Combinatorics on Words formalized
Binary codes that do not preserve primitivity

Štěpán Holub
Martin Raška

May 26, 2024

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

Contents

theory *Binary-Square-Interpretation*

imports
Combinatorics-Words.Submonoids
Combinatorics-Words.Equations-Basic
begin

0.1 Lemmas for covered x square

This section explores various variants of the situation when $x \cdot x$ is covered with $x \cdot y \ ^{@} \ k \cdot u \cdot v \cdot y \ ^{@} \ l \cdot x$, with $y = u \cdot v$, and the displayed dots being synchronized.

0.1.1 Two particular cases

lemma *pref-suf-pers-short*: **assumes** $x \leq p\ v \cdot x$ **and** $|v \cdot u| < |x|$ **and** $x \leq s\ r \cdot u \cdot v \cdot u$ **and** $r \in \langle\{u,v\}\rangle$
— $x \cdot x$ is covered by $(p \cdot u \cdot v \cdot u) \cdot v \cdot x$, the displayed dots being synchronized
— That is, the condition on the first x in $x \cdot y \ ^{@} \ k \cdot u \cdot v \cdot y \ ^{@} \ l \cdot x$ is relaxed
shows $u{\cdot}v = v{\cdot}u$
proof (*rule nemp-comm*)
have $v \cdot u <s\ x$
using *suf-prod-long-less*[*OF* - ‹$|v \cdot u| < |x|$›, *of* $r \cdot u$, *unfolded rassoc*, *OF* ‹$x \leq s\ r \cdot u \cdot v \cdot u$›].
assume $u \neq \varepsilon$ **and** $v \neq \varepsilon$
obtain q **where** $x = q \cdot v \cdot u$ **and** $q \neq \varepsilon$
using ‹$v \cdot u <s\ x$› **by** (*auto simp add*: *suffix-def*)
hence $q \leq s\ r \cdot u$
using ‹$x \leq s\ r \cdot u \cdot v \cdot u$› **by** (*auto simp add*: *suffix-def*)
from *suf-trans*[*OF primroot-suf this*]
have $\varrho\ q \leq s\ r \cdot u$.
have $q \cdot v = v \cdot q$
using *pref-marker*[*OF* ‹$x \leq p\ v{\cdot}x$›, *of* q] ‹$x = q \cdot v \cdot u$› **by** *simp*
from *suf-marker-per-root*[*OF* ‹$x \leq p\ v \cdot x$›, *of* q u, *unfolded rassoc* ‹$x = q \cdot v \cdot u$›]
have $u <p\ v \cdot u$
using ‹$v \neq \varepsilon$› **by** *blast*
from *per-root-primroot*[*OF this*]
comm-primroots'[*OF* ‹$q \neq \varepsilon$› ‹$v \neq \varepsilon$› ‹$q \cdot v = v \cdot q$›]
have $u \leq p\ \varrho\ q \cdot u$
by *force*

from *gen-prim*[*OF* ‹$r \in \langle\{u,\ v\}\rangle$›, *unfolded*]
have $r \in \langle\{u,\ \varrho\ q\}\rangle$
unfolding ‹$\varrho\ q = \varrho\ v$›.
from *two-elem-root-suf-comm*[*OF* ‹$u \leq p\ \varrho\ q \cdot u$› ‹$\varrho\ q \leq s\ r \cdot u$› *this*]
show $u \cdot v = v \cdot u$
using *comm-primroot-conv*[*of* - v, *folded* ‹$\varrho\ q = \varrho\ v$›] **by** *blast*
qed

lemma *pref-suf-pers-large-overlap*:
assumes
$p \leq_p x$ **and** $s \leq_s x$ **and** $p \leq_p r \cdot p$ **and** $s \leq_s s \cdot r$ **and** $|x| + |r| \leq |p| + |s|$
shows $x \cdot r = r \cdot x$
using *assms*
proof (*cases* $r = \varepsilon$)
assume $r \neq \varepsilon$ **hence** $r \neq \varepsilon$ **by** *blast*
have $|s| \leq |x|$
using ‹$s \leq_s x$› **unfolding** *suffix-def* **by** *force*
have $|p| \leq |x|$
using ‹$p \leq_p x$› **by** (*force simp add*: *prefix-def*)
have $|r| \leq |p|$
using ‹$|x| + |r| \leq |p| + |s|$› ‹$|s| \leq |x|$› **unfolding** *lenmorph* **by** *linarith*
have $|r| \leq |s|$
using ‹$|x| + |r| \leq |p| + |s|$› ‹$|p| \leq |x|$› **unfolding** *lenmorph* **by** *linarith*
obtain *p1 ov s1* **where** $p1 \cdot ov \cdot s1 = x$ **and** $p1 \cdot ov = p$ **and** $ov \cdot s1 = s$
using *pref-suf-overlapE*[*OF* ‹$p \leq_p x$› ‹$s \leq_s x$›] **using** ‹$|x| + |r| \leq |p| + |s|$›
by *auto*
have $|r| \leq |ov|$
using ‹$|x| + |r| \leq |p| + |s|$›[*folded* ‹$p1 \cdot ov \cdot s1 = x$› ‹$p1 \cdot ov = p$› ‹$ov \cdot s1 = s$›]
unfolding *lenmorph* **by** *force*
have $r \leq_p p$
using ‹$|r| \leq |p|$›[*unfolded swap-len*] *pref-prod-long*[*OF* ‹$p \leq_p r \cdot p$›] **by** *blast*
hence $r \leq_p x$
using ‹$p \leq_p x$› **by** *auto*
have $r \leq_s s$
using ‹$|r| \leq |s|$›[*unfolded swap-len*] *pref-prod-long*[*reversed*, *OF* ‹$s \leq_s s \cdot r$›]
by *blast*
hence $r \leq_s x$
using ‹$s \leq_s x$› **by** *auto*
obtain k **where** $p \leq_p r^{@} k$ $0 < k$
using *per-root-powE*[*OF per-rootI*[*OF* ‹$p \leq_p r \cdot p$› ‹$r \neq \varepsilon$›]] *sprefD1* **by** *metis*
hence $p1 \cdot ov \leq_f r^{@} k$
unfolding ‹$p1 \cdot ov = p$› **by** *blast*
obtain l **where** $s \leq_s r^{@}\ l$ ‹$0 < l$›
using *per-root-powE*[*reversed*, *OF per-rootI*[*reversed*, *OF* ‹$s \leq_s s \cdot r$› ‹$r \neq \varepsilon$›]]
ssufD1 **by** *metis*
hence $ov \cdot s1 \leq_f r^{@}\ l$
unfolding ‹$ov \cdot s1 = s$› **by** *blast*
from *per-glue-facs*[*OF* ‹$p1 \cdot ov \leq_f r^{@}\ k$› ‹$ov \cdot s1 \leq_f r^{@}\ l$› ‹$|r| \leq |ov|$›, *unfolded* ‹$p1 \cdot ov \cdot s1 = x$›]
obtain m **where** $x \leq_f r\ ^{@}\ m$**.**
show $x \cdot r = r \cdot x$
using *root-suf-comm*[*OF*
pref-prod-root[*OF marker-fac-pref*[*OF* ‹$x \leq_f r\ ^{@}\ m$› ‹$r \leq_p x$›]]
suffix-appendI[*OF* ‹$r \leq_s x$›]]**..**
qed *simp*

0.1.2 Main cases

locale *pref-suf-pers* =
  fixes x u v k m
  assumes
    x-pref: $x \leq p\ (v \cdot (u \cdot v)^{@} k) \cdot x$ — $\leq_p x\ (p \cdot x)$ and $\leq_p p\ (q \cdot p)$ where $q = v \cdot u$
    and
    x-suf: $x \leq s\ x \cdot (u \cdot v)^{@} m \cdot u$ — $\leq_s x\ (s \cdot x)$ and $\leq_s s\ (q' \cdot s)$ where $q' = u \cdot v$
    and *k-pos*: $0 < k$ **and** *m-pos*: $0 < m$
begin

lemma *pref-suf-commute-all-commutes*:
  assumes $|u \cdot v| \leq |x|$ **and** $u \cdot v = v \cdot u$
  shows *commutes* $\{u,v,x\}$
  using *assms*
proof (*cases* $u \cdot v = \varepsilon$)
  let *?p* $= (v \cdot (u \cdot v)^{@} k)$
  let *?s* $= (u \cdot v)^{@} m \cdot u$
  note *x-pref x-suf*

  assume $u \cdot v \neq \varepsilon$
  have *?p* $\neq \varepsilon$ **and** *?s* $\neq \varepsilon$ **and** $v \cdot u \neq \varepsilon$
    using ‹$u \cdot v \neq \varepsilon$› *m-pos k-pos* **by** *auto*
  obtain r **where** $u \in r*$ **and** $v \in r*$ **and** *primitive* r
    using ‹$u \cdot v = v \cdot u$› *comm-primrootE* **by** *metis*
  hence $r \neq \varepsilon$
    by *force*

  have *?p* $\in r*$ **and** *?s* $\in r*$ **and** $v \cdot u \in r*$ **and** $u \cdot v \in r*$
    using ‹$u \in r*$› ‹$v \in r*$›
    by (*simp-all add*: *add-roots root-pow-root*)

  have $x \leq p\ r \cdot x$
    using ‹*?p* $\in r*$› ‹$x \leq p$ *?p* $\cdot x$› ‹*?p* $\neq \varepsilon$› **by** *blast*
  have $v \cdot u \leq s\ x$
    using *ruler-le*[*reversed*, *OF* - - ‹$|u \cdot v| \leq |x|$›[*unfolded swap-len*[*of u*]],
      of $(x \cdot (u \cdot v)^{@} (m-1)) \cdot u) \cdot v \cdot u$, *OF triv-suf*, *unfolded rassoc*, *OF* ‹$x \leq s$ $x \cdot$ *?s*›[*unfolded pow-pos'*[*OF m-pos*] *rassoc*]].
  have $r \leq s\ \ v \cdot u$
    using ‹$v \cdot u \neq \varepsilon$› ‹$v \cdot u \in r*$› *per-root-suf* **by** *blast*
  have $r \leq s\ r \cdot x$
    using *suf-trans*[*OF* ‹$r \leq s\ \ v \cdot u$› ‹$v \cdot u \leq s\ x$›, *THEN suffix-appendI*] **by** *blast*
  have $x \cdot r\ = r \cdot x$
    using *root-suf-comm*[*OF* ‹$x \leq p\ r \cdot x$› ‹$r \leq s\ r \cdot x$›, *symmetric*].
  hence $x \in r*$
    by (*simp add*: ‹*primitive r*› *prim-comm-root*)
  thus *commutes* $\{u,v,x\}$
    using ‹$u \in r*$› ‹$v \in r*$› *commutesI-root*[*of* $\{u,v,x\}$] **by** *blast*
qed *simp*

lemma *no-overlap*:
assumes
len: $|v \cdot (u \cdot v)^{@}k| + |(u \cdot v)^{@}m \cdot u| \leq |x|$ (**is** $|?p| + |?s| \leq |x|$) **and**
$0 < k$ $0 < m$
shows *commutes* $\{u,v,x\}$
using *assms*
proof (*cases* $u \cdot v = \varepsilon$)
note *x-pref x-suf*
assume $u \cdot v \neq \varepsilon$
have $?p \neq \varepsilon$ **and** $?s \neq \varepsilon$
using ‹$u \cdot v \neq \varepsilon$› *m-pos k-pos* **by** *force+*
from *per-lemma-pref-suf*[*OF per-rootI*[*OF* ‹$x \leq p$ $?p \cdot x$› ‹$?p \neq \varepsilon$›] *per-rootI*[*reversed*, *OF* ‹$x \leq s$ $x \cdot ?s$› ‹$?s \neq \varepsilon$›] ‹$|?p| + |?s| \leq |x|$›]
obtain r s kp ks mw **where** $?p = (r \cdot s)^{@}kp$ **and** $?s = (s \cdot r)^{@}ks$ **and** $x = (r \cdot s)^{@}mw \cdot r$ **and** *primitive* $(r \cdot s)$**.**
hence ϱ $?p = r \cdot s$
using ‹$v \cdot (u \cdot v)^{@}k \neq \varepsilon$› *comm-primroots nemp-pow-nemp pow-comm prim-self-root* **by** *metis*
moreover have ϱ $?s = s \cdot r$
using *primroot-unique*[*OF* ‹$?s \neq \varepsilon$› - ‹$?s = (s \cdot r)^{@}ks$›] *prim-conjug*[*OF* ‹*primitive* $(r \cdot s)$›] **by** *blast*
ultimately have ϱ $?p \sim \varrho$ $?s$
by *force*
from *conj-pers-conj-comm*[*OF this k-pos m-pos*]
have $u \cdot v = v \cdot u$**.**

from *pref-suf-commute-all-commutes*[*OF* - *this*]
show *commutes* $\{u,v,x\}$
using *len* **by** *auto*
qed *simp*

lemma *no-overlap′*:
assumes
len: $|v \cdot (u \cdot v)^{@}k| + |(u \cdot v)^{@}m \cdot u| \leq |x|$ (**is** $|?p| + |?s| \leq |x|$)
and $0 < k$ $0 < m$
shows $u \cdot v = v \cdot u$
by (*rule commutesE*[*of* $\{u,v,x\}$], *simp-all add*: *no-overlap*[*OF assms*])

lemma *short-overlap*:
assumes
len1: $|x| < |v \cdot (u \cdot v)^{@}k| + |(u \cdot v)^{@}m \cdot u|$ (**is** $|x| < |?p| + |?s|$) **and**
len2: $|v \cdot (u \cdot v)^{@}k| + |(u \cdot v)^{@}m \cdot u| \leq |x| + |u|$ (**is** $|?p| + |?s| \leq |x| + |u|$)
shows *commutes* $\{u,v,x\}$
proof (*rule pref-suf-commute-all-commutes*)
show $|u \cdot v| \leq |x|$
using *len2* **unfolding** *pow-pos*[*OF k-pos*] *lenmorph* **by** *simp*
next
note *x-pref x-suf*
— obtain the overlap

have $|?p| \leq |x|$
using *len2* **unfolding** *lenmorph* **by** *linarith*
hence $?p \leq_p x$
using ‹$x \leq_p ?p \cdot x$› *pref-prod-long* **by** *blast*

have $|?s| \leq |x|$
using *len2* **unfolding** *pow-pos*[*OF k-pos*] *pow-len lenmorph* **by** *auto*
hence $?s \leq_s x$
using *suf-prod-long*[*OF* ‹$x \leq_s x \cdot ?s$›] **by** *blast*

from *pref-suf-overlapE*[*OF* ‹$?p \leq_p x$› ‹$?s \leq_s x$› *less-imp-le*[*OF len1*]]
obtain *p1 ov s1* **where** $p1 \cdot ov \cdot s1 = x$ **and** $p1 \cdot ov = ?p$ **and** $ov \cdot s1 = ?s$**.**

from *len1*[*folded this*]
have $ov \neq \varepsilon$
by *fastforce*

have $|ov| \leq |u|$
using *len2*[*folded* ‹$p1 \cdot ov \cdot s1 = x$› ‹$p1 \cdot ov = ?p$› ‹$ov \cdot s1 = ?s$›] **unfolding** *lenmorph* **by** *auto*

then obtain s' **where** $ov \cdot s' = u$ **and** $s' \cdot v \cdot (u \cdot v)^{@}(m-1) \cdot u = s1$
using *eqdE*[*OF* ‹$ov \cdot s1 = ?s$›[*unfolded pow-pos*[*OF m-pos*] *rassoc*]] **by** *auto*

— obtain the left complement

from *eqdE*[*reversed*, *of p1 ov* $v \cdot (u \cdot v)^{@}(k-1)\ u \cdot v$, *unfolded rassoc*, *OF* ‹$p1 \cdot ov = ?p$›[*unfolded pow-pos'*[*OF k-pos*]]] ‹$|ov| \leq |u|$›
have $v \cdot (u \cdot v)^{@}(k-1) \leq_p p1$
unfolding *lenmorph* **by** (*auto simp add*: *prefix-def*)

then obtain q **where** $v \cdot (u \cdot v)^{@}(k-1) \cdot q = p1$
by (*force simp add*: *prefix-def*)

— main proof using the lemma $[\![?u \cdot ?v \cdot ?v \cdot ?u \cdot ?p = ?q \cdot ?u \cdot ?v \cdot ?u;\ \leq_s ?p\ ?u;\ \leq_s ?q\ ?w;\ ?w \in \langle\{?u, ?v\}\rangle]\!] \Longrightarrow ?v \cdot ?u = ?u \cdot ?v$

show $u \cdot v = v \cdot u$
proof (*rule sym*, *rule uvu-suf-uvvu*)
show $s' \leq_s u$
using ‹$ov \cdot s' = u$› ‹$ov \neq \varepsilon$› **by** *blast*
show $u \cdot v \cdot v \cdot u \cdot s' = q \cdot u \cdot v \cdot u$ — the main fact: the overlap situation
proof–
have $u \cdot v \cdot u \leq_p ?s$
unfolding *pow-pos*[*OF m-pos*] *rassoc pref-cancel-conv shift-pow* **by** *blast*
hence $p1 \cdot u \cdot v \cdot u \leq_p x$
unfolding ‹$p1 \cdot ov \cdot s1 = x$›[*symmetric*] ‹$ov \cdot s1 = ?s$› *pref-cancel-conv*.
hence $v \cdot (u \cdot v)^{@}(k-1) \cdot q \cdot u \cdot v \cdot ov \leq_p x$

using ‹$v \cdot (u \cdot v)^{@}(k{-}1) \cdot q = p1$› ‹$ov \cdot s' = u$› **by** (*force simp add*: *prefix-def*)

have $v \cdot u \leq p\ x$
using ‹$?p \leq p\ x$›[*unfolded pow-pos*[*OF k-pos*]] **by** (*auto simp add*: *prefix-def*)
have $|?p \cdot v \cdot u| \leq |x|$
using *len2* **unfolding** *pow-pos*[*OF m-pos*] *lenmorph* **by** *force*
hence $?p \cdot v \cdot u \leq p\ x$
using ‹$x \leq p\ ?p \cdot x$› ‹$v \cdot u \leq p\ x$› *pref-prod-longer* **by** *blast*
hence $v \cdot (u \cdot v)^{@}(k{-}1) \cdot u \cdot v \cdot v \cdot u \leq p\ x$
unfolding *pow-pos'*[*OF k-pos*] *rassoc*.

