which every function dominates all succeeding functions in the above sense, allowing to decide propositions involving Landau symbols and functions that are products of powers of functions from this chain by reducing the proposition to a statement involving only logical connectives and comparisons on the exponents.

We will now give the mathematical background for this and implement reification to bring functions from this class into a canonical form, allowing the decision procedure to be implemented in a simproc.

2.4 Decision procedure

definition *powr-closure* $f \equiv \{\lambda x. f x \text{ powr } p :: \text{real } | p. \text{True}\}$

lemma *powr-closureI* [*simp*]: $(\lambda x. f x \text{ powr } p) \in \text{powr-closure } f$
 $\langle \text{proof} \rangle$

lemma *powr-closureE*:

assumes $g \in \text{powr-closure } f$
obtains p **where** $g = (\lambda x. f x \text{ powr } p)$
 $\langle \text{proof} \rangle$

locale *landau-function-family* =

fixes $F :: 'a \text{ filter}$ **and** $H :: ('a \Rightarrow \text{real}) \text{ set}$

assumes *F-nontrivial*: $F \neq \text{bot}$

assumes *pos*: $h \in H \Longrightarrow \text{eventually } (\lambda x. h x > 0) F$

assumes *linear*: $h1 \in H \Longrightarrow h2 \in H \Longrightarrow h1 \in o[F](h2) \vee h2 \in o[F](h1) \vee h1 \in \Theta[F](h2)$

assumes *mult*: $h1 \in H \Longrightarrow h2 \in H \Longrightarrow (\lambda x. h1 x * h2 x) \in H$

assumes *inverse*: $h \in H \Longrightarrow (\lambda x. \text{inverse } (h x)) \in H$

begin

lemma *div*: $h1 \in H \Longrightarrow h2 \in H \Longrightarrow (\lambda x. h1 x / h2 x) \in H$
 $\langle \text{proof} \rangle$

lemma *nonzero*: $h \in H \Longrightarrow \text{eventually } (\lambda x. h x \neq 0) F$
 $\langle \text{proof} \rangle$

lemma *landau-cases*:

assumes $h1 \in H \ h2 \in H$

obtains $h1 \in o[F](h2) \mid h2 \in o[F](h1) \mid h1 \in \Theta[F](h2)$

$\langle \text{proof} \rangle$

lemma *small-big-antisym*:

assumes $h1 \in H \ h2 \in H \ h1 \in o[F](h2) \ h2 \in O[F](h1)$ **shows** *False*
 $\langle \text{proof} \rangle$

lemma *small-antisym:*

assumes $h1 \in H$ $h2 \in H$ $h1 \in o[F](h2)$ $h2 \in o[F](h1)$ **shows** *False*

<proof>

end

locale *landau-function-family-pair =*

G: landau-function-family F G + H: landau-function-family F H for F G H +
fixes g

assumes *gs-dominate:* $g1 \in G \implies g2 \in G \implies h1 \in H \implies h2 \in H \implies g1 \in o[F](g2) \implies$

$(\lambda x. g1\ x * h1\ x) \in o[F](\lambda x. g2\ x * h2\ x)$

assumes $g: g \in G$

assumes *g-dominates:* $h \in H \implies h \in o[F](g)$

begin

sublocale *GH: landau-function-family F G * H*

<proof>

lemma *smallo-iff:*

assumes $g1 \in G$ $g2 \in G$ $h1 \in H$ $h2 \in H$

shows $(\lambda x. g1\ x * h1\ x) \in o[F](\lambda x. g2\ x * h2\ x) \longleftrightarrow$

$g1 \in o[F](g2) \vee (g1 \in \Theta[F](g2) \wedge h1 \in o[F](h2))$ (**is** $?P \longleftrightarrow ?Q$)

<proof>

lemma *bigo-iff:*

assumes $g1 \in G$ $g2 \in G$ $h1 \in H$ $h2 \in H$

shows $(\lambda x. g1\ x * h1\ x) \in O[F](\lambda x. g2\ x * h2\ x) \longleftrightarrow$

$g1 \in o[F](g2) \vee (g1 \in \Theta[F](g2) \wedge h1 \in O[F](h2))$ (**is** $?P \longleftrightarrow ?Q$)

<proof>

lemma *bigtheta-iff:*

$g1 \in G \implies g2 \in G \implies h1 \in H \implies h2 \in H \implies$

$(\lambda x. g1\ x * h1\ x) \in \Theta[F](\lambda x. g2\ x * h2\ x) \longleftrightarrow g1 \in \Theta[F](g2) \wedge h1 \in \Theta[F](h2)$

