

Combinatorics on Words formalized  
Lyndon Words

Štěpán Holub  
Štěpán Starosta

December 14, 2021

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

# Contents

<b>1</b>	<b>Lyndon words</b>	<b>2</b>
1.1	Definition and elementary properties . . . . .	2
1.1.1	Underlying order . . . . .	2
1.1.2	Lyndon word definition . . . . .	3
1.1.3	Code equations for Lyndon words . . . . .	4
1.1.4	Properties of Lyndon words . . . . .	5
1.2	Characterization by suffixes . . . . .	6
1.3	Unbordered prefix of a Lyndon word is Lyndon . . . . .	8
1.4	Concatenation of Lyndon words . . . . .	9
1.5	Longest Lyndon suffix . . . . .	10
1.6	Lyndon factorizations . . . . .	12
1.6.1	Standard factorization . . . . .	19
1.6.2	The minimal relation . . . . .	21
	<b>References</b>	<b>24</b>

```
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*:  $ord-class.lexordp\ u\ v \implies ord-class.lexordp\ v\ (u \cdot s) \implies u \leq_p v$   
**by** (*induct rule: lexordp-induct, simp-all*)

**lemma** (in *linorder*) *lexordp-ext*:  $ord-class.lexordp\ u\ v \implies \neg u \leq_p v \implies ord-class.lexordp\ (u \cdot w)\ (v \cdot z)$   
**by** (*induct rule: lexordp-induct, simp-all*)

**lemmas** [*code*] = *lexordp-simps*

**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**  $\leq_{lex}$  50)  
**where** *Lyndon-le* *xs ys*  $\equiv ord-class.lexordp-eq\ xs\ ys$

**interpretation** *rlex*: *linorder* ( $\leq_{lex}$ ) ( $<_{lex}$ )  
**using** *lexordp-linorder*.

**interpretation** *dual-rlex*: *linorder*  $\lambda x y. y \leq_{lex} x \lambda x y. y <_{lex} x$   
**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* = ( $w \neq \varepsilon \wedge (\forall n. 0 < n \wedge n < |w| \longrightarrow w <_{lex} \text{rotate } n w)$ )

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

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

**lemma** *LyndonI*:  $w \neq \varepsilon \Longrightarrow \forall n. 0 < n \wedge n < |w| \longrightarrow w <_{lex} \text{rotate } n w \Longrightarrow$   
*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**  $0 < n \Longrightarrow n < |w| \Longrightarrow \text{rotate } n w \neq w$  **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*:  $Lyndon (u \cdot v) \implies u \neq \varepsilon \implies v \neq \varepsilon \implies u \cdot v <_{lex} v \cdot u$   
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  $\Rightarrow$  nat  $\Rightarrow$  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**  $n < |a\#w| \implies 0 < n \implies Lyndon-rec$  ( $a\#w$ )  $n$   
**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**  $0 < n \implies n < |w| \implies Lyndon-rec$   $w$   $n$   
**proof**(*induction*  $n$ , *simp*)  
  **case** (Suc  $n$ )  
  **have**  $w <_{lex} rotate$  (Suc  $n$ )  $w$   
  **using** *LyndonD* *Suc.prem*s(2) *assms* **by** *blast*  
  **then show** ?*case*  
    using *Suc.IH*[*OF* - *Suc-lessD*[*OF*  $\langle$ Suc  $n$   $< |w|\rangle$ ], *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*:  $0 < n \implies Lyndon-rec$  ( $a\#w$ )  $n \implies (a\#w) <_{lex} rotate$   $n$  ( $a\#w$ ) **for**  $n$   
  **using** *Lyndon-rec.simps*(2)[of  $a\#w$ ] *gr0-implies-Suc*[of  $n$ ] **by** *metis*  
  **have** *bx*: *Lyndon-rec* ( $a \# w$ ) ( $|w|$ ) =  $(\forall n. n < |a\#w| \wedge 0 < n \implies Lyndon-rec$  ( $a\#w$ )  $n$ )