have $|v \cdot (u \cdot v)^{@}(k{-}1) \cdot u \cdot v \cdot v \cdot u| = |v \cdot (u \cdot v)^{@}(k{-}1) \cdot q \cdot u \cdot v \cdot ov|$
using *lenarg*[*OF* ‹$p1 \cdot ov = ?p$›[*folded* ‹$v \cdot (u \cdot v)^{@}(k{-}1) \cdot q = p1$›, *unfolded pow-pos*[*OF k-pos*] *rassoc cancel*]]
by *force*

from *ruler-eq-len*[*OF* ‹$v \cdot (u \cdot v)^{@}(k{-}1) \cdot u \cdot v \cdot v \cdot u \leq p\ x$› ‹$v \cdot (u \cdot v)^{@}(k{-}1) \cdot q \cdot u \cdot v \cdot ov \leq p\ x$› *this*, *unfolded cancel*]
have $u \cdot v \cdot v \cdot u = q \cdot u \cdot v \cdot ov$.

thus $u \cdot v \cdot v \cdot u \cdot s' = q \cdot u \cdot v \cdot u$
using ‹$ov \cdot s' = u$› **by** *auto*
qed
show $q \leq s\ v \cdot u$
proof (*rule ruler-le*[*reversed*])
show $q \leq s\ x$
proof (*rule suf-trans*)
show $p1 \leq s\ x$
using ‹$p1 \cdot ov \cdot s1 = x$›[*unfolded* ‹$ov \cdot s1 = ?s$›] ‹$x \leq s\ x \cdot ?s$› *same-suffix-suffix* **by** *blast*
show $q \leq s\ p1$
using ‹$v \cdot (u \cdot v)\ ^{@}\ (k{-}1) \cdot q = p1$› **by** *auto*
qed
show $v \cdot u \leq s\ x$
using ‹$?s \leq s\ x$›[*unfolded pow-pos'*[*OF m-pos*] *rassoc*] *suf-extD* **by** *metis*
show $|q| \leq |v \cdot u|$
using *lenarg*[*OF* ‹$u \cdot v \cdot v \cdot u \cdot s' = q \cdot u \cdot v \cdot u$›] *lenarg*[*OF* ‹$ov \cdot s' = u$›] **by** *force*
qed
qed *auto*
qed

lemma *medium-overlap*:
assumes
len1: $|x| + |u| < |v \cdot (u \cdot v)^{@}k| + |(u \cdot v)^{@}m \cdot u|$ (**is** $|x| + |u| < |?p| + |?s|$)
and
len2: $|v \cdot (u \cdot v)^{@}k| + |(u \cdot v)^{@}m \cdot u| < |x| + |u \cdot v|$ (**is** $|?p| + |?s| < |x| + |u \cdot v|$)

shows *commutes* $\{u,v,x\}$
proof (*rule pref-suf-commute-all-commutes*)
show $|u \cdot v| \leq |x|$
using *len2* **unfolding** *pow-pos*[*OF k-pos*] **by** *force*
next
note *x-pref x-suf*
have $|?p| \leq |x|$
using *len2* **unfolding** *pow-pos*[*OF m-pos*] **by** *auto*
hence $?p \leq_p x$
using $\langle x \leq_p ?p \cdot x \rangle$ *pref-prod-long* **by** *blast*
hence $v \cdot (u \cdot v)^{@}(k-1) \cdot u \cdot v \cdot v \leq_p ?p \cdot x$
using $\langle x \leq_p ?p \cdot x \rangle$ **unfolding** *pow-pos'*[*OF k-pos*] *rassoc* **by** (*auto simp add*: *prefix-def*)

have $|?s| \leq |x|$
using *len2* **unfolding** *pow-pos*[*OF k-pos*] *pow-len lenmorph* **by** *auto*
hence $?s \leq_s x$
using *suf-prod-long*[*OF* $\langle x \leq_s x \cdot ?s \rangle$] **by** *blast*
then obtain p' **where** $p' \cdot u \cdot v \leq_p x$ **and** $p' \cdot ?s = x$
unfolding *pow-pos*[*OF m-pos*] **by** (*auto simp add*: *suffix-def*)

have $|p' \cdot u \cdot v| \leq |?p \cdot v|$
using *len1*[*folded* $\langle p' \cdot ?s = x \rangle$] **by** *force*

have $|v \cdot (u \cdot v)^{@}(k-1)| < |p'|$
using *len2*[*folded* $\langle p' \cdot ?s = x \rangle$] **unfolding** *pow-pos'*[*OF k-pos*] **by** *force*

from *less-imp-le*[*OF this*]
obtain p **where** $v \cdot (u \cdot v)^{@}(k-1) \cdot p = p'$
using *ruler-le*[*OF* $\langle ?p \leq_p x \rangle$ $\langle p' \cdot u \cdot v \leq_p x \rangle$,
unfolded pow-pos'[*OF k-pos*] *lassoc*, *THEN pref-cancel-right*, *THEN pref-cancel-right*]
unfolding *lenmorph* **by** (*auto simp add*: *prefix-def*)

have $|p| \leq |v|$
using $\langle v \cdot (u \cdot v)^{@}(k-1) \cdot p = p' \rangle$ $\langle |p' \cdot u \cdot v| \leq |?p \cdot v| \rangle$ **unfolding** *pow-pos'*[*OF k-pos*] **by** *force*

show $u \cdot v = v \cdot u$
proof (*rule uv-fac-uvv*)
show $p \cdot u \cdot v \leq_p u \cdot v \cdot v$
proof (*rule pref-cancel*[*of* $v \cdot (u \cdot v)^{@}(k-1)$], *rule ruler-le*)
show $(v \cdot (u \cdot v)\ ^{@}\ (k-1)) \cdot p \cdot u \cdot v \leq_p ?p \cdot x$
unfolding *lassoc* $\langle v \cdot (u \cdot v)^{@}(k-1) \cdot p = p' \rangle$[*unfolded lassoc*]
using $\langle p' \cdot u \cdot v \leq_p x \rangle$ $\langle x \leq_p ?p \cdot x \rangle$ **unfolding** *pow-pos'*[*OF k-pos*] **by** *force*
show $(v \cdot (u \cdot v)\ ^{@}\ (k-1)) \cdot u \cdot v \cdot v \leq_p (v \cdot (u \cdot v)\ ^{@}\ k) \cdot x$
unfolding *pow-pos'*[*OF k-pos*] *rassoc*
using $\langle v \cdot (u \cdot v)\ ^{@}\ k \leq_p x \rangle$ **by** (*auto simp add*: *prefix-def*)
show $|(v \cdot (u \cdot v)\ ^{@}\ (k-1)) \cdot p \cdot u \cdot v| \leq |(v \cdot (u \cdot v)\ ^{@}\ (k-1)) \cdot u \cdot v \cdot v|$
using $\langle v \cdot (u \cdot v)^{@}(k-1) \cdot p = p' \rangle$ $\langle |p' \cdot u \cdot v| \leq |?p \cdot v| \rangle$ **unfolding**

pow-pos′[*OF k-pos*] **by** *force*
qed

have $p \leq s\ x$
using ‹$p' \cdot ?s = x$›[*folded* ‹$v \cdot (u \cdot v)^{@}(k{-}1) \cdot p = p'$›] ‹$x \leq s\ x \cdot ?s$› *suf-cancel suf-extD* **by** *metis*

from *ruler-le*[*reversed*, *OF this* ‹$?s \leq s\ x$›, *unfolded pow-pos′*[*OF m-pos*] *rassoc*]
show $p \leq s\ (u \cdot v)\ ^{@}\ (m{-}1) \cdot u \cdot v \cdot u$
using ‹$|p| \leq |v|$› **unfolding** *lenmorph* **by** *auto*

show $(u \cdot v)\ ^{@}\ (m{-}1) \cdot u \cdot v \cdot u \in \langle\{u, v\}\rangle$
by (*simp add*: *gen-in hull-closed power-in*)

show $p \neq \varepsilon$
using ‹$|v \cdot (u \cdot v)^{@}(k{-}1)|\ < |p'|$› ‹$v \cdot (u \cdot v)^{@}(k{-}1) \cdot p = p'$› **by** *force*
qed
qed

thm
no-overlap
short-overlap
medium-overlap

end

thm
pref-suf-pers.no-overlap
pref-suf-pers.short-overlap
pref-suf-pers.medium-overlap
pref-suf-pers-large-overlap

0.2 Square interpretation

In this section fundamental description is given of (the only) possible $\{x, y\}$-interpretation of the square $x \cdot x$, where $|y| \leq |x|$. The proof is divided into several locales.

lemma *cover-not-disjoint*:
shows *primitive* $(\mathfrak{a}{\cdot}\mathfrak{b}{\cdot}\mathfrak{a}{\cdot}\mathfrak{b}{\cdot}\mathfrak{a}{\cdot}\mathfrak{b}{\cdot}\mathfrak{a})$ (**is** *primitive ?x*) **and**
primitive $(\mathfrak{a}{\cdot}\mathfrak{b})$ (**is** *primitive ?y*) **and**
$(\mathfrak{a}{\cdot}\mathfrak{b}{\cdot}\mathfrak{a}{\cdot}\mathfrak{b}{\cdot}\mathfrak{a}{\cdot}\mathfrak{b}{\cdot}\mathfrak{a}) \cdot (\mathfrak{a}{\cdot}\mathfrak{b}) \neq (\mathfrak{a}{\cdot}\mathfrak{b}) \cdot (\mathfrak{a}{\cdot}\mathfrak{b}{\cdot}\mathfrak{a}{\cdot}\mathfrak{b}{\cdot}\mathfrak{a}{\cdot}\mathfrak{b}{\cdot}\mathfrak{a})$
(**is** $?x \cdot ?y \neq ?y \cdot ?x$) **and**
$\varepsilon\ (\mathfrak{a}{\cdot}\mathfrak{b}{\cdot}\mathfrak{a}{\cdot}\mathfrak{b}{\cdot}\mathfrak{a}{\cdot}\mathfrak{b}{\cdot}\mathfrak{a}) \cdot (\mathfrak{a}{\cdot}\mathfrak{b}{\cdot}\mathfrak{a}{\cdot}\mathfrak{b}{\cdot}\mathfrak{a}{\cdot}\mathfrak{b}{\cdot}\mathfrak{a})\ (\mathfrak{b}{\cdot}\mathfrak{a}{\cdot}\mathfrak{b}{\cdot}\mathfrak{a}) \sim_{\mathcal{I}} [(\mathfrak{a}{\cdot}\mathfrak{b}{\cdot}\mathfrak{a}{\cdot}\mathfrak{b}{\cdot}\mathfrak{a}{\cdot}\mathfrak{b}{\cdot}\mathfrak{a}),(\mathfrak{a}{\cdot}\mathfrak{b}),(\mathfrak{a}{\cdot}\mathfrak{b}),(\mathfrak{a}{\cdot}\mathfrak{b}{\cdot}\mathfrak{a}{\cdot}\mathfrak{b}{\cdot}\mathfrak{a}{\cdot}\mathfrak{b}{\cdot}\mathfrak{a})]$
(**is** $\varepsilon\ ?x \cdot ?x\ ?s \sim_{\mathcal{I}} [?x, ?y, ?y, ?x]$)
unfolding *factor-interpretation-def*
by *primitivity-inspection+ force*

0.2.1 Locale: interpretation

locale *square-interp* =
— The basic set of assumptions
— The goal is to arrive at $ws = [x] \cdot [y]\ ^{@}\ k \cdot [x]$ including the description of the interpretation in terms of the first and the second occurrence of x in the interpreted square.
fixes *x y p s ws*
assumes
non-comm: $x \cdot y \neq y \cdot x$ **and**
prim-x: *primitive x* **and**
y-le-x: $|y| \leq |x|$ **and**
ws-lists: $ws \in lists\ \{x,y\}$ **and**
nconjug: $\neg\ x \sim y$ **and**
disj-interp: $p\ [x,x]\ s \sim_{\mathcal{D}}\ ws$

begin

lemma *interp*: $p\ (x \cdot x)\ s \sim_{\mathcal{I}}\ ws$
using *disj-interpD*[*OF disj-interp*] **by** *force*

lemma *disjoint*: $p1 \leq p\ [x,x] \Longrightarrow p2 \leq p\ ws \Longrightarrow p \cdot concat\ p1 \neq concat\ p2$
using *disj-interpD1*[*OF disj-interp*]**.**

interpretation *binary-code x y*
using *non-comm* **by** *unfold-locales*

lemmas *interpret-concat* = *fac-interpD*(*3*)[*OF interp*]

lemma *p-nemp*: $p \neq \varepsilon$
using *disjoint*[*of* ε ε] **by** *auto*

lemma *s-nemp*: $s \neq \varepsilon$
using *disjoint*[*of* [*x*,*x*] *ws*] *interpret-concat* **by** *force*

lemma *x-root*: $\varrho\ x = x$
using *prim-x* **by** *blast*

lemma *ws-nemp*: $ws \neq \varepsilon$
using *bin-fst-nemp fac-interp-nemp interp* **by** *blast*

lemma *hd-ws-lists*: $hd\ ws \in \{x,\ y\}$
using *lists-hd-in-set ws-lists ws-nemp* **by** *auto*

lemma *last-ws-lists*: $last\ ws \in \{x,\ y\}$
using *lists-hd-in-set*[*reversed*, *OF ws-nemp ws-lists*]**.**

lemma *kE*: **obtains** *k* **where** $[hd\ ws] \cdot [y]^{@}k \cdot [last\ ws] = ws$
proof–

from *list.collapse*[*OF ws-nemp*] *hd-word*
obtain *ws′* **where** *ws* = [*hd ws*] · *ws′*
by *metis*
hence |*hd ws*| ≤ |*x*|
using *two-elem-cases*[*OF lists-hd-in-set*[*OF ws-nemp ws-lists*]] *y-le-x* **by** *blast*
hence |*x*| ≤ |*concat ws′*|
using *lenarg*[*OF interpret-concat*, *unfolded lenmorph*]
unfolding *concat.simps emp-simps arg-cong*[*OF* ‹*ws* = [*hd ws*] · *ws′*›, *of* λ *x*. |*concat x*|, *unfolded concat-morph lenmorph*]
by *linarith*
hence *ws′* ≠ *ε*
using *nemp-len*[*OF bin-fst-nemp*] **by** *fastforce*
then obtain *mid-ws* **where** *ws′* = *mid-ws* · [*last ws*]
using ‹*ws* = [*hd ws*] · *ws′*› *append-butlast-last-id last-appendR* **by** *metis*
note ‹*ws* = [*hd ws*] · *ws′*›[*unfolded this*]
fac-interpD[*OF interp*]
obtain *p′* **where** [*symmetric*]:*p* · *p′* = *hd ws* **and** *p′* ≠ *ε*
using *spref-exE*[*OF* ‹*p* <*p hd ws*›]**.**
obtain *s′* **where** [*symmetric*]: *s′*· *s* = *last ws* **and** *s′* ≠ *ε*
using *spref-exE*[*reversed*, *OF* ‹*s* <*s last ws*›]**.**
have *p′* · *concat mid-ws* · *s′* = *x* · *x*
using ‹*ws* = [*hd ws*] · *mid-ws* · [*last ws*]›[*unfolded* ‹*hd ws* = *p* · *p′*› ‹*last ws* = *s′*· *s*›]
‹*p* · (*x* · *x*) · *s* = *concat ws*› **by** *simp*
note *over* = *prim-overlap-sqE*[*OF prim-x*, *folded this*]
have *mid-ws* ∈ *lists* {*x*,*y*}
using ‹*ws* = [*hd ws*] · *ws′*› ‹*ws′* = *mid-ws* · [*last ws*]› *append-in-lists-conv ws-lists* **by** *metis*
have *x* ∉ *set mid-ws*
proof
assume *x* ∈ *set mid-ws*
then obtain *r q* **where** *concat mid-ws* = *r* · *x* · *q*
using *concat.simps*(*2*) *concat-morph in-set-conv-decomp-first* **by** *metis*
have (*p′* · *r*) · *x* · (*q* · *s′*) = *x* · *x*
using ‹*p′* · *concat mid-ws* · *s′* = *x* · *x*›[*unfolded* ‹*concat mid-ws* = *r* · *x* · *q*›]
unfolding *rassoc***.**
from *prim-overlap-sqE*[*OF prim-x this*]
show *False*
using ‹*p′* ≠ *ε*› ‹*s′* ≠ *ε*› **by** *blast*
qed
hence *mid-ws* ∈ *lists* {*y*}
using ‹*mid-ws* ∈ *lists* {*x*,*y*}› **by** *force*
from *that sing-lists-exp*[*OF this*]
show *thesis*
using ‹*ws* = [*hd ws*] · *mid-ws* · [*last ws*]› **by** *metis*
qed

lemma *l-mE*: **obtains** *m u v l* **where** (*hd ws*)·*y*[@]*m*·*u* = *p*·*x* **and** *v* · *y*[@]*l* · (*last ws*) = *x* · *s* **and**

