Landau Symbols

Manuel Eberl

March 17, 2025

Contents

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*⟩ **lemmas** *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*) ⇒ *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* (λ*y*. *f y* = *f x*) *xs of* (*xs'*, *xs''*) ⇒
fold merge xs' x # *group xs''*)

private lemma *f-fold-merge*: (⋀*y*. *y* ∈ *set xs* ⟹ *f y* = *f x*) ⟹ *f* (*fold merge xs x*) = *f x*
⟨*proof*⟩ **lemma** *f-group*: *x* ∈ *set* (*group xs*) ⟹ ∃ *x'*∈*set xs*. *f x* = *f x'*
⟨*proof*⟩ **lemma** *sorted-group*: *sorted* (*map f xs*) ⟹ *sorted* (*map f* (*group xs*))
⟨*proof*⟩ **lemma** *distinct-group*: *distinct* (*map f* (*group xs*))
⟨*proof*⟩ **lemma** *g-fold-same*:
assumes ⋀*z*. *z* ∈ *set xs* ⟹ *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 ⇒ *'a list* ⇒ (*'a list*) × *'a* × (*'a list*) ⇒ (*'a list*) × *'a* × (*'a list*)
where
group-part-aux p [] (*ls*, *eq*, *gs*) = (*ls*, *eq*, *gs*)
| *f x* < *p* ⟹ *group-part-aux p* (*x#xs*) (*ls*, *eq*, *gs*) = *group-part-aux p xs* (*x#ls*, *eq*, *gs*)
| *f x* > *p* ⟹ *group-part-aux p* (*x#xs*) (*ls*, *eq*, *gs*) = *group-part-aux p xs* (*ls*, *eq*, *x#gs*)
| *f x* = *p* ⟹ *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* (λ*x*. *f x* < *f p*) *xs*),
fold merge (*filter* (λ*x*. *f x* = *f p*) *xs*) *p*, *rev* (*filter* (λ*x*. *f x* > *f p*) *xs*))
⟨*proof*⟩

function *group-sort* :: *'a list* ⇒ *'a list* **where**
group-sort [] = []
| *group-sort* (*x#xs*) = (*case group-part x xs of* (*ls*, *eq*, *gs*) ⇒ *group-sort ls* @ *eq* # *group-sort gs*)
⟨*proof*⟩
termination ⟨*proof*⟩ **lemma** *group-append*:
assumes ⋀*x y*. *x* ∈ *set xs* ⟹ *y* ∈ *set ys* ⟹ *f x* ≠ *f y*
shows *group* (*xs* @ *ys*) = *group xs* @ *group ys*
⟨*proof*⟩ **lemma** *group-empty-iff* [*simp*]: *group xs* = [] ⟷ *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 of 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* ⟷ *eventually* (λ*x*. (*f x* :: - :: *real-normed-field*) ≠ *0*) *F*
definition *eventually-nonneg F f* ⟷ *eventually* (λ*x*. (*f x* :: - :: *linordered-field*) ≥ *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* ⟹ *eventually* (λ*x*. *f x* ≠ *0*) *F*

⟨*proof*⟩

lemma *eventually-nonzero-const* [*eventually-nonzero-simps*]:
eventually-nonzero F (λ-::-::*linorder*. *c*) ⟷ *F* = *bot* ∨ *c* ≠ *0*
⟨*proof*⟩

lemma *eventually-nonzero-inverse* [*eventually-nonzero-simps*]:
eventually-nonzero F (λ*x*. *inverse* (*f x*)) ⟷ *eventually-nonzero F f*
⟨*proof*⟩

lemma *eventually-nonzero-mult* [*eventually-nonzero-simps*]:
eventually-nonzero F (λ*x*. *f x* ∗ *g x*) ⟷ *eventually-nonzero F f* ∧ *eventually-nonzero F g*
⟨*proof*⟩

lemma *eventually-nonzero-pow* [*eventually-nonzero-simps*]:
eventually-nonzero F (λ*x*::-::*linorder*. *f x* ^ *n*) ⟷ *n* = *0* ∨ *eventually-nonzero F f*
⟨*proof*⟩

lemma *eventually-nonzero-divide* [*eventually-nonzero-simps*]:
eventually-nonzero F (λ*x*. *f x* / *g x*) ⟷ *eventually-nonzero F f* ∧ *eventually-nonzero F g*
⟨*proof*⟩

lemma *eventually-nonzero-ident-at-top-linorder* [*eventually-nonzero-simps*]:
eventually-nonzero at-top (λ*x*::′*a*::{*real-normed-field*,*linordered-field*}. *x*)
⟨*proof*⟩

lemma *eventually-nonzero-ident-nhds* [*eventually-nonzero-simps*]:
eventually-nonzero (*nhds a*) (λ*x*. *x*) ⟷ *a* ≠ *0*
⟨*proof*⟩

lemma *eventually-nonzero-ident-at-within* [*eventually-nonzero-simps*]:
eventually-nonzero (*at a within A*) (λ*x*. *x*)
⟨*proof*⟩

lemma *eventually-nonzero-ln-at-top* [*eventually-nonzero-simps*]:
eventually-nonzero at-top (λ*x*::*real*. *ln x*)
⟨*proof*⟩

lemma *eventually-nonzero-ln-const-at-top* [*eventually-nonzero-simps*]:
b > *0* ⟹ *eventually-nonzero at-top* (λ*x*. *ln* (*b* ∗ *x* :: *real*))
⟨*proof*⟩

lemma *eventually-nonzero-ln-const′-at-top* [*eventually-nonzero-simps*]:
b > *0* ⟹ *eventually-nonzero at-top* (λ*x*. *ln* (*x* ∗ *b* :: *real*))
⟨*proof*⟩