```

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 by 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**  $\neg$  bordered  $w$

```

proof
  assume bordered  $w$ 
  from bordered-dec[OF this]
  obtain  $u v$  where  $u \cdot v \cdot u = w$  and  $u \neq \varepsilon$ .
  hence  $v \cdot u \neq \varepsilon$  and  $u \cdot v \neq \varepsilon$  by blast+
  note lyn = ⟨Lyndon  $w$ ⟩[folded ⟨ $u \cdot v \cdot u = w$ ⟩]
  have  $u \cdot v \cdot u <_{\text{lex}} v \cdot u \cdot u$ 
    using Lyndon-conj-greater[of  $u v \cdot u$ , OF lyn ⟨ $u \neq \varepsilon$ ⟩ ⟨ $v \cdot u \neq \varepsilon$ ⟩, unfolded rassoc].
  from this[unfolded lassoc]
  have  $u \cdot v \neq v \cdot u$ 
    by force
  from lexord-suf-linorder[OF - this, of  $u u$ ]
  have  $u \cdot v <_{\text{lex}} v \cdot u$ 
    using ⟨ $u \cdot v \cdot u <_{\text{lex}} v \cdot u \cdot u$ ⟩ by simp
  from lexord-append-leftI-linorder[of  $u \cdot v v \cdot u$ , unfolded lassoc, OF this, unfolded
rassoc]
  have  $u \cdot u \cdot v <_{\text{lex}} u \cdot v \cdot u$ .
  from this Lyndon-conj-greater[of  $u \cdot v u$ , unfolded rassoc, OF lyn ⟨ $u \cdot v \neq \varepsilon$ ⟩ ⟨ $u$ 
≠ 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**  $w \neq \varepsilon$

**using** prim-nemp[OF ⟨primitive  $w$ ⟩].

**let** ?ConClass = {rotate  $n w$  |  $n$ .  $0 \leq n \wedge n < |w|$ }

**have**  $?ConClass \neq \{\}$   
**using**  $\langle w \neq \varepsilon \rangle$  **by** *blast*  
**have**  $w \in ?ConClass$   
**using**  $\langle w \neq \varepsilon \rangle$  *id-apply*[of  $w$ , *folded rotate0*]  
**by** *force*  
**have** *finite*  $?ConClass$   
**by** *simp*  
**have** *all-rot*: *rotate*  $m w \in ?ConClass$  **for**  $m$   
**using** *rotate-conv-mod*[of  $- w$ ] *mod-less-divisor*[of  $|w|$ ]  $\langle w \neq \varepsilon \rangle$   
**by** *blast*

**obtain**  $w' n$  **where**  $w' \in ?ConClass$  **and** *all-b*:  $\forall b \in ?ConClass. b \leq_{lex} w' \longrightarrow w' = b$  **and**  $w'$ :  $w' = \text{rotate } n w$   
**using** *rlx.finite-has-minimal*[OF  $\langle \text{finite } ?ConClass \rangle \langle ?ConClass \neq \{\} \rangle$ ] **by** *auto*

**have** *rotate*  $n w <_{lex} \text{rotate } na (\text{rotate } n w)$  **if**  $0 < na$  **and**  $na < |w|$  **for**  $na$   
**proof–**  
**from** *prim-no-rotate*[OF *assms*[*unfolded prim-rotate-conv*[of  $w n$ ], of  $na$ ]  $\langle na < |w| \rangle \langle 0 < na \rangle$   
**have** *rotate*  $na (\text{rotate } n w) \neq \text{rotate } n w$  **by** *force*  
**hence**  $\neg \text{rotate } na (\text{rotate } n w) \leq_{lex} \text{rotate } n w$   
**using** *all-b*[*rule-format*, OF *all-rot*[of  $na + n$ , *folded rotate-rotate*[of  $na n w$ ]]]  
**unfolding**  $w'$   
**by** *auto*  
**from** *rlx.not-le-imp-less*[OF *this*]  
**show** *rotate*  $n w <_{lex} \text{rotate } na (\text{rotate } n w)$ .  
**qed**  
**hence** *Lyndon*  $(\text{rotate } n w)$   
**using**  $\langle w \neq \varepsilon \rangle$  **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  $\langle \text{primitive } w \rangle$ ]  
**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**  $p = \text{take } |s| w$   
**have**  $|s| \leq |w|$   
**using** *nsD*[OF  $\langle s \leq_{ns} w s \rangle$ ] **by** *force*  
**have**  $p \leq_p w$  **and**  $|p| = |s|$   
**unfolding** *p-def*

**using** *take-is-prefix*  $\langle |s| \leq |w| \rangle$  *take-len* **by** *blast+*  
**hence**  $p \neq s$   
**using** *Lyndon-unbordered*[*OF*  $\langle \text{Lyndon } w \rangle$ ]  $\langle s \leq_{ns} w \rangle$   $\langle s \neq w \rangle$  *assms*  
**by** *auto*  
**define**  $p' s'$  **where**  $p' = \text{drop } |s| \ w$  **and**  $s' = \text{take } |p'| \ w$   
**have**  $p \cdot p' = w$  **and**  $s' \cdot s = w$   
**unfolding** *p'-def* *p-def* *s'-def*  
**using**  $\langle s \leq_{ns} w \rangle$  **by** *auto*  
**have**  $|p'| = |s'|$   
**using** *s'-def*  $\langle p \cdot p' = w \rangle$  **by** *auto*  
**have**  $w <_{lex} s \cdot s'$   
**using** *Lyndon-conj-greater*[*of*  $s' s$ , *unfolded*  $\langle s' \cdot s = w \rangle$ , *OF*  $\langle \text{Lyndon } w \rangle$ ]  $\langle p \neq s \rangle$   
**unfolding**  $\langle s' \cdot s = w \rangle$  *p-def* **using**  $\langle s' \cdot s = w \rangle$  *assms(3)* **by** *fastforce*  
**from** *lexord-suf-linorder*[*OF* -  $\langle p \neq s \rangle$   $\langle |p| = |s| \rangle$   $\langle |p'| = |s'| \rangle$ , *OF* *this*[*folded*  $\langle p \cdot p' = w \rangle$ ]]  
**have**  $p <_{lex} s$ .  
**from** *lexord-app-right-linorder*[*OF* *this*, *of*  $p' \varepsilon$ , *unfolded*  $\langle p \cdot p' = w \rangle$ ]  $\langle |p| = |s| \rangle$   
**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* *rlx.less-trans*)  
**show**  $p <_{lex} w$   
**using**  $\langle p \neq w \rangle$  *assms(2)* *lexordp-append-rightI* **by** *fastforce*  
**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**  $\neg$  *primitive*  $w$   
**obtain**  $q k$  **where**  $q \neq \varepsilon$   $1 < k$   $q^{\textcircled{a}} k = w$   $w \neq q$  — the exact match of  $\llbracket \neg$  *primitive*  $?w$ ;  $?w \neq \varepsilon$ ;  $\bigwedge r k. \llbracket r \neq \varepsilon$ ;  $1 < k$ ;  $r^{\textcircled{a}} k = ?w$ ;  $?w \neq r \rrbracket \implies ?thesis \rrbracket \implies ?thesis$   
fastens the proof considerably  