<proof>

end

lemma *landau-function-family-powr-closure:*

assumes $F \neq \text{bot}$ *filterlim f at-top F*

shows *landau-function-family F (powr-closure f)*

<proof>

lemma *landau-function-family-pair-trans:*

assumes *landau-function-family-pair Ftr F G f*

assumes *landau-function-family-pair Ftr G H g*

shows *landau-function-family-pair Ftr F (G*H) f*

<proof>

lemma *landau-function-family-pair-trans-powr*:
assumes *landau-function-family-pair* F (*powr-closure* g) H ($\lambda x. g\ x\ \text{powr}\ 1$)
assumes *filterlim* f *at-top* F
assumes $\bigwedge p. (\lambda x. g\ x\ \text{powr}\ p) \in o[F](f)$
shows *landau-function-family-pair* F (*powr-closure* f) (*powr-closure* $g * H$) ($\lambda x. f\ x\ \text{powr}\ 1$)
 $\langle \text{proof} \rangle$

definition *dominates* :: *'a filter* \Rightarrow (*'a* \Rightarrow *real*) \Rightarrow (*'a* \Rightarrow *real*) \Rightarrow *bool* **where**
dominates $F\ f\ g = (\forall p. (\lambda x. g\ x\ \text{powr}\ p) \in o[F](f))$

lemma *dominates-trans*:
assumes *eventually* ($\lambda x. g\ x > 0$) F
assumes *dominates* $F\ f\ g$ *dominates* $F\ g\ h$
shows *dominates* $F\ f\ h$
 $\langle \text{proof} \rangle$

fun *landau-dominating-chain* **where**
landau-dominating-chain F ($f \# g \# gs$) \longleftrightarrow
dominates $F\ f\ g \wedge$ *landau-dominating-chain* F ($g \# gs$)
 $|$ *landau-dominating-chain* F $[f]$ $\longleftrightarrow (\lambda x. 1) \in o[F](f)$
 $|$ *landau-dominating-chain* F $[]$ $\longleftrightarrow \text{True}$

primrec *landau-dominating-chain'* **where**
landau-dominating-chain' F $[]$ $\longleftrightarrow \text{True}$
 $|$ *landau-dominating-chain'* F ($f \# gs$) \longleftrightarrow
landau-function-family-pair F (*powr-closure* f) (*prod-list* (*map* *powr-closure* gs))
 $(\lambda x. f\ x\ \text{powr}\ 1) \wedge$
landau-dominating-chain' $F\ gs$

primrec *nonneg-list* **where**
nonneg-list $[]$ $\longleftrightarrow \text{True}$
 $|$ *nonneg-list* ($x \# xs$) $\longleftrightarrow x > 0 \vee (x = 0 \wedge \text{nonneg-list}\ xs)$

primrec *pos-list* **where**
pos-list $[]$ $\longleftrightarrow \text{False}$
 $|$ *pos-list* ($x \# xs$) $\longleftrightarrow x > 0 \vee (x = 0 \wedge \text{pos-list}\ xs)$

lemma *dominating-chain-imp-dominating-chain'*:
 $Ftr \neq \text{bot} \Longrightarrow (\bigwedge g. g \in \text{set}\ gs \Longrightarrow \text{filterlim}\ g\ \text{at-top}\ Ftr) \Longrightarrow$
landau-dominating-chain $Ftr\ gs \Longrightarrow$ *landau-dominating-chain'* $Ftr\ gs$
 $\langle \text{proof} \rangle$

```

locale landau-function-family-chain =
  fixes F :: 'b filter
  fixes gs :: 'a list
  fixes get-param :: 'a  $\Rightarrow$  real
  fixes get-fun :: 'a  $\Rightarrow$  ('b  $\Rightarrow$  real)
  assumes F-nontrivial: F  $\neq$  bot
  assumes gs-pos: g  $\in$  set (map get-fun gs)  $\implies$  filterlim g at-top F
  assumes dominating-chain: landau-dominating-chain F (map get-fun gs)
begin

lemma dominating-chain': landau-dominating-chain' F (map get-fun gs)
  <proof>

lemma gs-powr-0-eq-one:
  eventually ( $\lambda x. (\prod g \leftarrow gs. \text{get-fun } g \ x \ \text{powr } 0) = 1$ ) F
  <proof>

lemma listmap-gs-in-listmap:
  ( $\lambda x. \prod g \leftarrow fs. h \ g \ x \ \text{powr } p \ g$ )  $\in$  prod-list (map powr-closure (map h fs))
  <proof>

lemma smallo-iff:
  ( $\lambda-. 1$ )  $\in$  o[F]( $\lambda x. \prod g \leftarrow gs. \text{get-fun } g \ x \ \text{powr } \text{get-param } g$ )  $\longleftrightarrow$  pos-list (map
  get-param gs)
  <proof>

lemma bigo-iff:
  ( $\lambda-. 1$ )  $\in$  O[F]( $\lambda x. \prod g \leftarrow gs. \text{get-fun } g \ x \ \text{powr } \text{get-param } g$ )  $\longleftrightarrow$  nonneg-list (map
  get-param gs)
  <proof>

lemma bigtheta-iff:
  ( $\lambda-. 1$ )  $\in$   $\Theta$ [F]( $\lambda x. \prod g \leftarrow gs. \text{get-fun } g \ x \ \text{powr } \text{get-param } g$ )  $\longleftrightarrow$  list-all ((=) 0)
  (map get-param gs)
  <proof>

end

lemma fun-chain-at-top-at-top:
  assumes filterlim (f :: ('a::order)  $\Rightarrow$  'a) at-top at-top
  shows filterlim (f  $\widetilde{\sim}$  n) at-top at-top
  <proof>

lemma const-smallo-ln-chain: ( $\lambda-. 1$ )  $\in$  o((ln::real $\Rightarrow$ real)  $\widetilde{\sim}$  n)
  <proof>

lemma ln-fun-in-smallo-fun:

```

assumes *filterlim f at-top at-top*
shows $(\lambda x. \text{ln } (f x) \text{ powr } p :: \text{real}) \in o(f)$
 $\langle \text{proof} \rangle$

lemma *ln-chain-dominates: $m > n \implies \text{dominates at-top } ((\text{ln}::\text{real} \Rightarrow \text{real}) \sim^n)$*
 $(\text{ln} \sim^m)$
 $\langle \text{proof} \rangle$

datatype *primfun = LnChain nat*

instantiation *primfun :: linorder*
begin

fun *less-eq-primfun :: primfun \Rightarrow primfun \Rightarrow bool where*
 $\text{LnChain } x \leq \text{LnChain } y \longleftrightarrow x \leq y$

fun *less-primfun :: primfun \Rightarrow primfun \Rightarrow bool where*
 $\text{LnChain } x < \text{LnChain } y \longleftrightarrow x < y$

instance
 $\langle \text{proof} \rangle$

end

fun *eval-primfun' :: - \Rightarrow - \Rightarrow real where*
 $\text{eval-primfun}' (\text{LnChain } n) = (\lambda x. (\text{ln} \sim^n) x)$

fun *eval-primfun :: - \Rightarrow - \Rightarrow real where*
 $\text{eval-primfun } (f, e) = (\lambda x. \text{eval-primfun}' f x \text{ powr } e)$

lemma *eval-primfun-altdef: $\text{eval-primfun } f x = \text{eval-primfun}' (fst f) x \text{ powr } snd f$*
 $\langle \text{proof} \rangle$

fun *merge-primfun where*
 $\text{merge-primfun } (x::\text{primfun}, a) (y, b) = (x, a + b)$

fun *inverse-primfun where*
 $\text{inverse-primfun } (x::\text{primfun}, a) = (x, -a)$

fun *powr-primfun where*
 $\text{powr-primfun } (x::\text{primfun}, a) e = (x, e * a)$

lemma *primfun-cases:*

assumes $(\bigwedge n e. P (LnChain\ n, e))$
shows $P\ x$
 $\langle proof \rangle$

lemma *eval-primfun'-at-top: filterlim (eval-primfun' f) at-top at-top*
 $\langle proof \rangle$

lemma *primfun-dominates:*
 $f < g \implies \text{dominates at-top (eval-primfun' f) (eval-primfun' g)}$
 $\langle proof \rangle$

lemma *eval-primfun-pos: eventually $(\lambda x::real. \text{eval-primfun } f\ x > 0)$ at-top*
 $\langle proof \rangle$

lemma *eventually-nonneg-primfun: eventually-nonneg at-top (eval-primfun f)*
 $\langle proof \rangle$

lemma *eval-primfun-nonzero: eventually $(\lambda x. \text{eval-primfun } f\ x \neq 0)$ at-top*
 $\langle proof \rangle$

lemma *eval-merge-primfun:*
 $\text{fst } f = \text{fst } g \implies$
 $\text{eval-primfun (merge-primfun } f\ g)\ x = \text{eval-primfun } f\ x * \text{eval-primfun } g\ x$
 $\langle proof \rangle$

lemma *eval-inverse-primfun:*
 $\text{eval-primfun (inverse-primfun } f)\ x = \text{inverse (eval-primfun } f\ x)$
 $\langle proof \rangle$

lemma *eval-powr-primfun:*
 $\text{eval-primfun (powr-primfun } f\ e)\ x = \text{eval-primfun } f\ x \text{ powr } e$
 $\langle proof \rangle$

definition *eval-primfuns where*
 $\text{eval-primfuns } fs\ x = (\prod f \leftarrow fs. \text{eval-primfun } f\ x)$

lemma *eval-primfuns-pos: eventually $(\lambda x. \text{eval-primfuns } fs\ x > 0)$ at-top*
 $\langle proof \rangle$

lemma *eval-primfuns-nonzero: eventually $(\lambda x. \text{eval-primfuns } fs\ x \neq 0)$ at-top*
 $\langle proof \rangle$

2.5 Reification

definition *LANDAU-PROD' where*
 $\text{LANDAU-PROD' } L\ c\ f = L(\lambda x. c * f\ x)$

definition *LANDAU-PROD* where

LANDAU-PROD $L\ c1\ c2\ fs \longleftrightarrow (\lambda-. c1) \in L(\lambda x. c2 * \text{eval-primfun}\ fs\ x)$

definition *BIGTHETA-CONST'* where *BIGTHETA-CONST'* $c = \Theta(\lambda x. c)$

definition *BIGTHETA-CONST* where *BIGTHETA-CONST* $c\ A = \text{set-mult}\ \Theta(\lambda-. c)\ A$

definition *BIGTHETA-FUN* where *BIGTHETA-FUN* $f = \Theta(f)$

lemma *BIGTHETA-CONST'-tag*: $\Theta(\lambda x. c) = \text{BIGTHETA-CONST}'\ c$ *<proof>*

lemma *BIGTHETA-CONST-tag*: $\Theta(f) = \text{BIGTHETA-CONST}\ 1\ \Theta(f)$
<proof>

lemma *BIGTHETA-FUN-tag*: $\Theta(f) = \text{BIGTHETA-FUN}\ f$
<proof>

lemma *set-mult-is-times*: $\text{set-mult}\ A\ B = A * B$
<proof>

lemma *set-powr-mult*:

assumes *eventually-nonneg F f* and *eventually-nonneg F g*

shows $\Theta[F](\lambda x. (f\ x * g\ x :: \text{real})\ \text{powr}\ p) = \text{set-mult}\ (\Theta[F](\lambda x. f\ x\ \text{powr}\ p))$
 $(\Theta[F](\lambda x. g\ x\ \text{powr}\ p))$
<proof>

lemma *eventually-nonneg-bigtheta-pow-realpow*:

$\Theta(\lambda x. \text{eval-primfun}\ f\ x\ \hat{e}) = \Theta(\lambda x. \text{eval-primfun}\ f\ x\ \text{powr}\ \text{real}\ e)$
<proof>

lemma *BIGTHETA-CONST-fold*:

$\text{BIGTHETA-CONST}\ (c::\text{real})\ (\text{BIGTHETA-CONST}\ d\ A) = \text{BIGTHETA-CONST}\ (c*d)\ A$

bigtheta-pow at-top $(\text{BIGTHETA-CONST}\ c\ \Theta(\text{eval-primfun}\ pf))\ k =$

$\text{BIGTHETA-CONST}\ (c\ \hat{k})\ \Theta(\lambda x. \text{eval-primfun}\ pf\ x\ \text{powr}\ k)$

set-inverse $(\text{BIGTHETA-CONST}\ c\ \Theta(f)) = \text{BIGTHETA-CONST}\ (\text{inverse}\ c)$
 $\Theta(\lambda x. \text{inverse}\ (f\ x))$

set-mult $(\text{BIGTHETA-CONST}\ c\ \Theta(f))\ (\text{BIGTHETA-CONST}\ d\ \Theta(g)) =$

$\text{BIGTHETA-CONST}\ (c*d)\ \Theta(\lambda x. f\ x*g\ x)$

BIGTHETA-CONST' $(c::\text{real}) = \text{BIGTHETA-CONST}\ c\ \Theta(\lambda-. 1)$

BIGTHETA-FUN $(f::\text{real}\Rightarrow\text{real}) = \text{BIGTHETA-CONST}\ 1\ \Theta(f)$

<proof>

lemma *fold-fun-chain*:

$g\ x = (g\ \hat{\hat{1}}\ 1)\ x\ (g\ \hat{\hat{m}}\ m)\ ((g\ \hat{\hat{n}}\ n)\ x) = (g\ \hat{\hat{(m+n)}}\ (m+n))\ x$
<proof>

lemma *reify-ln-chain-1*:

$\Theta(\lambda x. (\ln\ \hat{\hat{n}}\ n)\ x) = \Theta(\text{eval-primfun}\ (\text{LnChain}\ n,\ 1))$
<proof>

lemma *reify-monom-1*:

$\Theta(\lambda x::\text{real}. x) = \Theta(\text{eval-primfun } (\text{LnChain } 0, 1))$
<proof>

lemma *reify-monom-pow*:

$\Theta(\lambda x::\text{real}. x \wedge e) = \Theta(\text{eval-primfun } (\text{LnChain } 0, \text{real } e))$
<proof>

lemma *reify-monom-powr*:

$\Theta(\lambda x::\text{real}. x \text{ powr } e) = \Theta(\text{eval-primfun } (\text{LnChain } 0, e))$
<proof>

lemmas *reify-monom* = *reify-monom-1 reify-monom-pow reify-monom-powr*

lemma *reify-ln-chain-pow*:

$\Theta(\lambda x. (\text{ln } \wedge n) x \wedge e) = \Theta(\text{eval-primfun } (\text{LnChain } n, \text{real } e))$
<proof>

lemma *reify-ln-chain-powr*:

$\Theta(\lambda x. (\text{ln } \wedge n) x \text{ powr } e) = \Theta(\text{eval-primfun } (\text{LnChain } n, e))$
<proof>

lemmas *reify-ln-chain* = *reify-ln-chain-1 reify-ln-chain-pow reify-ln-chain-powr*

lemma *numeral-power-Suc*: *numeral* $n \wedge \text{Suc } a = \text{numeral } n * \text{numeral } n \wedge a$
<proof>

lemmas *landau-product-preprocess* =

one-add-one one-plus-numeral numeral-plus-one arith-simps numeral-power-Suc
power-0
fold-fun-chain[where g = ln] reify-ln-chain reify-monom

lemma *LANDAU-PROD'-fold*:

BIGTHETA-CONST $e \Theta(\lambda-. d) = \text{BIGTHETA-CONST } (e*d) \Theta(\text{eval-primfuns } [])$
LANDAU-PROD' c $(\lambda-. 1) = \text{LANDAU-PROD' } c (\text{eval-primfuns } [])$
eval-primfun $f = \text{eval-primfuns } [f]$
eval-primfuns $fs x * \text{eval-primfuns } gs x = \text{eval-primfuns } (fs @ gs) x$
<proof>

lemma *inverse-prod-list-field*:

prod-list $(\text{map } (\lambda x. \text{inverse } (f x)) xs) = \text{inverse } (\text{prod-list } (\text{map } f xs :: - :: \text{field list}))$
<proof>

lemma *landau-prod-meta-cong*:
assumes *landau-symbol* $L L' Lr$
assumes $\Theta(f) \equiv \text{BIGTHETA-CONST } c1 \ (\Theta(\text{eval-primfun } fs))$
assumes $\Theta(g) \equiv \text{BIGTHETA-CONST } c2 \ (\Theta(\text{eval-primfun } gs))$
shows $f \in L \text{ at-top } (g) \equiv \text{LANDAU-PROD } (L \text{ at-top}) \ c1 \ c2 \ (\text{map inverse-primfun } fs \ @ \ gs)$
 $\langle \text{proof} \rangle$

