Combinatorics on Words formalized
Lyndon Words

Štěpán Holub
Štěpán Starosta

March 17, 2025

Funded by the Czech Science Foundation grant GAČR 20-20621S.

Contents

theory *Lyndon*
imports *Combinatorics-Words.CoWBasic*
begin

Chapter 1

Lyndon words

A Lyndon word is a non-empty word that is lexicographically strictly smaller than any other word in its conjugacy class, i.e., than any its rotations. They are named after R. Lyndon who introduced them in [4] as "standard" sequences.

We present elementary results on Lyndon words, mostly covered by results in [3, Chapter 5] and [1, 2].

This definition assumes a linear order on letters given by the context.

1.1 Definition and elementary properties

1.1.1 Underlying order

lemma (**in** *linorder*) *lexordp-mid-pref*: *ord-class.lexordp u v* $\Longrightarrow$ *ord-class.lexordp v* $(u{\cdot}s)$ $\Longrightarrow$
  $u \le p\ v$
  by (*induct rule*: *lexordp-induct*, *simp-all*)

lemma (**in** *linorder*) *lexordp-ext*: *ord-class.lexordp u v* $\Longrightarrow$ $\neg\ u \le p\ v$ $\Longrightarrow$
  ord-class.lexordp $(u{\cdot}w)$ $(v{\cdot}z)$
  by (*induct rule*: *lexordp-induct*, *simp-all*)

context *linorder*
begin

abbreviation *Lyndon-less* :: *'a list* $\Rightarrow$ *'a list* $\Rightarrow$ *bool* (**infixl** ‹<*lex*› *50*)
  where *Lyndon-less xs ys* $\equiv$ *ord-class.lexordp xs ys*

abbreviation *Lyndon-le* :: *'a list* $\Rightarrow$ *'a list* $\Rightarrow$ *bool* (**infixl** ‹≤*lex*› *50*)
  where *Lyndon-le xs ys* $\equiv$ *ord-class.lexordp-eq xs ys*

interpretation *rlex*: *linorder* (≤*lex*) (<*lex*)
  using *lexordp-linorder*.

interpretation *dual-rlex*: *linorder* λ *x y. y* $\leq$*lex x* λ *x y. y* <*lex x*
using *rlex.dual-linorder*.

lemma *sorted-dual-rev-iff*: *dual-rlex.sorted ws* $\longleftrightarrow$ *rlex.sorted* (*rev ws*)
unfolding *rlex.sorted-rev-iff-nth-mono dual-rlex.sorted-iff-nth-mono* **by** *blast*

Several useful lemmas that are formulated for relations, interpreted for the default linear order.

lemmas *lexord-suf-linorder* = *lexord-sufE*[*of* - - - - {(*x*, *y*). *x* < *y*}, *folded lexordp-conv-lexord*]
and *lexord-append-leftI-linorder* = *lexord-append-leftI*[*of* - - {(*x*, *y*). *x* < *y*} -, *folded lexordp-conv-lexord*]
and *lexord-app-right-linorder* = *lexord-sufI*[*of* - - {(*x*, *y*). *x* < *y*} -, *folded lexordp-conv-lexord*]
and *lexord-take-index-conv-linorder* = *lexord-take-index-conv*[*of* - - {(*x*, *y*). *x* < *y*}, *folded lexordp-conv-lexord*]
and *mismatch-lexord-linorder* = *mismatch-lexord*[*of* - - {(*x*, *y*). *x* < *y*}, *folded lexordp-conv-lexord*]
and *lexord-cancel-right-linorder* = *lexord-cancel-right*[*of* - - - - {(*a*,*b*). *a* < *b*}, *folded lexordp-conv-lexord*]

1.1.2 Lyndon word definition

fun *Lyndon* :: *'a list* $\Rightarrow$ *bool*
where *Lyndon w* = (*w* $\neq \varepsilon$ $\wedge$ ($\forall$ *n*. *0* < *n* $\wedge$ *n* < |*w*| $\longrightarrow$ *w* <*lex rotate n w*))

lemma *LyndonD*: *Lyndon w* $\Longrightarrow$ *0* < *n* $\Longrightarrow$ *n* < |*w*| $\Longrightarrow$ *w* <*lex rotate n w*
unfolding *Lyndon.simps* **by** *blast*

lemma *LyndonD-nemp*: *Lyndon w* $\Longrightarrow$ *w* $\neq \varepsilon$
unfolding *Lyndon.simps* **by** *blast*

lemma *LyndonI*: *w* $\neq \varepsilon$ $\Longrightarrow$ $\forall$ *n*. *0* < *n* $\wedge$ *n* < |*w*| $\longrightarrow$ *w* <*lex rotate n w* $\Longrightarrow$ *Lyndon w*
unfolding *Lyndon.simps* **by** *blast*

lemma *Lyndon-sing*: *Lyndon* [*a*]
unfolding *Lyndon.simps* **by** *auto*

lemma *Lyndon-prim*: **assumes** *Lyndon w*
shows *primitive w*
proof–
have *0* < *n* $\Longrightarrow$ *n* < |*w*| $\Longrightarrow$ *rotate n w* $\neq$ *w* **for** *n*
using *LyndonD*[*OF* ‹*Lyndon w*›, *of n*] *rlex.less-irrefl*[*of w*] **by** *argo*
from *no-rotate-prim*[*OF LyndonD-nemp*[*OF* ‹*Lyndon w*›]] *this*
show *?thesis* **by** *blast*
qed

lemma *Lyndon-conj-greater*: *Lyndon* $(u \cdot v) \Longrightarrow u \neq \varepsilon \Longrightarrow v \neq \varepsilon \Longrightarrow u \cdot v$ *<lex* $v \cdot u$
  using *LyndonD*[*of* $u \cdot v$ $|u|$, *unfolded rotate-append*[*of u v*]]
  by *force*

1.1.3 Code equations for Lyndon words

primrec *Lyndon-rec* :: *'a list* ⇒ *nat* ⇒ *bool* **where**
  Lyndon-rec w 0 = *True* |
  Lyndon-rec w (*Suc n*) = (*if w <lex rotate* (*Suc n*) *w then Lyndon-rec w n else False*)

lemma *Lyndon-rec-all*: **assumes** *Lyndon-rec* (*a* # *w*) ($|w|$)
  shows $n < |a\#w| \Longrightarrow 0 < n \Longrightarrow$ *Lyndon-rec* (*a*#*w*) *n*
proof(*induction n rule*: *strict-inc-induct*)
  case (*base i*)
  then show *?case*
    using *assms* **by** *auto*
next
  case (*step i*)
  then show *?case*
    by (*meson Lyndon-rec.simps*(*2*) *zero-less-Suc*)
qed

lemma *Lyndon-Lyndon-rec*: **assumes** *Lyndon w*
  shows $0 < n \Longrightarrow n < |w| \Longrightarrow$ *Lyndon-rec w n*
proof(*induction n*, *simp*)
  case (*Suc n*)
  have *w <lex rotate* (*Suc n*) *w*
    using *LyndonD Suc.prems*(*2*) *assms* **by** *blast*
  then show *?case*
    using *Suc.IH*[*OF* - *Suc-lessD*[*OF* ‹*Suc n* < $|w|$›], *folded neq0-conv*] *Lyndon-rec.simps*(*1*)[*of w*]
    unfolding *Lyndon-rec.simps*(*2*)
    by *metis*
qed