**using** *non-prim*[*OF*  $\langle \neg$  *primitive*  $w \rangle$   $\langle w \neq \varepsilon \rangle$ ] **by** *blast*  
**hence**  $q \leq_{ns} w$   
**unfolding** *nonempty-suffix-def* *pow-eq-if-list*[*of*  $q k$ ] *pow-commutes-list*[*symmetric*]  
**using** *sufI*[*of*  $q^{\textcircled{a}} (k - 1) q w$ ]  
**by** *presburger*



**have**  $q <_p w$   
**using**  $\langle 1 < k \rangle \langle q @ k = w \rangle$   
**unfolding**  $\text{pow-eq-if-list}[of\ q\ k] \text{pow-eq-if-list}[of\ q\ k-1]$   
**using**  $\langle w \neq \varepsilon \rangle$  **by** *auto*  
**from**  $\text{lexordp-append-rightI}[of\ q^{-1} > w\ q,$   
 $\text{unfolded}\ \text{lq-pref}[OF\ \text{sprefD1}[OF\ \text{this}]],\ OF\ \text{lq-spref}[OF\ \text{this}]$   
**have**  $q <_{lex} w$ .  
**thus** *Lyndon*  $w$   
**unfolding** *Lyndon.simps* **using**  $\langle q \leq_{ns} w \rangle \langle w \neq \varepsilon \rangle$  *assms(2)* *rlex.order.strict-trans*  
**by** *blast*  
**next**  
**assume** *primitive*  $w$   
**have**  $w <_{lex} \text{rotate}\ l\ w$  **if** *assms-l*:  $0 < l < |w|$  **for**  $l$   
**proof-**  
**have**  $\text{take}\ l\ w \leq_{np} w$  **and**  $|\text{take}\ l\ w| = l$   
**using** *assms-l* *take-is-prefix*  $\langle l < |w| \rangle$  **by** *auto*  
**have**  $\text{drop}\ l\ w \leq_{ns} w$   
**using**  $\langle l < |w| \rangle$  *suffix-drop* **by** *auto*  
**have**  $\text{drop}\ l\ w \neq w$   
**using** *append-take-drop-id*[*of*  $l\ w$ ] *npD'*[*OF*  $\langle \text{take}\ l\ w \leq_{np} w \rangle$ ] **by** *force*  
**have**  $\text{drop}\ l\ w \cdot \text{take}\ l\ w = \text{rotate}\ l\ w$   
**using** *rotate-append*[*of*  $\text{take}\ l\ w\ \text{drop}\ l\ w$ , *symmetric*, *unfolded*  $\langle \text{take}\ l\ w =$   
 $l \rangle,$   
 $\text{unfolded}\ \text{append-take-drop-id}$ ].  
  
**have**  $w <_{lex} \text{drop}\ l\ w$   
**using**  $\langle \text{drop}\ l\ w \leq_{ns} w \rangle \langle \text{drop}\ l\ w \neq w \rangle$  *assms(2)* **by** *blast*  
**from** *lexord-app-right-linorder*[*OF* *this* *suffix-length-le*[*OF* *conjunct2*[*OF*  $\langle \text{drop}\ l$   
 $w \leq_{ns} w \rangle$ ][*unfolded* *nonempty-suffix-def*]]], *of*  $\varepsilon\ \text{take}\ l\ w$ , *unfolded* *append.right-neutral*]  
**have**  $w <_{lex} \text{drop}\ l\ w \cdot \text{take}\ l\ w$ .  
**thus**  $w <_{lex} \text{rotate}\ l\ w$   
**by** (*simp* *add*:  $\langle \text{drop}\ l\ w \cdot \text{take}\ l\ w = \text{rotate}\ l\ w \rangle$ )  
**qed**  
**thus** *Lyndon*  $w$   
**by** (*simp* *add*:  $\langle w \neq \varepsilon \rangle$  *local.LyndonI*)  
**qed**

**corollary** *Lyndon-suf-char*:  $w \neq \varepsilon \implies \text{Lyndon}\ w \iff (\forall s. s \leq_{ns} w \implies s \neq w \implies w <_{lex} s)$   
**using** *Lyndon-suf-less* *suf-less-Lyndon* **by** *blast*

**lemma** *Lyndon-suf-le*:  $\text{Lyndon}\ w \implies s \leq_{ns} w \implies w \leq_{lex} s$   
**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*:  $\text{Lyndon}\ (u \cdot v) \implies u \neq \varepsilon \implies \neg \text{bordered}\ u \implies \text{Lyndon}\ u$

**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*  $\langle Lyndon (u \cdot v) \rangle$ , *of*  $s \cdot v$ ] **by** *auto*  
**have**  $\neg s \leq_p u$   
**using**  $\langle \neg bordered\ u \rangle \langle s \leq_{ns} u \rangle \langle s \neq u \rangle$  **by** *auto*  
**moreover have**  $\neg u \leq_p s$   
**using** *suf-pref-eq*[*OF* *nsD*[*OF*  $\langle s \leq_{ns} u \rangle$ ]]  $\langle s \neq u \rangle$  **by** *blast*  
**ultimately show**  $u <_{lex} s$   
**using** *lexord-cancel-right-linorder*[*OF*  $\langle u \cdot v <_{lex} s \cdot v \rangle$ ] **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**  $u \leq_p v$   
**then obtain**  $z$  **where**  $u \cdot z = v$  **and**  $z \leq_{ns} v$   
**using** *assms*(3) *dual-rlex.less-imp-neq* **by** *auto*  
**from** *Lyndon-suf-less*[*OF*  $\langle Lyndon\ v \rangle$  *this*(2), *THEN* *lexord-append-left-linorder*,  
*of*  $u$ ]  
*LyndonD-nemp*[*OF*  $\langle Lyndon\ u \rangle$ ] *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*  $\langle u <_{lex} v \rangle$ , *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*  $\langle Lyndon\ v \rangle$  *this*]  
**have**  $z \neq v \implies v <_{lex} z$   
**by** *blast*  
**thus**  $u \cdot v <_{lex} z$   
**using**  $\langle u \cdot v <_{lex} v \rangle$  *rlex.less-trans*  
**by** *fast*