$u{\cdot}v = y$ $u \neq \varepsilon$ $v \neq \varepsilon$ **and** $x \cdot (v \cdot u) \neq (v \cdot u) \cdot x$
proof−
note *fac-interpD*[*OF interp*]
obtain k **where** $[hd\ ws] \cdot [y]^{@}k \cdot [last\ ws] = ws$
using *kE*.
from *arg-cong*[*OF this*, *of concat*, *folded interpret-concat*, *unfolded concat-morph rassoc concat-sing′ concat-sing-pow*]
have $hd\ ws \cdot y^{@}k \cdot last\ ws = p \cdot x \cdot x \cdot s$.
have $|hd\ ws| \leq |p \cdot x|$
unfolding *lenmorph* **by** (*rule two-elem-cases*[*OF hd-ws-lists*])
(*use dual-order.trans*[*OF le-add2 y-le-x*] *le-add2*[*of* $|x|$] **in** *fast*)+
from *eqd*[*OF - this*]
obtain *ya* **where** $hd\ ws \cdot ya = p \cdot x$
using ‹$hd\ ws \cdot y^{@}k \cdot last\ ws = p \cdot x \cdot x \cdot s$› **by** *auto*
have $|last\ ws| \leq |x|$
unfolding *lenmorph* **using** *dual-order.trans last-ws-lists y-le-x* **by** *auto*
hence $|last\ ws| < |x \cdot s|$
unfolding *lenmorph* **using** *nemp-len*[*OF s-nemp*] **by** *linarith*
from *eqd*[*reversed*, *OF - less-imp-le*[*OF this*]]
obtain *yb* **where** $yb \cdot (last\ ws) = x \cdot s$
using ‹$(hd\ ws) \cdot y^{@}k \cdot (last\ ws) = p \cdot x \cdot x \cdot s$› *rassoc* **by** *metis*
hence $yb \neq \varepsilon$
using *s-nemp* ‹$|last\ ws| < |x \cdot s|$› **by** *force*
have $ya \cdot yb = y^{@}k$
using ‹$(hd\ ws) \cdot y^{@}k \cdot (last\ ws) = p \cdot x \cdot x \cdot s$›[*folded* ‹$yb \cdot (last\ ws) = x \cdot s$›, *unfolded lassoc cancel-right*, *folded* ‹$(hd\ ws) \cdot ya = p \cdot x$›, *unfolded rassoc cancel*, *symmetric*].
from *pref-mod-pow′*[*OF sprefI*[*OF prefI*[*OF this*]], *folded this*]
obtain $m\ u$ **where** $m < k$ **and** $u <p\ y$ **and** $y^{@}m \cdot u = ya$
using ‹$yb \neq \varepsilon$› **by** *blast*
have $y^{@}m \cdot u \cdot (u^{-1>}y) \cdot y^{@}(k - m - 1) = y^{@}m \cdot y \cdot y^{@}(k - m - 1)$
using ‹$u <p\ y$› **by** (*auto simp add*: *prefix-def*)
also have $\ldots = y^{@}(m + 1 + (k{-}m{-}1))$
using *rassoc add-exps pow-1* **by** *metis*
also have $\ldots = y^{@}k$
using ‹$m < k$› **by** *auto*
finally obtain $l\ v$ **where** $u{\cdot}v = y$ **and** $y^{@}m \cdot u \cdot v \cdot y^{@}l = y^{@}k$
using ‹$u <p\ y$› *lq-pref* **by** *blast*
have *concat* $([hd\ ws]{\cdot}[y]\ ^{@}\ m) = hd\ ws \cdot y\ ^{@}\ m$
by *simp*
have $v \neq \varepsilon$
using ‹$u <p\ y$› ‹$u{\cdot}v = y$› **by** *force*
have $[hd\ ws] \cdot [y]\ ^{@}\ m \leq p\ ws$
using ‹$[hd\ ws] \cdot [y]^{@}k \cdot [last\ ws] = ws$›[*folded pop-pow*[*OF less-imp-le*[*OF* ‹$m < k$›]]] **by** *fastforce*
from *disjoint*[*OF - this*, *of* $[x]$, *unfolded* ‹*concat* $([hd\ ws] \cdot [y]\ ^{@}\ m) = hd\ ws \cdot y\ ^{@}\ m$›]
have $u \neq \varepsilon$
using ‹$(hd\ ws) \cdot ya = p \cdot x$›[*folded* ‹$y^{@}m \cdot u = ya$›] *s-nemp* **by** *force*

```
      have x · (v · u) ≠ (v · u) · x
  proof
    assume x · v · u = (v · u) · x
    from comm-primroots'[OF  bin-fst-nemp suf-nemp[OF ‹u ≠ ε›] this, unfolded
x-root]
    have x = ϱ (v · u).
    thus False
      using ‹u · v = y›  nconjug y-le-x
     using conjugI' nle-le pref-same-len primroot-emp primroot-len-le primroot-pref
swap-len by metis
  qed
  with that[of m u u^-1> y l, OF ‹hd ws · ya = p · x›[folded ‹y @ m · u = ya›],
folded ‹yb · last ws = x · s› ‹u · v = y›,
     unfolded lq-triv lassoc cancel-right, OF - - ‹u ≠ ε› ‹v ≠ ε› this[unfolded lassoc]]
  show thesis
    using ‹y @ m · u · v · y @ l = y @ k›[folded ‹ya · yb = y @ k› ‹y @ m · u =
ya›, unfolded rassoc cancel, folded ‹u · v = y›] by blast
qed

lemma last-ws: last ws = x
proof(rule ccontr)
  assume last ws ≠ x
  hence last ws = y
    using last-ws-lists by blast
  obtain l m u v where (hd ws)·y@m·u = p·x and v · y@l · (last ws) = x · s and
    u·v = y and u ≠ ε and v ≠ ε and x · v · u ≠ (v · u) · x
    using l-mE by metis
  note y-le-x[folded ‹u · v = y›,unfolded swap-len[of u]]

  from ‹v · y@l · (last ws) = x · s›[unfolded ‹last ws = y›, folded ‹u · v = y›]
  have x ≤p (v · u)@Suc l · v
    unfolding pow-Suc' rassoc using append-eq-appendI prefix-def shift-pow by
metis
  moreover have (v · u) @ Suc l · v ≤p (v · u) · (v · u) @ Suc l · v
    unfolding lassoc pow-comm[symmetric] using rassoc by blast
  ultimately have x ≤p (v · u) · x
    using pref-keeps-per-root by blast

  thus False
  proof (cases m = 0)
    assume m ≠ 0
    have v · u ≤s x
      using ‹(hd ws)·y@m·u = p·x›[folded ‹u · v = y›, unfolded pow-pos'[OF ‹m ≠
0›[unfolded neq0-conv]] rassoc]
        suf-extD[THEN suf-prod-long[OF - ‹|v · u| ≤ |x|›], of p hd ws · (u · v) @
(m−1) · u, unfolded rassoc] by simp
    have [symmetric]: (v · u) · x = x · (v · u)
      using root-suf-comm'[OF  ‹x ≤p (v · u) · x› ‹(v · u) ≤s x›].
    thus False
```

```
    using ‹x · v · u ≠ (v · u) · x› by blast
  next
    assume m = 0
    thus False
    proof (cases hd ws = y)
      assume hd ws = y
      have p · (x · x) · s = y @ Suc (Suc (Suc (m+l)))
        unfolding rassoc ‹v · y @ l · (last ws) = x · s›[unfolded ‹last ws = y›,
symmetric] power-Suc2
        unfolding lassoc ‹(hd ws)·y @ m·u = p·x›[unfolded ‹hd ws = y›, symmetric]
          ‹u · v = y›[symmetric]
        by comparison
      have ϱ x ∼ ϱ y
      proof (rule fac-two-conjug-primroot')
        show x ≠ ε and y ≠ ε using bin-fst-nemp bin-snd-nemp.
        show x · x ≤f  y @ Suc (Suc (Suc (m+l)))
         using facI[of x·x p s,unfolded ‹p · (x · x) · s = y @ Suc (Suc (Suc (m+l)))›].
        show x · x ≤f x @ 2
          unfolding pow-two by blast
        show |x| + |y| ≤ |x · x|
          using y-le-x unfolding lenmorph by auto
      qed
      thus False
        unfolding x-root  using nconjug y-le-x
        by (metis conjug-len long-pref primroot-pref)
    next
      assume hd ws ≠ y
      hence hd ws = x
        using hd-ws-lists by auto

      have x ≤s x · u
        using  ‹(hd ws)·y @ m·u = p·x›[unfolded ‹m = 0› ‹hd ws = x› pow-zero
emp-simps]
        by (simp add: suffix-def)
      have v · u ≤p x
        using ‹x ≤p (v · u) · x› y-le-x[folded ‹u · v = y›,unfolded swap-len[of u]]
          pref-prod-long by blast
      hence |v · u| < |x|
        using  nconjug conjugI[OF - ‹u · v = y›, of x] ‹|v · u| ≤ |x|›
          le-neq-implies-less pref-same-len by blast
      have u · v = v · u
      proof (rule pref-suf-pers-short[reversed])
        from ‹x ≤p (v · u) @ Suc l · v›
        show x ≤p ((v · u) · v) · (u · v) @ l
          by comparison
        show (u · v) @ l ∈ ⟨{v, u}⟩
          by blast
      qed fact+
      from pref-extD[OF ‹v · u ≤p x›[folded ‹u · v = v · u›]]
```

```
    have x · u = u · x
      using ‹x ≤s x · u› suf-root-pref-comm by blast
    with comm-trans[OF this ‹u · v = v · u›[symmetric] ‹u ≠ ε›]
    have x · (v · u) = (v · u) · x
      using comm-prod by blast
    thus False
      using ‹x · v · u ≠ (v · u) · x› by blast
  qed
 qed
qed

lemma rev-square-interp:
 square-interp (rev x) (rev y) (rev s) (rev p) (rev (map rev ws))
proof (unfold-locales)
 show rev (map rev ws) ∈ lists {rev x, rev y}
  using ws-lists by force
 show |rev y| ≤ |rev x|
  using y-le-x by simp
 show ¬ (rev x) ∼ (rev y)
  by (simp add: conjug-rev-conv nconjug)
 show primitive (rev x)
  using prim-x
  by (simp-all add: prim-rev-iff)
 show (rev s) [rev x, rev x] (rev p) ∼𝒟 (rev (map rev ws))
 proof
  show (rev s) (concat [rev x, rev x]) (rev p) ∼ℐ rev (map rev ws)
   using interp rev-fac-interp by fastforce
  show ⋀p1 p2. p1 ≤p [rev x, rev x] ⟹ p2 ≤p rev (map rev ws) ⟹ rev s ·
concat p1 ≠ concat p2
  proof
   fix p1′ p2′ assume p1′ ≤p [rev x, rev x] and p2′ ≤p rev (map rev ws) and
rev s · concat p1′ = concat p2′
   obtain p1 p2 where p1′ · p1 = [rev x, rev x] and p2′·p2 = rev (map rev ws)
    using ‹p1′ ≤p [rev x, rev x]› ‹p2′ ≤p rev (map rev ws)› by (auto simp add:
prefix-def)
   hence rev s · (concat p1′ · concat p1) · rev p = concat p2′ · concat p2
    unfolding concat-morph[symmetric] using ‹(rev s) (concat[rev x,rev x]) (rev
p) ∼ℐ rev (map rev ws)›
      fac-interpD(3) by force
   from this[unfolded lassoc, folded ‹rev s · concat p1′ = concat p2′›, unfolded
rassoc cancel]
   have concat p1 · rev p = concat p2.
   hence p · (concat (rev  (map rev p1))) = concat (rev (map rev p2))
    using rev-append rev-concat rev-map rev-rev-ident by metis
   have rev  (map rev p1) ≤p [x,x]
     using arg-cong[OF ‹p1′ · p1 = [rev x, rev x]›, of λ x. rev (map rev x),
unfolded map-append rev-append]
    by fastforce
   have rev  (map rev p2) ≤p ws
```

using *arg-cong*[*OF* ‹*p2'*·*p2* = *rev* (*map rev ws*)›, *of* λ *x*. *rev* (*map rev x*), *unfolded map-append rev-append rev-map*
rev-rev-ident map-rev-involution, *folded rev-map*] **by** *blast*
from *disjoint*[*OF* ‹*rev* (*map rev p1*) ≤*p* [*x*,*x*]› ‹*rev* (*map rev p2*) ≤*p ws*›]
show *False*
using ‹*p* · (*concat* (*rev* (*map rev p1*))) = *concat* (*rev* (*map rev p2*))› **by** *blast*
qed
qed
show *rev x* · *rev y* ≠ *rev y* · *rev x*
using *non-comm* **unfolding** *comm-rev-iff*.
qed

lemma *hd-ws*: *hd ws* = *x*
using *square-interp.last-ws*[*reversed*, *OF rev-square-interp*]
unfolding *hd-map*[*OF ws-nemp*]
by *simp*

lemma *p-pref*: *p* <*p x*
using *fac-interpD*(*1*) *hd-ws interp* **by** *auto*

lemma *s-suf*: *s* <*s x*
using *fac-interpD*(*2*) *last-ws interp* **by** *auto*

end

0.2.2 Locale with additional parameters

locale *square-interp-plus* = *square-interp* +
fixes *l m u v*
assumes *fst-x*: *x* · *y*[@]*m* · *u* = *p* · *x* **and**
snd-x: *v* · *y*[@]*l* · *x* = *x* · *s* **and**
uv-y: *u* · *v* = *y* **and**
u-nemp: *u* ≠ *ε* **and** *v-nemp*: *v* ≠ *ε* **and**
vu-x-non-comm: *x* · (*v* · *u*) ≠ (*v* · *u*) · *x*
begin

interpretation *binary-code x y*
using *non-comm* **by** *unfold-locales*

lemma *rev-square-interp-plus*: *square-interp-plus* (*rev x*) (*rev y*) (*rev s*) (*rev p*) (*rev* (*map rev ws*)) *m l* (*rev v*) (*rev u*)
proof−
note *fac-interpD*[*OF interp*, *unfolded hd-ws last-ws*]
note ‹*s* <*s x*›[*unfolded strict-suffix-to-prefix*]
note ‹*p* <*p x*›[*unfolded spref-rev-suf-iff*]

interpret *i*: *square-interp* (*rev x*) (*rev y*) (*rev s*) (*rev p*) (*rev* (*map rev ws*))

using *rev-square-interp*.
show *?thesis*
by *standard*
(*simp-all del*: *rev-append add*: *rev-pow*[*symmetric*] *rev-append*[*symmetric*],
simp-all add: *fst-x snd-x uv-y v-nemp u-nemp vu-x-non-comm*[*symmetric*, *unfolded rassoc*])
qed

Exactly one of the exponents is zero: impossible.

Uses lemma $[\![\leq_p ?x\ (?v \cdot ?x);\ |?v \cdot ?u| < |?x|;\ \leq_s ?x\ (?r \cdot ?u \cdot ?v \cdot ?u);\ ?r \in \langle\{?u, ?v\}\rangle]\!] \Longrightarrow ?u \cdot ?v = ?v \cdot ?u$ and exploits the symmetric interpretation.

lemma *fst-exp-zero*: **assumes** $m = 0$ **and** $0 < l$ **shows** *False*
proof (*rule notE*[*OF vu-x-non-comm*])
note *y-le-x*[*folded uv-y*, *unfolded swap-len*[*of u*]]
have $x \leq p\ (v \cdot (u \cdot v)\ ^{@}\ l) \cdot x$
unfolding *rassoc* **using** *snd-x*[*folded uv-y*] **by** *blast*
have $v \cdot (u \cdot v)\ ^{@}\ l \neq \varepsilon$
using *v-nemp* **by** *force*
obtain *exp* **where** $x \leq p\ (v \cdot (u \cdot v)\ ^{@}\ l)^{@}exp\ 0 < exp$
using *per-root-powE*[*OF per-rootI*[*OF* ‹$x \leq p\ (v \cdot (u \cdot v)\ ^{@}\ l) \cdot x$› ‹$v \cdot (u \cdot v)\ ^{@}\ l \neq \varepsilon$›], *of thesis*] **by** *blast*

have $x \leq s\ x \cdot u$
using *fst-x*[*unfolded* ‹$m = 0$› *pow-zero emp-simps*] **by** (*simp add*: *suffix-def*)
have $((v \cdot u) \cdot v) \cdot ((u \cdot v)^{@}(l-1)) \cdot (v \cdot (u \cdot v)\ ^{@}\ l)^{@}(exp-1) = (v \cdot (u \cdot v)\ ^{@}\ l)^{@}exp$
(**is** $((v \cdot u) \cdot v) \cdot ?suf = (v \cdot (u \cdot v)\ ^{@}\ l)^{@}exp$)
using ‹$0 < l$› ‹$0 < exp$› **by** *comparison*
have $v \cdot u \leq p\ x$
using *pref-prod-longer*[*OF* ‹$x \leq p\ (v \cdot (u \cdot v)\ ^{@}\ l) \cdot x$›[*unfolded rassoc*] - ‹$|v \cdot u| \leq |x|$›]
unfolding *pow-pos*[*OF* ‹$0 < l$›] *rassoc* **by** *blast*
hence $|v \cdot u| < |x|$
using *nconjug conjugI*[*OF* - *uv-y*, *of x*] ‹$|v \cdot u| \leq |x|$›
le-neq-implies-less pref-same-len **by** *blast*
have $u \cdot v = v \cdot u$
proof (*rule pref-suf-pers-short*[*reversed*])
show $x \leq p\ ((v \cdot u) \cdot v) \cdot ?suf$
unfolding ‹$((v \cdot u) \cdot v) \cdot ?suf = (v \cdot (u \cdot v)\ ^{@}\ l)^{@}exp$› **by** *fact*
show $((u \cdot v)^{@}(l-1)) \cdot (v \cdot (u \cdot v)\ ^{@}\ l)^{@}(exp-1) \in \langle\{v,u\}\rangle$
by (*simp add*: *gen-in hull-closed power-in*)
qed *fact+*
show $x \cdot v \cdot u = (v \cdot u) \cdot x$
using *root-suf-comm*[*OF* - ‹$x \leq s\ x \cdot u$›] *pref-keeps-per-root comm-trans*[*OF* ‹$u \cdot v = v \cdot u$›[*symmetric*] - *u-nemp*, *symmetric*] ‹$v \cdot u \leq p\ x$› *comm-prod prefI*
by *metis*
qed

lemma *snd-exp-zero*: **assumes** $0 < m$ **and** $l = 0$ **shows** *False*
using *square-interp-plus.fst-exp-zero*[*OF rev-square-interp-plus*, *reversed*, *rotated*, *OF assms*].