lemma *Lyndon-code* [*code*]:
  Lyndon Nil = *False*
  Lyndon (*a* # *w*) = *Lyndon-rec* (*a* # *w*) ($|w|$)
proof–
  show *Lyndon Nil* = *False* **by** *simp*
  have *a* # $w \neq \varepsilon$
    by *simp*
  have *ax*: $0 < n \Longrightarrow$ *Lyndon-rec* (*a*#*w*) *n* $\Longrightarrow$ (*a*#*w*) *<lex rotate n* (*a*#*w*) **for** *n*
    using *Lyndon-rec.simps*(*2*)[*of a*#*w*] *gr0-implies-Suc*[*of n*] **by** *metis*
  have *bx*: *Lyndon-rec* (*a* # *w*) ($|w|$) = ($\forall n.\ n < |a\#w| \wedge 0 < n \longrightarrow$ *Lyndon-rec* (*a*#*w*) *n*)
  proof(*cases* $w = \varepsilon$, *simp*)
    assume $w \neq \varepsilon$

from *this*[*folded length-greater-0-conv*]
show *?thesis*
using *Lyndon-rec-all*[*of a w*] *length-Cons*[*of a w*] *lessI*[*of* |*w*|]
by *fastforce*
qed
show *Lyndon* (*a* # *w*) = *Lyndon-rec* (*a* # *w*) |*w*|
unfolding *bx Lyndon.simps*
using *ax LyndonI*[*OF* ‹*a* # *w* ≠ *ε*›]*Lyndon-Lyndon-rec* **by** *blast*
qed

1.1.4 Properties of Lyndon words

Lyndon words are unbordered

theorem *Lyndon-unbordered*: **assumes** *Lyndon w* **shows** ¬ *bordered w*
proof
assume *bordered w*
from *bordered-dec*[*OF this*]
obtain *u v* **where** *u*·*v*·*u* = *w* **and** *u* ≠ *ε*.
hence *v* · *u* ≠ *ε* **and** *u* · *v* ≠ *ε* **by** *blast+*
note *lyn* = ‹*Lyndon w*›[*folded* ‹*u*·*v*·*u* = *w*›]
have *u*·*v*·*u* <*lex v*·*u*·*u*
using *Lyndon-conj-greater*[*of u v*·*u*, *OF lyn* ‹*u* ≠ *ε*› ‹*v* · *u* ≠ *ε*›, *unfolded rassoc*].
from *this*[*unfolded lassoc*]
have *u* · *v* ≠ *v* · *u*
by *force*
from *lexord-suf-linorder*[*OF - this*, *of u u*]
have *u*·*v* <*lex v*·*u*
using ‹*u*·*v*·*u* <*lex v*·*u*·*u*› **by** *simp*
from *lexord-append-leftI-linorder*[*of u*·*v v*·*u*, *unfolded lassoc*, *OF this*, *unfolded rassoc*]
have *u*·*u*·*v* <*lex u*·*v*·*u*.
from *this Lyndon-conj-greater*[*of u*·*v u*, *unfolded rassoc*, *OF lyn* ‹*u* · *v* ≠ *ε*› ‹*u* ≠ *ε*›]
show *False*
by *simp*
qed

Each conjugacy class contains a Lyndon word

lemma *conjug-Lyndon-ex*: **assumes** *primitive w*
obtains *n* **where** *Lyndon* (*rotate n w*)
proof−
have *w* ≠ *ε*
using *prim-nemp*[*OF* ‹*primitive w*›].

let *?ConClass* = {*rotate n w* | *n*. *0* ≤ *n* ∧ *n* < |*w*|}

have *?ConClass* ≠ {}
using ‹*w* ≠ *ε*› **by** *blast*

have *finite ?ConClass*
by *force*
have $w \in$ *?ConClass*
by (*rule CollectI*)
(*use le0*[*of 0*] *nemp-pos-len*[*OF* ‹$w \neq \varepsilon$›] *id-apply*[*of w*, *folded rotate0*] **in** *metis*)
have *all-rot*: *rotate m w* $\in$ *?ConClass* **for** *m*
using *rotate-conv-mod*[*of - w*] *mod-less-divisor*[*of* $|w|$] ‹$w \neq \varepsilon$›
by *blast*

obtain w' n **where** $w' \in$ *?ConClass* **and** *all-b*: $\forall$ $b \in$ *?ConClass*. $b \leq$*lex* $w' \longrightarrow w' = b$ **and** w': w' = *rotate n w*
using *rlex.finite-has-minimal*[*OF* ‹*finite ?ConClass*› ‹*?ConClass* $\neq \{\}$›] **by** *auto*

have *rotate n w* <*lex rotate na* (*rotate n w*) **if** $0 < na$ **and** $na < |w|$ **for** *na*
proof−
from *prim-no-rotate*[*OF assms*[*unfolded prim-rotate-conv*[*of w n*]], *of na*] ‹$na < |w|$› ‹$0 < na$›
have *rotate na* (*rotate n w*) $\neq$ *rotate n w* **by** *force*
hence $\neg$ *rotate na* (*rotate n w*) $\leq$*lex rotate n w*
using *all-b*[*rule-format*, *OF all-rot*[*of na* + *n*, *folded rotate-rotate*[*of na n w*]]] **unfolding** w'
by *auto*
from *rlex.not-le-imp-less*[*OF this*]
show *rotate n w* <*lex rotate na* (*rotate n w*).
qed
hence *Lyndon* (*rotate n w*)
using ‹$w \neq \varepsilon$› **by** *auto*
from *that*[*OF this*] **show** *thesis*.
qed

lemma *conjug-Lyndon-ex'*: **assumes** *primitive w*
obtains *v* **where** $w \sim v$ **and** *Lyndon v*
unfolding *conjug-rotate-iff*
using *conjug-Lyndon-ex*[*OF* ‹*primitive w*›]
by *metis*

1.2 Characterization by suffixes

lemma *Lyndon-suf-less*: **assumes** *Lyndon w s* $\leq$*ns w s* $\neq$ *w*
shows *w* <*lex s*
proof−
define *p* **where** *p* = *take* $|s|$ *w*
have $|s| \leq |w|$
using *nsD*[*OF* ‹$s \leq$*ns w*›]
by (*simp add*: *suffix-length-le*)
have $p \leq$*p w* **and** $|p| = |s|$
unfolding *p-def*
using *take-is-prefix* ‹$|s| \leq |w|$› *take-len* **by** *blast+*
hence $p \neq s$

using *Lyndon-unbordered*[*OF* ‹*Lyndon w*›] ‹$s \leq ns\ w$› ‹$s \neq w$› *assms*
by *auto*
define p' s' **where** $p' = drop\ |s|\ w$ **and** $s' = take\ |p'|\ w$
have $p \cdot p' = w$
unfolding *p'-def p-def s'-def* **by** *simp*
have $s' \cdot s = w$
unfolding *p'-def p-def s'-def*
using *suf-len*[*OF nsD*[*OF* ‹$s \leq ns\ w$›]] *nsD*[*OF* ‹$s \leq ns\ w$›]
length-drop suffix-take **by** *metis*
have $|p'| = |s'|$
using *s'-def* ‹$p{\cdot}p' = w$› **by** *auto*
have $w <lex\ s{\cdot}s'$
using *Lyndon-conj-greater*[*of s' s*, *unfolded* ‹$s' \cdot s = w$›, *OF* ‹*Lyndon w*›] ‹$p \neq s$›
unfolding ‹$s' \cdot s = w$› *p-def* **using** ‹$s' \cdot s = w$› *assms*(*3*) **by** *fastforce*
from *lexord-suf-linorder*[*OF* - ‹$p \neq s$› ‹$|p| = |s|$› ‹$|p'| = |s'|$›, *OF this*[*folded* ‹$p \cdot p' = w$›]]
have $p <lex\ s$**.**
from *lexord-app-right-linorder*[*OF this*, *of p' ε*, *unfolded* ‹$p \cdot p' = w$›] ‹$|p| = |s|$›
show $w <lex\ s$
by *simp*
qed

