

Combinatorics on Words formalized
Lyndon Words

Štěpán Holub
Štěpán Starosta

June 17, 2024

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

Contents

1	Lyndon words	2
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	3
1.1.4	Properties of Lyndon words	4
1.2	Characterization by suffixes	4
1.3	Unbordered prefix of a Lyndon word is Lyndon	5
1.4	Concatenation of Lyndon words	5
1.5	Longest Lyndon suffix	5
1.6	Lyndon factorizations	6
1.6.1	Standard factorization	8
1.6.2	The minimal relation	9
	References	10

```
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*: $ord-class.lexordp\ u\ v \implies ord-class.lexordp\ v\ (u \cdot s) \implies u \leq_p v$
<proof>

lemma (in *linorder*) *lexordp-ext*: $ord-class.lexordp\ u\ v \implies \neg u \leq_p v \implies ord-class.lexordp\ (u \cdot w)\ (v \cdot z)$
<proof>

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*)
<proof>

interpretation *dual-rlex*: $\text{linorder } \lambda x y. y \leq_{\text{lex}} x \lambda x y. y <_{\text{lex}} x$
 ⟨proof⟩

lemma *sorted-dual-rev-iff*: $\text{dual-rlex.sorted } ws \longleftrightarrow \text{rlex.sorted } (\text{rev } ws)$
 ⟨proof⟩

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* $w = (w \neq \varepsilon \wedge (\forall n. 0 < n \wedge n < |w| \longrightarrow w <_{\text{lex}} \text{rotate } n w))$

lemma *LyndonD*: $\text{Lyndon } w \Longrightarrow 0 < n \Longrightarrow n < |w| \Longrightarrow w <_{\text{lex}} \text{rotate } n w$
 ⟨proof⟩

lemma *LyndonD-nemp*: $\text{Lyndon } w \Longrightarrow w \neq \varepsilon$
 ⟨proof⟩

lemma *LyndonI*: $w \neq \varepsilon \Longrightarrow \forall n. 0 < n \wedge n < |w| \longrightarrow w <_{\text{lex}} \text{rotate } n w \Longrightarrow \text{Lyndon } w$
 ⟨proof⟩

lemma *Lyndon-sing*: $\text{Lyndon } [a]$
 ⟨proof⟩

lemma *Lyndon-prim*: **assumes** *Lyndon* w
shows *primitive* w
 ⟨proof⟩

lemma *Lyndon-conj-greater*: $\text{Lyndon } (u \cdot v) \Longrightarrow u \neq \varepsilon \Longrightarrow v \neq \varepsilon \Longrightarrow u \cdot v <_{\text{lex}} v \cdot u$
 ⟨proof⟩

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>

lemma *Lyndon-Lyndon-rec*: **assumes** $Lyndon\ w$
shows $0 < n \implies n < |w| \implies Lyndon-rec\ w\ n$
<proof>

lemma *Lyndon-code* [code]:
 $Lyndon\ Nil = False$
 $Lyndon\ (a\ \#\ w) = Lyndon-rec\ (a\ \#\ w)\ (|w|)$
<proof>

1.1.4 Properties of Lyndon words

Lyndon words are unbordered

theorem *Lyndon-unbordered*: **assumes** $Lyndon\ w$ **shows** $\neg\ bordered\ w$
<proof>

Each conjugacy class contains a Lyndon word

lemma *conjug-Lyndon-ex*: **assumes** $primitive\ w$
obtains n **where** $Lyndon\ (rotate\ n\ w)$
<proof>

lemma *conjug-Lyndon-ex'*: **assumes** $primitive\ w$
obtains v **where** $w \sim v$ **and** $Lyndon\ v$
<proof>

1.2 Characterization by suffixes

lemma *Lyndon-suf-less*: **assumes** $Lyndon\ w\ s \leq_{ns}\ w\ s \neq w$
shows $w <_{lex}\ s$
<proof>

lemma *Lyndon-pref-suf-less*: **assumes** $Lyndon\ w\ p \leq_p\ w\ s \leq_{ns}\ w\ s \neq w$
shows $p <_{lex}\ s$
<proof>

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>

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

<proof>

lemma *Lyndon-suf-le*: $\text{Lyndon } w \implies s \leq_{ns} w \implies w \leq_{lex} s$

<proof>

1.3 Unbordered prefix of a Lyndon word is Lyndon

lemma *unbordered-pref-Lyndon*: $\text{Lyndon } (u \cdot v) \implies u \neq \varepsilon \implies \neg \text{bordered } u \implies \text{Lyndon } u$

<proof>

1.4 Concatenation of Lyndon words

theorem *Lyndon-concat*: **assumes** $\text{Lyndon } u$ **and** $\text{Lyndon } v$ **and** $u <_{lex} v$
shows $\text{Lyndon } (u \cdot v)$

<proof>

1.5 Longest Lyndon suffix

fun *longest-Lyndon-suffix*:: 'a list \Rightarrow 'a list (*LynSuf*) **where**

longest-Lyndon-suffix $\varepsilon = \varepsilon$ |

longest-Lyndon-suffix $(a \# w) = (\text{if } \text{Lyndon } (a \# w) \text{ then } a \# w \text{ else } \text{longest-Lyndon-suffix } w)$

lemma *longest-Lyndon-suf-ext*: $\neg \text{Lyndon } (a \# w) \implies \text{LynSuf } w = \text{LynSuf } (a \# w)$

<proof>

lemma *longest-Lyndon-suf-suf*: $w \neq \varepsilon \implies \text{LynSuf } w \leq_s w$

<proof>

lemma *longest-Lyndon-suf-max*:

$v \leq_s w \implies \text{Lyndon } v \implies v \leq_s (\text{LynSuf } w)$

<proof>

lemma *longest-Lyndon-suf-Lyndon-id*: **assumes** $\text{Lyndon } w$
shows $\text{LynSuf } w = w$

<proof>