lemma *eventually-nonzero-powr-at-top* [*eventually-nonzero-simps*]:
eventually-nonzero at-top (λx::*real*. f x *powr* p) $\longleftrightarrow$ *eventually-nonzero at-top* f
⟨*proof*⟩

lemma *eventually-nonneg-const* [*eventually-nonzero-simps*]:
eventually-nonneg F (λ-. c) $\longleftrightarrow$ F = *bot* $\vee$ $c \geq 0$
⟨*proof*⟩

lemma *eventually-nonneg-inverse* [*eventually-nonzero-simps*]:
eventually-nonneg F (λx. *inverse* (f x)) $\longleftrightarrow$ *eventually-nonneg* F f
⟨*proof*⟩

lemma *eventually-nonneg-add* [*eventually-nonzero-simps*]:
assumes *eventually-nonneg* F f *eventually-nonneg* F g
shows *eventually-nonneg* F (λx. f x + g x)
⟨*proof*⟩

lemma *eventually-nonneg-mult* [*eventually-nonzero-simps*]:
assumes *eventually-nonneg* F f *eventually-nonneg* F g
shows *eventually-nonneg* F (λx. f x $*$ g x)
⟨*proof*⟩

lemma *eventually-nonneg-mult′* [*eventually-nonzero-simps*]:
assumes *eventually-nonneg* F (λx. $-f$ x) *eventually-nonneg* F (λx. $-$ g x)
shows *eventually-nonneg* F (λx. f x $*$ g x)
⟨*proof*⟩

lemma *eventually-nonneg-divide* [*eventually-nonzero-simps*]:
assumes *eventually-nonneg* F f *eventually-nonneg* F g
shows *eventually-nonneg* F (λx. f x / g x)
⟨*proof*⟩

lemma *eventually-nonneg-divide′* [*eventually-nonzero-simps*]:
assumes *eventually-nonneg* F (λx. $-f$ x) *eventually-nonneg* F (λx. $-$ g x)
shows *eventually-nonneg* F (λx. f x / g x)
⟨*proof*⟩

lemma *eventually-nonneg-ident-at-top* [*eventually-nonzero-simps*]:
eventually-nonneg at-top (λx. x) ⟨*proof*⟩

lemma *eventually-nonneg-ident-nhds* [*eventually-nonzero-simps*]:
fixes a :: $'a$ :: {*linorder-topology*, *linordered-field*}
shows $a > 0 \Longrightarrow$ *eventually-nonneg* (*nhds* a) (λx. x) ⟨*proof*⟩

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

⟨*proof*⟩

lemma *eventually-nonneg-pow* [*eventually-nonzero-simps*]:
eventually-nonneg F f $\Longrightarrow$ *eventually-nonneg F* ($\lambda x.$ *f x* ^ *n*)
⟨*proof*⟩

lemma *eventually-nonneg-powr* [*eventually-nonzero-simps*]:
eventually-nonneg F ($\lambda x.$ *f x powr y* :: *real*) ⟨*proof*⟩

lemma *eventually-nonneg-ln-at-top* [*eventually-nonzero-simps*]:
eventually-nonneg at-top ($\lambda x.$ *ln x* :: *real*)
⟨*proof*⟩

lemma *eventually-nonneg-ln-const* [*eventually-nonzero-simps*]:
b > *0* $\Longrightarrow$ *eventually-nonneg at-top* ($\lambda x.$ *ln* (*b*∗*x*) :: *real*)
⟨*proof*⟩

lemma *eventually-nonneg-ln-const'* [*eventually-nonzero-simps*]:
b > *0* $\Longrightarrow$ *eventually-nonneg at-top* ($\lambda x.$ *ln* (*x*∗*b*) :: *real*)
⟨*proof*⟩

lemma *eventually-nonzero-bigtheta'*:
$f \in \Theta[F](g) \Longrightarrow$ *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* ⇒ *'b* :: {*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*) $\Longrightarrow$ *eventually-nonneg F f*
⟨*proof*⟩

lemma *eventually-nonzero-at-top-ASSUMPTION* [*eventually-nonzero-simps*]:
ASSUMPTION (*filterlim f* (*at-top* :: *'a* :: {*linordered-field*, *real-normed-field*} *filter*) *F*) $\Longrightarrow$
eventually-nonzero F f
⟨*proof*⟩

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

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

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

lemma *eventually-nonneg-add1* [*eventually-nonzero-simps*]:
assumes $\mathit{eventually\text{-}nonneg}\ F\ f\ g \in o[F](f)$
shows $\mathit{eventually\text{-}nonneg}\ F\ (\lambda x.\ f\ x + g\ x :: \mathit{real})$
⟨*proof*⟩

lemma *eventually-nonneg-add2* [*eventually-nonzero-simps*]:
assumes $\mathit{eventually\text{-}nonneg}\ F\ g\ f \in o[F](g)$
shows $\mathit{eventually\text{-}nonneg}\ F\ (\lambda x.\ f\ x + g\ x :: \mathit{real})$
⟨*proof*⟩

lemma *eventually-nonneg-diff1* [*eventually-nonzero-simps*]:
assumes $\mathit{eventually\text{-}nonneg}\ F\ f\ g \in o[F](f)$
shows $\mathit{eventually\text{-}nonneg}\ F\ (\lambda x.\ f\ x - g\ x :: \mathit{real})$
⟨*proof*⟩

lemma *eventually-nonneg-diff2* [*eventually-nonzero-simps*]:
assumes $\mathit{eventually\text{-}nonneg}\ F\ (\lambda x.\ -\ g\ x)\ f \in o[F](g)$
shows $\mathit{eventually\text{-}nonneg}\ F\ (\lambda x.\ f\ x - g\ x :: \mathit{real})$
⟨*proof*⟩