lemma *Lyndon-pref-suf-less*: **assumes** *Lyndon w* $p \leq p\ w$ $s \leq ns\ w$ $s \neq w$
shows $p <lex\ s$
proof(*cases* $p = w$, *simp add*: *Lyndon-suf-less*[*OF assms*(*1*) *assms*(*3*) *assms*(*4*)])
assume $p \neq w$
show $p <lex\ s$
proof(*rule rlex.less-trans*)
show $p <lex\ w$
using ‹$p \neq w$› *assms*(*2*) *lexordp-append-rightI*
by (*fastforce simp add*: *prefix-def*)
show $w <lex\ s$
using *Lyndon-suf-less assms*(*1*) *assms*(*3*) *assms*(*4*) **by** *blast*
qed
qed

lemma *suf-less-Lyndon*: **assumes** $w \neq \varepsilon$ **and** $\forall\, s.\ (s \leq ns\ w \longrightarrow s \neq w \longrightarrow w <lex\ s)$
shows *Lyndon w*
proof (*cases primitive w*)
assume ¬ *primitive w*
obtain *q k* **where** $q \neq \varepsilon$ $1 < k$ $q^{@}k{=}w$ $w{\neq}q$ — the exact match of $[\![\neg\ primitive\ ?w;\ ?w \neq \varepsilon;\ \bigwedge r\ k.\ [\![r \neq \varepsilon;\ 1 < k;\ r\ ^{@}\ k = ?w;\ ?w \neq r]\!] \Longrightarrow ?thesis]\!] \Longrightarrow ?thesis$ fastens the proof considerably
using *non-prim*[*OF* ‹¬ *primitive w*› ‹$w \neq \varepsilon$›] **by** *blast*
hence $q \leq ns\ w$
unfolding *nonempty-suffix-def pow-eq-if-list*[*of q k*] *pow-comm*[*symmetric*]
using *sufI*[*of* $q\ ^{@}\ (k - 1)\ q\ w$]

by *presburger*

have $q <p\ w$
using ‹$1 < k$› ‹q $^@$ $k = w$›
unfolding *pow-eq-if-list*[*of* q k] *pow-eq-if-list*[*of* q $k-1$]
using ‹$w \neq \varepsilon$› **by** *auto*
from *lexordp-append-rightI*[*of* $q^{-1>}w$ q,
unfolded lq-pref[*OF sprefD1*[*OF this*]], *OF lq-spref*[*OF this*]]
have $q <lex\ w$.
thus *Lyndon w*
unfolding *Lyndon.simps* **using** ‹$q \leq ns\ w$› ‹$w \neq \varepsilon$› *assms*(*2*) *rlex.order.strict-trans*
by *blast*
next
assume *primitive w*
have $w <lex$ *rotate l w* **if** *assms-l*: $0 < l$ $l < |w|$ **for** l
proof–
have *take l w* $\leq np$ w **and** $|$*take l w*$| = l$
using *assms-l take-is-prefix* ‹$l < |w|$› **by** *auto*
have *drop l w* $\leq ns$ w
using ‹$l < |w|$› *suffix-drop* **by** *auto*
have *drop l w* $\neq w$
using *append-take-drop-id*[*of l w*] *npD′*[*OF* ‹*take l w* $\leq np$ w›] **by** *force*
have *drop l w* $\cdot$ *take l w* $=$ *rotate l w*
using *rotate-append*[*of take l w drop l w*, *symmetric*, *unfolded* ‹$|$*take l w*$| = l$›,
unfolded append-take-drop-id].

have $w <lex$ *drop l w*
using ‹*drop l w* $\leq ns$ w› ‹*drop l w* $\neq w$› *assms*(*2*) **by** *blast*
from *lexord-app-right-linorder*[*OF this suffix-length-le*[*OF conjunct2*[*OF* ‹*drop l w* $\leq ns$ w›[*unfolded nonempty-suffix-def*]]], *of* ε *take l w*, *unfolded append.right-neutral*]
have $w <lex$ *drop l w* $\cdot$ *take l w*.
thus $w <lex$ *rotate l w*
by (*simp add*: ‹*drop l w* $\cdot$ *take l w* $=$ *rotate l w*›)
qed
thus *Lyndon w*
by (*simp add*: ‹$w \neq \varepsilon$› *local.LyndonI*)
qed

corollary *Lyndon-suf-char*: $w \neq \varepsilon \Longrightarrow$ *Lyndon w* $\longleftrightarrow$ ($\forall s.\ s \leq ns\ w \longrightarrow s \neq w \longrightarrow w <lex\ s$)
using *Lyndon-suf-less suf-less-Lyndon* **by** *blast*

lemma *Lyndon-suf-le*: *Lyndon w* $\Longrightarrow s \leq ns\ w \Longrightarrow w \leq lex\ s$
using *Lyndon-suf-less rlex.not-less rlex.order.asym* **by** *blast*

1.3 Unbordered prefix of a Lyndon word is Lyndon

lemma *unbordered-pref-Lyndon*: *Lyndon* $(u \cdot v) \Longrightarrow u \neq \varepsilon \Longrightarrow \neg$ *bordered* $u \Longrightarrow$ *Lyndon u*

unfolding *Lyndon-suf-char*
proof(*standard*+)
  fix *s*
  assume *Lyndon* $(u \cdot v)$ **and** $u \neq \varepsilon$ **and** $\neg$ *bordered* u **and** $s \leq ns\ u$ **and** $s \neq u$
    hence $u \cdot v <lex\ s \cdot v$
      using *Lyndon-suf-less*[*OF* ‹*Lyndon* $(u \cdot v)$›, *of* $s \cdot v$] **by** *auto*
    have $\neg\ s \leq p\ u$
      using ‹$\neg$ *bordered* u› ‹$s \leq ns\ u$› ‹$s \neq u$› **by** *auto*
    moreover have $\neg\ u \leq p\ s$
      using *suf-pref-eq*[*OF nsD*[*OF*‹$s \leq ns\ u$›]] ‹$s \neq u$› **by** *blast*
    ultimately show $u <lex\ s$
      using *lexord-cancel-right-linorder*[*OF* ‹$u \cdot v <lex\ s \cdot v$›] **by** *blast*
qed

1.4 Concatenation of Lyndon words

theorem *Lyndon-concat*: **assumes** *Lyndon* u **and** *Lyndon* v **and** $u <lex\ v$
  shows *Lyndon* $(u{\cdot}v)$
proof–
  have $u{\cdot}v <lex\ v$
  proof(*cases* $u \leq p\ v$)
    assume $u \leq p\ v$
    obtain z **where** $u{\cdot}z = v$ **and** $z \leq ns\ v$
    using *lq-pref*[*OF* ‹$u \leq p\ v$›] *nsI′ rlex.less-imp-neq*[*OF* ‹$u <lex\ v$›] *self-append-conv*
by *metis*
   from *Lyndon-suf-less*[*OF* ‹*Lyndon* v› *this*(*2*), *THEN lexord-append-leftI-linorder*,
of u]
      LyndonD-nemp[*OF* ‹*Lyndon* u›] *this*(*1*)
    show *?thesis*
      by *blast*
  next
    assume $\neg\ u \leq p\ v$
    then show *?thesis*
      using *local.lexordp-linear*[*of* v $u{\cdot}v$]
        local.lexordp-mid-pref[*OF* ‹$u <lex\ v$›,*of* v]
        prefixI[*of* v u v]
      by *argo*
  qed