```

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

```

## 1.5 Longest Lyndon suffix

```

fun longest-Lyndon-suffix:: 'a list  $\Rightarrow$  '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:  $\neg$  Lyndon ( $a \# w$ )  $\implies$  LynSuf  $w =$  LynSuf ( $a \#$ 
w)
  using longest-Lyndon-suffix.simps(2) by presburger

```

```

lemma longest-Lyndon-suf-suf:  $w \neq \varepsilon \implies$  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 suf-
fix-ConsI, of  $a$ ]

```

```

      Lyndon-sing False

```

```

      show ?thesis by blast

```

```

  qed

```

```

qed

```

**lemma** *longest-Lyndon-suf-max*:  
 $v \leq_s w \implies \text{Lyndon } v \implies v \leq_s (\text{LynSuf } w)$   
**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-conv*  
  **show** ?*thesis* **by** *blast*  
**qed**

**lemma** *longest-Lyndon-suf-longest*:  $w \neq \varepsilon \implies v' \leq_s w \implies \text{Lyndon } v' \implies |v'| \leq |(\text{LynSuf } w)|$   
**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*:  $w \neq \varepsilon \implies \text{Lyndon } (\text{LynSuf } w)$   
**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:  $w \neq \varepsilon \implies \text{LynSuf } w \neq \varepsilon$ 
  using longest-Lyndon-suf-Lyndon[THEN LyndonD-nemp].

lemma longest-Lyndon-sufI:
  assumes  $q \leq s$   $w$  and Lyndon  $q$  and all-s:  $(\forall s. (s \leq s \ w \ \wedge \ \text{Lyndon } s) \longrightarrow s \leq s$ 
 $q)$ 
  shows  $\text{LynSuf } w = q$ 
proof(cases  $w = \varepsilon$ )
  case True
  then show ?thesis
    using assms(1) by fastforce
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:  $\forall s. (s \leq s \ w \ \wedge \ \text{Lyndon } s) \longrightarrow |s| \leq$ 
 $|q|$ 
  shows  $\text{LynSuf } w = q$ 
  using longest-Lyndon-sufI[OF  $\langle q \leq s \ w \rangle \ \langle \text{Lyndon } q \rangle$ ] suf-ruler-le all-s  $\langle q \leq s \ w \rangle$ 
  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  $|w|^{<-1}(\text{LynSuf } w) < |w|$ 
  using nemp-len[OF longest-Lyndon-suf-nemp[OF  $\langle w \neq \varepsilon \rangle$ ]] arg-cong[OF rq-suf[OF
  longest-Lyndon-suf-suf[OF  $\langle w \neq \varepsilon \rangle$ ]], of length]
  unfolding length-append by linarith

```

## 1.6 Lyndon factorizations

```

function Lyndon-fac::'a list  $\Rightarrow$  'a list list (LynFac)
  where Lyndon-fac  $w = (if \ w \neq \varepsilon \ then \ ((\text{Lyndon-fac } (w^{<-1}(\text{LynSuf } w))) \cdot$ 
 $[\text{LynSuf } w]) \ else \ \varepsilon)$ 
  using longest-Lyndon-suffix.cases by blast+
termination
proof(relation measure length, simp)

```

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

The factorization  $\text{LynFac } w$  obtained by taking always the longest Lyndon suffix is well defined, and called “Lyndon factorization (of  $w$ )”.

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

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

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

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

Lyndon factorization is composed of Lyndon words ...