2.2 Rewriting Landau symbols

lemma *bigtheta-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 $\Theta(\lambda x.\ \mathit{log}\ \mathit{2}\ x * x)$ to $\Theta(\lambda x.\ \mathit{ln}\ x * x)$, we need to bring the term into something that contains $\Theta(\mathit{log}\ \mathit{2})$ 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 $\mathit{set\text{-}mult}\ A\ B = \{\lambda x.\ f\ x * g\ x\ |f\ g.\ f \in A \wedge g \in B\}$
definition $\mathit{set\text{-}inverse}\ A = \{\lambda x.\ \mathit{inverse}\ (f\ x)\ |f.\ f \in A\}$
definition $\mathit{set\text{-}divide}\ A\ B = \{\lambda x.\ f\ x\ /\ g\ x\ |f\ g.\ f \in A \wedge g \in B\}$
definition $\mathit{set\text{-}pow}\ A\ n = \{\lambda x.\ f\ x\ \widehat{}\ n\ |f.\ f \in A\}$
definition $\mathit{set\text{-}powr}\ A\ y = \{\lambda x.\ f\ x\ \mathit{powr}\ y\ |f.\ f \in A\}$

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

lemma *bigtheta-inverse-eq-set-inverse*:
shows $\Theta[F](\lambda x.\ \mathit{inverse}\ (f\ x)) = \mathit{set\text{-}inverse}\ (\Theta[F](f))$
⟨*proof*⟩

lemma *set-divide-inverse*:
$\mathit{set\text{-}divide}\ (A :: (\text{-} \Rightarrow (\text{-} :: \mathit{division\text{-}ring}))\ \mathit{set})\ B = \mathit{set\text{-}mult}\ A\ (\mathit{set\text{-}inverse}\ B)$
⟨*proof*⟩

lemma *bigtheta-divide-eq-set-divide*:
shows $\Theta[F](\lambda x.\ f\ x\ /\ g\ x) = \mathit{set\text{-}divide}\ (\Theta[F](f))\ (\Theta[F](g))$
⟨*proof*⟩

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

lemma *bigtheta-pow-eq-set-pow*: $\Theta[F](\lambda x.\ f\ x\ \hat{}\ n) = \mathit{bigtheta\text{-}pow}\ F\ (\Theta[F](f))\ n$
⟨*proof*⟩

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

lemma *bigtheta-powr-eq-set-powr*:
assumes *eventually-nonneg F f*
shows $\Theta[F](\lambda x.\ f\ x\ \mathit{powr}\ (y{::}\mathit{real})) = \mathit{bigtheta\text{-}powr}\ F\ (\Theta[F](f))\ y$
⟨*proof*⟩

lemmas *bigtheta-factors-eq* =
bigtheta-mult-eq-set-mult bigtheta-inverse-eq-set-inverse bigtheta-divide-eq-set-divide

bigtheta-pow-eq-set-pow bigtheta-powr-eq-set-powr

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

lemma (**in** *landau-symbol*) *meta-cong-bigtheta*: $\Theta[F](f) \equiv \Theta[F](g) \Longrightarrow L\ F\ (f) \equiv L\ F\ (g)$
⟨*proof*⟩

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

2.3 Preliminary facts

lemma *real-powr-at-top*:
assumes $(p{::}\mathit{real}) > 0$
shows $\mathit{filterlim}\ (\lambda x.\ x\ \mathit{powr}\ p)\ \mathit{at\text{-}top}\ \mathit{at\text{-}top}$

⟨*proof*⟩

lemma *tendsto-ln-over-powr*:
assumes $(a::real) > 0$
shows $((\lambda x.\ ln\ x\ /\ x\ powr\ a) \longrightarrow 0)\ at\text{-}top$
⟨*proof*⟩

lemma *tendsto-ln-powr-over-powr*:
assumes $(a::real) > 0\ b > 0$
shows $((\lambda x.\ ln\ x\ powr\ a\ /\ x\ powr\ b) \longrightarrow 0)\ at\text{-}top$
⟨*proof*⟩

lemma *tendsto-ln-powr-over-powr'*:
assumes $b > 0$
shows $((\lambda x::real.\ ln\ x\ powr\ a\ /\ x\ powr\ b) \longrightarrow 0)\ at\text{-}top$
⟨*proof*⟩

lemma *tendsto-ln-over-ln*:
assumes $(a::real) > 0\ c > 0$
shows $((\lambda x.\ ln\ (a*x)\ /\ ln\ (c*x)) \longrightarrow 1)\ at\text{-}top$
⟨*proof*⟩

lemma *tendsto-ln-powr-over-ln-powr*:
assumes $(a::real) > 0\ c > 0$
shows $((\lambda x.\ ln\ (a*x)\ powr\ d\ /\ ln\ (c*x)\ powr\ d) \longrightarrow 1)\ at\text{-}top$
⟨*proof*⟩

lemma *tendsto-ln-powr-over-ln-powr'*:
$c > 0 \Longrightarrow ((\lambda x::real.\ ln\ x\ powr\ d\ /\ ln\ (c*x)\ powr\ d) \longrightarrow 1)\ at\text{-}top$
⟨*proof*⟩

lemma *tendsto-ln-powr-over-ln-powr''*:
$a > 0 \Longrightarrow ((\lambda x::real.\ ln\ (a*x)\ powr\ d\ /\ ln\ x\ powr\ d) \longrightarrow 1)\ at\text{-}top$
⟨*proof*⟩

lemma *bigtheta-const-ln-powr* [*simp*]: $a > 0 \Longrightarrow (\lambda x::real.\ ln\ (a*x)\ powr\ d) \in \Theta(\lambda x.\ ln\ x\ powr\ d)$
⟨*proof*⟩

lemma *bigtheta-const-ln-pow* [*simp*]: $a > 0 \Longrightarrow (\lambda x::real.\ ln\ (a*x)\ \hat{}\ d) \in \Theta(\lambda x.\ ln\ x\ \hat{}\ d)$
⟨*proof*⟩