Both exponents positive: impossible

lemma *both-exps-pos*: **assumes** $0 < m$ **and** $0 < l$ **shows** *False*
proof–
note *fac-interpD*[*OF interp*, *unfolded hd-ws last-ws*]
have $|p| \leq |x|$ **and** $|s| \leq |x|$
using *pref-len*[*OF sprefD1*[*OF* ‹$p <p x$›]] *suf-len*[*OF ssufD1*[*OF* ‹$s <s x$›]].

have $x \leq p\ (v \cdot (u \cdot v)^{@} l) \cdot x$
(**is** $x \leq p$ *?pref* $\cdot\ x$)
using *snd-x*[*folded uv-y*] **by** *force*
moreover have $x \leq s\ x \cdot ((u \cdot v)^{@} m \cdot u)$
(**is** $x \leq s\ x \cdot$ *?suf*)
using *fst-x*[*folded uv-y*] **by** *force*

ultimately interpret *pref-suf-pers x u v l m*
using ‹$0 < l$› ‹$0 < m$› **by** *unfold-locales*

have *?pref* $\leq p\ x$
using *snd-x*[*folded uv-y rassoc*, *symmetric*] *eqd*[*reversed*, *OF* - ‹$|s| \leq |x|$›] **by** *blast*
have *?suf* $\leq s\ x$
using *fst-x*[*folded uv-y*, *symmetric*] *eqd*[*OF* - ‹$|p| \leq |x|$›] **by** *blast*

have *in-particular*: *commutes* $\{u,v,x\} \Longrightarrow x \cdot (v \cdot u) = (v \cdot u) \cdot x$
unfolding *commutes-def* **by** (*rule comm-prod*) *blast*+

— Case analysis based on (slightly modified) lemmas for covered x square.

note *no-overlap-comm* = *no-overlap*[*THEN in-particular*] **and**
short-overlap-comm = *short-overlap*[*THEN in-particular*] **and**
medium-overlap-comm = *medium-overlap*[*THEN in-particular*] **and**
large-overlap-conjug = *pref-suf-pers-large-overlap*[*OF* ‹*?pref* $\leq p\ x$› ‹*?suf* $\leq s\ x$›, *of* $v \cdot u$]

consider
(*no-overlap*) $|?pref| + |?suf| \leq |x|$ |
(*short-overlap*) $|x| < |?pref| + |?suf| \wedge |?pref| + |?suf| \leq |x| + |u|$ |
(*medium-overlap*) $|x| + |u| < |?pref| + |?suf| \wedge |?pref| + |?suf| < |x| + |u \cdot v|$ |
(*large-overlap*) $|x| + |v \cdot u| \leq |?pref| + |?suf|$
unfolding *swap-len*[*of v*] **by** *linarith*
thus *False*
proof (*cases*)
case *no-overlap*

then show *False*
using *no-overlap-comm vu-x-non-comm* ‹*0 < l*› ‹*0 < m*› **by** *blast*
next
case *short-overlap*
then show *False*
using *short-overlap-comm vu-x-non-comm* **by** *blast*
next
case *medium-overlap*
then show *False*
using *medium-overlap-comm vu-x-non-comm* **by** *blast*
next
case *large-overlap*
show *False*
thm *large-overlap-conjug nconjug*
proof (*rule notE*[*OF vu-x-non-comm*], *rule large-overlap-conjug*[*OF - - large-overlap*])
have $(u \cdot v) \ ^{@} \ (l-1) \leq p \ (u \cdot v) \ ^{@} \ Suc \ (l-1)$
using *pref-pow-ext* **by** *blast*
thus $v \cdot (u \cdot v) \ ^{@} \ l \leq p \ (v \cdot u) \cdot v \cdot (u \cdot v) \ ^{@} \ l$
unfolding *pow-pos*[*OF* ‹*0 < l*›] *pow-Suc rassoc pref-cancel-conv*.
show $(u \cdot v) \ ^{@} \ m \cdot u \leq s \ ((u \cdot v) \ ^{@} \ m \cdot u) \cdot v \cdot u$
by *comparison*
qed
qed
qed

thm *suf-cancel-conv*

end

0.2.3 Back to the main locale

context *square-interp*

begin

definition u **where** $u = x^{-1>}(p \cdot x)$
definition v **where** $v = (x \cdot s)^{<-1}x$

lemma *cover-xyx*: $ws = [x,y,x]$ **and** *vu-x-non-comm*: $x \cdot (v \cdot u) \neq (v \cdot u) \cdot x$ **and** *uv-y*: $u \cdot v = y$ **and**
px-xu: $p \cdot x = x \cdot u$ **and** *vx-xs*: $v \cdot x = x \cdot s$ **and** *u-nemp*: $u \neq \varepsilon$ **and** *v-nemp*: $v \neq \varepsilon$
proof−
obtain k **where** *ws*: $[x] \cdot [y]^{@}k \cdot [x] = ws$
using *kE*[*unfolded hd-ws last-ws*].
obtain $m \ u' \ v' \ l$ **where** $x \cdot y \ ^{@} \ m \cdot u' = p \cdot x$ **and** $v' \cdot y \ ^{@} \ l \cdot x = x \cdot s$ **and** $u' \cdot v' = y$
and $u' \neq \varepsilon$ **and** $v' \neq \varepsilon$ **and** $x \cdot v' \cdot u' \neq (v' \cdot u') \cdot x$
using *l-mE*[*unfolded hd-ws last-ws*].

then interpret *square-interp-plus x y p s ws l m u′ v′*
by (*unfold-locales*)
have $m = 0$ **and** $l = 0$ **and** $y \neq \varepsilon$
using *both-exps-pos snd-exp-zero fst-exp-zero* ‹$u' \cdot v' = y$› ‹$u' \neq \varepsilon$› **by** *blast+*
have $u' = u$
unfolding *u-def*
using *conjug-lq*[*OF fst-x*[*unfolded* ‹$m = 0$› *pow-zero emp-simps*, *symmetric*]].
have $v' = v$
unfolding *v-def*
using *conjug-lq*[*reversed*, *OF snd-x*[*unfolded* ‹$l = 0$› *pow-zero emp-simps*, *symmetric*]].
have $x \cdot y \,^{@}\, m \cdot (u' \cdot v') \cdot y^{@}l \cdot x = concat\ ws$
unfolding *interpret-concat*[*symmetric*] **using** *fst-x snd-x* **by** *force*
from *this*[*folded ws*, *unfolded* ‹$u' \cdot v' = y$› ‹$m = 0$› ‹$l = 0$› *pow-zero emp-simps*]
have $k = 1$
unfolding *eq-pow-exp*[*OF* ‹$y \neq \varepsilon$›, *of k 1*, *symmetric*] *pow-1 concat-morph concat-pow*
by *simp*
from *ws*[*unfolded this pow-1*]
show $ws = [x,y,x]$ **by** *simp*
show $u \cdot v = y$
unfolding ‹$u' = u$›[*symmetric*] ‹$v' = v$›[*symmetric*] **by** *fact+*
show $p \cdot x = x \cdot u$
using ‹$x \cdot y \,^{@}\, m \cdot u' = p \cdot x$›[*unfolded* ‹$m = 0$› ‹$u' = u$› *pow-zero emp-simps*, *symmetric*].
show $v \cdot x = x \cdot s$
using ‹$v' \cdot y \,^{@}\, l \cdot x = x \cdot s$›[*unfolded* ‹$l = 0$› ‹$v' = v$› *pow-zero emp-simps*].
show $x \cdot (v \cdot u) \neq (v \cdot u) \cdot x$
using ‹$x \cdot v' \cdot u' \neq (v' \cdot u') \cdot x$›[*unfolded* ‹$u' = u$› ‹$v' = v$›].
show $u \neq \varepsilon$ **and** $v \neq \varepsilon$
using ‹$u' \neq \varepsilon$› ‹$v' \neq \varepsilon$› **unfolding** ‹$u' = u$› ‹$v' = v$›.
qed

lemma *cover*: $x \cdot y \cdot x = p \cdot x \cdot x \cdot s$
using *interpret-concat cover-xyx* **by** *auto*

lemma *conjug-facs*: $\varrho\ u \sim \varrho\ v$
proof−
note *sufI*[*OF px-xu*]
have $u \neq \varepsilon$
using *p-nemp px-xu* **by** *force*
obtain *expu* **where** $x <_s u^{@}\ expu\ 0 < expu$
using *per-root-powE*[*reversed*, *OF per-rootI*[*reversed*, *OF* ‹ $x \leq_s x \cdot u$› ‹$u \neq \varepsilon$›]].
hence $x \leq_f u^{@}\ expu$
using *ssufD1* **by** *blast*

note *prefI*[*OF vx-xs*[*symmetric*]]
have $v \neq \varepsilon$

```
    using s-nemp vx-xs by force
  obtain expv where x <p v^@expv 0 < expv
    using per-root-powE[OF per-rootI[OF ‹x ≤p v · x› ‹v ≠ ε›]].
  hence x ≤f v^@expv  by blast

  show ϱ u ∼ ϱ v
   proof(rule fac-two-conjug-primroot'[OF ‹x ≤f u^@expu› ‹x ≤f v^@expv› ‹u ≠ ε›
‹v ≠ ε›])
    show |u| + |v| ≤ |x|
      using y-le-x[folded uv-y, unfolded lenmorph] by fastforce
  qed
qed

term square-interp.v

— We have a detailed information about all words
lemma bin-sq-interpE: obtains r t m k l
  where (t · r)^@k = u and  (r · t)^@l = v and
    (r · t)^@m · r = x and (t · r)^@k · (r · t)^@ l = y
    and (r · t)^@k = p and  (t · r)^@ l = s  and r · t ≠ t · r and
    0 < k and 0 < m and 0 < l
proof–

  obtain r t k m where (r · t)^@ k = p and (t · r)^@ k = u and (r · t)^@m · r = x
and
    t ≠ ε and 0 < k  and primitive (r · t)
    using conjug-eq-primrootE[OF px-xu p-nemp].
  have t · r = ϱ u
   using  prim-conjug[OF ‹primitive (r · t)›, THEN primroot-unique[OF u-nemp],
OF conjugI' ‹(t · r)^@k = u›[symmetric]]..

  have 0 < m
  proof (rule ccontr)
    assume ¬ 0 < m
    hence x = r using ‹(r · t)^@m · r = x› by simp
    show False
      using ‹0 < k› ‹(r · t) ^@ k = p› ‹x = r› comp-pows-pref-zero p-pref by blast
  qed


  from ‹(r · t)^@m · r = x›[unfolded pow-pos[OF ‹0 < m›]]
  have r · t ≤p x
    by auto

  have r · t = ϱ v
  proof (rule ruler-eq-len[of ϱ v x r · t, symmetric])
    have |ϱ v| ≤ |x|
      unfolding conjug-len[OF conjug-facs, symmetric] ‹t · r = ϱ u›[symmetric]
      unfolding ‹(r · t)^@m · r = x›[symmetric] pow-pos[OF ‹0 < m›]
```

lenmorph pow-len **by** *auto*
from *ruler-le*[*OF - - this*, *of* $v \cdot x$]
show $\varrho\ v \leq_p x$
using *vx-xs prefI prefix-prefix primroot-pref v-nemp* **by** *metis*
show $r \cdot t \leq_p x$ **by** *fact*
show $|\varrho\ v| = |r \cdot t|$
unfolding *conjug-len*[*OF conjug-facs*, *symmetric*, *folded* ‹$t \cdot r = \varrho\ u$›] *lenmorph*
by *simp*
qed

then obtain l **where** $(r \cdot t)^{@}\ l = v$ **and** $0 < l$
using *primroot-expE v-nemp* **by** *metis*

have $(t \cdot r)^{@}\ l = s$
using *vx-xs*[*folded* ‹$(r \cdot t)^{@} m \cdot r = x$› ‹$(r \cdot t)^{@}\ l = v$›, *unfolded lassoc pows-comm*[*of - - m*],
unfolded rassoc cancel, *unfolded shift-pow cancel*].

have $r \cdot t \neq t \cdot r$
proof
assume $r \cdot t = t \cdot r$
hence *aux*: $r \cdot (t \cdot r)^{@}\ e = (t \cdot r)^{@}\ e \cdot r$ **for** e
by *comparison*
have $x \cdot (v \cdot u) = (v \cdot u) \cdot x$
unfolding ‹$(t \cdot r)^{@}\ k = u$›[*symmetric*] ‹$(r \cdot t)^{@}\ l = v$›[*symmetric*]
unfolding ‹$(r \cdot t)^{@} m \cdot r = x$›[*symmetric*] *add-exps*[*symmetric*] ‹$r \cdot t = t \cdot r$› *aux rassoc*
unfolding *lassoc cancel-right add-exps*[*symmetric*]
by (*simp add*: *add.commute*)
thus *False*
using *vu-x-non-comm* **by** *blast*
qed

show *thesis*
using *that*[*OF* ‹$(t \cdot r)^{@}\ k = u$› ‹$(r \cdot t)^{@}\ l = v$› ‹$(r \cdot t)^{@} m \cdot r = x$›
uv-y[*folded* ‹$(t \cdot r)^{@}\ k = u$› ‹$(r \cdot t)^{@}\ l = v$›] ‹$(r \cdot t)^{@}\ k = p$› ‹$(t \cdot r)^{@}\ l = s$› ‹$r \cdot t \neq t \cdot r$›
‹$0 < k$› ‹$0 < m$› ‹$0 < l$›].
qed

end

0.2.4 Locale: Extendable interpretation

Further specification follows from the assumption that the interpretation is extendable, that is, the covered $x \cdot x$ is a factor of a word composed of $\{x, y\}$. Namely, u and v are then conjugate by x.

locale *square-interp-ext* = *square-interp* +
assumes *p-extend*: $\exists\ pe.\ pe \in \langle\{x,y\}\rangle \wedge p \leq_s pe$ **and**

s-extend: $\exists\ se.\ se \in \langle\{x,y\}\rangle \wedge s \leq p\ se$

begin

lemma *s-pref-y*: $s \leq p\ y$
proof–
obtain *sy ry eu ev ex*
where $(ry \cdot sy)^{@} eu = u$ **and** $(sy \cdot ry)^{@}\ ev = v$ **and**
$(sy \cdot ry)^{@}\ eu = p$ **and** $(ry \cdot sy)^{@}\ ev = s$ **and**
$(sy \cdot ry)^{@}\ ex \cdot sy = x$ **and** $sy \cdot ry \neq ry \cdot sy$ **and**
$0 < eu$ **and** $0 < ev$ **and** $0 < ex$
using *bin-sq-interpE*.

obtain *se* **where** $se \in \langle\{x,y\}\rangle$ **and** $s \leq p\ se$
using *s-extend* **by** *blast*
hence $se \neq \varepsilon$ **using** *s-nemp* **by** *force*

from ‹$(sy \cdot ry)^{@}\ ex \cdot sy = x$›
have $sy \cdot ry \leq p\ x$
unfolding *pow-pos*[*OF* ‹$0 < ex$›] *rassoc* **by** *force*

have $x \leq p\ se \vee y \leq p\ se$
using ‹$se \neq \varepsilon$› *hull.cases*[*OF* ‹$se \in \langle\{x,y\}\rangle$›, *of* $x \leq p\ se \vee y \leq p\ se$]
prefix-append triv-pref two-elem-cases **by** *blast*
moreover have $\neg\ x \leq p\ se$
proof
assume $x \leq p\ se$
from *ruler-eq-len*[*of* $sy \cdot ry\ se\ ry \cdot sy$, *OF pref-trans*[*OF* ‹$sy \cdot ry \leq p\ x$› *this*]]
show *False*
using ‹$s \leq p\ se$›[*folded* ‹$(ry \cdot sy)^{@}\ ev = s$›[*unfolded pow-pos*[*OF* ‹$0 < ev$›]]]
‹$sy \cdot ry \neq ry \cdot sy$› **by** (*force simp add*: *prefix-def*)
qed
ultimately have *y-pref-se*: $y \leq p\ se$ **by** *blast*

from *ruler-le*[*OF* ‹$s \leq p\ se$› *this*]
show $s \leq p\ y$
using *lenarg*[*OF vx-xs*] **unfolding** *uv-y*[*symmetric*] *lenmorph* **by** *linarith*
qed

lemma *rev-square-interp-ext*: *square-interp-ext* (*rev x*) (*rev y*) (*rev s*) (*rev p*) (*rev* (*map rev ws*))
proof–
interpret *i*: *square-interp* (*rev x*) (*rev y*) (*rev s*) (*rev p*) (*rev* (*map rev ws*))
using *rev-square-interp*.
show *?thesis*
proof
show $\exists pe.\ pe \in \langle\{rev\ x,\ rev\ y\}\rangle \wedge rev\ s \leq s\ pe$
using *s-pref-y* **unfolding** *pref-rev-suf-iff* **by** *blast*
obtain *pe* **where** $pe \in \langle\{x,\ y\}\rangle$ **and** $p \leq s\ pe$

using *p-extend* **by** *blast*
hence *rev pe* ∈ ⟨{*rev x*, *rev y*}⟩
by (*simp add*: *rev-hull rev-in-conv*)
thus ∃ *se*. *se* ∈ ⟨{*rev x*, *rev y*}⟩ ∧ *rev p* ≤*p se*
using ‹*p* ≤*s pe*›[*unfolded suf-rev-pref-iff prefix-def*] *rev-rev-ident* **by** *blast*
qed
qed

lemma *p-suf-y*: *p* ≤*s y*
proof–
interpret *i*: *square-interp-ext* (*rev x*) (*rev y*) (*rev s*) (*rev p*) (*rev* (*map rev ws*))
using *rev-square-interp-ext*.

from *i.s-pref-y*[*reversed*]
show *p* ≤*s y*.
qed

theorem *bin-sq-interp-extE*: **obtains** *r t k m* **where** (*r* · *t*)^@*m* · *r* = *x* **and** (*t* · *r*)^@*k* · (*r* · *t*)^@ *k* = *y*
(*r* · *t*)^@ *k* = *p* **and** (*t* · *r*)^@ *k* = *s* **and** *r* · *t* ≠ *t* · *r* **and** *u* = *s* **and** *v* = *p* **and** |*p*| = |*s*| **and**
0 < *k* **and** *0* < *m*
proof–
obtain *r t k k′ m*
where *u*: (*t* · *r*)^@ *k* = *u* **and** *v*: (*r* · *t*)^@ *k′* = *v* **and**
p: (*r* · *t*)^@ *k* = *p* **and** *s*: (*t* · *r*)^@ *k′* = *s* **and**
x: (*r* · *t*)^@ *m* · *r* = *x* **and** *code*: *r* · *t* ≠ *t* · *r* **and**
0 < *k′ 0* < *m 0* < *k*
using *bin-sq-interpE*.
have |*u* · *v*| = |*s* · *p*|
using *lenarg*[*OF px-xu*, *unfolded lenmorph*] *lenarg*[*OF vx-xs*, *unfolded lenmorph*]
by *simp*
hence *u* · *v* = *s* · *p*
unfolding *uv-y* **using** *s-pref-y p-suf-y* **by** (*auto simp add*: *prefix-def suffix-def*)
note *eq* = ‹*u* · *v* = *s* · *p*›[*unfolded* ‹(*t* · *r*)^@ *k* = *u*›[*symmetric*] ‹(*r* · *t*)^@ *k′* = *v*›[*symmetric*],
unfolded ‹(*t* · *r*)^@ *k′* = *s*›[*symmetric*] ‹(*r* · *t*)^@ *k* = *p*›[*symmetric*]]
from *pows-comm-comm*[*OF this*]
have *k* = *k′*
using ‹*r* · *t* ≠ *t* · *r*› *eqd-eq*(*1*)[*OF - swap-len*, *of t r*] **by** *fastforce*
have |*p*| = |*s*|
using *lenarg*[*OF p*] *lenarg*[*OF s*] **unfolding** ‹*k* = *k′*› *pow-len lenmorph add.commute*[*of* |*r*|] **by** *fastforce*
thus *thesis*
using *that*[*OF x uv-y*[*folded u v* ‹*k* = *k′*›] *p s*[*folded* ‹*k* = *k′*›] *code - - -* ‹*0* < *k*› ‹*0* < *m*›] *u v p s* **unfolding** ‹*k* = *k′*› **by** *argo*
qed

lemma *ps-len*: |*p*| = |*s*| **and** *p-eq-v*: *p* = *v* **and** *s-eq-u*: *s* = *u*

using *bin-sq-interp-extE* **by** *blast+*

lemma *v-x-x-u*: $v \cdot x = x \cdot u$
using *vx-xs* **unfolding** *s-eq-u*.