**lemma** *Lyndon-fac-set*:  $z \in \text{set } (\text{Lyndon-fac } w) \implies \text{Lyndon } z$

**proof**(*induction w rule: Lyndon-fac.induct*)

**case** ( $1 w$ )

**then show**  $\text{Lyndon } z$

**proof** (*cases w = ε*)

**assume**  $w \neq \varepsilon$

**have**  $\text{Lyndon-fac } w = (\text{Lyndon-fac } (w^{<-1} (\text{LynSuf } w))) \cdot [\text{LynSuf } w]$

**using** *Lyndon-fac-simp[OF ‹w ≠ ε›]*.

**from** *set-ConsD[OF 1.prem1[unfolding rotate1.simps(2)[of LynSuf w Lyndon-fac (w^{<-1} (LynSuf w)), folded this, symmetric], unfolded set-rotate1]*

**have**  $z = \text{LynSuf } w \vee z \in \text{set } (\text{Lyndon-fac } (w^{<-1} (\text{LynSuf } w)))$ .

**thus**  $\text{Lyndon } z$

**using** *1.IH[OF ‹w ≠ ε› longest-Lyndon-suf-Lyndon[OF ‹w ≠ ε›]*

**by** *blast*

**next**

**assume**  $w = \varepsilon$

**thus**  $\text{Lyndon } z$

**using** *1.prem1*

**unfolding** *Lyndon-fac-emp[folded ‹w = ε› list.set(1) empty-iff]*

**by** *blast*

**qed**

**qed**

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

**lemma** *Lyndon-fac-longest-dec*:  $\text{concat } (\text{Lyndon-fac } w) = w$

**proof**(*induction w rule: Lyndon-fac.induct*)

**case** ( $1 w$ )

**thus**  $\text{concat } (\text{LynFac } w) = w$

**proof** (*cases w = ε, simp*)

```

assume  $w \neq \varepsilon$ 
  have  $eq$ :  $concat (Lyndon\text{-}fac\ w) = concat ( (Lyndon\text{-}fac (w^{<^{-1}}(LynSuf\ w) ))$ 
)  $\cdot concat ([LynSuf\ w])$ 
  unfolding  $Lyndon\text{-}fac\text{-}simp[OF\ \langle w \neq \varepsilon \rangle]$   $concat\text{-}morph..$ 
  from  $this[unfolded\ 1.IH[OF\ \langle w \neq \varepsilon \rangle]\ concat\text{-}sing'\ rq\text{-}suf[OF\ longest\text{-}Lyndon\text{-}suf\text{-}suf[OF\ \langle w \neq \varepsilon \rangle]]]$ 
  show  $?case.$ 
qed
qed

```

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

**lemma**  $Lyndon\text{-}fac\text{-}longest\text{-}pref$ :  $us \leq_p Lyndon\text{-}fac\ w \implies Lyndon\text{-}fac (concat\ us) = us$

**proof**( $induction\ w\ arbitrary$ ;  $us\ rule$ :  $Lyndon\text{-}fac.induct$ )

**case**  $(1\ w)$

**thus**  $LynFac (concat\ us) = us$

**proof** ( $cases\ w = \varepsilon, simp$ )

**assume**  $w \neq \varepsilon$

**have**  $step$ :  $Lyndon\text{-}fac\ w = (Lyndon\text{-}fac (w^{<^{-1}}(LynSuf\ w))) \cdot [LynSuf\ w]$

**using**  $Lyndon\text{-}fac\text{-}simp[OF\ \langle w \neq \varepsilon \rangle]$ .

**consider** ( $neq$ )  $us \neq Lyndon\text{-}fac\ w \mid$  ( $eq$ )  $us = Lyndon\text{-}fac\ w$

**using**  $1.prem\ le\text{-}neq\text{-}implies\text{-}less$  **by**  $blast$

**then show**  $LynFac (concat\ us) = us$

**proof**( $cases$ )

**case**  $neq$

**hence**  $us \leq_p Lyndon\text{-}fac (w^{<^{-1}}LynSuf\ w)$

**using**  $1.prem\ last\text{-}no\text{-}split[of\ us\ Lyndon\text{-}fac (w^{<^{-1}}LynSuf\ w)\ LynSuf\ w]$

**unfolding**  $step[symmetric]$  **by**  $blast$

**thus**  $LynFac (concat\ us) = us$

**using**  $1.IH\ \langle w \neq \varepsilon \rangle$  **by**  $blast$

**next**

**case**  $eq$

**show**  $LynFac (concat\ us) = us$

**using**  $Lyndon\text{-}fac\text{-}longest\text{-}dec[of\ w, folded\ eq]$   $eq$  **by**  $simp$

**qed**

**qed**

**qed**

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

**definition**  $Lyndon\text{-}mono$  ::  $'a\ list\ list \implies bool$  **where**

$Lyndon\text{-}mono\ ws \iff (\forall\ u \in set\ ws.\ Lyndon\ u) \wedge (rlex.sorted (rev\ ws))$