lemma *bigtheta-const-ln* [*simp*]: $a > 0 \Longrightarrow (\lambda x::real.\ ln\ (a*x)) \in \Theta(\lambda x.\ ln\ x)$
⟨*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, $\lambda x.\ ln\ (ln\ x)$, ... 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* ≡ {λx. *f x powr p* :: *real* |*p*. *True*}

lemma *powr-closureI* [*simp*]: (λx. *f x powr p*) ∈ *powr-closure f*
  ⟨*proof*⟩

lemma *powr-closureE*:
  assumes *g* ∈ *powr-closure f*
  obtains *p* **where** *g* = (λx. *f x powr p*)
  ⟨*proof*⟩

locale *landau-function-family* =
  fixes *F* :: *'a filter* **and** *H* :: (*'a* ⇒ *real*) *set*
  assumes *F-nontrivial*: *F* ≠ *bot*
  assumes *pos*: *h* ∈ *H* ⟹ *eventually* (λx. *h x* > *0*) *F*
  assumes *linear*: *h1* ∈ *H* ⟹ *h2* ∈ *H* ⟹ *h1* ∈ *o*[*F*](*h2*) ∨ *h2* ∈ *o*[*F*](*h1*) ∨ *h1* ∈ Θ[*F*](*h2*)
  assumes *mult*: *h1* ∈ *H* ⟹ *h2* ∈ *H* ⟹ (λx. *h1 x* ∗ *h2 x*) ∈ *H*
  assumes *inverse*: *h* ∈ *H* ⟹ (λx. *inverse* (*h x*)) ∈ *H*
begin

lemma *div*: *h1* ∈ *H* ⟹ *h2* ∈ *H* ⟹ (λx. *h1 x* / *h2 x*) ∈ *H*
  ⟨*proof*⟩

lemma *nonzero*: *h* ∈ *H* ⟹ *eventually* (λx. *h x* ≠ *0*) *F*
  ⟨*proof*⟩

lemma *landau-cases*:
  assumes *h1* ∈ *H* *h2* ∈ *H*
  obtains *h1* ∈ *o*[*F*](*h2*) | *h2* ∈ *o*[*F*](*h1*) | *h1* ∈ Θ[*F*](*h2*)
  ⟨*proof*⟩

lemma *small-big-antisym*:
  assumes *h1* ∈ *H* *h2* ∈ *H* *h1* ∈ *o*[*F*](*h2*) *h2* ∈ *O*[*F*](*h1*) **shows** *False*
⟨*proof*⟩

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*: $g1 \in G \Longrightarrow g2 \in G \Longrightarrow h1 \in H \Longrightarrow h2 \in H \Longrightarrow g1 \in o[F](g2) \Longrightarrow$
$(\lambda x.\ g1\ x * h1\ x) \in o[F](\lambda x.\ g2\ x * h2\ x)$
assumes *g*: $g \in G$
assumes *g-dominates*: $h \in H \Longrightarrow h \in o[F](g)$
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* ⟷ *?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* ⟷ *?Q*)
⟨*proof*⟩

lemma *bigtheta-iff*:
$g1 \in G \Longrightarrow g2 \in G \Longrightarrow h1 \in H \Longrightarrow h2 \in H \Longrightarrow$
$(\lambda x.\ g1\ x * h1\ x) \in \Theta[F](\lambda x.\ g2\ x * h2\ x) \longleftrightarrow g1 \in \Theta[F](g2) \wedge h1 \in \Theta[F](h2)$
⟨*proof*⟩

end

lemma *landau-function-family-powr-closure*:
assumes $F \neq$ *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* $(\lambda x.\ g\ x\ powr\ 1)$
  assumes *filterlim f at-top F*
  assumes $\bigwedge p.\ (\lambda x.\ g\ x\ powr\ p) \in o[F](f)$
  shows *landau-function-family-pair F (powr-closure f) (powr-closure g* $*$ *H)* $(\lambda x.$ $f\ x\ powr\ 1)$
⟨*proof*⟩

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

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*
⟨*proof*⟩

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* $[] \longleftrightarrow$ *True*

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

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

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

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

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*) $\Longrightarrow$ *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 (λx. ($\prod g \leftarrow gs$. *get-fun g x powr 0*) = *1*) *F*
⟨*proof*⟩

lemma *listmap-gs-in-listmap*:
  (λx. $\prod g \leftarrow fs$. *h g x powr p g*) $\in$ *prod-list* (*map powr-closure* (*map h fs*))
⟨*proof*⟩

lemma *smallo-iff*:
  (λ-. *1*) $\in$ $o[F]$(λx. $\prod g \leftarrow gs$. *get-fun g x powr get-param g*) $\longleftrightarrow$ *pos-list* (*map get-param gs*)
⟨*proof*⟩

lemma *bigo-iff*:
  (λ-. *1*) $\in$ $O[F]$(λx. $\prod g \leftarrow gs$. *get-fun g x powr get-param g*) $\longleftrightarrow$ *nonneg-list* (*map get-param gs*)
⟨*proof*⟩

lemma *bigtheta-iff*:
  (λ-. *1*) $\in$ $\Theta[F]$(λx. $\prod g \leftarrow gs$. *get-fun g x powr 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 ^^ n) *at-top at-top*
  ⟨*proof*⟩

lemma *const-smallo-ln-chain*: (λ-. *1*) $\in$ o((*ln*::*real*⇒*real*) ^^ n)
⟨*proof*⟩

lemma *ln-fun-in-smallo-fun*:

assumes *filterlim f at-top at-top*
shows $(\lambda x.\ ln\ (f\ x)\ powr\ p :: real) \in o(f)$
⟨*proof*⟩