  { **fix** z
    assume $z \leq ns\ (u{\cdot}v)$ $z \neq u{\cdot}v$
    have $u{\cdot}v <lex\ z$
    proof(*cases* $z \leq ns\ v$)
      assume $z \leq ns\ v$
      from *Lyndon-suf-less*[*OF* ‹*Lyndon* v› *this*]
      have $z \neq v \Longrightarrow v <lex\ z$
        by *blast*
      thus $u{\cdot}v <lex\ z$
        using ‹$u \cdot v <lex\ v$› *rlex.less-trans*

by *fast*
next
assume ¬ $z \leq ns\ v$
then obtain z' **where** $z' \leq ns\ u\ z' \neq u\ z' \cdot v = z$
using ‹$z \leq ns\ u \cdot v$› ‹$z \neq u \cdot v$› *suffix-append*[*of z u v*]
unfolding *nonempty-suffix-def*
by *force*
from *Lyndon-suf-less*[*OF* ‹*Lyndon u*› *this*(*1*) *this*(*2*)]
have $u <lex\ z'$.
thus $u \cdot v <lex\ z$
using ‹$z' \leq ns\ u$› *lexord-app-right-linorder*[*of u z' v v*] *suffix-length-le*[*of z' u*]
unfolding *nonempty-suffix-def* ‹$z' \cdot v = z$›
by *blast*
qed
}
thus *?thesis*
using *suf-nemp*[*OF LyndonD-nemp*[*OF* ‹*Lyndon v*›], *of u*, *THEN suf-less-Lyndon*]
by *blast*
qed

1.5 Longest Lyndon suffix

fun *longest-Lyndon-suffix*:: *'a list* ⇒ *'a list* (‹*LynSuf*›) **where**
longest-Lyndon-suffix $\varepsilon = \varepsilon$ |
longest-Lyndon-suffix (*a#w*) = (*if Lyndon* (*a#w*) *then a#w else longest-Lyndon-suffix w*)

lemma *longest-Lyndon-suf-ext*: ¬ *Lyndon* (*a # w*) ⟹ *LynSuf w* = *LynSuf* (*a # w*)
using *longest-Lyndon-suffix.simps*(*2*) **by** *presburger*

lemma *longest-Lyndon-suf-suf*: $w \neq \varepsilon$ ⟹ *LynSuf* $w \leq s\ w$
proof(*induction w rule*: *longest-Lyndon-suffix.induct*)
case *1*
then show *?case* **by** *simp*
next
case (*2 a w*)
show *?case*
proof(*cases Lyndon* (*a#w*))
case *True*
then show *?thesis* **by** *auto*
next
case *False*
from *2.IH*[*OF this*, *unfolded longest-Lyndon-suf-ext*[*OF this*], *THEN suffix-ConsI*, *of a*]
Lyndon-sing False
show *?thesis* **by** *blast*
qed

qed

lemma *longest-Lyndon-suf-max*:
$v \leq s\ w \implies Lyndon\ v \implies v \leq s\ (LynSuf\ w)$
proof(*induction w arbitrary*: *v rule*: *longest-Lyndon-suffix.induct*)
case *1*
then show *?case*
using *longest-Lyndon-suffix.simps*(*1*) **by** *presburger*
next
case (*2 a w*)
show *?case*
proof(*cases Lyndon* (*a#w*))
case *True*
then show *?thesis*
using *2.prems*(*1*) *longest-Lyndon-suffix.simps*(*2*) **by** *presburger*
next
case *False*
have $v \neq a \# w$
using *2.prems*(*2*) *False* **by** *blast*
from *2.IH*[*OF False - 2.prems*(*2*), *unfolded longest-Lyndon-suf-ext*[*OF False*]]
2.prems(*1*)[*unfolded suffix-Cons*] *this*
show *?thesis* **by** *fast*
qed
qed

lemma *longest-Lyndon-suf-Lyndon-id*: **assumes** *Lyndon w*
shows *LynSuf w* = *w*
proof(*cases* $w = \varepsilon$, *simp*)
case *False*
from *longest-Lyndon-suf-suf*[*OF this*]
suffix-order.order-refl[*THEN longest-Lyndon-suf-max*[*OF - assms*]]
suffix-order.antisym
show *?thesis* **by** *blast*
qed

lemma *longest-Lyndon-suf-longest*: $w \neq \varepsilon \implies v' \leq s\ w \implies Lyndon\ v' \implies |v'| \leq |(LynSuf\ w)|$
using *longest-Lyndon-suf-max suffix-length-le* **by** *blast*

lemma *longest-Lyndon-suf-sing*: *LynSuf* [*a*] = [*a*]
using *Lyndon-sing longest-Lyndon-suf-Lyndon-id* **by** *blast*

lemma *longest-Lyndon-suf-Lyndon*: $w \neq \varepsilon \implies Lyndon\ (LynSuf\ w)$
proof(*induction w rule*: *longest-Lyndon-suffix.induct*, *blast*)
case (*2 a w*)
show *?case*
proof(*cases Lyndon* (*a#w*))
case *True*
then show *?thesis*

    using *longest-Lyndon-suf-Lyndon-id* **by** *presburger*
  next
    case *False*
    from *2.IH[OF this, unfolded longest-Lyndon-suf-ext[OF this]] Lyndon-sing*
    show *?thesis* **by** *fastforce*
  qed
qed

lemma *longest-Lyndon-suf-nemp*: $w \neq \varepsilon \Longrightarrow LynSuf\ w \neq \varepsilon$
  using *longest-Lyndon-suf-Lyndon[THEN LyndonD-nemp]*.

lemma *longest-Lyndon-sufI*:
  assumes $q \leq s\ w$ **and** *Lyndon q* **and** *all-s*: $(\forall\ s.\ (s \leq s\ w \wedge Lyndon\ s) \longrightarrow s \leq s\ q)$
  shows *LynSuf* $w = q$
proof(*cases* $w = \varepsilon$)
  case *True*
  then show *?thesis*
    using *assms(1) longest-Lyndon-suffix.simps(1) suffix-bot.extremum-uniqueI* **by** *blast*
next
  case *False*
  from *all-s longest-Lyndon-suf-Lyndon[OF this] longest-Lyndon-suf-max[OF assms(1) assms(2)]*
    longest-Lyndon-suf-suf[OF this] suffix-order.eq-iff
  show *?thesis* **by** *blast*
qed

corollary *longest-Lyndon-sufI′*:
  assumes $q \leq s\ w$ **and** *Lyndon q* **and** *all-s*: $\forall\ s.\ (s \leq s\ w \wedge Lyndon\ s) \longrightarrow |s| \leq |q|$
  shows *LynSuf* $w = q$
  using *longest-Lyndon-sufI[OF* ‹$q \leq s\ w$› ‹*Lyndon q*›*] suf-ruler-le all-s* ‹$q \leq s\ w$› **by** *blast*

The next lemma is fabricated to suit the upcoming definition of longest Lyndon factorization.

lemma *longest-Lyndon-suf-shorter*: **assumes** $w \neq \varepsilon$
  shows $|w^{<-1}(LynSuf\ w)| < |w|$
  using *nemp-len[OF longest-Lyndon-suf-nemp[OF* ‹$w \neq \varepsilon$›*]] arg-cong[OF rq-suf[OF longest-Lyndon-suf-suf[OF* ‹$w \neq \varepsilon$›*]], of length]*
  unfolding *lenmorph* **by** *linarith*