**lemma**  $Lyndon\text{-}mono\text{-}set$ :  $Lyndon\text{-}mono\ ws \implies u \in set\ ws \implies Lyndon\ u$

**unfolding**  $Lyndon\text{-}mono\text{-}def$  **by**  $blast$

**lemma**  $Lyndon\text{-}mono\text{-}sorted$ :  $Lyndon\text{-}mono\ ws \implies rlex.sorted (rev\ ws)$

**unfolding** *Lyndon-mono-def* **by** *blast*

**lemma** *Lyndon-mono-nth*: *Lyndon-mono*  $ws \implies i \leq j \implies j < |ws| \implies ws!j \leq_{lex} ws!i$

**unfolding** *Lyndon-mono-def* **using** *rlex.sorted-rev-nth-mono* **by** *blast*

**lemma** *Lyndon-mono-empty[simp]*: *Lyndon-mono*  $\varepsilon$

**unfolding** *Lyndon-mono-def* **by** *auto*

**lemma** *Lyndon-mono-sing*: *Lyndon*  $u \implies \text{Lyndon-mono } [u]$

**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**  $\forall x \in (\text{set } ps). \text{Lyndon } x$

**using**  $\langle \text{Lyndon-mono } ws \rangle [\text{unfolded } \text{Lyndon-mono-def}] \text{set-mono-sublist}[OF \langle ps \leq_f ws \rangle]$  **by** *blast*

**show** *linorder.sorted*  $(\leq_{lex}) (rev \ ps)$

**using** *rlex.sorted-append*  $\langle \text{Lyndon-mono } ws \rangle [\text{unfolded } \text{Lyndon-mono-def}] \langle ps \leq_f ws \rangle [\text{unfolded sublist-def}]$  **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*  $(\text{Lyndon-fac } w)$

**proof** (*induct Lyndon-fac w arbitrary: w rule: rev-induct, simp*)

**case**  $(\text{snoc } x \ xs)$

**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*  $(\text{LynFac } w)$

**proof** (*cases xs = ε*)

**assume**  $xs = \varepsilon$

**show** *Lyndon-mono*  $(\text{LynFac } w)$

**unfolding** *Lyndon-mono-def*  $\langle xs \cdot [x] = \text{LynFac } w \rangle [\text{symmetric}] \langle xs = \varepsilon \rangle$   
*append.left-neutral*

*rlex.sorted1*[*of x*]

**using**  $\langle \text{Lyndon } x \rangle$  **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**  $x = \text{LynSuf } w$  **and**  $xs = \text{LynFac } (w^{<-1} \text{LynSuf } w)$

**using** *Lyndon-fac.simps*[*of w, folded snoc.hyps(2)*]  $\langle w \neq \varepsilon \rangle$

*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<sup>-1</sup>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<sup>-1</sup>LynSuf w, folded ‹xs = LynFac (w<sup>-1</sup>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 (λx 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*:  $Lyndon\text{-mono } ws \implies s \leq ns \text{ (concat } ws) \implies last\ ws \leqlex s$

**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] ≠ ε›, unfolded this] ‹Lyndon-mono (ws · [a])›[unfolded
Lyndon-mono-def]

```

```

have Lyndon a
  by blast
show ?case
proof(cases s ≤ns a)
  case True
  from Lyndon-suf-le[OF ‹Lyndon a›] this
  show ?thesis
  by simp
next
  case False
  hence ws ≠ ε
  using snoc.prem(2) by force
  obtain s' where s = s' · a
  using False snoc.prem(2)[unfolding concat-append[of ws [a], unfolded con-
cat-sing1]] suffix-append[of s concat ws a]
  unfolding nonempty-suffix-def
  by blast
  hence s' ≤ns concat ws
  using False snoc.prem(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' ≤ns concat ws›]
  have last ws ≤lex s'
  by auto
  hence last ws ≤lex s' · a
  using local.lexordp-eq-trans ord.lexordp-eq-pref by blast
  have a ≤lex last ws
  using ‹Lyndon-mono (ws · [a])›
  unfolding Lyndon-mono-def
  by (simp add: ‹ws ≠ ε› last-ConsL)
  from dual-rlex.order-trans[OF ‹last ws ≤lex s' · a› this, folded ‹s = s' · 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**  $\text{LynSuf}(\text{concat } ws) = \text{last } ws$

**proof**–

```

have Lyndon (last ws)
  using Lyndon-mono-set assms(1) assms(2) last-in-set by blast
hence last ws ≠ ε
  using LyndonD-nemp by blast
hence last ws ≤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 ≠ ε
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) ≤ns concat ws
unfolding nonempty-suffix-def
by simp

show ?thesis
using Lyndon-mono-last-smallest[OF ⟨Lyndon-mono ws⟩ ⟨LynSuf (concat ws)
≤ns concat ws⟩]
Lyndon-suf-le[OF longest-Lyndon-suf-Lyndon[OF ⟨concat ws ≠ ε⟩], OF ⟨last
ws ≤ns LynSuf (concat ws)⟩]
by simp
qed