lemma *ln-chain-dominates*: $m > n \Longrightarrow$ *dominates at-top* $((ln::real \Rightarrow real)$^^$n)$ $(ln$^^$m)$
⟨*proof*⟩

datatype *primfun* = *LnChain nat*

instantiation *primfun* :: *linorder*
begin

fun *less-eq-primfun* :: *primfun* ⇒ *primfun* ⇒ *bool* **where**
LnChain x ≤ *LnChain y* ⟷ *x* ≤ *y*

fun *less-primfun* :: *primfun* ⇒ *primfun* ⇒ *bool* **where**
LnChain x < *LnChain y* ⟷ *x* < *y*

instance
⟨*proof*⟩

end

fun *eval-primfun′* :: - ⇒ - ⇒ *real* **where**
eval-primfun′ (*LnChain n*) = (λ*x*. (*ln*^^*n*) *x*)

fun *eval-primfun* :: - ⇒ - ⇒ *real* **where**
eval-primfun (*f*, *e*) = (λ*x*. *eval-primfun′ f x powr e*)

lemma *eval-primfun-altdef*: *eval-primfun f x* = *eval-primfun′* (*fst f*) *x powr snd f*
⟨*proof*⟩

fun *merge-primfun* **where**
merge-primfun (*x*::*primfun*, *a*) (*y*, *b*) = (*x*, *a* + *b*)

fun *inverse-primfun* **where**
inverse-primfun (*x*::*primfun*, *a*) = (*x*, −*a*)

fun *powr-primfun* **where**
powr-primfun (*x*::*primfun*, *a*) *e* = (*x*, *e*∗*a*)

lemma *primfun-cases*:

assumes $(\bigwedge n\ e.\ P\ (LnChain\ n,\ e))$
shows $P\ x$
⟨*proof*⟩

lemma *eval-primfun′-at-top*: $filterlim\ (eval\text{-}primfun'\ f)\ at\text{-}top\ at\text{-}top$
⟨*proof*⟩

lemma *primfun-dominates*:
$f < g \Longrightarrow dominates\ at\text{-}top\ (eval\text{-}primfun'\ f)\ (eval\text{-}primfun'\ g)$
⟨*proof*⟩

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

lemma *eventually-nonneg-primfun*: $eventually\text{-}nonneg\ at\text{-}top\ (eval\text{-}primfun\ f)$
⟨*proof*⟩

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

lemma *eval-merge-primfun*:
$fst\ f = fst\ g \Longrightarrow$
$eval\text{-}primfun\ (merge\text{-}primfun\ f\ g)\ x = eval\text{-}primfun\ f\ x * eval\text{-}primfun\ g\ x$
⟨*proof*⟩

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

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

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

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

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

2.5 Reification

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

definition *LANDAU-PROD* **where**
LANDAU-PROD L c1 c2 fs ⟷ (λ-. *c1*) ∈ *L*(λ*x*. *c2* ∗ *eval-primfuns fs x*)

definition *BIGTHETA-CONST′* **where** *BIGTHETA-CONST′ c* = Θ(λ*x*. *c*)
definition *BIGTHETA-CONST* **where** *BIGTHETA-CONST c A* = *set-mult* Θ(λ-. *c*) *A*
definition *BIGTHETA-FUN* **where** *BIGTHETA-FUN f* = Θ(*f*)

lemma *BIGTHETA-CONST′-tag*: Θ(λ*x*. *c*) = *BIGTHETA-CONST′ c* ⟨*proof*⟩
lemma *BIGTHETA-CONST-tag*: Θ(*f*) = *BIGTHETA-CONST 1* Θ(*f*)
⟨*proof*⟩
lemma *BIGTHETA-FUN-tag*: Θ(*f*) = *BIGTHETA-FUN f*
⟨*proof*⟩

lemma *set-mult-is-times*: *set-mult A B* = *A* ∗ *B*
⟨*proof*⟩

lemma *set-powr-mult*:
assumes *eventually-nonneg F f* **and** *eventually-nonneg F g*
shows Θ[*F*](λ*x*. (*f x* ∗ *g x* :: *real*) *powr p*) = *set-mult* (Θ[*F*](λ*x*. *f x powr p*)) (Θ[*F*](λ*x*. *g x powr p*))
⟨*proof*⟩

lemma *eventually-nonneg-bigtheta-pow-realpow*:
Θ(λ*x*. *eval-primfun f x* ^ *e*) = Θ(λ*x*. *eval-primfun f x powr real e*)
⟨*proof*⟩

lemma *BIGTHETA-CONST-fold*:
BIGTHETA-CONST (*c*::*real*) (*BIGTHETA-CONST d A*) = *BIGTHETA-CONST* (*c*∗*d*) *A*
bigtheta-pow at-top (*BIGTHETA-CONST c* Θ(*eval-primfun pf*)) *k* =
BIGTHETA-CONST (*c* ^ *k*) Θ(λ*x*. *eval-primfun pf x powr k*)
set-inverse (*BIGTHETA-CONST c* Θ(*f*)) = *BIGTHETA-CONST* (*inverse c*) Θ(λ*x*. *inverse* (*f x*))
set-mult (*BIGTHETA-CONST c* Θ(*f*)) (*BIGTHETA-CONST d* Θ(*g*)) =
BIGTHETA-CONST (*c*∗*d*) Θ(λ*x*. *f x*∗*g x*)
BIGTHETA-CONST′ (*c*::*real*) = *BIGTHETA-CONST c* Θ(λ-. *1*)
BIGTHETA-FUN (*f*::*real*⇒*real*) = *BIGTHETA-CONST 1* Θ(*f*)
⟨*proof*⟩

lemma *fold-fun-chain*:
g x = (*g* ^^ *1*) *x* (*g* ^^ *m*) ((*g* ^^ *n*) *x*) = (*g* ^^ (*m*+*n*)) *x*
⟨*proof*⟩