lemma *sp-y*: $s \cdot p = y$
using *p-eq-v s-eq-u uv-y* **by** *auto*

lemma *p-x-x-s*: $p \cdot x = x \cdot s$
by (*simp add*: *px-xu s-eq-u*)

lemma *xxy-root*: $x \cdot x \cdot y = (x \cdot p) \cdot (x \cdot p)$
using *p-x-x-s sp-y* **by** *force*

theorem *sq-ext-interp*: *ws* $= [x, y, x]$ $s \cdot p = y$ $p \cdot x = x \cdot s$
using *cover-xyx sp-y p-x-x-s*.

end

theorem *bin-sq-interpE*:
assumes $x \cdot y \neq y \cdot x$ **and** *primitive* x **and** $|y| \leq |x|$ **and** $ws \in$ *lists* $\{x, y\}$ **and**
$\neg\ x \sim y$ **and**
$p\ [x,x]\ s \sim_{\mathcal{D}} ws$
obtains $r\ t\ m\ k\ l$ **where** $(r \cdot t)^{@}\ m \cdot r = x$ **and** $(t \cdot r)^{@}\ k \cdot (r \cdot t)^{@} l = y$
$(r \cdot t)^{@}\ k = p$ **and** $(t \cdot r)^{@}\ l = s$ **and** $r \cdot t \neq t \cdot r$ **and** $0 < k\ 0 < m\ 0 < l$
using *square-interp.bin-sq-interpE*[*OF square-interp.intro*, *OF assms*, *of thesis*].

theorem *bin-sq-interp*:
assumes $x \cdot y \neq y \cdot x$ **and** *primitive* x **and** $|y| \leq |x|$ **and** $ws \in$ *lists* $\{x, y\}$ **and**
$\neg\ x \sim y$ **and**
$p\ [x,x]\ s \sim_{\mathcal{D}} ws$
shows $ws = [x,y,x]$
using *square-interp.cover-xyx*[*OF square-interp.intro*, *OF assms*].

theorem *bin-sq-interp-extE*:
assumes $x \cdot y \neq y \cdot x$ **and** *primitive* x **and** $|y| \leq |x|$ **and** $ws \in$ *lists* $\{x, y\}$ **and**
$\neg\ x \sim y$ **and**
$p\ [x,x]\ s \sim_{\mathcal{D}} ws$ **and**
p-extend: $\exists\ pe.\ pe \in \langle\{x,y\}\rangle \wedge p \leq s\ pe$ **and**
s-extend: $\exists\ se.\ se \in \langle\{x,y\}\rangle \wedge s \leq p\ se$
obtains $r\ t\ m\ k$ **where** $(r \cdot t)^{@}\ m \cdot r = x$ **and** $(t \cdot r)^{@}\ k \cdot (r \cdot t)^{@}\ k = y$
$(r \cdot t)^{@}\ k = p$ **and** $(t \cdot r)^{@}\ k = s$ **and** $r \cdot t \neq t \cdot r$ **and** $0 < k$ **and** $0 < m$
using *square-interp-ext.bin-sq-interp-extE*[*OF square-interp-ext.intro*, *OF square-interp.intro square-interp-ext-axioms.intro*, *OF assms*, *of thesis*].

end

theory *Binary-Code-Imprimitive*

imports
Combinatorics-Words-Graph-Lemma.Glued-Codes
Binary-Square-Interpretation
begin

This theory focuses on the characterization of imprimitive words which are concatenations of copies of two words (forming a binary code). We follow the article [1] (mainly Théorème 2.1 and Lemme 3.1), while substantially optimizing the proof. See also [3] for an earlier result on this question, and [2] for another proof.

0.3 General primitivity not preserving codes

context *code*

begin

Two nontrivially conjugate elements generated by a code induce a disjoint interpretation.

lemma *shift-disjoint*:
assumes $ws \in lists\ \mathcal{C}$ **and** $ws' \in lists\ \mathcal{C}$ **and** $z \notin \langle\mathcal{C}\rangle$ **and** $z \cdot concat\ ws = concat\ ws' \cdot z$
$us \leq_p ws^{@}n$ **and** $vs \leq_p ws'^{@}n$
shows $z \cdot concat\ us \neq concat\ vs$
using ‹$z \notin \langle\mathcal{C}\rangle$›
proof (*elim contrapos-nn*)
assume $z \cdot concat\ us = concat\ vs$
have $z \neq \varepsilon$
using ‹$z \notin \langle\mathcal{C}\rangle$› **by** *blast*
obtain us' **where** $ws^{@}n = us \cdot us'$
using *prefixE*[*OF* ‹$us \leq_p ws^{@}n$›]**.**
obtain vs' **where** $ws'^{@}n = vs \cdot vs'$
using *prefixE*[*OF* ‹$vs \leq_p ws'^{@}n$›]**.**
from *conjug-pow*[*OF* ‹$z \cdot concat\ ws = concat\ ws' \cdot z$›[*symmetric*], *symmetric*]
have $z \cdot concat\ (ws^{@}n) = concat\ (ws'^{@}n) \cdot z$
unfolding *concat-pow***.**
from *this*[*unfolded* ‹$ws^{@}n = us \cdot us'$› ‹$ws'^{@}n = vs \cdot vs'$› *concat-morph rassoc*
‹$z \cdot concat\ us = concat\ vs$›[*symmetric*] *cancel*]
have $concat\ vs' \cdot z = concat\ us'$**..**
show $z \in \langle\mathcal{C}\rangle$
proof (*rule stability*)
have $us \in lists\ \mathcal{C}$ **and** $us' \in lists\ \mathcal{C}$ **and** $vs \in lists\ \mathcal{C}$ **and** $vs' \in lists\ \mathcal{C}$
using ‹$ws \in lists\ \mathcal{C}$› ‹$ws' \in lists\ \mathcal{C}$› ‹$ws'^{@}n = vs \cdot vs'$› ‹$ws^{@}n = us \cdot us'$›
by *inlists*
thus $z \cdot concat\ us \in \langle\mathcal{C}\rangle$ **and** $concat\ vs' \in \langle\mathcal{C}\rangle$ **and** $concat\ us \in \langle\mathcal{C}\rangle$ **and** $concat\ vs' \cdot z \in \langle\mathcal{C}\rangle$
unfolding ‹$concat\ vs' \cdot z = concat\ us'$› ‹$z \cdot concat\ us = concat\ vs$›
by (*simp-all add*: *concat-in-hull'*)

qed
qed

This in particular yields a disjoint extendable interpretation of any prefix

lemma *shift-interp*:
assumes $ws \in lists\ \mathcal{C}$ **and** $ws' \in lists\ \mathcal{C}$ **and** $z \notin \langle \mathcal{C} \rangle$ **and**
conjug: $z \cdot concat\ ws = concat\ ws' \cdot z$ **and** $|z| \leq |concat\ ws'|$
and $us \leq p\ ws$ **and** $us \neq \varepsilon$
obtains $p\ s\ vs\ ps$ **where**
$p\ us\ s \sim_{\mathcal{D}} vs$ **and** $vs \in lists\ \mathcal{C}$
and $s \leq p\ concat\ (us^{-1>}(ws \cdot ws))$ **and** $p \leq s\ concat\ ws$ — extendable
and $ps \cdot vs \leq p\ ws' \cdot ws'$ **and** $concat\ ps \cdot p = z$
proof–
have $ws' \cdot ws' \in lists\ \mathcal{C}$
using ‹$ws' \in lists\ \mathcal{C}$› **by** *inlists*
have $concat\ us \neq \varepsilon$
using ‹$us \neq \varepsilon$› **unfolding** *code-concat-eq-emp-iff*[*OF pref-in-lists*[*OF* ‹$us \leq p\ ws$› ‹$ws \in lists\ \mathcal{C}$›]].
have $|concat\ ws'| = |concat\ ws|$
using *lenarg*[*OF conjug*, *unfolded lenmorph*] **by** *linarith*
have $z \cdot concat(ws \cdot ws) = concat\ (ws' \cdot ws') \cdot z$
unfolding *rassoc concat-morph conjug*[*symmetric*] **unfolding** *lassoc cancel-right*
using *conjug*.
hence $concat\ (ws' \cdot ws') \leq p\ z \cdot concat\ (ws \cdot ws)$
by *blast*
have $z \cdot concat\ ws \leq p\ concat\ (ws' \cdot ws')$
unfolding *concat-morph conjug pref-cancel-conv* **using** *eq-le-pref*[*OF conjug* ‹$|z| \leq |concat\ ws'|$›].
from *prefixE*[*OF pref-shorten*[*OF pref-concat-pref*[*OF* ‹$us \leq p\ ws$›] *this*], *unfolded rassoc*]
obtain *su* **where** *fac-u*[*symmetric*]: $concat\ (ws' \cdot ws') = z \cdot concat\ us \cdot su$.

from *obtain-fac-interp*[*OF fac-u* ‹$concat\ us \neq \varepsilon$›]
obtain $ps\ ss'\ p\ s\ vs$ **where** $p\ (concat\ us)\ s \sim_{\mathcal{I}} vs$ **and**
$ps \cdot vs \cdot ss' = ws' \cdot ws'$ **and** $concat\ ps \cdot p = z$ **and** $s \cdot concat\ ss' = su$.
note *fac-interpD*[*OF* ‹$p\ (concat\ us)\ s \sim_{\mathcal{I}} vs$›]

let $?ss = us^{-1>}(ws \cdot ws)$
have $us \cdot ?ss = ws \cdot ws$
using ‹$us \leq p\ ws$› **by** *auto*

have $ps \cdot vs \leq p\ ws' \cdot ws'$
unfolding ‹$ps \cdot vs \cdot ss' = ws' \cdot ws'$›[*symmetric*] *lassoc* **using** *triv-pref*.

hence $vs \in lists\ \mathcal{C}$
using ‹$ws' \in lists\ \mathcal{C}$›
by *inlists*

have $s \leq p\ concat\ ?ss$

using ‹*concat* $(ws' \cdot ws') \leq p\ z \cdot$ *concat* $(ws \cdot ws)$›
unfolding *arg-cong*[*OF* ‹$ps \cdot vs \cdot ss' = ws' \cdot ws'$›, *of concat*, *symmetric*] ‹*concat* $ps \cdot p = z$›[*symmetric*]
arg-cong[*OF* ‹$us \cdot$ *?ss* $= ws \cdot ws$›, *of concat*, *symmetric*]
unfolding *concat-morph rassoc pref-cancel-conv*
‹$p \cdot$ *concat* $us \cdot s =$ *concat* vs›[*symmetric*]
using *append-prefixD* **by** *auto*

have $|p| \leq |$*concat* $ws|$
using ‹$|z| \leq |$*concat* $ws'|$›[*folded lenarg*[*OF* ‹*concat* $ps \cdot p = z$›], *unfolded* ‹$|$*concat* $ws'| = |$*concat* $ws|$›]
by *simp*

with *eqd*[*reversed*, *OF conjug*[*folded* ‹*concat* $ps \cdot p = z$›, *unfolded lassoc*, *symmetric*] *this*]
have $p \leq s$ *concat* ws
by *blast*

have *disjoint*: $p \cdot$ *concat* $us' \neq$ *concat* vs' **if** $us' \leq p\ us\ vs' \leq p\ vs$ **for** $us'\ vs'$
proof
have $us' \leq p\ ws \cdot ws$
using ‹$us \leq p\ ws$› ‹$us' \leq p\ us$› **by** *auto*
have $ps \cdot vs' \leq p\ ws' \cdot ws'$
using ‹$vs' \leq p\ vs$› ‹$ps \cdot vs \leq p\ ws' \cdot ws'$› *pref-trans same-prefix-prefix* **by** *metis*
assume $p \cdot$ *concat* $us' =$ *concat* vs'
hence $z \cdot$ *concat* $us' =$ *concat* $(ps \cdot vs')$
unfolding *concat-morph* ‹*concat* $ps \cdot p = z$›[*symmetric*] *rassoc cancel*.
thus *False*
using *shift-disjoint*[*OF* ‹$ws \in$ *lists* $\mathcal{C}$› ‹$ws' \in$ *lists* $\mathcal{C}$› ‹$z \notin \langle\mathcal{C}\rangle$›
‹$z \cdot$ *concat* $ws =$ *concat* $ws' \cdot z$› ‹$us' \leq p\ ws \cdot ws$›[*folded pow-two*] ‹$ps \cdot vs' \leq p\ ws' \cdot ws'$›[*folded pow-two*]] **by** *fast*
qed
from *disjoint*[*of* $\varepsilon\ \varepsilon$]
have $p \neq \varepsilon$ **by** *blast*
have $s \neq \varepsilon$
using ‹$p \cdot$ *concat* $us \cdot s =$ *concat* vs› *disjoint* **by** *auto*

from *disjoint-interpI*[*OF* ‹p (*concat* us) $s \sim_{\mathcal{I}} vs$›] *disjoint*
have $p\ us\ s \sim_{\mathcal{D}} vs$
by *blast*

from *that*[*OF this* ‹$vs \in$ *lists* $\mathcal{C}$›
‹$s \leq p$ *concat ?ss*› ‹$p \leq s$ *concat* ws› ‹$ps \cdot vs \leq p\ ws' \cdot ws'$› ‹*concat* $ps \cdot p = z$›]
show *thesis*.
qed

The conditions are in particular met by imprimitivity witnesses

lemma *imprim-witness-shift*:
assumes $ws \in$ *lists* $\mathcal{C}$ **and** *primitive* ws **and** $\neg$ *primitive* (*concat* ws)

obtains $z\ n$ **where** $concat\ ws = z^{@}n\ z \notin \langle \mathcal{C} \rangle$ **and**
$z \cdot concat\ ws = concat\ ws \cdot z$ **and** $|z| < |concat\ ws|$ **and** $2 \leq n$
proof−
have $concat\ ws \neq \varepsilon$
using ‹$primitive\ ws$› $emp\text{-}concat\text{-}emp'[OF$ ‹$ws \in lists\ \mathcal{C}$›$]$ $emp\text{-}not\text{-}prim$ **by** *blast*
obtain $z\ n$ **where** $[symmetric]$: $z^{@}n = concat\ ws$ **and** $2 \leq n$
using $not\text{-}prim\text{-}primroot\text{-}expE[OF$ ‹$\neg\ primitive\ (concat\ ws)$›$]$ **by** *metis*

hence $z \neq \varepsilon$
using ‹$concat\ ws \neq \varepsilon$› **by** *force*

have $z \notin \langle \mathcal{C} \rangle$
proof
assume $z \in \langle \mathcal{C} \rangle$
then obtain zs **where** $zs \in lists\ \mathcal{C}$ **and** $concat\ zs = z$
using *hull-concat-lists0* **by** *blast*
from $is\text{-}code[OF$ ‹$ws \in lists\ \mathcal{C}$› $pow\text{-}in\text{-}lists[OF$ ‹$zs \in lists\ \mathcal{C}$›$]$,
unfolded concat-pow ‹$concat\ ws = z^{@}n$› ‹$concat\ zs = z$›, *of* $n]$
show *False*
using ‹$primitive\ ws$› ‹$2 \leq n$› $pow\text{-}nemp\text{-}imprim$ **by** *blast*
qed

have $|z| < |concat\ ws|$
unfolding $lenarg[OF$ ‹$concat\ ws = z^{@}n$›, *unfolded lenmorph pow-len*$]$
using $nemp\text{-}len[OF$ ‹$z \neq \varepsilon$›$]$ ‹$2 \leq n$› **by** *simp*

from $that[OF$ ‹$concat\ ws = z^{@}n$› ‹$z \notin \langle \mathcal{C} \rangle$› - *this* ‹$2 \leq n$›$]$
show *thesis*
unfolding ‹$concat\ ws = z^{@}n$› *pow-comm* **by** *blast*

qed

end

0.4 Covered uniform square

lemma *cover-xy-xxx*: **assumes** $|x| = |y|$ **and** $p \cdot x \cdot y \cdot s = x \cdot x \cdot x$
shows $x = y$
using *append-assoc assms(1) assms(2) eq-le-pref le-refl long-pref lq-triv prefI pref-comm-eq'* **by** *metis*

lemma *cover-xy-yyy*: **assumes** $|x| = |y|$ **and** *eq*: $p \cdot x \cdot y \cdot s = y \cdot y \cdot y$
shows $x = y$
using *cover-xy-xxx*[*reversed*, *unfolded rassoc*, *OF* ‹$|x| = |y|$›[*symmetric*] *eq*, *symmetric*]**.**

lemma *cover-xy-xxy*: **assumes** $|x| = |y|$ **and** $s \neq \varepsilon$ **and** *eq*: $p \cdot x \cdot y \cdot s = x \cdot x \cdot y$
shows $x = y$

proof–
have $|p| < |x|$
using *lenarg*[*OF eq*] *nemp-pos-len*[*OF* ‹$s \neq \varepsilon$›] **unfolding** *lenmorph* **by** *linarith*
then obtain t **where** x: $x = p \cdot t$ **and** $t \neq \varepsilon$
using *eqd*[*OF eq*] **by** *force*
from *eq*[*unfolded this rassoc cancel*]
have $p \cdot t = t \cdot p$
by *mismatch*
hence $x \leq p\ t \cdot x$
unfolding x **by** *auto*
from *eq*[*unfolded x*]
have $y \leq p\ t \cdot y$
using ‹$p \cdot t = t \cdot p$› ‹$p \cdot t \cdot y \cdot s = t \cdot p \cdot t \cdot y$› *pref-cancel′ suf-marker-per-root triv-pref* **by** *metis*
show $x = y$
using *same-len-nemp-root-eq*[*OF per-rootI*[*OF* ‹$x \leq p\ t \cdot x$› ‹$t \neq \varepsilon$›]
per-rootI[*OF* ‹$y \leq p\ t \cdot y$› ‹$t \neq \varepsilon$›] ‹$|x| = |y|$›]**.**
qed

lemma *cover-xy-xyy*: **assumes** $|x| = |y|$ **and** $p \neq \varepsilon$ **and** *eq*: $p \cdot x \cdot y \cdot s = x \cdot y \cdot y$
shows $x = y$
using *cover-xy-xxy*[*reversed*, *unfolded rassoc*, *OF assms*(*1*)[*symmetric*] *assms*(*2*) *eq*]**..**

lemma *cover-xy-yyx*: **assumes** $|x| = |y|$ **and** *eq*: $p \cdot x \cdot y \cdot s = y \cdot y \cdot x$
shows $x = y$
proof–
have $|p| \leq |y|$
using *lenarg*[*OF eq*] **unfolding** *lenmorph* **by** *linarith*
then obtain t **where** y: $y = p \cdot t$
using *eqd*[*OF eq*] **by** *force*
from *eqd-eq*[*OF* - ‹$|x| = |y|$›[*unfolded y swap-len*[*of p*]], *unfolded rassoc*] *eq*[*unfolded this rassoc cancel*]
have x: $x = t \cdot p$ **by** *blast*
from *eq*[*unfolded x y rassoc cancel*]
have $p \cdot t = t \cdot p$
by *mismatch*
thus $x = y$
unfolding x y**..**
qed