```

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

**lemma** *Lyndon-mono-fac:*

*Lyndon-mono ws ⇒ 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])<-1x = concat xs
by simp
have concat (xs · [x]) ≠ ε
using Lyndon-mono-set snoc.premis by auto
from this
show ?case
using Lyndon-fac.simps[of concat (xs · [x]), folded ⟨x = LynSuf (concat (xs ·
[x])⟩),
unfolded ⟨concat (xs · [x])<-1x = concat xs⟩, folded ⟨xs = LynFac (concat
xs)⟩]
by presburger
qed

```

This implies that the Lyndon factorization can be characterized in two equiv-

alent 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*  $w$   $\longleftrightarrow$   $w = \text{LynFac}(\text{concat } w)$   
**using** *Lyndon-mono-fac fac-Lyndon-mono*[*of concat*  $w$ ] **by** *fastforce*

**corollary** *Lyndon-mono-unique*: **assumes** *Lyndon-mono*  $w$  **and** *Lyndon-mono*  $z$  **and**  $\text{concat } w = \text{concat } z$

**shows**  $w = z$

**using** *Lyndon-mono-fac*[*OF*  $\langle \text{Lyndon-mono } w \rangle$ ] *Lyndon-mono-fac*[*OF*  $\langle \text{Lyndon-mono } z \rangle$ ]

**unfolding**  $\langle \text{concat } w = \text{concat } z \rangle$  **by** *simp*

### 1.6.1 Standard factorization

**lemma** *Lyndon-std*: **assumes** *Lyndon*  $w$   $1 < |w|$

**obtains**  $l\ m$  **where**  $w = l \cdot m$  **and** *Lyndon*  $l$  **and** *Lyndon*  $m$  **and**  $l <_{\text{lex}} m$

**proof**–

**have**  $w \neq \varepsilon$   $tl\ w \neq \varepsilon$

**using**  $\langle 1 < |w| \rangle$  *long-list-tl* **by** *auto*

**define**  $m$  **where**  $m = \text{LynSuf}(tl\ w)$

**hence** *Lyndon*  $m$

**using**  $\langle tl\ w \neq \varepsilon \rangle$  *local.longest-Lyndon-suf-Lyndon* **by** *blast*

**have**  $m \leq_s w$

**unfolding** *m-def*

**using** *suffix-order.trans*[*OF* *longest-Lyndon-suf-suf*[*OF*  $\langle tl\ w \neq \varepsilon \rangle$ ] *suffix-tl*[*of*  $w$ ]].

**moreover** **have**  $m \neq w$

**unfolding** *m-def* **using** *hd-word'*[*OF*  $\langle w \neq \varepsilon \rangle$ ] *list.simps(3)* *longest-Lyndon-suf-suf*[*OF*  $\langle tl\ w \neq \varepsilon \rangle$ ] *same-suffix-nil*

**by** *fastforce*

**ultimately obtain**  $l$  **where**  $w = l \cdot m$  **and**  $l \neq \varepsilon$

**by** *force*

**have** *Lyndon*  $l$

**proof** (*rule* *unbordered-pref-Lyndon*[*OF*  $\langle \text{Lyndon } w \rangle$ ][*unfolded*  $\langle w = l \cdot m \rangle$ ]  $\langle l \neq \varepsilon \rangle$ , *rule*)

**assume** *bordered*  $l$

**from** *unbordered-border*[*OF* *this*, *unfolded* *border-def*]

**obtain**  $s$  **where**  $s \neq \varepsilon$  **and**  $s \neq l$  **and**  $s \leq_p l$  **and**  $s \leq_s l$  **and**  $\neg$  *bordered*  $s$

**by** *blast*

**have** *Lyndon*  $s$

**using** *unbordered-pref-Lyndon*[*OF*  $\langle s \neq \varepsilon \rangle$ ]  $\langle \neg$  *bordered*  $s$  $\rangle$ , *of*  $s^{-1} > l \cdot m$ , *unfolded* *lassoc lq-pref*[*OF*  $\langle s \leq_p l \rangle$ ]

$\langle \text{Lyndon } w \rangle$ ][*unfolded*  $\langle w = l \cdot m \rangle$ ] **by** *blast*

**have**  $s <_{\text{lex}} m$

**using** *Lyndon-pref-suf-less*[*OF*  $\langle \text{Lyndon } w \rangle$ ]  $-$  *nsI*[*OF* *LyndonD-nemp*[*OF*  $\langle \text{Lyndon } m \rangle$ ]  $\langle m \leq_s w \rangle$ ]

$\langle m \neq w \rangle$ , *of*  $s$ ] *Lyndon.elims(2)*[*OF*  $\langle \text{Lyndon } m \rangle$ ]

$\langle s \leq_p l \rangle$  *prefix-append*[*of*  $s\ l\ m$ , *folded*  $\langle w = l \cdot m \rangle$ ]

**by** *presburger*  
**from** *Lyndon-concat*[*OF*  $\langle \text{Lyndon } s \rangle \langle \text{Lyndon } m \rangle$  *this*]  
**have** *Lyndon*  $(s \cdot m)$ .  
**moreover have**  $s \cdot m \leq_s tl \ w$   
**unfolding**  $\langle w = l \cdot m \rangle$  **using**  $\langle s \neq l \rangle \langle s \leq_s l \rangle$  *list.collapse*[*OF*  $\langle w \neq \varepsilon \rangle$ ,  
*unfolded*  $\langle w = l \cdot m \rangle$ ] **by force**  
**ultimately show** *False*  
**using** *m-def*  $\langle s \neq \varepsilon \rangle$  *longest-Lyndon-suf-max same-suffix-nil* **by blast**  
**qed**

**have**  $l <_{lex} m$   
**using** *Lyndon-pref-suf-less*[*OF*  $\langle \text{Lyndon } w \rangle$  *prefI*[*OF*  $\langle w = l \cdot m \rangle$  [*symmetric*]]  
*nsI*[*OF* *longest-Lyndon-suf-nemp*[*OF*  $\langle tl \ w \neq \varepsilon \rangle$ , *folded* *m-def*]  $\langle m \leq_s w \rangle$ ]  