lemma *longest-Lyndon-suf-longest*: $w \neq \varepsilon \implies v' \leq_s w \implies \text{Lyndon } v' \implies |v'| \leq |(\text{LynSuf } w)|$

<proof>

lemma *longest-Lyndon-suf-sing*: $\text{LynSuf } [a] = [a]$

<proof>

lemma *longest-Lyndon-suf-Lyndon*: $w \neq \varepsilon \implies \text{Lyndon} (\text{LynSuf } w)$
 ⟨proof⟩

lemma *longest-Lyndon-suf-nemp*: $w \neq \varepsilon \implies \text{LynSuf } w \neq \varepsilon$
 ⟨proof⟩

lemma *longest-Lyndon-sufI*:
assumes $q \leq s$ **and** *Lyndon* q **and** *all-s*: $(\forall s. (s \leq s \ w \wedge \text{Lyndon } s) \longrightarrow s \leq q)$
shows $\text{LynSuf } w = q$
 ⟨proof⟩

corollary *longest-Lyndon-sufI'*:
assumes $q \leq s$ **and** *Lyndon* q **and** *all-s*: $\forall s. (s \leq s \ w \wedge \text{Lyndon } s) \longrightarrow |s| \leq |q|$
shows $\text{LynSuf } w = q$
 ⟨proof⟩

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

lemma *longest-Lyndon-suf-shorter*: **assumes** $w \neq \varepsilon$
shows $|w^{<-1}(\text{LynSuf } w)| < |w|$
 ⟨proof⟩

1.6 Lyndon factorizations

function *Lyndon-fac*::'a list \Rightarrow 'a list list (*LynFac*)
where *Lyndon-fac* $w = (\text{if } w \neq \varepsilon \text{ then } ((\text{Lyndon-fac } (w^{<-1}(\text{LynSuf } w))) \cdot [\text{LynSuf } w]) \text{ else } \varepsilon)$
 ⟨proof⟩

termination
 ⟨proof⟩

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*: $w \neq \varepsilon \implies \text{Lyndon-fac } w = \text{Lyndon-fac } (w^{<-1} \text{LynSuf } w) \cdot (\text{LynSuf } w \ \# \ \varepsilon)$
 ⟨proof⟩

lemma *Lyndon-fac-emp*: $\text{Lyndon-fac } \varepsilon = \varepsilon$
 ⟨proof⟩

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

lemma *Lyndon-fac-longest-Lyndon-id*: $\text{Lyndon } w \implies \text{Lyndon-fac } w = [w]$
 ⟨proof⟩

Lyndon factorization is composed of Lyndon words ...

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

<proof>

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

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

<proof>

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

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

<proof>

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

definition *Lyndon-mono* :: 'a list list \implies bool **where**

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

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

<proof>

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

<proof>

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

<proof>

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

<proof>

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

<proof>

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

assumes $ps \leq_f ws$ **and** $\text{Lyndon-mono } ws$ **shows** $\text{Lyndon-mono } ps$

<proof>

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

theorem *fac-Lyndon-mono*: $\text{Lyndon-mono} (\text{Lyndon-fac } w)$

<proof>

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*: $\text{Lyndon-mono } ws \implies s \leq_{ns} (\text{concat } ws) \implies \text{last } ws \leq_{\text{lex}} s$

<proof>

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>

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

lemma *Lyndon-mono-fac*:

$Lyndon\ mono\ ws \implies ws = Lyndon\ fac (concat\ ws)$

<proof>

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*: $Lyndon\ mono\ ws \iff ws = LynFac (concat\ ws)$

<proof>

corollary *Lyndon-mono-unique*: **assumes** *Lyndon-mono* ws **and** *Lyndon-mono* zs **and** $concat\ ws = concat\ zs$

shows $ws = zs$

<proof>

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** $l <_{lex} m$

<proof>

corollary *Lyndon-std-iff*:

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

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

<proof>

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

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) $\longleftrightarrow (\exists r. \text{strict-linear-order } r \wedge \text{rotate-rel } w \subseteq r)$ (**is** ? $L \longleftrightarrow ?R$)
<proof>

lemma *slo-linorder*: *strict-linear-order* $r \implies \text{class.linorder } (\lambda a b. (a,b) \in r^=)$ ($\lambda a b. (a,b) \in r$)
<proof>

Application examples

lemma **assumes** $w \neq \varepsilon$ **and** *acyclic* (*rotate-rel* w) **shows** *primitive* w
<proof>

lemma **assumes** $w \neq \varepsilon$ **and** *acyclic* (*rotate-rel* w) **shows** \neg *bordered* w
<proof>

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.