1.6 Lyndon factorizations

function *Lyndon-fac::′a list* ⇒ *′a list list* (‹*LynFac*›)
  where *Lyndon-fac w* $=$ (*if* $w \neq \varepsilon$ *then* $((Lyndon\text{-}fac\ (w^{<-1}(LynSuf\ w))) \cdot [LynSuf\ w])$ *else* ε)
  using *longest-Lyndon-suffix.cases* **by** *blast+*

termination
proof(*relation measure length*, *simp*)
show $\bigwedge w.\ w \neq \varepsilon \Longrightarrow (w^{<-1} LynSuf\ w,\ w) \in measure\ length$
unfolding *measure-def inv-image-def* **using** *longest-Lyndon-suf-shorter* **by** *blast*
qed

The factorization *LynFac w* obtained by taking always the longest Lyndon suffix is well defined, and called "Lyndon factorization (of *w*)".

lemma *Lyndon-fac-simp*: $w \neq \varepsilon \Longrightarrow Lyndon\text{-}fac\ w = Lyndon\text{-}fac\ (w^{<-1} LynSuf\ w) \cdot (LynSuf\ w \# \varepsilon)$
using *Lyndon-fac.simps*[*of w*] **by** *meson*

lemma *Lyndon-fac-emp*: $Lyndon\text{-}fac\ \varepsilon = \varepsilon$
by *simp*

Note that the Lyndon factorization of a Lyndon word is trivial.

lemma *Lyndon-fac-longest-Lyndon-id*: $Lyndon\ w \Longrightarrow Lyndon\text{-}fac\ w = [w]$
by (*simp add*: *longest-Lyndon-suf-Lyndon-id*)

Lyndon factorization is composed of Lyndon words ...

lemma *Lyndon-fac-set*: $z \in set\ (Lyndon\text{-}fac\ w) \Longrightarrow Lyndon\ z$
proof(*induction w rule*: *Lyndon-fac.induct*)
case (*1 w*)
then show *Lyndon z*
proof (*cases* $w = \varepsilon$)
assume $w \neq \varepsilon$
have $Lyndon\text{-}fac\ w = (Lyndon\text{-}fac\ (w\ ^{<-1}(LynSuf\ w)\ )) \cdot [LynSuf\ w]$
using *Lyndon-fac-simp*[*OF* ‹$w \neq \varepsilon$›].
from *set-ConsD*[*OF 1.prems*(*1*)[*unfolded rotate1.simps*(*2*)[*of LynSuf w Lyndon-fac* ($w\ ^{<-1}(LynSuf\ w)$), *folded this*, *symmetric*], *unfolded set-rotate1*]]
have $z = LynSuf\ w \lor z \in set\ (Lyndon\text{-}fac\ (w\ ^{<-1}(LynSuf\ w)\ ))$.
thus *Lyndon z*
using *1.IH*[*OF* ‹$w \neq \varepsilon$›] *longest-Lyndon-suf-Lyndon*[*OF* ‹$w \neq \varepsilon$›]
by *blast*
next
assume $w = \varepsilon$
thus *Lyndon z*
using *1.prems*
unfolding *Lyndon-fac-emp*[*folded* ‹$w = \varepsilon$›] *list.set*(*1*) *empty-iff*
by *blast*
qed
qed

...and it indeed is a factorization of the argument.

lemma *Lyndon-fac-longest-dec*: $concat\ (Lyndon\text{-}fac\ w) = w$
proof(*induction w rule*: *Lyndon-fac.induct*)
case (*1 w*)

thus *concat* (*LynFac w*) = *w*
proof (*cases w* = *ε*, *simp*)
assume *w* ≠ *ε*
have *eq*: *concat* (*Lyndon-fac w*) = *concat* ((*Lyndon-fac* (*w* $^{<-1}$(*LynSuf w*)))
) · *concat* ([*LynSuf w*])
unfolding *Lyndon-fac-simp*[*OF* ‹*w* ≠ *ε*›] *concat-morph*..
from *this*[*unfolded 1.IH*[*OF* ‹*w* ≠ *ε*›] *concat-sing′ rq-suf*[*OF longest-Lyndon-suf-suf*[*OF*
‹*w* ≠ *ε*›]]]
show *?case*.
qed
qed

The following lemma makes explicit the inductive character of the definition of *LynFac*.

lemma *Lyndon-fac-longest-pref*: *us* ≤*p* *Lyndon-fac w* ⟹ *Lyndon-fac* (*concat us*)
= *us*
proof(*induction w arbitrary*: *us rule*: *Lyndon-fac.induct*)
case (*1 w*)
thus *LynFac* (*concat us*) = *us*
proof (*cases w* = *ε*, *simp*)
assume *w* ≠ *ε*
have *step*: *Lyndon-fac w* = (*Lyndon-fac* (*w* $^{<-1}$(*LynSuf w*))) · [*LynSuf w*]
using *Lyndon-fac-simp*[*OF* ‹*w* ≠ *ε*›].

consider (*neq*) *us* ≠ *Lyndon-fac w* | (*eq*) *us* = *Lyndon-fac w*
using *1.prems le-neq-implies-less* **by** *blast*
then show *LynFac* (*concat us*) = *us*
proof(*cases*)
case *neq*
hence *us* ≤*p* *Lyndon-fac* (*w*$^{<-1}$*LynSuf w*)
using *1.prems last-no-split*[*of us Lyndon-fac* (*w*$^{<-1}$*LynSuf w*) *LynSuf w*]
unfolding *step*[*symmetric*] **by** *blast*
thus *LynFac* (*concat us*) = *us*
using *1.IH* ‹*w* ≠ *ε*› **by** *blast*
next
case *eq*
show *LynFac* (*concat us*) = *us*
using *Lyndon-fac-longest-dec*[*of w*, *folded eq*] *eq* **by** *simp*
qed
qed
qed

We give name to an important predicate: monotone (nonincreasing) list of Lyndon words.

definition *Lyndon-mono* :: *′a list list* ⇒ *bool* **where**
Lyndon-mono ws ⟷ (∀ *u* ∈ *set ws*. *Lyndon u*) ∧ (*rlex.sorted* (*rev ws*))

lemma *Lyndon-mono-set*: *Lyndon-mono ws* ⟹ *u* ∈ *set ws* ⟹ *Lyndon u*
unfolding *Lyndon-mono-def* **by** *blast*

lemma *Lyndon-mono-sorted*: *Lyndon-mono ws* $\Longrightarrow$ *rlex.sorted* (*rev ws*)
  unfolding *Lyndon-mono-def* **by** *blast*

lemma *Lyndon-mono-nth*: *Lyndon-mono ws* $\Longrightarrow i \leq j \Longrightarrow j < |ws| \Longrightarrow ws!j \leq lex$ $ws!i$
  unfolding *Lyndon-mono-def* **using** *rlex.sorted-rev-nth-mono* **by** *blast*

lemma *Lyndon-mono-empty*[*simp*]: *Lyndon-mono* ε
  unfolding *Lyndon-mono-def* **by** *auto*

lemma *Lyndon-mono-sing*: *Lyndon u* $\Longrightarrow$ *Lyndon-mono* [*u*]
  unfolding *Lyndon-mono-def* **by** *auto*

lemma *Lyndon-mono-fac-Lyndon-mono*:
  assumes *ps* $\leq f$ *ws* **and** *Lyndon-mono ws* **shows** *Lyndon-mono ps*
unfolding *Lyndon-mono-def*
proof
  show $\forall x \in$ (*set ps*). *Lyndon x*
    using ‹*Lyndon-mono ws*›[*unfolded Lyndon-mono-def*] *set-mono-sublist*[*OF* ‹*ps* $\leq f$ *ws*›] **by** *blast*
  show *linorder.sorted* ($\leq lex$) (*rev ps*)
    using *rlex.sorted-append* ‹*Lyndon-mono ws*›[*unfolded Lyndon-mono-def*] ‹*ps* $\leq f$ *ws*›[*unfolded sublist-def*] **by** *fastforce*
qed