 $\langle m \neq w \rangle$ ].  
**from** *that*[*OF*  $\langle w = l \cdot m \rangle \langle \text{Lyndon } l \rangle \langle \text{Lyndon } m \rangle$  *this*]  
**show** *thesis*.  
**qed**

**corollary** *Lyndon-std-iff*:

$\text{Lyndon } w \longleftrightarrow (|w| = 1 \vee (\exists l \ m. w = l \cdot m \wedge \text{Lyndon } l \wedge \text{Lyndon } m \wedge l <_{lex} m))$

(**is**  $?L \longleftrightarrow ?R$ )

**proof**

**assume**  $?L$

**show**  $?R$

**using** *Lyndon-std*[*OF*  $\langle \text{Lyndon } w \rangle$ ]

*nemp-pos-len*[*OF* *LyndonD-nemp*[*OF*  $\langle \text{Lyndon } w \rangle$ ], *unfolded le-eq-less-or-eq*]

**by** *metis*

**next**

**assume**  $?R$

**thus**  $?L$

**proof**(*rule disjE, fastforce*)

**show**  $\langle \exists l \ m. w = l \cdot m \wedge \text{Lyndon } l \wedge \text{Lyndon } m \wedge l <_{lex} m \implies \text{Lyndon } w \rangle$

**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  $\Rightarrow$  'a rel **for**  $w$   
**where**  $0 < n \implies n < |w| \implies (\text{mismatch-pair } w (\text{rotate } n w)) \in \text{rotate-rel } w$

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**  $\text{Lyndon } w \iff \text{rotate-rel } w \subseteq \{(x,y). x < y\}$  (**is**  $?L \iff ?R$ )

**proof**

**assume** *Lyndon*  $w$  **show**  $\text{rotate-rel } w \subseteq \{(x,y). x < y\}$

**proof**

**fix**  $x$  **assume**  $x \in \text{rotate-rel } w$

**then obtain**  $n$  **where**  $x = \text{mismatch-pair } w (\text{rotate } n w)$  **and**  $0 < n$  **and**  $n < |w|$

**using** *rotate-rel.cases* **by** *blast*

**have**  $w <_{\text{lex}} \text{rotate } n w$

**using** *LyndonD*[*OF*  $\langle \text{Lyndon } w \rangle \langle 0 < n \rangle \langle n < |w| \rangle$ ].

**from** *this*[*unfolded lexordp-conv-lexord*]

*prim-no-rotate*[*OF Lyndon-prim*[*OF*  $\langle \text{Lyndon } w \rangle \langle 0 < n \rangle \langle n < |w| \rangle$ ]

**show**  $x \in \{(a, b). a < b\}$

**using** *lexord-mismatch*[*of*  $w \text{ rotate } n w \{(a,b). a < b\}$ , *folded*  $\langle x = \text{mismatch-pair } w (\text{rotate } n w) \rangle$ ]

$\langle \text{rotate } n w \neq w \rangle \text{rotate-comp-eq}[*of* w n]$

**unfolding** *irrefl-def* **by** *blast*

**qed**

**next**

**assume**  $?R$

**show**  $?L$

**unfolding** *Lyndon.simps*

**proof**(*simp add: assms*)

**have**  $w <_{\text{lex}} \text{rotate } n w$  **if**  $0 < n$   $n < |w|$  **for**  $n$

**proof**–

**have**  $\neg w \bowtie \text{rotate } n w$

**using** *rotate-comp-eq*[*of*  $w n$ ] *subsetD*[*OF*  $\langle ?R \rangle$ , *OF* *rotate-rel.intros*[*OF*  $\langle 0 < n \rangle \langle n < |w| \rangle$ ]]

*mismatch-pair-lcp*[*of*  $w \text{ rotate } n w$ ] **by** *fastforce*

**from** *mismatch-lexord-linorder*[*OF this subsetD*[*OF*  $\langle ?R \rangle$ , *OF* *rotate-rel.intros*[*OF*  $\langle 0 < n \rangle \langle n < |w| \rangle$ ]]]

**show**  $w <_{\text{lex}} \text{rotate } n w$ .

**qed**

**thus**  $\forall n. 0 < n \wedge n < |w| \implies w <_{\text{lex}} \text{rotate } n w$  **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  $r \implies$  class.linorder ( $\lambda a b. (a,b) \in r^\equiv$ ) ( $\lambda a b. (a,b) \in r$ )*

**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** *auto*

**interpret** *r*: *linorder  $\lambda a b. (a,b) \in r^\equiv \lambda a b. (a,b) \in r$*

**using** *slo-linorder[OF  $\langle$ strict-linear-order  $r$  $\rangle$ ]*.

**have** *r.Lyndon w*

**using** *r.rotate-rel-iff[OF  $\langle$ w  $\neq$   $\varepsilon$  $\rangle$   $\langle$ rotate-rel w  $\subseteq$  r $\rangle$*  **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** *auto*

**interpret** *r*: *linorder  $\lambda a b. (a,b) \in r^\equiv \lambda a b. (a,b) \in r$*

**using** *slo-linorder[OF  $\langle$ strict-linear-order  $r$  $\rangle$ ]*.

**have** *r.Lyndon w*

**using** *r.rotate-rel-iff[OF  $\langle$ w  $\neq$   $\varepsilon$  $\rangle$   $\langle$ rotate-rel w  $\subseteq$  r $\rangle$*  **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.