lemma *reify-ln-chain-1*:
Θ(λ*x*. (*ln* ^^ *n*) *x*) = Θ(*eval-primfun* (*LnChain n*, *1*))
⟨*proof*⟩

lemma *reify-monom-1*:
$\Theta(\lambda x::real.\ x) = \Theta(eval\text{-}primfun\ (LnChain\ 0,\ 1))$
⟨*proof*⟩

lemma *reify-monom-pow*:
$\Theta(\lambda x::real.\ x\ \hat{}\ e) = \Theta(eval\text{-}primfun\ (LnChain\ 0,\ real\ e))$
⟨*proof*⟩

lemma *reify-monom-powr*:
$\Theta(\lambda x::real.\ x\ powr\ e) = \Theta(eval\text{-}primfun\ (LnChain\ 0,\ e))$
⟨*proof*⟩

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

lemma *reify-ln-chain-pow*:
$\Theta(\lambda x.\ (ln\ \hat{}\hat{}\ n)\ x\ \hat{}\ e) = \Theta(eval\text{-}primfun\ (LnChain\ n,\ real\ e))$
⟨*proof*⟩

lemma *reify-ln-chain-powr*:
$\Theta(\lambda x.\ (ln\ \hat{}\hat{}\ n)\ x\ powr\ e) = \Theta(eval\text{-}primfun\ (LnChain\ n,\ e))$
⟨*proof*⟩

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

lemma *numeral-power-Suc*: *numeral n* ^ *Suc a* = *numeral n* ∗ *numeral n* ^ *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 Θ(λ-. *d*) = *BIGTHETA-CONST* (*e*∗*d*) Θ(*eval-primfuns* [])
LANDAU-PROD′ c (λ-. *1*) = *LANDAU-PROD′ c* (*eval-primfuns* [])
eval-primfun f = *eval-primfuns* [*f*]
eval-primfuns fs x ∗ *eval-primfuns gs x* = *eval-primfuns* (*fs* @ *gs*) *x*
⟨*proof*⟩

lemma *inverse-prod-list-field*:
prod-list (*map* (λ*x*. *inverse* (*f x*)) *xs*) = *inverse* (*prod-list* (*map f xs* :: - :: *field list*))
⟨*proof*⟩

lemma *landau-prod-meta-cong*:
  assumes *landau-symbol L L′ Lr*
  assumes $\Theta(f) \equiv$ *BIGTHETA-CONST c1* (Θ(*eval-primfuns fs*))
  assumes $\Theta(g) \equiv$ *BIGTHETA-CONST c2* (Θ(*eval-primfuns gs*))
  shows $f \in L$ *at-top* $(g) \equiv$ *LANDAU-PROD* (*L at-top*) *c1 c2* (*map inverse-primfun fs @ gs*)
⟨*proof*⟩

fun *pos-primfun-list* **where**
  pos-primfun-list [] ⟷ *False*
| *pos-primfun-list* ((-,*x*)#*xs*) ⟷ *x* > *0* ∨ (*x* = *0* ∧ *pos-primfun-list xs*)

fun *nonneg-primfun-list* **where**
  nonneg-primfun-list [] ⟷ *True*
| *nonneg-primfun-list* ((-,*x*)#*xs*) ⟷ *x* > *0* ∨ (*x* = *0* ∧ *nonneg-primfun-list xs*)

fun *iszero-primfun-list* **where**
  iszero-primfun-list [] ⟷ *True*
| *iszero-primfun-list* ((-,*x*)#*xs*) ⟷ *x* = *0* ∧ *iszero-primfun-list xs*

definition *group-primfuns* ≡ *groupsort.group-sort fst merge-primfun*

lemma *list-ConsCons-induct*:
  assumes *P* [] ⋀*x*. *P* [*x*] ⋀*x y xs*. *P* (*y*#*xs*) ⟹ *P* (*x*#*y*#*xs*)
  shows *P xs*
⟨*proof*⟩

lemma *landau-function-family-chain-primfuns*:
  assumes *sorted* (*map fst fs*)
  assumes *distinct* (*map fst fs*)
  shows *landau-function-family-chain at-top fs* (*eval-primfun′ o fst*)
⟨*proof*⟩

lemma (**in** *monoid-mult*) *fold-plus-prod-list-rev*:
  fold times xs = *times* (*prod-list* (*rev xs*))
⟨*proof*⟩

interpretation *groupsort-primfun*: *groupsort fst merge-primfun eval-primfuns*
⟨*proof*⟩

lemma *nonneg-primfun-list-iff*: *nonneg-primfun-list fs* = *nonneg-list* (*map snd fs*)
  ⟨*proof*⟩

lemma *pos-primfun-list-iff*: *pos-primfun-list fs* = *pos-list* (*map snd fs*)
  ⟨*proof*⟩

lemma *iszero-primfun-list-iff*: *iszero-primfun-list fs* = *list-all* ((=) *0*) (*map snd*

fs)
⟨*proof*⟩

lemma *landau-primfuns-iff*:
((λ-. *1*) ∈ *O*(*eval-primfuns fs*)) = *nonneg-primfun-list* (*group-primfuns fs*) (**is** *?A*)
((λ-. *1*) ∈ *o*(*eval-primfuns fs*)) = *pos-primfun-list* (*group-primfuns fs*) (**is** *?B*)
((λ-. *1*) ∈ Θ(*eval-primfuns fs*)) = *iszero-primfun-list* (*group-primfuns fs*) (**is** *?C*)
⟨*proof*⟩

lemma *LANDAU-PROD-bigo-iff*:
LANDAU-PROD (*bigo at-top*) *c1 c2 fs* ⟷ *c1* = *0* ∨ (*c2* ≠ *0* ∧ *nonneg-primfun-list* (*group-primfuns fs*))
⟨*proof*⟩

lemma *LANDAU-PROD-smallo-iff*:
LANDAU-PROD (*smallo at-top*) *c1 c2 fs* ⟷ *c1* = *0* ∨ (*c2* ≠ *0* ∧ *pos-primfun-list* (*group-primfuns fs*))
⟨*proof*⟩

lemma *LANDAU-PROD-bigtheta-iff*:
LANDAU-PROD (*bigtheta at-top*) *c1 c2 fs* ⟷ (*c1* = *0* ∧ *c2* = *0*) ∨ (*c1* ≠ *0* ∧ *c2* ≠ *0* ∧
iszero-primfun-list (*group-primfuns fs*))
⟨*proof*⟩