Lyndon factorization is monotone! Altogether, we have shown that the Lyndon factorization is a monotone factorization into Lyndon words.

theorem *fac-Lyndon-mono*: *Lyndon-mono* (*Lyndon-fac w*)
proof (*induct Lyndon-fac w arbitrary*: *w rule*: *rev-induct*, *simp*)
  case (*snoc x xs*)
  have *Lyndon x*
    using *Lyndon-fac-set*[*of x*, *unfolded in-set-conv-decomp*, *of w*, *folded snoc.hyps*(*2*)] **by** *fast*
  have *concat* (*xs* $\cdot$ [*x*]) = *w*
    using *Lyndon-fac-longest-dec*[*of w*, *folded snoc.hyps*(*2*)] **by** *auto*
  then show *Lyndon-mono* (*LynFac w*)
  proof (*cases xs* = ε)
    assume *xs* = ε
    show *Lyndon-mono* (*LynFac w*)
      unfolding *Lyndon-mono-def* ‹*xs* $\cdot$ [*x*] = *LynFac w*›[*symmetric*] ‹*xs* = ε› *append.left-neutral*
        rlex.sorted1[*of x*]
      using ‹*Lyndon x*› **by** *force*
  next
    assume *xs* $\neq \varepsilon$
    have *concat* (*xs* $\cdot$ [*x*]) $\neq \varepsilon$ **and** *w* $\neq \varepsilon$
      using *Lyndon-fac-longest-dec snoc.hyps*(*2*) **by** *auto*
    have *x* = *LynSuf w* **and** *xs* = *LynFac* ($w^{<-1}$*LynSuf w*)

using *Lyndon-fac.simps*[*of w*, *folded snoc.hyps*(*2*)] ‹*w* ≠ *ε*›
Lyndon-fac-longest-dec append1-eq-conv[*of xs x LynFac* ($w^{<-1}$*LynSuf w*) *LynSuf w*] **by** *presburger*+
from *Lyndon-mono-sorted*[*OF snoc.hyps*(*1*)[*OF* ‹*xs* = *LynFac* ($w^{<-1}$*LynSuf w*)›], *folded this*]
have *dual-rlex.sorted xs*
unfolding *sorted-dual-rev-iff*.
have *Lyndon* (*last xs*)
using *Lyndon-fac-set*[*of last xs* $w^{<-1}$*LynSuf w*, *folded* ‹*xs* = *LynFac* ($w^{<-1}$*LynSuf w*)›, *OF last-in-set*[*OF* ‹*xs* ≠ *ε*›]].
have *x* ≤*lex last xs*
proof(*rule ccontr*)
assume ¬ *x* ≤*lex last xs* **hence** *last xs* <*lex x* **by** *auto*
from *Lyndon-concat*[*OF* ‹*Lyndon* (*last xs*)› ‹*Lyndon x*› *this*]
have *Lyndon* ((*last xs*) · *x*).
have (*last xs*) · *x* ≤*s concat* (*xs* · [*x*])
using ‹*xs* ≠ *ε*› *concat-last-suf* **by** *auto*
from *longest-Lyndon-suf-longest*[*OF* ‹*concat* (*xs* · [*x*]) ≠ *ε*› *this* ‹*Lyndon* ((*last xs*) · *x*)›,
unfolded ‹*concat* (*xs* · [*x*]) = *w*›, *folded* ‹*x* = *LynSuf w*›]
show *False*
using ‹*Lyndon* (*last xs*)› **by** *simp*
qed
have *dual-rlex.sorted* (*butlast xs* · [*last xs*])
by (*simp add*: ‹*linorder.sorted* (λ*x y*. *y* ≤*lex x*) *xs*› ‹*xs* ≠ *ε*›)
from *this* ‹*x* ≤*lex last xs*›
have *dual-rlex.sorted* (*butlast xs* · [*last xs*,*x*])
using *dual-rlex.sorted-append* **by** *auto*
from *this*[*unfolded hd-word*[*of last xs* [*x*]] *lassoc append-butlast-last-id*[*OF* ‹*xs* ≠ *ε*›]]
have *rlex.sorted* (*rev* (*LynFac w*))
unfolding ‹*xs* · [*x*] = *LynFac w*›[*symmetric*] *sorted-dual-rev-iff*[*symmetric*].
thus *Lyndon-mono* (*LynFac w*)
unfolding *Lyndon-mono-def* **using** *Lyndon-fac-set* **by** *blast*
qed
qed

Now we want to show the converse: any monotone factorization into Lyndon words is the Lyndon factorization

The last element in the Lyndon factorization is the smallest suffix.

lemma *Lyndon-mono-last-smallest*: *Lyndon-mono ws* ⟹*s* ≤*ns* (*concat ws*) ⟹ *last ws* ≤*lex s*
proof(*induct ws arbitrary*: *s rule*: *rev-induct*, *fastforce*)
case (*snoc a ws*)
have *ws*·[*a*] ≠ *ε*
by *blast*
have *last* (*ws*·[*a*]) = *a*
by *simp*

from *last-in-set*[*OF* ‹*ws*·[*a*] $\neq \varepsilon$›, *unfolded this*] ‹*Lyndon-mono* (*ws* · [*a*])›[*unfolded Lyndon-mono-def*]
have *Lyndon a*
by *blast*
show *?case*
proof(*cases s* $\leq$*ns a*)
case *True*
from *Lyndon-suf-le*[*OF* ‹*Lyndon a*›] *this*
show *?thesis*
by *simp*
next
case *False*
hence *ws* $\neq \varepsilon$
using *snoc.prems*(*2*) **by** *force*
obtain s' **where** $s = s'\cdot a$
using *False snoc.prems*(*2*)[*unfolded concat-append*[*of ws* [*a*], *unfolded concat-sing'*]] *suffix-append*[*of s concat ws a*]
unfolding *nonempty-suffix-def*
by *blast*
hence s' $\leq$*ns concat ws*
using *False snoc.prems*(*2*) **by** *fastforce*
have *Lyndon-mono ws*
using ‹*Lyndon-mono* (*ws*·[*a*])› *Lyndon-mono-fac-Lyndon-mono*
by *blast*
from *snoc.hyps*[*OF this* ‹s' $\leq$*ns concat ws*›]
have *last ws* $\leq$*lex* s'
by *auto*
hence *last ws* $\leq$*lex* $s'\cdot a$
using *local.lexordp-eq-trans ord.lexordp-eq-pref* **by** *blast*
have *a* $\leq$*lex last ws*
using ‹*Lyndon-mono* (*ws*·[*a*])›
unfolding *Lyndon-mono-def*
by (*simp add*: ‹*ws* $\neq \varepsilon$› *last-ConsL*)
from *dual-rlex.order-trans*[*OF* ‹*last ws* $\leq$*lex* $s' \cdot a$› *this*, *folded* ‹$s = s' \cdot a$›]
show *?thesis*
unfolding ‹*last* (*ws*·[*a*]) = *a*›
by *blast*
qed
qed