fun *pos-primfun-list* **where**
 $\text{pos-primfun-list } [] \longleftrightarrow \text{False}$
 $|\ \text{pos-primfun-list } ((-,x)\#xs) \longleftrightarrow x > 0 \vee (x = 0 \wedge \text{pos-primfun-list } xs)$

fun *nonneg-primfun-list* **where**
 $\text{nonneg-primfun-list } [] \longleftrightarrow \text{True}$
 $|\ \text{nonneg-primfun-list } ((-,x)\#xs) \longleftrightarrow x > 0 \vee (x = 0 \wedge \text{nonneg-primfun-list } xs)$

fun *iszero-primfun-list* **where**
 $\text{iszero-primfun-list } [] \longleftrightarrow \text{True}$
 $|\ \text{iszero-primfun-list } ((-,x)\#xs) \longleftrightarrow x = 0 \wedge \text{iszero-primfun-list } xs$

definition *group-primfun* $\equiv \text{groupsort.group-sort fst merge-primfun}$

lemma *list-ConsCons-induct*:
assumes $P [] \wedge x. P [x] \wedge x \ y \ xs. P (y\#xs) \implies P (x\#y\#xs)$
shows $P xs$
 $\langle \text{proof} \rangle$

lemma *landau-function-family-chain-primfun*:
assumes *sorted* $(\text{map fst } fs)$
assumes *distinct* $(\text{map fst } fs)$
shows *landau-function-family-chain at-top* $fs \ (\text{eval-primfun}' \ o \ \text{fst})$
 $\langle \text{proof} \rangle$

lemma **(in** *monoid-mult*) *fold-plus-prod-list-rev*:
 $\text{fold times } xs = \text{times } (\text{prod-list } (\text{rev } xs))$
 $\langle \text{proof} \rangle$

interpretation *groupsort-primfun*: *groupsort fst merge-primfun eval-primfun*
 $\langle \text{proof} \rangle$

lemma *nonneg-primfun-list-iff*: *nonneg-primfun-list* $fs = \text{nonneg-list } (\text{map snd } fs)$
 $\langle \text{proof} \rangle$

lemma *pos-primfun-list-iff*: *pos-primfun-list* $fs = \text{pos-list } (\text{map snd } fs)$
 $\langle \text{proof} \rangle$

lemma *iszero-primfun-list-iff*: *iszero-primfun-list* $fs = \text{list-all } ((=) \ 0) \ (\text{map snd } fs)$

fs)
<proof>

lemma *landau-primfun-iff*:

$((\lambda-. 1) \in O(\text{eval-primfun-} fs)) = \text{nonneg-primfun-list } (group\text{-primfun-} fs)$ (**is** ?A)
 $((\lambda-. 1) \in o(\text{eval-primfun-} fs)) = \text{pos-primfun-list } (group\text{-primfun-} fs)$ (**is** ?B)
 $((\lambda-. 1) \in \Theta(\text{eval-primfun-} fs)) = \text{iszero-primfun-list } (group\text{-primfun-} fs)$ (**is** ?C)
<proof>

lemma *LANDAU-PROD-bigo-iff*:

$LANDAU\text{-}PROD (bigo\text{ at-top})\ c1\ c2\ fs \longleftrightarrow c1 = 0 \vee (c2 \neq 0 \wedge \text{nonneg-primfun-list } (group\text{-primfun-} fs))$
<proof>

lemma *LANDAU-PROD-smallo-iff*:

$LANDAU\text{-}PROD (smallo\text{ at-top})\ c1\ c2\ fs \longleftrightarrow c1 = 0 \vee (c2 \neq 0 \wedge \text{pos-primfun-list } (group\text{-primfun-} fs))$
<proof>

lemma *LANDAU-PROD-bigtheta-iff*:

$LANDAU\text{-}PROD (bigtheta\text{ at-top})\ c1\ c2\ fs \longleftrightarrow (c1 = 0 \wedge c2 = 0) \vee (c1 \neq 0 \wedge c2 \neq 0 \wedge \text{iszero-primfun-list } (group\text{-primfun-} fs))$
<proof>

lemmas *LANDAU-PROD-iff = LANDAU-PROD-bigo-iff LANDAU-PROD-smallo-iff LANDAU-PROD-bigtheta-iff*

lemmas *landau-real-prod-simps [simp] =*

group-sort-primfun.group-part-def
group-primfun-def group-sort-primfun.group-sort.simps
group-sort-primfun.group-part-aux.simps pos-primfun-list.simps
nonneg-primfun-list.simps iszero-primfun-list.simps

end

3 Simplification procedures

theory *Landau-Simprocs*

imports *Landau-Real-Products*

begin

3.1 Simplification under Landau symbols

The following can be seen as simpset for terms under Landau symbols. When given a rule $f \in \Theta(g)$, the simproc will attempt to rewrite any occurrence of f under a Landau symbol to g .

named-theorems *landau-simp* *BigTheta* rules for simplification of Landau symbols
 (ML)

lemma *bigheta-const* [*landau-simp*]:

NO-MATCH $1\ c \implies c \neq 0 \implies (\lambda x. c) \in \Theta(\lambda x. 1)$ (proof)

lemmas [*landau-simp*] = *bigheta-const-ln* *bigheta-const-ln-powr* *bigheta-const-ln-pow*

lemma *bigheta-const-ln'* [*landau-simp*]:

$0 < a \implies (\lambda x::real. \ln (x * a)) \in \Theta(\ln)$
 (proof)

lemma *bigheta-const-ln-powr'* [*landau-simp*]:

$0 < a \implies (\lambda x::real. \ln (x * a) \text{ powr } p) \in \Theta(\lambda x. \ln x \text{ powr } p)$
 (proof)

lemma *bigheta-const-ln-pow'* [*landau-simp*]:

$0 < a \implies (\lambda x::real. \ln (x * a) \hat{\ } p) \in \Theta(\lambda x. \ln x \hat{\ } p)$
 (proof)

3.2 Simproc setup

lemma *landau-gt-1-cong*:

landau-symbol $L\ L'\ Lr \implies (\bigwedge x::real. x > 1 \implies f\ x = g\ x) \implies L\ \text{at-top}\ (f) = L\ \text{at-top}\ (g)$
 (proof)

lemma *landau-gt-1-in-cong*:

landau-symbol $L\ L'\ Lr \implies (\bigwedge x::real. x > 1 \implies f\ x = g\ x) \implies f \in L\ \text{at-top}\ (h) \iff g \in L\ \text{at-top}\ (h)$
 (proof)

lemma *landau-prop-equalsI*:

landau-symbol $L\ L'\ Lr \implies (\bigwedge x::real. x > 1 \implies f1\ x = f2\ x) \implies (\bigwedge x. x > 1 \implies g1\ x = g2\ x) \implies f1 \in L\ \text{at-top}\ (g1) \iff f2 \in L\ \text{at-top}\ (g2)$
 (proof)

lemma *ab-diff-conv-add-uminus'*: $(a:::ab\text{-group-add}) - b = -b + a$ (proof)

lemma *extract-diff-middle*: $(a:::ab\text{-group-add}) - (x + b) = -x + (a - b)$ (proof)

lemma *divide-inverse'*: $(a :: \{ \text{division-ring, ab-semigroup-mult} \}) / b = \text{inverse } b * a$

<proof>

lemma *extract-divide-middle*: $(a :: \{ \text{field} \}) / (x * b) = \text{inverse } x * (a / b)$

<proof>

lemmas *landau-cancel* = *landau-symbol.mult-cancel-left*

lemmas *mult-cancel-left'* = *landau-symbol.mult-cancel-left*[*OF - bitheta-refl eventually-nonzeroD*]

lemma *mult-cancel-left-1*:

assumes *landau-symbol L L' Lr eventually-nonzero F f*

shows $f \in L F (\lambda x. f x * g2 x) \longleftrightarrow (\lambda-. 1) \in L F (g2)$

$(\lambda x. f x * f2 x) \in L F (f) \longleftrightarrow f2 \in L F (\lambda-. 1)$

$f \in L F (f) \longleftrightarrow (\lambda-. 1) \in L F (\lambda-. 1)$

<proof>

lemmas *landau-mult-cancel-simps* = *mult-cancel-left' mult-cancel-left-1*

<ML>

lemmas *bitheta-simps* =

landau-theta.cong-bitheta[*OF bitheta-const-ln*]

landau-theta.cong-bitheta[*OF bitheta-const-ln-powr*]

The following simproc attempts to cancel common factors in Landau symbols, i. e. in a goal like $f(x)h(x) \in L(g(x)h(x))$, the common factor $h(x)$ will be cancelled. This only works if the simproc can prove that $h(x)$ is eventually non-zero, for which it uses some heuristics.

<ML>

The next simproc attempts to cancel dominated summands from Landau symbols; e. g. $O(x + \ln x)$ is simplified to $O(x)$, since $\ln x \in o(x)$. This can be very slow on large terms, so it is not enabled by default.

<ML>

This simproc attempts to simplify factors of an expression in a Landau symbol statement independently from another, i. e. in something like $O(f(x)g(x))$, a simp rule that rewrites $O(f(x))$ to $O(f'(x))$ will also rewrite $O(f(x)g(x))$ to $O(f'(x)g(x))$ without any further setup.

<ML>

Lastly, the next very specialised simproc can solve goals of the form $f(x) \in L(g(x))$ where f and g are real-valued functions consisting only of multiplications, powers of x , and powers of iterated logarithms of x . This is done by rewriting both sides into the form $x^a(\ln x)^b(\ln \ln x)^c$ etc. and then comparing the exponents lexicographically.

Note that for historic reasons, this only works for $x \rightarrow \infty$.