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

lemmas *landau-real-prod-simps* [*simp*] =
groupsort-primfun.group-part-def
group-primfuns-def groupsort-primfun.group-sort.simps
groupsort-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 *bigtheta-const* [*landau-simp*]:
$NO\text{-}MATCH\ 1\ c \Longrightarrow c \neq 0 \Longrightarrow (\lambda x.\ c) \in \Theta(\lambda x.\ 1)$ ⟨*proof*⟩

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

lemma *bigtheta-const-ln'* [*landau-simp*]:
$0 < a \Longrightarrow (\lambda x{::}real.\ ln\ (x * a)) \in \Theta(ln)$
⟨*proof*⟩

lemma *bigtheta-const-ln-powr'* [*landau-simp*]:
$0 < a \Longrightarrow (\lambda x{::}real.\ ln\ (x * a)\ powr\ p) \in \Theta(\lambda x.\ ln\ x\ powr\ p)$
⟨*proof*⟩

lemma *bigtheta-const-ln-pow'* [*landau-simp*]:
$0 < a \Longrightarrow (\lambda x{::}real.\ ln\ (x * a)\ \hat{}\ p) \in \Theta(\lambda x.\ ln\ x\ \hat{}\ p)$
⟨*proof*⟩

3.2 Simproc setup

lemma *landau-gt-1-cong*:
$landau\text{-}symbol\ L\ L'\ Lr \Longrightarrow (\bigwedge x{::}real.\ x > 1 \Longrightarrow f\ x = g\ x) \Longrightarrow L\ at\text{-}top\ (f) = L\ at\text{-}top\ (g)$
⟨*proof*⟩

lemma *landau-gt-1-in-cong*:
$landau\text{-}symbol\ L\ L'\ Lr \Longrightarrow (\bigwedge x{::}real.\ x > 1 \Longrightarrow f\ x = g\ x) \Longrightarrow f \in L\ at\text{-}top\ (h) \longleftrightarrow g \in L\ at\text{-}top\ (h)$
⟨*proof*⟩

lemma *landau-prop-equalsI*:
$landau\text{-}symbol\ L\ L'\ Lr \Longrightarrow (\bigwedge x{::}real.\ x > 1 \Longrightarrow f1\ x = f2\ x) \Longrightarrow (\bigwedge x.\ x > 1 \Longrightarrow g1\ x = g2\ x) \Longrightarrow$
$f1 \in L\ at\text{-}top\ (g1) \longleftrightarrow f2 \in L\ at\text{-}top\ (g2)$
⟨*proof*⟩

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

lemma *divide-inverse'*: (*a*::-::{*division-ring*,*ab-semigroup-mult*}) / *b* = *inverse b* ∗ *a*
⟨*proof*⟩
lemma *extract-divide-middle*:(*a*::-::{*field*}) / (*x* ∗ *b*) = *inverse x* ∗ (*a* / *b*)
⟨*proof*⟩

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

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

lemma *mult-cancel-left-1*:
assumes *landau-symbol L L' Lr eventually-nonzero F f*
shows *f* ∈ *L F* (λ*x*. *f x* ∗ *g2 x*) ⟷ (λ-. *1*) ∈ *L F* (*g2*)
(λ*x*. *f x* ∗ *f2 x*) ∈ *L F* (*f*) ⟷ *f2* ∈ *L F* (λ-. *1*)
f ∈ *L F* (*f*) ⟷ (λ-. *1*) ∈ *L F* (λ-. *1*)
⟨*proof*⟩

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

⟨*ML*⟩

lemmas *bigtheta-simps* =
landau-theta.cong-bigtheta[*OF bigtheta-const-ln*]
landau-theta.cong-bigtheta[*OF bigtheta-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 $x \to \infty$.

⟨*ML*⟩

3.3 Tests

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

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

3.3.1 Product simplification tests

lemma $(\lambda x{::}real.\ f\ x * x) \in O(\lambda x.\ g\ x\ /\ (h\ x\ /\ x)) \longleftrightarrow f \in O(\lambda x.\ g\ x\ /\ h\ x)$
⟨*proof*⟩

lemma $(\lambda x{::}real.\ x) \in \omega(\lambda x.\ g\ x\ /\ (h\ x\ /\ x)) \longleftrightarrow (\lambda x.\ 1) \in \omega(\lambda x.\ g\ x\ /\ h\ x)$
⟨*proof*⟩

3.3.2 Real product decision procure tests

lemma $(\lambda x.\ x\ powr\ 1) \in O(\lambda x.\ x\ powr\ 2 :: real)$
⟨*proof*⟩

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

lemma $p < q \Longrightarrow (\lambda x{::}real.\ c * x\ powr\ p * ln\ x\ powr\ r) \in o(\lambda x{::}real.\ x\ powr\ q)$
⟨*proof*⟩

lemma $c \neq 0 \Longrightarrow p > q \Longrightarrow (\lambda x{::}real.\ c * x\ powr\ p * ln\ x\ powr\ r) \in \omega(\lambda x{::}real.\ x\ powr\ q)$
⟨*proof*⟩

lemma $b > 0 \Longrightarrow (\lambda x{::}real.\ x\ /\ ln\ (2{*}b{*}x) * 2) \in o(\lambda x.\ x * ln\ (b{*}x))$
⟨*proof*⟩
lemma $o(\lambda x{::}real.\ x * ln\ (3{*}x)) = o(\lambda x.\ ln\ x * x)$
⟨*proof*⟩
lemma $(\lambda x{::}real.\ x) \in o(\lambda x.\ x * ln\ (3{*}x))$ ⟨*proof*⟩