lemma *cover-xy-yxx*: **assumes** $|x| = |y|$ **and** *eq*: $p \cdot x \cdot y \cdot s = y \cdot x \cdot x$
shows $x = y$
using *cover-xy-yyx*[*reversed*, *unfolded rassoc*, *OF assms*(*1*)[*symmetric*] *eq*]**..**

lemma *cover-xy-xyx*: **assumes** $|x| = |y|$ **and** $p \neq \varepsilon$ **and** $s \neq \varepsilon$ **and** *eq*: $p \cdot x \cdot y \cdot s = x \cdot y \cdot x$
shows ¬ *primitive* $(x \cdot y)$

proof
assume *primitive* $(x \cdot y)$
have $p \cdot (x \cdot y) \cdot (s \cdot y) = (x \cdot y) \cdot (x \cdot y)$
unfolding *lassoc eq*[*unfolded lassoc*]**..**
from *prim-overlap-sqE*[*OF* ‹*primitive* $(x \cdot y)$› *this*]
show *False*
using ‹$p \neq \varepsilon$› ‹$s \neq \varepsilon$› **by** *blast*
qed

lemma *cover-xy-yxy*: **assumes** $|x| = |y|$ **and** $p \neq \varepsilon$ **and** ‹$s \neq \varepsilon$› **and** *eq*: $p \cdot x \cdot y \cdot s = y \cdot x \cdot y$
shows ¬ *primitive* $(x \cdot y)$
using *cover-xy-xyx*[*reversed*, *unfolded rassoc*, *OF assms*(*1*)[*symmetric*] *assms*(*3*) *assms*(*2*) *eq*]**.**

theorem *uniform-square-interp*: **assumes** $x{\cdot}y \neq y{\cdot}x$ **and** $|x| = |y|$ **and** $vs \in lists\ \{x,y\}$
and $p\ \ (x \cdot y)\ \ s \sim_{\mathcal{I}} vs$ **and** $p \neq \varepsilon$
shows ¬ *primitive* $(x{\cdot}y)$ **and** $vs = [x,y,x] \vee vs = [y,x,y]$
proof−
note *fac-interpD*[*OF* ‹$p\ \ (x \cdot y)\ \ s \sim_{\mathcal{I}} vs$›]

have $vs \neq \varepsilon$
using ‹$p \cdot (x \cdot y) \cdot s = concat\ vs$› *assms*(*5*) **by** *force*
have $|p| < |x|$
using *prefix-length-less*[*OF* ‹$p <_p hd\ vs$›] *lists-hd-in-set*[*OF* ‹$vs \neq \varepsilon$› ‹$vs \in lists\ \{x,y\}$›]
‹$|x| = |y|$›
by *fastforce*
have $|s| < |x|$
using *suffix-length-less*[*OF* ‹$s <_s last\ vs$›] ‹$|x| = |y|$› *lists-hd-in-set*[*reversed*, *OF* ‹$vs \neq \varepsilon$› ‹$vs \in lists\ \{x,y\}$›]
by *fastforce*
have $|concat\ vs| = |x| * |vs|$
using *assms*(*2*−*3*)
proof (*induction vs*)
case (*Cons a vs*)
have $|a| = |x|$ **and** $|a\ \#\ vs| = Suc\ |vs|$ **and**
$|concat\ (a\ \#\ vs)| = |a| + |concat\ vs|$ **and** $|concat\ vs| = |x| * |vs|$
using ‹$a\#vs \in lists\ \{x,y\}$› ‹$|x| = |y|$› *Cons.IH Cons.prems* **by** *auto*
then show *?case* **by** *force*
qed *simp*
note *leneq* = *lenarg*[*OF* ‹$p \cdot (x \cdot y) \cdot s = concat\ vs$›, *unfolded this lenmorph* ‹$|x| = |y|$›[*symmetric*]]
hence $|x| * |vs| < |x| * 4$ **and** $2 * |x| < |x| * |vs|$
using ‹$|p| < |x|$› ‹$|s| < |x|$› *nemp-pos-len*[*OF* ‹$p \neq \varepsilon$›] **by** *linarith*+
hence $|vs| = 3$
by *force*
hence $s \neq \varepsilon$

using *leneq* ‹$|p| < |x|$› **by** *force*

have $x \neq y$
using *assms(1)* **by** *blast*
with ‹$|vs| = 3$› ‹$vs \in lists\ \{x,y\}$› ‹$p \cdot (x \cdot y) \cdot s = concat\ vs$›
have $(\neg\ primitive\ (x{\cdot}y)) \wedge (vs = [x,y,x] \vee vs = [y,x,y])$
proof(*list-inspection*, *simp-all*)
assume $p \cdot x \cdot y \cdot s = x \cdot x \cdot x$
from *cover-xy-xxx*[*OF* ‹$|x| = |y|$› *this*]
show *False*
using ‹$x \neq y$› **by** *blast*
next
assume $p \cdot x \cdot y \cdot s = x \cdot x \cdot y$
from *cover-xy-xxy*[*OF* ‹$|x| = |y|$› ‹$s \neq \varepsilon$› *this*]
show *False*
using ‹$x \neq y$› **by** *blast*
next
assume $p \cdot x \cdot y \cdot s = x \cdot y \cdot x$
from *cover-xy-xyx*[*OF* ‹$|x| = |y|$› ‹$p \neq \varepsilon$› ‹$s \neq \varepsilon$› *this*]
show $\neg$ *primitive* $(x \cdot y)$
by *blast*
next
assume $p \cdot x \cdot y \cdot s = x \cdot y \cdot y$
from *cover-xy-xyy*[*OF* ‹$|x| = |y|$› ‹$p \neq \varepsilon$› *this*]
show *False*
using ‹$x \neq y$› **by** *blast*
next
assume $p \cdot x \cdot y \cdot s = y \cdot x \cdot x$
from *cover-xy-yxx*[*OF* ‹$|x| = |y|$› *this*]
show *False*
using ‹$x \neq y$› **by** *blast*
next
assume $p \cdot x \cdot y \cdot s = y \cdot x \cdot y$
from *cover-xy-yxy*[*OF* ‹$|x| = |y|$› ‹$p \neq \varepsilon$› ‹$s \neq \varepsilon$› *this*]
show $\neg$ *primitive* $(x \cdot y)$
by *blast*
next
assume $p \cdot x \cdot y \cdot s = y \cdot y \cdot x$
from *cover-xy-yyx*[*OF* ‹$|x| = |y|$› *this*]
show *False*
using ‹$x \neq y$› **by** *blast*
next
assume $p \cdot x \cdot y \cdot s = y \cdot y \cdot y$
from *cover-xy-yyy*[*OF* ‹$|x| = |y|$› *this*]
show *False*
using ‹$x \neq y$› **by** *blast*
qed
thus $\neg$ *primitive* $(x{\cdot}y)$ $vs = [x,y,x] \vee vs = [y,x,y]$
by *blast*+

qed

0.4.1 Primitivity (non)preserving uniform binary codes

theorem *bin-uniform-prim-morph*:
assumes $x \cdot y \neq y \cdot x$ **and** $|x| = |y|$ **and** *primitive* $(x \cdot y)$
and $ws \in$ *lists* $\{x,y\}$ **and** $2 \leq |ws|$
shows *primitive ws* $\longleftrightarrow$ *primitive* (*concat ws*)
proof (*standard*, *rule ccontr*)
assume ‹*primitive ws*› **and** ‹¬ *primitive* (*concat ws*)›
from *bin-prim-long-pref*[*OF* ‹$ws \in$ *lists* $\{x,y\}$› ‹*primitive ws*› ‹$2 \leq |ws|$›]
obtain ws' **where** $ws \sim ws'$ $[x, y] \leq p$ ws'**.**
have $ws' \in$ *lists* $\{x,y\}$
using *conjug-in-lists'*[*OF* ‹$ws \sim ws'$› ‹$ws \in$ *lists* $\{x,y\}$›]**.**
have *primitive* ws'
using *prim-conjug*[*OF* ‹*primitive ws*› ‹$ws \sim ws'$›]**.**
have ¬ *primitive* (*concat ws'*)
using *conjug-concat-prim-iff* ‹¬ *primitive* (*concat ws*)› ‹$ws \sim ws'$› **by** *auto*
interpret *code* $\{x,y\}$
using *bin-code-code*[*OF* ‹$x \cdot y \neq y \cdot x$›]**.**

have $[x,y] \neq \varepsilon$ **by** *blast*
from *imprim-witness-shift*[*OF* ‹$ws' \in$ *lists* $\{x,y\}$› ‹*primitive ws'*› ‹¬ *primitive* (*concat ws'*)›]
obtain z n **where** *concat* $ws' = z$ [@] n $z \notin \langle\{x, y\}\rangle$ $z \cdot$ *concat* $ws' =$ *concat* $ws' \cdot z$ $|z| < |$*concat ws'*|**.**
from *shift-interp*[*OF* ‹$ws' \in$ *lists* $\{x,y\}$› ‹$ws' \in$ *lists* $\{x,y\}$› *this*(*2*−*3*) *less-imp-le*[*OF this*(*4*)] ‹$[x,y] \leq p$ ws'› ‹$[x,y] \neq \varepsilon$›]
obtain p s *vs ps* **where** p $[x, y]$ $s \sim_{\mathcal{D}}$ *vs vs* $\in$ *lists* $\{x, y\}$ $s \leq p$ *concat* ($[x, y]^{-1>}(ws' \cdot ws')$)
$p \leq s$ *concat ws' ps* $\cdot$ *vs* $\leq p$ *ws'* $\cdot$ *ws'* *concat ps* $\cdot$ $p = z$**.**
from *uniform-square-interp*(*1*)[*OF* ‹$x \cdot y \neq y \cdot x$› ‹$|x| = |y|$› ‹*vs* $\in$ *lists* $\{x,y\}$› - -]
‹*primitive* $(x \cdot y)$› *disj-interpD*[*OF this*(*1*), *simplified*] *disj-interp-nemp*(*1*)[*OF this*(*1*)]
show *False* **by** *force*
qed (*simp add*: *prim-concat-prim*)

— A stronger version is implied by the following lemma.

lemma *bin-uniform-imprim*: **assumes** $x \cdot y \neq y \cdot x$ **and** $|x| = |y|$ **and** ¬ *primitive* $(x \cdot y)$
shows *primitive x*
proof−
have $x \cdot y \neq \varepsilon$ **and** $x \neq \varepsilon$ **and** $y \neq \varepsilon$
using ‹$x \cdot y \neq y \cdot x$› **by** *blast*+
from *not-prim-expE*[*OF* ‹¬ *primitive* $(x \cdot y)$› ‹$x \cdot y \neq \varepsilon$›]
obtain z k **where** *primitive z* **and** $2 \leq k$ **and** z[@]$k = x \cdot y$**.**
hence $0 < k$

by *simp*
from *split-pow*[*OF* ‹$z^{@}k = x \cdot y$›[*symmetric*] ‹$0 < k$› ‹$y \neq \varepsilon$›]
obtain $u\ v\ l\ m$ **where** [*symmetric*]: $z = u \cdot v$ **and** $v \neq \varepsilon\ x = (u{\cdot}v)\ ^{@}\ l \cdot u\ \ y = (v \cdot u)\ ^{@}\ m \cdot v\ k = l + m + 1$**.**
have $u \cdot v \neq v \cdot u$
using ‹$x \cdot y \neq y \cdot x$› **unfolding** ‹$x = (u{\cdot}v)\ ^{@}\ l \cdot u$› ‹$y = (v \cdot u)\ ^{@}\ m \cdot v$›
shifts **unfolding** *add-exps*[*symmetric*] *add.commute*[*of m*] **by** *force*
have $u \neq \varepsilon$ **and** $v \neq \varepsilon$ **and** $u \neq v$
using ‹$u \cdot v \neq v \cdot u$› **by** *blast+*
have $m = l$ **and** $|u| = |v|$
using *almost-equal-equal*[*OF nemp-len*[*OF* ‹$u \neq \varepsilon$›] *nemp-len*[*OF* ‹$v \neq \varepsilon$›], *of l m*] *lenarg*[*OF* ‹$x = (u{\cdot}v)^{@}l \cdot u$›, *unfolded* ‹$|x| = |y|$›, *unfolded lenarg*[*OF* ‹$y = (v \cdot u)\ ^{@}\ m \cdot v$›]]
unfolding *lenmorph pow-len lenarg*[*OF* ‹$u \cdot v = z$›, *symmetric*] **by** *algebra+*
from ‹$k = l + m + 1$›[*folded Suc-eq-plus1*, *symmetric*]
have $l \neq 0$
using ‹$2 \leq k$›[*folded* ‹$Suc(l{+}m) = k$›, *unfolded* ‹$m = l$›] **by** *force*
let *?w* = $[u,v]^{@}l \cdot [u]$
have *?w* ∈ *lists* $\{u,v\}$
by (*induct l*, *simp-all*)
have $2 \leq |?w|$
using ‹$l \neq 0$› **unfolding** *lenmorph pow-len* **by** *fastforce*
have *concat ?w* = x
using ‹$x = (u \cdot v)\ ^{@}\ l \cdot u$› **by** *simp*
from *bin-uniform-prim-morph*[*OF* ‹$u \cdot v \neq v \cdot u$› ‹$|u| = |v|$› ‹*primitive z*›[*folded* ‹$u \cdot v = z$›] ‹$?w \in lists\ \{u,v\}$› ‹$2 \leq |?w|$›]
show *primitive x*
unfolding ‹*concat ?w* = x› **using** *alternate-prim*[*OF* ‹$u \neq v$›] **by** *blast*
qed

theorem *bin-uniform-prim-morph′*:
assumes $x \cdot y \neq y \cdot x$ **and** $|x| = |y|$ **and** *primitive* $(x \cdot y) \vee \neg$ *primitive* $x \vee \neg$ *primitive* y
and *ws* ∈ *lists* $\{x,y\}$ **and** $2 \leq |ws|$
shows *primitive ws* ⟷ *primitive* (*concat ws*)
using *bin-uniform-prim-morph*[*OF assms*(*1−2*) - *assms*(*4−5*)] *bin-uniform-imprim*[*OF assms*(*1−2*)]
bin-uniform-imprim[*OF assms*(*1−2*)[*symmetric*], *unfolded conjug-prim-iff′*[*of y*]]
assms(*3*) **by** *blast*

0.5 The main theorem

0.5.1 Imprimitive words with single y

If the shorter word occurs only once, the result is straightforward from the parametric solution of the Lyndon-Schutzenberger equation.

lemma *bin-imprim-single-y*:
  assumes *non-comm*: $x \cdot y \neq y \cdot x$ **and**
    $ws \in lists\ \{x,y\}$ **and**
    $|y| \leq |x|$ **and**
    $2 \leq count\text{-}list\ ws\ x$ **and**
    $count\text{-}list\ ws\ y < 2$ **and**
    $primitive\ ws$ **and**
    $\neg\ primitive\ (concat\ ws)$
  shows $ws \sim [x,x,y]$ **and** $primitive\ x$ **and** $primitive\ y$
proof−
  have $x \neq y$
    using *non-comm* **by** *blast*
  have $count\text{-}list\ ws\ y \neq 0$
  proof
    assume $count\text{-}list\ ws\ y = 0$
    from *bin-lists-count-zero'*[*OF* ‹$ws \in lists\ \{x,y\}$› *this*]
    have $ws \in lists\ \{x\}$**.**
    from *prim-exp-one*[*OF* ‹*primitive ws*› *sing-lists-exp-count*[*OF this*]]
    show *False*
      using ‹$2 \leq count\text{-}list\ ws\ x$› **by** *simp*
  qed
  hence $count\text{-}list\ ws\ y = 1$
    using ‹$count\text{-}list\ ws\ y < 2$› **by** *linarith*

  from *this bin-count-one-conjug*[*OF* ‹$ws \in lists\ \{x,y\}$› - *this*]
  have $ws \sim [x]^{@} count\text{-}list\ ws\ x \cdot [y]$
    using *non-comm* (*1*) **by** *metis*
  from *conjug-concat-prim-iff*[*OF this*]
  have $\neg\ primitive\ (x^{@}(count\text{-}list\ ws\ x) \cdot y)$
    using ‹$\neg\ primitive\ (concat\ ws)$› **by** *simp*

  from *not-prim-primroot-expE*[*OF this*]
  obtain $z\ l$ **where** [*symmetric*]: $z^{@} l = x^{@}(count\text{-}list\ ws\ x) \cdot y^{@} 1$ **and** $2 \leq l$
    unfolding *pow-1***.**

  interpret *LS-len-le x y count-list ws x 1 l z*
    by (*unfold-locales*)
    (*use* ‹$2 \leq count\text{-}list\ ws\ x$› ‹$x \cdot y \neq y \cdot x$› ‹$|y| \leq |x|$›
     ‹$x\ @\ count\text{-}list\ ws\ x \cdot y\ @\ 1 = z\ @\ l$› ‹$2 \leq l$› **in** *force*)+

  from *case-j2k1*[*OF* ‹$2 \leq count\text{-}list\ ws\ x$› *refl*]
  have $primitive\ x$ **and** $primitive\ y$ **and** $count\text{-}list\ ws\ x = 2$ **by** *blast*+

  with ‹$ws \sim [x]^{@} count\text{-}list\ ws\ x \cdot [y]$›[*unfolded this*(*3*) *pow-two append-Cons append-Nil*]
  show $primitive\ x$ **and** $primitive\ y$ **and** $ws \sim [x,x,y]$
    by *simp-all*

qed

0.5.2 Conjugate words

lemma *bin-imprim-not-conjug*:
assumes *ws* ∈ *lists* {*x*,*y*} **and**
x · *y* ≠ *y* · *x* **and**
2 ≤ |*ws*| **and**
primitive ws **and**
¬ *primitive* (*concat ws*)
shows ¬ *x* ∼ *y*
proof
assume *x* ∼ *y*
hence |*x*| = |*y*| **by** *force*
from *bin-uniform-prim-morph*[*OF* ‹*x* · *y* ≠ *y* · *x*› *this* - ‹*ws* ∈ *lists* {*x*,*y*}› ‹*2* ≤ |*ws*|›]
have ¬ *primitive* (*x*·*y*)
using ‹*primitive ws*› ‹¬ *primitive* (*concat ws*)› **by** *blast*

from *Lyndon-Schutzenberger-conjug*[*OF* ‹*x* ∼ *y*› *this*]
show *False*
using ‹*x* · *y* ≠ *y* · *x*› **by** *blast*
qed

0.5.3 Square factor of the longer word and both words primitive (was all_assms)

The main idea of the proof is as follows: Imprimitivity of the concatenation yields (at least) two overlapping factorizations into {*x*, *y*}. Due to the presence of the square *x* · *x*, these two can be synchronized, which yields that the situation coincides with the canonical form.