$\langle ML \rangle$

3.3 Tests

lemma *asympt-equiv-plus-const-left*: $(\lambda n. c + \text{real } n) \sim[at-top] (\lambda n. \text{real } n)$
 $\langle proof \rangle$

lemma *asympt-equiv-plus-const-right*: $(\lambda n. \text{real } n + c) \sim[at-top] (\lambda n. \text{real } n)$
 $\langle proof \rangle$

3.3.1 Product simplification tests

lemma $(\lambda x::\text{real}. f\ x * x) \in O(\lambda x. g\ x / (h\ x / x)) \iff f \in O(\lambda x. g\ x / h\ x)$
 $\langle proof \rangle$

lemma $(\lambda x::\text{real}. x) \in \omega(\lambda x. g\ x / (h\ x / x)) \iff (\lambda x. 1) \in \omega(\lambda x. g\ x / h\ x)$
 $\langle proof \rangle$

3.3.2 Real product decision procure tests

lemma $(\lambda x. x \text{ powr } 1) \in O(\lambda x. x \text{ powr } 2 :: \text{real})$
 $\langle proof \rangle$

lemma $\Theta(\lambda x::\text{real}. 2*x \text{ powr } 3 - 4*x \text{ powr } 2) = \Theta(\lambda x::\text{real}. x \text{ powr } 3)$
 $\langle proof \rangle$

lemma $p < q \implies (\lambda x::\text{real}. c * x \text{ powr } p * \ln x \text{ powr } r) \in o(\lambda x::\text{real}. x \text{ powr } q)$
 $\langle proof \rangle$

lemma $c \neq 0 \implies p > q \implies (\lambda x::\text{real}. c * x \text{ powr } p * \ln x \text{ powr } r) \in \omega(\lambda x::\text{real}. x \text{ powr } q)$
 $\langle proof \rangle$

lemma $b > 0 \implies (\lambda x::\text{real}. x / \ln (2*b*x) * 2) \in o(\lambda x. x * \ln (b*x))$
 $\langle proof \rangle$

lemma $o(\lambda x::\text{real}. x * \ln (3*x)) = o(\lambda x. \ln x * x)$
 $\langle proof \rangle$

lemma $(\lambda x::\text{real}. x) \in o(\lambda x. x * \ln (3*x))$ $\langle proof \rangle$

$\langle ML \rangle$

lemma $(\lambda x. 3 * \ln x * \ln x / x * \ln (\ln (\ln (\ln x)))) \in$
 $\omega(\lambda x::\text{real}. 5 * \ln (\ln x) ^ 2 / (2*x) \text{ powr } 1.5 * \text{inverse } 2)$
 $\langle proof \rangle$

3.3.3 Sum cancelling tests

lemma $\Theta(\lambda x::\text{real}. 2 * x \text{ powr } 3 + x * x^2 / \ln x) = \Theta(\lambda x::\text{real}. x \text{ powr } 3)$
 $\langle proof \rangle$

lemma $\Theta(\lambda x :: \text{real}. 2 * x \text{ powr } 3 + x * x^2 / \ln x + 42 * x \text{ powr } 9 + 213 * x \text{ powr } 5 - 4 * x \text{ powr } 7) =$
 $\Theta(\lambda x :: \text{real}. x^3 + x / \ln x * x \text{ powr } (3/2) - 2 * x \text{ powr } 9)$
 ⟨proof⟩

lemma $(\lambda x :: \text{real}. x + x * \ln (3 * x)) \in o(\lambda x :: \text{real}. x^2 + \ln (2 * x) \text{ powr } 3)$ ⟨proof⟩

end

theory *Landau-More*

imports

HOL-Library.Landau-Symbols

Landau-Simprocs

begin

lemma *bigconst-inverse* [simp]:
assumes *filterlim f at-top F F ≠ bot*
shows $(\lambda-. c) \in O[F](\lambda x. \text{inverse } (f x) :: \text{real}) \longleftrightarrow c = 0$
 ⟨proof⟩

lemma *smallo-const-inverse* [simp]:
 $\text{filterlim } f \text{ at-top } F \implies F \neq \text{bot} \implies (\lambda-. c :: \text{real}) \in o[F](\lambda x. \text{inverse } (f x)) \longleftrightarrow$
 $c = 0$
 ⟨proof⟩

lemma *const-in-smallo-const* [simp]: $(\lambda-. b) \in o(\lambda- :: - :: \text{linorder}. c) \longleftrightarrow b = 0$
 (is ?lhs \longleftrightarrow ?rhs)
 ⟨proof⟩

lemma *smallomega-1-conv-filterlim*: $f \in \omega[F](\lambda-. 1) \longleftrightarrow \text{filterlim } f \text{ at-infinity } F$
 ⟨proof⟩

lemma *bigtheta-powr-1* [landau-simp]:
 $\text{eventually } (\lambda x. (f x :: \text{real}) \geq 0) F \implies (\lambda x. f x \text{ powr } 1) \in \Theta[F](f)$
 ⟨proof⟩

lemma *bigtheta-powr-0* [landau-simp]:
 $\text{eventually } (\lambda x. (f x :: \text{real}) \neq 0) F \implies (\lambda x. f x \text{ powr } 0) \in \Theta[F](\lambda-. 1)$
 ⟨proof⟩

lemma *bigtheta-powr-nonzero* [landau-simp]:
 $\text{eventually } (\lambda x. (f x :: \text{real}) \neq 0) F \implies (\lambda x. \text{if } f x = 0 \text{ then } g x \text{ else } h x) \in$
 $\Theta[F](h)$
 ⟨proof⟩

lemma *bigtheta-powr-nonzero'* [landau-simp]:

eventually $(\lambda x. (f x :: \text{real}) \neq 0) F \implies (\lambda x. \text{if } f x \neq 0 \text{ then } g x \text{ else } h x) \in \Theta[F](g)$
 ⟨proof⟩

lemma *bigtheta-powr-nonneg* [landau-simp]:

eventually $(\lambda x. (f x :: \text{real}) \geq 0) F \implies (\lambda x. \text{if } f x \geq 0 \text{ then } g x \text{ else } h x) \in \Theta[F](g)$
 ⟨proof⟩

lemma *bigtheta-powr-nonneg'* [landau-simp]:

eventually $(\lambda x. (f x :: \text{real}) \geq 0) F \implies (\lambda x. \text{if } f x < 0 \text{ then } g x \text{ else } h x) \in \Theta[F](h)$
 ⟨proof⟩

lemma *bigo-powr-iff*:

assumes $0 < p$ eventually $(\lambda x. f x \geq 0) F$ eventually $(\lambda x. g x \geq 0) F$
 shows $(\lambda x. (f x :: \text{real}) \text{ powr } p) \in O[F](\lambda x. g x \text{ powr } p) \longleftrightarrow f \in O[F](g)$ (is ?lhs
 \longleftrightarrow ?rhs)
 ⟨proof⟩

lemma *inverse-powr* [simp]:

assumes $(x :: \text{real}) \geq 0$
 shows $\text{inverse } x \text{ powr } y = \text{inverse } (x \text{ powr } y)$
 ⟨proof⟩

lemma *bigo-neg-powr-iff*:

assumes $p < 0$ eventually $(\lambda x. f x \geq 0) F$ eventually $(\lambda x. g x \geq 0) F$
 eventually $(\lambda x. f x \neq 0) F$ eventually $(\lambda x. g x \neq 0) F$
 shows $(\lambda x. (f x :: \text{real}) \text{ powr } p) \in O[F](\lambda x. g x \text{ powr } p) \longleftrightarrow g \in O[F](f)$ (is ?lhs
 \longleftrightarrow ?rhs)
 ⟨proof⟩

lemma *smallo-powr-iff*:

assumes $0 < p$ eventually $(\lambda x. f x \geq 0) F$ eventually $(\lambda x. g x \geq 0) F$
 shows $(\lambda x. (f x :: \text{real}) \text{ powr } p) \in o[F](\lambda x. g x \text{ powr } p) \longleftrightarrow f \in o[F](g)$ (is ?lhs
 \longleftrightarrow ?rhs)
 ⟨proof⟩

lemma *smallo-neg-powr-iff*:

assumes $p < 0$ eventually $(\lambda x. f x \geq 0) F$ eventually $(\lambda x. g x \geq 0) F$
 eventually $(\lambda x. f x \neq 0) F$ eventually $(\lambda x. g x \neq 0) F$
 shows $(\lambda x. (f x :: \text{real}) \text{ powr } p) \in o[F](\lambda x. g x \text{ powr } p) \longleftrightarrow g \in o[F](f)$ (is ?lhs
 \longleftrightarrow ?rhs)
 ⟨proof⟩

lemma *const-smallo-powr*:

assumes *filterlim f at-top F F* $F \neq \text{bot}$
 shows $(\lambda c. c :: \text{real}) \in o[F](\lambda x. f x \text{ powr } p) \longleftrightarrow p > 0 \vee c = 0$
 ⟨proof⟩

lemma *bigO-const-powr*:

assumes *filterlim f at-top F F ≠ bot*

shows $(\lambda-. c :: \text{real}) \in O[F](\lambda x. f x \text{ powr } p) \longleftrightarrow p \geq 0 \vee c = 0$
<proof>

lemma *filterlim-powr-at-top*:

$(b :: \text{real}) > 1 \implies \text{filterlim } (\lambda x. b \text{ powr } x) \text{ at-top at-top}$

<proof>

lemma *power-smallo-exponential*:

fixes $b :: \text{real}$

assumes $b: b > 1$

shows $(\lambda x. x \text{ powr } n) \in o(\lambda x. b \text{ powr } x)$
<proof>

lemma *powr-fast-growth-tendsto*:

assumes $gf: g \in O[F](f)$

and $n: n \geq 0$

and $k: k > 1$

and $f: \text{filterlim } f \text{ at-top } F$

and $g: \text{eventually } (\lambda x. g x \geq 0) F$

shows $(\lambda x. g x \text{ powr } n) \in o[F](\lambda x. k \text{ powr } f x :: \text{real})$
<proof>

lemma *bigO-abs-powr-iff [simp]*:

$0 < p \implies (\lambda x. |f x :: \text{real}| \text{ powr } p) \in O[F](\lambda x. |g x| \text{ powr } p) \longleftrightarrow f \in O[F](g)$

<proof>

lemma *smallo-abs-powr-iff [simp]*:

$0 < p \implies (\lambda x. |f x :: \text{real}| \text{ powr } p) \in o[F](\lambda x. |g x| \text{ powr } p) \longleftrightarrow f \in o[F](g)$

<proof>

lemma *const-smallo-inverse-powr*:

assumes *filterlim f at-top at-top*

shows $(\lambda- :: - :: \text{linorder}. c :: \text{real}) \in o(\lambda x. \text{inverse } (f x \text{ powr } p)) \longleftrightarrow (p \geq 0 \longrightarrow c = 0)$

<proof>

lemma *bigO-const-inverse-powr*:

assumes *filterlim f at-top at-top*

shows $(\lambda- :: - :: \text{linorder}. c :: \text{real}) \in O(\lambda x. \text{inverse } (f x \text{ powr } p)) \longleftrightarrow c = 0 \vee p \leq 0$

<proof>

end