⟨*ML*⟩

lemma $(\lambda x.\ 3 * ln\ x * ln\ x\ /\ x * ln\ (ln\ (ln\ (ln\ x)))) \in$
$\omega(\lambda x{::}real.\ 5 * ln\ (ln\ x)\ \hat{}\ 2\ /\ (2{*}x)\ powr\ 1.5 * inverse\ 2)$
⟨*proof*⟩

3.3.3 Sum cancelling tests

lemma $\Theta(\lambda x{::}real.\ 2 * x\ powr\ 3 + x * x\hat{}2/ln\ x) = \Theta(\lambda x{::}real.\ x\ powr\ 3)$
⟨*proof*⟩

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

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

end

theory *Landau-More*
imports
HOL−Library.Landau-Symbols
Landau-Simprocs
begin

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

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

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

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

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

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

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

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

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

lemma *bigtheta-powr-nonneg* [*landau-simp*]:
eventually $(\lambda x.\ (f\ x :: real) \geq 0)$ $F \Longrightarrow (\lambda x.\ if\ f\ x \geq 0\ then\ g\ x\ else\ h\ x) \in \Theta[F](g)$
⟨*proof*⟩

lemma *bigtheta-powr-nonneg′* [*landau-simp*]:
eventually $(\lambda x.\ (f\ x :: real) \geq 0)$ $F \Longrightarrow (\lambda x.\ if\ f\ x < 0\ then\ g\ x\ else\ h\ x) \in \Theta[F](h)$
⟨*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 $(\lambda x.\ (f\ x :: real)\ powr\ p) \in O[F](\lambda x.\ g\ x\ powr\ p) \longleftrightarrow f \in O[F](g)$ (**is** *?lhs* $\longleftrightarrow$ *?rhs*)
⟨*proof*⟩

lemma *inverse-powr* [*simp*]:
assumes $(x::real) \geq 0$
shows *inverse x powr y = inverse (x 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 $(\lambda x.\ (f\ x :: real)\ powr\ p) \in O[F](\lambda x.\ g\ x\ powr\ p) \longleftrightarrow g \in O[F](f)$ (**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 $(\lambda x.\ (f\ x :: real)\ powr\ p) \in o[F](\lambda x.\ g\ x\ powr\ p) \longleftrightarrow f \in o[F](g)$ (**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 $(\lambda x.\ (f\ x :: real)\ powr\ p) \in o[F](\lambda x.\ g\ x\ powr\ p) \longleftrightarrow g \in o[F](f)$ (**is** *?lhs* $\longleftrightarrow$ *?rhs*)
⟨*proof*⟩

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

lemma *bigo-const-powr*:
  assumes *filterlim f at-top F F* $\neq$ *bot*
  shows $(\lambda\text{-}.\ c :: real) \in O[F](\lambda x.\ f\ x\ powr\ p) \longleftrightarrow p \geq 0 \lor c = 0$
⟨*proof*⟩

lemma *filterlim-powr-at-top*:
  $(b::real) > 1 \Longrightarrow$ *filterlim* $(\lambda x.\ b\ powr\ x)$ *at-top at-top*
  ⟨*proof*⟩

lemma *power-smallo-exponential*:
  fixes *b* :: *real*
  assumes *b*: $b > 1$
  shows $(\lambda x.\ x\ powr\ n) \in o(\lambda x.\ b\ powr\ x)$
⟨*proof*⟩

lemma *powr-fast-growth-tendsto*:
  assumes *gf*: $g \in O[F](f)$
  and *n*: $n \geq 0$
  and *k*: $k > 1$
  and *f*: *filterlim f at-top F*
  and *g*: *eventually* $(\lambda x.\ g\ x \geq 0)$ *F*
  shows $(\lambda x.\ g\ x\ powr\ n) \in o[F](\lambda x.\ k\ powr\ f\ x :: real)$
⟨*proof*⟩

lemma *bigo-abs-powr-iff* [*simp*]:
  $0 < p \Longrightarrow (\lambda x.\ |f\ x :: real|\ powr\ p) \in O[F](\lambda x.\ |g\ x|\ powr\ p) \longleftrightarrow f \in O[F](g)$
  ⟨*proof*⟩

lemma *smallo-abs-powr-iff* [*simp*]:
  $0 < p \Longrightarrow (\lambda x.\ |f\ x :: real|\ powr\ p) \in o[F](\lambda x.\ |g\ x|\ powr\ p) \longleftrightarrow f \in o[F](g)$
  ⟨*proof*⟩

lemma *const-smallo-inverse-powr*:
  assumes *filterlim f at-top at-top*
  shows $(\lambda\text{-} :: \text{-} :: linorder.\ c :: real) \in o(\lambda x.\ inverse\ (f\ x\ powr\ p)) \longleftrightarrow (p \geq 0 \longrightarrow c = 0)$
⟨*proof*⟩

lemma *bigo-const-inverse-powr*:
  assumes *filterlim f at-top at-top*
  shows $(\lambda\text{-} :: \text{-} :: linorder.\ c :: real) \in O(\lambda x.\ inverse\ (f\ x\ powr\ p)) \longleftrightarrow c = 0 \lor p \leq 0$
⟨*proof*⟩

end