A monotone list, if seen as a factorization, must end with the longest suffix

lemma *Lyndon-mono-last-longest*: **assumes** *ws* $\neq \varepsilon$ **and** *Lyndon-mono ws*
shows *LynSuf* (*concat ws*) = *last ws*
proof–
have *Lyndon* (*last ws*)
using *Lyndon-mono-set assms*(*1*) *assms*(*2*) *last-in-set* **by** *blast*
hence *last ws* $\neq \varepsilon$
using *LyndonD-nemp* **by** *blast*
hence *last ws* $\leq$*ns LynSuf* (*concat ws*)

using *longest-Lyndon-suf-max*[*OF* *concat-last-suf*[*OF assms*(*1*)] ‹*Lyndon* (*last ws*)›]
unfolding *nonempty-suffix-def*
by *blast*

have *concat ws* $\neq \varepsilon$
using *Lyndon.simps assms*(*2*)[*unfolded Lyndon-mono-def*] *set-nemp-concat-nemp*[*OF assms*(*1*)]
by *blast*
from *longest-Lyndon-suf-nemp*[*OF this*] *longest-Lyndon-suf-suf*[*OF this*]
have *LynSuf* (*concat ws*) $\leq$*ns concat ws*
unfolding *nonempty-suffix-def*
by *simp*

show *?thesis*
using *Lyndon-mono-last-smallest*[*OF* ‹*Lyndon-mono ws*› ‹*LynSuf* (*concat ws*) $\leq$*ns concat ws*›]
Lyndon-suf-le[*OF longest-Lyndon-suf-Lyndon*[*OF* ‹*concat ws* $\neq \varepsilon$›], *OF* ‹*last ws* $\leq$*ns LynSuf* (*concat ws*)›]
eq-iff
by *simp*
qed

Therefore, by construction, any monotone list is the Lyndon factorization of its concatenation

lemma *Lyndon-mono-fac*:
Lyndon-mono ws $\Longrightarrow$ *ws* = *Lyndon-fac* (*concat ws*)
proof (*induct ws rule*: *rev-induct*, *simp*)
case (*snoc x xs*)
have *Lyndon-mono xs*
using ‹*Lyndon-mono* (*xs* · [*x*])›
unfolding *Lyndon-mono-def*
by *simp*
from *snoc.hyps*[*OF this*]
have *xs* = *LynFac* (*concat xs*)**.**
have *x* = *LynSuf* (*concat* (*xs* · [*x*]))
using *Lyndon-mono-last-longest*[*OF* - ‹*Lyndon-mono* (*xs* · [*x*])›, *unfolded last-snoc*] **by** *simp*
have *concat* (*xs* · [*x*])$^{<-1}$*x* = *concat xs*
by *simp*
have *concat* (*xs* · [*x*]) $\neq \varepsilon$
using *Lyndon-mono-set snoc.prems* **by** *auto*
from *this*
show *?case*
using *Lyndon-fac.simps*[*of concat* (*xs* · [*x*]), *folded* ‹*x* = *LynSuf* (*concat* (*xs* · [*x*]))›,
unfolded ‹*concat* (*xs* · [*x*])$^{<-1}$*x* = *concat xs*›, *folded* ‹*xs* = *LynFac* (*concat xs*)›]
by *presburger*

qed

This implies that the Lyndon factorization can be characterized in two equivalent ways: as the (unique) monotone factorization (into Lyndon words) or as the "suffix greedy" factorization (into Lyndon words).

corollary *Lyndon-mono-fac-iff*: *Lyndon-mono ws* $\longleftrightarrow$ *ws* = *LynFac* (*concat ws*)
  using *Lyndon-mono-fac fac-Lyndon-mono*[*of concat ws*] **by** *fastforce*

corollary *Lyndon-mono-unique*: **assumes** *Lyndon-mono ws* **and** *Lyndon-mono zs* **and** *concat ws* = *concat zs*
  shows *ws* = *zs*
   using *Lyndon-mono-fac*[*OF* ‹*Lyndon-mono ws*›] *Lyndon-mono-fac*[*OF* ‹*Lyndon-mono zs*›]
  unfolding ‹*concat ws* = *concat zs*› **by** *simp*

1.6.1 Standard factorization

lemma *Lyndon-std*: **assumes** *Lyndon w 1* < |*w*|
  obtains *l m* **where** *w* = *l*·*m* **and** *Lyndon l* **and** *Lyndon m* **and** *l* <*lex m*
proof−
  have *w* ≠ ε *tl w* ≠ ε
    using ‹*1* < |*w*|› *long-list-tl* **by** *auto*
  define *m* **where** *m* = *LynSuf* (*tl w*)
  hence *Lyndon m*
    using ‹*tl w* ≠ ε› *local.longest-Lyndon-suf-Lyndon* **by** *blast*
  have *m* ≤*s w*
    unfolding *m-def*
     using *suffix-order.trans*[*OF longest-Lyndon-suf-suf*[*OF* ‹*tl w* ≠ ε›] *suffix-tl*[*of w*]].
  moreover have *m* ≠ *w*
    unfolding *m-def* **using** *hd-tl*[*OF* ‹*w* ≠ ε›] *longest-Lyndon-suf-suf*[*OF* ‹*tl w* ≠ ε›] *same-suffix-nil*
    not-Cons-self2 **by** *metis*
  ultimately obtain *l* **where** *w* = *l*·*m* **and** *l* ≠ ε
   by (*auto simp add*: *suffix-def*)

  have *Lyndon l*
  proof (*rule unbordered-pref-Lyndon*[*OF* ‹*Lyndon w*›[*unfolded* ‹*w* = *l*·*m*›] ‹*l* ≠ ε›], *rule*)
    assume *bordered l*
    from *unbordered-border*[*OF this*, *unfolded border-def*]
    obtain *s* **where** *s* ≠ ε **and** *s* ≠ *l* **and** *s* ≤*p l* **and** *s* ≤*s l* **and** ¬ *bordered s*
      by *blast*
    have *Lyndon s*
     using *unbordered-pref-Lyndon*[*OF* - ‹*s* ≠ ε› ‹¬ *bordered s*›, *of* $s^{-1>}$*l*·*m*, *unfolded lassoc lq-pref*[*OF* ‹*s* ≤*p l*›]]
      ‹*Lyndon w*›[*unfolded* ‹*w* = *l* · *m*›] **by** *blast*
    have *s* <*lex m*
        using *Lyndon-pref-suf-less*[*OF* ‹*Lyndon w*› - *nsI*[*OF LyndonD-nemp*[*OF*

‹*Lyndon m*›] ‹*m* ≤*s w*›]
      ‹*m* ≠ *w*›, *of s*] *Lyndon.elims*(*2*)[*OF* ‹*Lyndon m*›]
      ‹*s* ≤*p l*› *prefix-append*[*of s l m*, *folded* ‹*w* = *l* · *m*›]
    by *presburger*
  from *Lyndon-concat*[*OF* ‹*Lyndon s*› ‹*Lyndon m*› *this*]
  have *Lyndon* (*s*·*m*)**.**
  moreover have *s*·*m* ≤*s tl w*
    unfolding ‹*w* = *l* · *m*› **using** ‹*s* ≠ *l*› ‹*s* ≤*s l*› *list.collapse*[*OF* ‹*w* ≠ *ε*›, *unfolded* ‹*w* = *l* · *m*›]
    by (*auto simp add*: *suffix-def*)
  ultimately show *False*
    using *m-def* ‹*s* ≠ *ε*› *longest-Lyndon-suf-max same-suffix-nil* **by** *blast*
 qed

 have *l* <*lex m*
  using *Lyndon-pref-suf-less*[*OF* ‹*Lyndon w*› *prefI*[*OF* ‹*w* = *l* · *m*›[*symmetric*]]
    nsI[*OF longest-Lyndon-suf-nemp*[*OF* ‹*tl w* ≠ *ε*›, *folded m-def*] ‹*m* ≤*s w*›] ‹*m* ≠ *w*›]**.**
 from *that*[*OF* ‹*w* = *l* · *m*› ‹*Lyndon l*› ‹*Lyndon m*› *this*]
 show *thesis***.**
qed