lemma *bin-imprim-primitive*:
assumes *x* · *y* ≠ *y* · *x*
and *primitive x* **and** *primitive y*
and |*y*| ≤ |*x*|
and *ws* ∈ *lists* {*x*, *y*}
and *primitive ws* **and** ¬ *primitive* (*concat ws*)
and [*x*, *x*] ≤*f* *ws* · *ws*
shows *ws* ∼ [*x*, *x*, *y*]
proof–
— Preliminaries
have *x* ≠ *y*
using *assms*(*1*) **by** *blast*
have |*ws*| ≠ *1*

using *len-one-concat-in*[*OF* ‹*ws* ∈ *lists* {*x*, *y*}›] ‹¬ *primitive* (*concat ws*)› ‹*primitive x*› ‹*primitive y*›
by *blast*
with *prim-nemp*[*OF* ‹*primitive ws*›, *THEN nemp-le-len*]
have *2* ≤ |*ws*|
by *auto*
hence |[*x*, *x*]| ≤ |*ws*|
by *force*
have ¬ *x* ∼ *y*
by (*rule bin-imprim-not-conjug*) *fact+*
have *primitive* [*x*,*x*,*y*]
using ‹*x* ≠ *y*› **by** *primitivity-inspection*
have *concat* [*x*,*x*] = *x* · *x*
by *simp*
interpret *xy*: *binary-code x y*
using ‹*x* · *y* ≠ *y* · *x*› **by** (*unfold-locales*)

— Rotate *ws* in order to obtain a list with a prefix [*x* · *x*]
obtain *ws*′ **where** *ws* ∼ *ws*′ [*x*,*x*] ≤*p* *ws*′
using *rotate-into-pos-sq*[*of* *ε* [*x*,*x*] *- thesis*, *unfolded emp-simps*, *OF* ‹[*x*, *x*] ≤*f* *ws* · *ws*›
le0 ‹|[*x*, *x*]| ≤ |*ws*|›] **by** *blast*

have *ws*′ ∈ *lists* {*x*,*y*} **and** *primitive ws*′ **and** ¬ *primitive* (*concat ws*′)
using *conjug-in-lists*′[*OF* ‹*ws* ∼ *ws*′› ‹*ws* ∈ *lists* {*x*, *y*}›]
prim-conjug[*OF* ‹*primitive ws*› ‹*ws* ∼ *ws*′›]
‹¬ *primitive* (*concat ws*)›[*unfolded conjug-concat-prim-iff*[*OF* ‹*ws* ∼ *ws*′›]].
have *2* ≤ |*ws*′| **and** [*x*,*x*] ≠ *ε* **and** *ws*′ ≠ *ε*
using ‹[*x*,*x*] ≤*p* *ws*′› **unfolding** *prefix-def* **by** *auto*
have *concat ws*′ ≠ *ε*
using ‹*primitive x*› ‹[*x*,*x*] ≤*p* *ws*′› **by** (*fastforce simp add*: *prefix-def*)
have *ws*′ · *ws*′ · *ws*′ ∈ *lists* {*x*, *y*} **and** *ws*′ · *ws*′ ∈ *lists* {*x*, *y*}
using ‹*ws*′ ∈ *lists* {*x*,*y*}› **by** *inlists*

— The core of the proof
have *ws*′ = [*x*,*x*,*y*]
proof(*rule ccontr*)
assume *ws*′ ≠ [*x*,*x*,*y*]

from *xy.imprim-witness-shift*[*OF* ‹*ws*′ ∈ *lists* {*x*,*y*}› ‹*primitive ws*′› ‹¬ *primitive* (*concat ws*′)›]
obtain *z n* **where** *con-ws*: *concat ws*′ = *z* @ *n* **and** *z* ∉ ⟨{*x*, *y*}⟩ **and** *z* · *concat ws*′ = *concat ws*′ · *z*
and |*z*| < |*concat ws*′| **and** *2* ≤ *n*.
have *0* < *n*
using ‹*2* ≤ *n*› **by** *simp*
from *xy.shift-interp*[*OF* ‹*ws*′ ∈ *lists* {*x*,*y*}› ‹*ws*′ ∈ *lists* {*x*,*y*}› ‹*z* ∉ ⟨{*x*, *y*}⟩› ‹*z* · *concat ws*′ = *concat ws*′ · *z*›
less-imp-le[*OF* ‹|*z*| < |*concat ws*′|›] ‹[*x*,*x*] ≤*p* *ws*′› ‹[*x*,*x*] ≠ *ε*›]

obtain $p\ s\ vs\ ps$ **where** *dis*: $p\ [x,x]\ s \sim_{\mathcal{D}} vs$ **and** $\langle vs \in lists\ \{x,\ y\}\rangle$ **and**
$s \leq p\ concat\ ([x,x]^{-1>}(ws' \cdot ws'))$ **and** $p \leq s\ concat\ ws'$ **and** $ps \cdot vs \leq p\ ws' \cdot ws'$
and $concat\ ps \cdot p = z$**.**

from *disj-interp-nemp*(*1*)[*OF this*(*1*)]
have $p \neq \varepsilon$ **by** *simp*

have $p \cdot concat\ p1 \neq concat\ p2$ **if** $p1 \leq p\ [x,\ x]$ **and** $p2 \leq p\ vs$ **for** $p1\ p2$
using $\langle p\ [x,x]\ s \sim_{\mathcal{D}} vs\rangle$ *disj-interpD1 that* **by** *blast*

have $ps \in lists\ \{x,y\}$
using $\langle ps \cdot vs \leq p\ ws' \cdot ws'\rangle$ $\langle ws' \in lists\ \{x,y\}\rangle$ $\langle ws' \cdot ws' \in lists\ \{x,\ y\}\rangle$
append-prefixD pref-in-lists **by** *metis*
have $vs \in lists\ \{x,y\}$
using $\langle ws' \in lists\ \{x,y\}\rangle$ *pref-in-lists*[*OF* $\langle ps \cdot vs \leq p\ ws' \cdot ws'\rangle$] **by** *inlists*
have $[x,x]^{-1>}(ws' \cdot ws') \in lists\ \{x,y\}$
using $\langle ws' \in lists\ \{x,y\}\rangle$ **by** *inlists*
have $p\ x \cdot x\ s \sim_{\mathcal{I}} vs$
using *disj-interpD*[*OF* $\langle p\ [x,x]\ s \sim_{\mathcal{D}} vs\rangle$] **by** *simp*

interpret *square-interp-ext x y p s vs*
proof (*rule square-interp-ext.intro*[*OF square-interp.intro*, *unfolded square-interp-ext-axioms-def*])
show $(\exists\, pe.\ pe \in \langle\{x,\ y\}\rangle \wedge p \leq s\ pe) \wedge (\exists\, se.\ se \in \langle\{x,\ y\}\rangle \wedge s \leq p\ se)$
using $\langle s \leq p\ concat\ ([x,x]^{-1>}(ws' \cdot ws'))\rangle$ $\langle p \leq s\ concat\ ws'\rangle$
$\langle [x,\ x]^{-1>}(ws' \cdot ws') \in lists\ \{x,\ y\}\rangle$ $\langle ws' \in lists\ \{x,\ y\}\rangle$ *concat-in-hull'* **by**
meson
qed *fact+*

— Establishing the connection between ws' = [x,x,y] and z = xp.

define *xp* **where** $xp = x \cdot p$

have $concat\ [x,x,y] = xp \cdot xp$
by (*simp add*: *xxy-root xp-def*)

hence $ws' \cdot [x,x,y] \neq [x,x,y] \cdot ws'$
using *comm-prim*[*OF* $\langle primitive\ ws'\rangle$ $\langle primitive\ [x,x,y]\rangle$] $\langle ws' \neq [x,x,y]\ \rangle$**by**
force

have $z \cdot xp \neq xp \cdot z$
proof
assume $z \cdot xp = xp \cdot z$
from *comm-add-exp*[*symmetric*, *OF this*[*symmetric*], *of 2*,
THEN comm-add-exp, *of n*, *unfolded pow-two*]
have $z^{@}n \cdot xp \cdot xp = xp \cdot xp \cdot z^{@}n$
unfolding *rassoc***.**
hence $concat\ ws' \cdot concat\ [x,x,y] = concat\ [x,x,y] \cdot concat\ ws'$
unfolding *con-ws* $\langle concat\ [x,x,y] = xp \cdot xp\rangle$ *rassoc* **by** *simp*
from *xy.is-code*[*OF - - this*[*folded concat-morph*]]

have $ws' \cdot [x,\ x,\ y] = [x,x,y] \cdot ws'$
using *append-in-lists* ‹$ws' \in$ *lists* $\{x,y\}$› **by** *simp*
thus *False*
using ‹$ws' \cdot [x,\ x,\ y] \neq [x,x,y] \cdot ws'$› **by** *fastforce*
qed

then interpret *binary-code z xp*
by (*unfold-locales*)

have ¬ *concat* $(ws' \cdot [x,\ x,\ y]) \bowtie$ *concat* $([x,\ x,\ y] \cdot ws')$
proof (*rule notI*)
assume *concat* $(ws' \cdot [x,\ x,\ y]) \bowtie$ *concat* $([x,\ x,\ y] \cdot ws')$
from *comm-comp-eq*[*OF this*[*unfolded concat-morph*], *unfolded* ‹*concat* [*x*,*x*,*y*] = *xp* · *xp*› *con-ws*]
have $z \ ^{@}\ n \cdot xp^{@}Suc(Suc\ 0) = xp^{@}Suc(Suc\ 0) \cdot z \ ^{@}\ n$
unfolding *pow-Suc pow-zero emp-simps rassoc*.
from *comm-drop-exps*[*OF this*]
show *False*
using ‹$z \cdot xp \neq xp \cdot z$› ‹$2 \leq n$› **by** *force*
qed

— How the xp/z mismatch is reflected by mismatch in lists x,y?
— Looking at the first occurrence of z:

define *lcp-ws* **where** *lcp-ws* = $ws' \cdot [x,x,y] \wedge_p [x,x,y] \cdot ws'$

have *lcp-ws* ∈ *lists* $\{x,y\}$
unfolding *lcp-ws-def* **by** *inlists*

have *lcp-xp-z*: *concat* $(ws' \cdot [x,x,y]) \wedge_p$ *concat* $([x,x,y] \cdot ws') =$ *bin-lcp z* $(x \cdot p)$
unfolding *concat-morph con-ws* ‹*concat* [*x*,*x*,*y*] = *xp* · *xp*› *add-exps*[*symmetric*]
using *bin-lcp-pows*[*OF* ‹0 < *n*›, *of 2*]
unfolding *pow-two pow-pos*[*OF* ‹0 < *n*›] *rassoc xp-def* **by** *force*

have (*concat lcp-ws*) · *bin-lcp x y* = *bin-lcp z* $(x \cdot p)$
proof (*rule xy.bin-code-lcp-concat'*[*OF - -* ‹¬ *concat* $(ws' \cdot [x,\ x,\ y]) \bowtie$ *concat* $([x,\ x,\ y] \cdot ws')$›, *folded lcp-ws-def*, *unfolded lcp-xp-z*, *symmetric*])
show $ws' \cdot [x,\ x,\ y] \in$ *lists* $\{x,\ y\}$ **and** $[x,\ x,\ y] \cdot ws' \in$ *lists* $\{x,\ y\}$
by *inlists*
qed

— Looking at the second occurrence of z:

define ws'' **where** $ws'' = ps \cdot [x,y]$
define *lcp-ws'* **where** *lcp-ws'* = $ws' \cdot ws'' \wedge_p ws'' \cdot ws'$

have *lcp-ws'* ∈ *lists* $\{x,y\}$
unfolding *lcp-ws'-def*
using ‹$ps \in$ *lists* $\{x,\ y\}$› ‹$ws' \in$ *lists* $\{x,\ y\}$› *ws''-def* **by** *inlists*

have *concat ws'' = z · xp*
unfolding *ws''-def xp-def* **using** ‹*concat ps · p = z*› *xxy-root* **by** *fastforce*

have *ws' · ws'' ≠ ws'' · ws'*
proof
assume *ws' · ws'' = ws'' · ws'*
from *arg-cong*[*OF this*, *of concat*, *unfolded concat-morph con-ws*
‹*concat ws'' = z · xp*›,
unfolded lassoc pow-comm, *unfolded rassoc cancel*]
show *False*
using ‹*z · xp ≠ xp · z*› *comm-drop-exp'*[*OF* - ‹*0 < n*›] **by** *blast*
qed

have
lcp-xp-z': *concat* (*ws' · ws''*) ∧$_p$ *concat* (*ws'' · ws'*) = *z · bin-lcp z* (*x · p*)
unfolding *concat-morph con-ws* ‹*concat ws'' = z · xp*› *pow-Suc*
unfolding *lcp-ext-left*[*symmetric*] *bin-lcp-def shifts*
unfolding *rassoc lcp-ext-left cancel*
using *bin-lcp-pows*[*OF* ‹*0 < n*›, *of 1 ε z*$^{@}$(*n*−*1*), *unfolded pow-1*, *folded pow-pos*[*OF* ‹*0 < n*›]]
unfolding *bin-lcp-def xp-def rassoc emp-simps* **by** *linarith*

have *z · bin-lcp z* (*x · p*) = *concat* (*lcp-ws'*) · *bin-lcp x y*
unfolding *lcp-xp-z'*[*symmetric*] *lcp-ws'-def*
proof (*rule xy.bin-code-lcp-concat'*)
show *ws' · ws'' ∈ lists* {*x*, *y*}
unfolding *ws''-def* **using** ‹*ws' · ws' · ws' ∈ lists* {*x*, *y*}› ‹*ps ∈ lists* {*x*,*y*}› **by** *inlists*
thus *ws'' · ws' ∈ lists* {*x*, *y*}
by *inlists*
show ¬ *concat* (*ws' · ws''*) ⋈ *concat* (*ws'' · ws'*)
unfolding *concat-morph con-ws* ‹*concat ws'' = z · xp*› *pow-pos*[*OF* ‹*0 < n*›]
unfolding *rassoc comp-cancel*
unfolding *lassoc pow-pos*[*OF* ‹*0 < n*›, *symmetric*] *pow-pos'*[*OF* ‹*0 < n*›, *symmetric*]
comm-comp-eq-conv
using *comm-drop-exp'*[*OF* - ‹*0 < n*›, *of z n xp*] *non-comm* **by** *argo*
qed

have *concat lcp-ws' = z · concat lcp-ws*
unfolding *cancel-right*[*of concat lcp-ws' bin-lcp x y z · concat lcp-ws*, *symmetric*]
unfolding *rassoc*[*of z*] ‹*concat* (*lcp-ws*) · *bin-lcp x y = bin-lcp z* (*x · p*)› ‹*z · bin-lcp z* (*x · p*) = *concat* (*lcp-ws'*) · *bin-lcp x y*›**..**

have *lcp-ws* ≤*p ws' · [x,x,y]*
unfolding *lcp-ws-def* **using** *longest-common-prefix-prefix1***.**
have *lcp-ws ≠ ws' · [x,x,y]*
unfolding *lcp-ws-def lcp-pref-conv*

using ‹$ws' \cdot [x,\ x,\ y] \neq [x,\ x,\ y] \cdot ws'$› *pref-comm-eq* **by** *blast*
have *lcp-ws* $\leq p$ $ws' \cdot [x,x]$
using *spref-butlast-pref*[*OF* ‹*lcp-ws* $\leq p$ $ws' \cdot [x,x,y]$› ‹*lcp-ws* $\neq$ $ws' \cdot [x,x,y]$›]
unfolding *butlast-append* **by** *simp*
from *prefixE*[*OF pref-prolong*[*OF this* ‹$[x,x] \leq p$ ws'›]]
obtain ws''_1 **where** $ws' \cdot ws' \cdot ws' =$ *lcp-ws* $\cdot\ ws''_1$ **using** *rassoc* **by** *metis*

have $ws' \cdot ps \cdot [x,y] \leq p$ $ws' \cdot ps \cdot [x,y,x]$
by *simp*
from *pref-trans*[*OF pref-trans*[*OF longest-common-prefix-prefix1 this*]]
have *lcp-ws'* $\leq p$ $ws' \cdot ws' \cdot ws'$
unfolding *lcp-ws'-def ws''-def* **using** ‹$ps \cdot vs \leq p$ $ws' \cdot ws'$›[*unfolded cover-xyx*, *unfolded pref-cancel-conv*]
unfolding *pref-cancel-conv*[*symmetric*, *of* $ps \cdot [x,y,x]$ $ws' \cdot ws'$ ws'] **by** *blast*
from *prefixE*[*OF this*]
obtain ws''_2 **where** $ws' \cdot ws' \cdot ws' =$ *lcp-ws'* $\cdot\ ws''_2$**.**

have *concat lcp-ws'*$\cdot$ *concat* $ws''_1 = z \cdot$ *concat*(*lcp-ws*) $\cdot$ *concat* ws''_1
unfolding *lassoc* ‹*concat lcp-ws'* $= z \cdot$ *concat lcp-ws*›**..**
also have ... $= z \cdot$ *concat* $(ws' \cdot ws' \cdot ws')$
unfolding *rassoc* ‹$ws' \cdot ws' \cdot ws' =$ *lcp-ws* $\cdot\ ws''_1$› *concat-morph***..**
also have ... = *concat* $(ws' \cdot ws' \cdot ws') \cdot z$
unfolding *concat-morph con-ws add-exps*[*symmetric*]
pow-Suc[*symmetric*] *pow-Suc'*[*symmetric*]**..**
also have ... = *concat lcp-ws'*$\cdot$ *concat* $ws''_2 \cdot z$
unfolding ‹$ws' \cdot ws' \cdot ws' =$ *lcp-ws'* $\cdot\ ws''_2$› *concat-morph rassoc***..**
finally have *concat* $ws''_1 =$ *concat* $ws''_2 \cdot z$
unfolding *cancel***.**

from *xy.stability*[*of concat* ws''_2 *concat lcp-ws z*,
folded ‹*concat* $ws''_1 =$ *concat* $ws''_2 \cdot z$› ‹*concat lcp-ws'* $= z \cdot$ *concat lcp-ws*›]
have $z \in \langle\{x,\ y\}\rangle$
using ‹$ws' \cdot ws' \cdot ws' =$ *lcp-ws* $\cdot\ ws''_1$› ‹$ws' \cdot ws' \cdot ws' =$ *lcp-ws'* $\cdot\ ws''_2$› ‹$ws' \cdot ws' \cdot ws' \in$ *lists* $\{x,\ y\}$›
append-in-lists-dest append-in-lists-dest' concat-in-hull' **by** *metis*
thus *False*
using ‹$z \notin \langle\{x,y\}\rangle$› **by** *blast*
qed
thus $ws \sim [x,\ x,\ y]$
using ‹$ws \sim ws'$› **by** *blast*
qed