corollary *Lyndon-std-iff*:
 Lyndon w ⟷ (|*w*| = *1* ∨ (∃ *l m*. *w* = *l*·*m* ∧ *Lyndon l* ∧ *Lyndon m* ∧ *l* <*lex m*))
 (**is** *?L* ⟷ *?R*)
proof
 assume *?L*
 show *?R*
  using *Lyndon-std*[*OF* ‹*Lyndon w*›]
   nemp-le-len[*OF LyndonD-nemp*[*OF* ‹*Lyndon w*›], *unfolded le-eq-less-or-eq*]
  by *metis*
next
 assume *?R*
 thus *?L*
 proof(*rule disjE*, *fastforce*)
  show ‹∃ *l m*. *w* = *l* · *m* ∧ *Lyndon l* ∧ *Lyndon m* ∧ *l* <*lex m* ⟹ *Lyndon w*›
   using *Lyndon-concat* **by** *blast*
 qed
qed

end — end context linorder

end

theory *Lyndon-Addition*
 imports *Lyndon Szpilrajn.Szpilrajn*

begin

1.6.2 The minimal relation

We define the minimal relation which guarantees the lexicographic minimality of w compared to its nontrivial conjugates.

inductive-set *rotate-rel* :: *'a list* ⇒ *'a rel* **for** *w*
  where $0 < n \Longrightarrow n < |w| \Longrightarrow$ (*mismatch-pair w* (*rotate n w*)) ∈ *rotate-rel w*

A word is Lyndon iff the corresponding order of letters is compatible with *rotate-rel w*.

lemma (**in** *linorder*) *rotate-rel-iff*: **assumes** $w \neq \varepsilon$
  shows *Lyndon w* ⟷ *rotate-rel w* ⊆ {(*x*,*y*). *x* < *y*} (**is** *?L* ⟷ *?R*)
proof
  assume *Lyndon w* **show** *rotate-rel w* ⊆ {(*x*,*y*). *x* < *y*}
  proof
    fix *x* **assume** *x* ∈ *rotate-rel w*
    then obtain *n* **where** *x* = *mismatch-pair w* (*rotate n w*) **and** *0* < *n* **and** *n* <
|*w*|
      using *rotate-rel.cases* **by** *blast*
    have *w* <*lex rotate n w*
      using *LyndonD*[*OF* ‹*Lyndon w*› ‹*0* < *n*› ‹*n* < |*w*|›]**.**
    from *this*[*unfolded lexordp-conv-lexord*]
      prim-no-rotate[*OF Lyndon-prim*[*OF* ‹*Lyndon w*›] ‹*0* < *n*› ‹*n* < |*w*|›]
    show *x* ∈ {(*a*, *b*). *a* < *b*}
    using *lexord-mismatch*[*of w rotate n w* {(*a*,*b*). *a* < *b*}, *folded* ‹*x* = *mismatch-pair*
w (*rotate n w*)›]
        ‹*rotate n w* ≠ *w*› *rotate-comp-eq*[*of w n*]
      unfolding *irrefl-def* **by** *blast*
  qed
next
  assume *?R*
  show *?L*
    unfolding *Lyndon.simps*
  proof(*simp add*: *assms*)
    have *w* <*lex rotate n w* **if** *0* < *n* *n* < |*w*| **for** *n*
    proof–
      have ¬ *w* ⋈ *rotate n w*
        using *rotate-comp-eq*[*of w n*] *subsetD*[*OF* ‹*?R*›, *OF rotate-rel.intros*[*OF* ‹*0*
< *n*› ‹*n* < |*w*|›]]
          mismatch-pair-lcp[*of w rotate n w*] **by** *fastforce*
    from *mismatch-lexord-linorder*[*OF this subsetD*[*OF* ‹*?R*›, *OF rotate-rel.intros*[*OF*
‹*0* < *n*› ‹*n* < |*w*|›]]]
      show *w* <*lex rotate n w***.**
    qed
    thus ∀ *n*. *0* < *n* ∧ *n* < |*w*| ⟶ *w* <*lex rotate n w* **by** *blast*
  qed
qed

It is well known that an acyclic order can be extended to a total strict linear order. This means that a word is Lyndon with respect to some order iff its *rotate-rel w* is acyclic.

lemma *Lyndon-rotate-rel-iff*:
  acyclic (*rotate-rel w*) $\longleftrightarrow$ ($\exists$ *r*. *strict-linear-order r* $\wedge$ *rotate-rel w* $\subseteq$ *r*) (**is** *?L* $\longleftrightarrow$ *?R*)
proof
  assume *?R* **thus** *?L*
    unfolding *strict-linear-order-on-def acyclic-def irrefl-def*
    using *trancl-id trancl-mono* **by** *metis*
next
  assume *?L* **thus** *?R*
    using *acyclic-order-extension* **by** *auto*
qed

lemma *slo-linorder*: *strict-linear-order r* $\Longrightarrow$ *class.linorder* (λ *a b*. (*a*,*b*) $\in$ $r^=$) (λ *a b*. (*a*,*b*) $\in$ *r*)
    unfolding *strict-linear-order-on-def strict-partial-order-def irrefl-def trans-def total-on-def*
    by *unfold-locales blast+*

Application examples

lemma assumes $w \neq \varepsilon$ **and** *acyclic* (*rotate-rel w*) **shows** *primitive w*
proof–
  obtain *r* **where** *strict-linear-order r* **and** *rotate-rel w* $\subseteq$ *r*
    using *Lyndon-rotate-rel-iff assms* **by** *blast*

  interpret *r*: *linorder* λ *a b*. (*a*,*b*) $\in$ $r^=$ λ *a b*. (*a*,*b*) $\in$ *r*
    using *slo-linorder*[*OF* ‹*strict-linear-order r*›]**.**

  have *r.Lyndon w*
    using *r.rotate-rel-iff*[*OF* ‹$w \neq \varepsilon$›] ‹*rotate-rel w* $\subseteq$ *r*› **by** *blast*

  from *r.Lyndon-prim*[*OF this*]
  show *primitive w***.**

qed

lemma assumes $w \neq \varepsilon$ **and** *acyclic* (*rotate-rel w*) **shows** $\neg$ *bordered w*
proof–
  obtain *r* **where** *strict-linear-order r* **and** *rotate-rel w* $\subseteq$ *r*
    using *Lyndon-rotate-rel-iff assms* **by** *blast*

  interpret *r*: *linorder* λ *a b*. (*a*,*b*) $\in$ $r^=$ λ *a b*. (*a*,*b*) $\in$ *r*
    using *slo-linorder*[*OF* ‹*strict-linear-order r*›]**.**

  have *r.Lyndon w*
    using *r.rotate-rel-iff*[*OF* ‹$w \neq \varepsilon$›] ‹*rotate-rel w* $\subseteq$ *r*› **by** *blast*

from *r.Lyndon-unbordered*[*OF this*]
show $\neg$ *bordered w*.
qed

end

References

[1] J.-P. Duval. Mots de Lyndon et périodicité. *RAIRO - Theoretical Informatics and Applications - Informatique Théorique et Applications*, 14(2):181–191, 1980.

[2] J. P. Duval. Factorizing words over an ordered alphabet. *Journal of Algorithms*, 4(4):363–381, Dec. 1983.

[3] M. Lothaire. *Combinatorics on Words*, volume 17 of *Encyclopaedia of Mathematics and its Applications*. Addison-Wesley, Reading, Mass., 1983. Reprinted in the *Cambridge Mathematical Library*, Cambridge University Press, Cambridge UK, 1997.

[4] R. C. Lyndon. On Burnside's problem. *Transactions of the American Mathematical Society*, 77(2):202–202, Feb. 1954.