0.5.4 Obtaining primitivity with two squares (refining)

lemma *bin-imprim-both-squares-prim*:
assumes $x \cdot y \neq y \cdot x$
and $ws \in$ *lists* $\{x,\ y\}$
and *primitive ws* **and** $\neg$ *primitive* (*concat ws*)
and $[x,\ x] \leq f$ $ws \cdot ws$

and $[y,\ y] \leq_f ws \cdot ws$
and *primitive* x **and** *primitive* y
shows *False*
proof−
have $x \neq y$ **using** ‹$x \cdot y \neq y \cdot x$›
by *blast*
from *bin-imprim-primitive*[*OF* ‹$x \cdot y \neq y \cdot x$› ‹*primitive* x› ‹*primitive* y›
- ‹$ws \in$ *lists* $\{x,y\}$› ‹*primitive* ws› ‹¬ *primitive* (*concat* ws)› ‹$[x,\ x] \leq_f ws \cdot ws$›]
bin-imprim-primitive[*OF* ‹$x \cdot y \neq y \cdot x$›[*symmetric*] ‹*primitive* y› ‹*primitive* x›
- ‹$ws \in$ *lists* $\{x,y\}$›[*unfolded insert-commute*[*of* x]] ‹*primitive* ws› ‹¬ *primitive* (*concat* ws)›
‹$[y,\ y] \leq_f ws \cdot ws$›]
have $ws \sim [x,\ x,\ y] \vee ws \sim [y,\ y,\ x]$
using ‹$x \cdot y \neq y \cdot x$›
by *force*
hence $|ws| = 3$
using *conjug-len* **by** *force*
note[*simp*] = *sublist-code*(3)
from ‹$|ws| = 3$› ‹$ws \in$ *lists* $\{x,y\}$› ‹$x \neq y$›
‹$[x,\ x] \leq_f ws \cdot ws$› ‹$[y,\ y] \leq_f ws \cdot ws$›
show *False*
by *list-inspection simp-all*
qed

lemma *bin-imprim-both-squares*:
assumes $x \cdot y \neq y \cdot x$
and $ws \in$ *lists* $\{x,\ y\}$
and *primitive* ws **and** ¬ *primitive* (*concat* ws)
and $[x,\ x] \leq_f ws \cdot ws$
and $[y,\ y] \leq_f ws \cdot ws$
shows *False*
proof (*rule bin-imprim-both-squares-prim*)
have $x \neq \varepsilon$ **and** $y \neq \varepsilon$ **and** $x \neq y$
using ‹$x \cdot y \neq y \cdot x$› **by** *blast*+
let *?R* = $\lambda\ x.\ [\varrho\ x]^{@}(e_\varrho\ x)$
define ws' **where** $ws' =$ *concat* (*map ?R* ws)
show $\varrho\ x \cdot \varrho\ y \neq \varrho\ y \cdot \varrho\ x$
using ‹$x \cdot y \neq y \cdot x$›[*unfolded comp-primroot-conv'*[*of* $x\ y$]].
have [*simp*]: $a = x \vee a = y \Longrightarrow [\varrho\ a]\ ^{@}\ e_\varrho\ a \in$ *lists* $\{\varrho\ x,\ \varrho\ y\}$ **for** a
using *insert-iff sing-pow-lists*[*of* - $\{\varrho\ x,\ \varrho\ y\}$] **by** *metis*
show $ws' \in$ *lists* $\{\varrho\ x,\ \varrho\ y\}$
unfolding ws'*-def* **using** ‹$ws \in$ *lists* $\{x,y\}$›
by (*induction* ws, *simp-all*)

— The primitivity of ws' is obtained from the fact that the decompositions into roots is a primitive morphism
interpret *binary-code* $x\ y$
using ‹$x \cdot y \neq y \cdot x$› **by** *unfold-locales*

note[*simp*] = *sublist-code*(*3*)
have |*ws*| ≤ *3* ⟹ *ws* ∈ *lists* {*x*,*y*} ⟹ *x* ≠ *y* ⟹ [*x*, *x*] ≤*f* *ws* · *ws* ⟹ [*y*, *y*] ≤*f* *ws* · *ws* ⟹ *False*
by *list-inspection simp-all*
from *this*[*OF* - ‹ *ws* ∈ *lists* {*x*,*y*}› ‹*x* ≠ *y*› ‹[*x*, *x*] ≤*f* *ws* · *ws*› ‹[*y*, *y*] ≤*f* *ws* · *ws*›] *roots-prim-morph*[*OF* ‹*ws* ∈ *lists* {*x*,*y*}› - ‹*primitive ws*›]
show *primitive ws′*
unfolding *ws′-def* **by** *fastforce*

show ¬ *primitive* (*concat ws′*)
unfolding *ws′-def concat-root-dec-eq-concat*[*OF* ‹*ws* ∈ *lists*{*x*,*y*}›] **by** *fact*

have *concat*(*map ?R* [*x*,*x*]) ≤*f* *ws′* · *ws′* **and** *concat*(*map ?R* [*y*,*y*]) ≤*f* *ws′* · *ws′*
unfolding *ws′-def*
using *concat-mono-fac*[*OF map-mono-sublist*[*OF* ‹[*x*,*x*] ≤*f* *ws* · *ws*›]]
concat-mono-fac[*OF map-mono-sublist*[*OF* ‹[*y*,*y*] ≤*f* *ws* · *ws*›]]
unfolding *concat-morph map-append*.

have *Suc* (*Suc* (e_ϱ *x* + e_ϱ *x* − *2*)) = e_ϱ *x* + e_ϱ *x*
using *Suc-minus2 primroot-exp-nemp*[*OF* ‹*x* ≠ *ε*›] **by** *simp*
have *concat* (*map ?R* [*x*,*x*]) = [*ϱ x*] @ (*Suc* (e_ϱ *x* −*1*) + *Suc* (e_ϱ *x* − *1*))
unfolding *Suc-minus-pos*[*OF primroot-exp-nemp*[*OF* ‹*x* ≠ *ε*›]] **by** (*simp add*: *add-exps*)
hence [*ϱ x*, *ϱ x*] ≤*f* *concat* (*map ?R* [*x*,*x*])
by *auto*
thus [*ϱ x*, *ϱ x*] ≤*f* *ws′* · *ws′*
using *fac-trans*[*OF* - ‹*concat*(*map ?R* [*x*,*x*]) ≤*f* *ws′* · *ws′*›] **by** *blast*

have *Suc* (*Suc* (e_ϱ *y* + e_ϱ *y* − *2*)) = e_ϱ *y* + e_ϱ *y*
using *Suc-minus2 primroot-exp-nemp*[*OF* ‹*y* ≠ *ε*›] **by** *simp*
have *concat* (*map ?R* [*y*,*y*]) = [*ϱ y*] @ (*Suc* (e_ϱ *y* −*1*) + *Suc* (e_ϱ *y* − *1*))
unfolding *Suc-minus-pos*[*OF primroot-exp-nemp*[*OF* ‹*y* ≠ *ε*›]] **by** (*simp add*: *add-exps*)
hence [*ϱ y*, *ϱ y*] ≤*f* *concat* (*map ?R* [*y*,*y*])
by *auto*
thus [*ϱ y*, *ϱ y*] ≤*f* *ws′* · *ws′*
using *fac-trans*[*OF* - ‹*concat*(*map ?R* [*y*,*y*]) ≤*f* *ws′* · *ws′*›] **by** *blast*

show *primitive* (*ϱ x*) **and** *primitive* (*ϱ y*)
using *primroot-prim* ‹*x* ≠ *ε*› ‹*y* ≠ *ε*› **by** *blast+*
qed

0.5.5 Obtaining the square of the longer word (gluing)

lemma *bin-imprim-longer-twice*:
— 1. If there are both squares, then contradiction; 2. If a square is missing: a) if y appears once: the positive conclusion b) if y appears twice, then gluing preserves presence of the longer word at least twice (because both appear twice) and induction yields [x',x',y'] where y' is a suffix of x', a contradiction with primitivity of words

of the form xyxyy;
assumes $x \cdot y \neq y \cdot x$
and $ws \in lists\ \{x, y\}$
and $|y| \leq |x|$
and $count\text{-}list\ ws\ x \geq 2$
and $primitive\ ws$ **and** $\neg\ primitive\ (concat\ ws)$
shows $ws \sim [x,x,y] \wedge primitive\ x \wedge primitive\ y$
using *assms* **proof** (*induction* $|ws|$ *arbitrary*: *x y ws rule*: *less-induct*)
case *less*
then show *?case*
proof (*cases*)
assume $[x, x] \leq_f ws \cdot ws \wedge [y, y] \leq_f ws \cdot ws$
with *bin-imprim-both-squares*[*OF* ‹$x \cdot y \neq y \cdot x$› ‹$ws \in lists\ \{x,y\}$› ‹*primitive ws*› ‹$\neg\ primitive\ (concat\ ws)$›]
have *False* **by** *blast*
thus *?case* **by** *blast*
next
assume *missing-sq*: $\neg\ ([x, x] \leq_f ws \cdot ws \wedge [y, y] \leq_f ws \cdot ws)$
then show *?case*
proof (*cases*)
assume $count\text{-}list\ ws\ y < 2$
with *bin-imprim-single-y*[*OF less.prems*(*1−4*) *this less.prems*(*5−6*)]
show $ws \sim [x,x,y] \wedge primitive\ x \wedge primitive\ y$
by *blast*
next
assume $\neg\ count\text{-}list\ ws\ y < 2$ **hence** $2 \leq count\text{-}list\ ws\ y$ **by** *simp*

— Missing square and two y's allow gluing
define x' **where** $x' = (if\ \neg\ [x, x] \leq_f ws \cdot ws\ then\ x\ else\ y)$
define y' **where** $y' = (if\ \neg\ [x, x] \leq_f ws \cdot ws\ then\ y\ else\ x)$

have $\{x', y'\} = \{x, y\}$
by (*simp add*: *doubleton-eq-iff x′-def y′-def*)
note *cases* = *disjE*[*OF this*[*unfolded doubleton-eq-iff*]]
have $\neg\ [x', x'] \leq_f ws \cdot ws$
using *missing-sq x′-def* **by** *presburger*
have $count\text{-}list\ ws\ x' \geq 2$ **and** $count\text{-}list\ ws\ y' \geq 2$
unfolding *x′-def y′-def* **using** ‹$2 \leq count\text{-}list\ ws\ x$› ‹$2 \leq count\text{-}list\ ws\ y$› **by** *presburger*+
have $x' \cdot\ y' \neq y' \cdot x'$
by (*rule cases*, *simp-all add*: ‹$x \cdot y \neq y \cdot x$› ‹$x \cdot y \neq y \cdot x$›[*symmetric*])
have $x' \neq \varepsilon$ **and** $x' \neq y'$ **and** $x' \cdot y' \neq y'$
using ‹$x' \cdot y' \neq y' \cdot x'$› **by** *auto*

— rotating last if necessary for successful gluing
note *prim-nemp*[*OF* ‹*primitive ws*›]
hence *rot*: $last\ ws = x' \Longrightarrow hd\ ws = x' \Longrightarrow butlast\ ws \cdot [x',x'] \cdot tl\ ws = ws \cdot ws$
using *append-butlast-last-id hd-tl hd-word rassoc* **by** *metis*

from *this*[*THEN facI'*]
have *last ws* = *x'* ⟹ *hd ws* ≠ *x'*
using ‹¬ [*x'*, *x'*] ≤*f ws* · *ws*› **by** *blast*
define *ws'* **where** *ws'* = (*if last ws* ≠ *x' then ws else tl ws* · [*hd ws*])
have *cond*: *ws'* = ε ∨ *last ws'* ≠ *x'* — gluing condition
unfolding *ws'-def* **using** ‹*last ws* = *x'* ⟹ *hd ws* ≠ *x'*› **by** *simp*
have *ws'* ∼ *ws*
unfolding *ws'-def* **using** ‹*ws* ≠ ε› **by** *fastforce*
hence *counts'*: *count-list ws' x'* ≥ *2 count-list ws' y'* ≥ *2*
by (*simp-all add*: ‹*2* ≤ *count-list ws x'*› ‹*2* ≤ *count-list ws y'*› *count-list-conjug*)

— verify induction assumptions of the glued word
let *?ws* = *glue x' ws'*
have *c1*: |*?ws*| < |*ws*|
using *len-glue*[*OF cond*] *conjug-len*[*OF* ‹*ws'* ∼ *ws*›] ‹*count-list ws' x'* ≥ *2*›
by *linarith*
hence *c2*: (*x'* · *y'*) · *y'* ≠ *y'* · *x'* · *y'*
using ‹*x'* · *y'* ≠ *y'* · *x'*› **by** *force*

have *ws'* ≤*f ws* · *ws*
using *conjugE*[*OF* ‹*ws'* ∼ *ws*›] *rassoc sublist-appendI* **by** *metis*
hence ¬ [*x'*, *x'*] ≤*f ws'*
using ‹¬ [*x'*,*x'*] ≤*f ws* · *ws*› **by** *blast*
have *ws'* ∈ *lists* {*x'*,*y'*}
using *conjug-in-lists*[*OF* ‹*ws'* ∼ *ws*› ‹*ws* ∈ *lists* {*x*,*y*}›[*folded* ‹{*x'*,*y'*} = {*x*,*y*}›]].
have *c3*: *?ws* ∈ *lists* {*x'* · *y'*, *y'*}
using *single-bin-glue-in-lists*[*OF cond* ‹¬ [*x'*, *x'*] ≤*f ws'*› ‹*ws'* ∈ *lists* {*x'*,*y'*}›].

have *c4*: *2* ≤ *count-list* (*glue x' ws'*) (*x'* · *y'*)
using ‹*2* ≤ *count-list ws' x'*›
unfolding *count-list-single-bin-glue*(*1*)[*OF* ‹*x'* ≠ ε› ‹*x'* ≠ *y'*› *cond* ‹*ws'* ∈ *lists* {*x'*,*y'*}› ‹¬ [*x'*,*x'*] ≤*f ws'*›].

from ‹*primitive ws*›[*folded conjug-prim-iff*[*OF* ‹*ws'* ∼ *ws*›]]
have *c5*: *primitive* (*glue x' ws'*)
using *prim-bin-glue* [*OF* ‹*ws'* ∈ *lists* {*x'*,*y'*}› ‹*x'* ≠ ε› *cond*] **by** *blast*

have *count-list ws' x'* ≥ *2*
using ‹*count-list ws x* ≥ *2*› ‹*count-list ws y* ≥ *2*› ‹{*x'*, *y'*} = {*x*, *y*}›
count-list-conjug[*OF* ‹*ws'* ∼ *ws*›] *x'-def* **by** *metis*

have *concat* (*glue x' ws'*) = *concat ws'*
by (*simp add*: *cond*)
have *c6*: ¬ *primitive* (*concat* (*glue x' ws'*))
unfolding ‹*concat* (*glue x' ws'*) = *concat ws'*› **using** ‹¬ *primitive* (*concat ws*)› ‹*ws'* ∼ *ws*›
conjug-concat-conjug prim-conjug **by** *metis*
— The claim holds by induction

from *less.hyps*[*OF c1 c2 c3 - c4 c5 c6*]
have *glue x′ ws′* ∼ [*x′* · *y′*, *x′* · *y′*, *y′*] **by** *simp*
— Which is impossible after gluing
from *prim-xyxyy*[*OF* ‹*x′* · *y′* ≠ *y′* · *x′*›] *conjug-prim-iff*[*OF conjug-concat-conjug*[*OF this*]]
have *False*
using ‹¬ *primitive* (*concat* (*glue x′ ws′*))› **by** *simp*
thus *?case* **by** *blast*
qed
qed
qed

lemma *bin-imprim-both-twice*:
assumes *x* · *y* ≠ *y* · *x*
and *ws* ∈ *lists* {*x*, *y*}
and *count-list ws x* ≥ *2*
and *count-list ws y* ≥ *2*
and *primitive ws* **and** ¬ *primitive* (*concat ws*)
shows *False*
proof−
have *x* ≠ *y*
using ‹*x* · *y* ≠ *y* · *x*› **by** *blast*
from *bin-imprim-longer-twice*[*OF assms*(*1*−*2*) - *assms*(*3*) *assms*(*5*−*6*)]
bin-imprim-longer-twice[*OF assms*(*1*)[*symmetric*] *assms*(*2*)[*unfolded insert-commute*[*of x*]] - *assms*(*4*) *assms*(*5*−*6*)]
have *or*: *ws* ∼ [*x*, *x*, *y*] ∨ *ws* ∼ [*y*, *y*, *x*] **by** *linarith*
thus *False*
proof (*rule disjE*)
assume *ws* ∼ [*x*, *x*, *y*]
from ‹*count-list ws y* ≥ *2*›[*unfolded count-list-conjug*[*OF this*]]
show *False*
using ‹*x* ≠ *y*› **by** *force*
next
assume *ws* ∼ [*y*, *y*, *x*]
from ‹*count-list ws x* ≥ *2*›[*unfolded count-list-conjug*[*OF this*]]
show *False*
using ‹*x* ≠ *y*› **by** *force*
qed
qed

0.6 Examples

lemma *x* ≠ *ε* ⟹ *ε* (*x*·*x*) *ε* ∼$_{\mathcal{I}}$ [*x*,*x*]
unfolding *factor-interpretation-def*
by *simp*

lemma assumes *x* = [(*0*::*nat*),*1*,*0*,*1*,*0*] **and** *y* = [*1*,*0*,*0*,*1*]
shows [*0*,*1*] (*x*·*x*) [*1*,*0*] ∼$_{\mathcal{I}}$ [*x*,*y*,*x*]
unfolding *factor-interpretation-def assms* **by** (*simp add*: *suffix-def*)

0.7 Primitivity non-preserving binary code

In this section, we give the final form of imprimitive words over a given binary code $\{x, y\}$. We start with a lemma, then we show that the only possibility is that such word is conjugate with $x \,^{@}\, j \cdot y \,^{@}\, k$.

lemma *bin-imprim-expsE-y*: **assumes** $x \cdot y \neq y \cdot x$ **and**
  $ws \in lists\ \{x,y\}$ **and**
  $2 \leq |ws|$ **and**
  primitive ws **and**
  $\neg$ *primitive* (*concat ws*) **and**
  count-list ws y = *1*
obtains *j k* **where** *1* $\leq$ *j 1* $\leq$ *k j* = *1* $\vee$ *k* = *1*
  $ws \sim [x]^{@}j \cdot [y]^{@}k$
proof–
  have $x \neq y$ **using** ‹$x \cdot y \neq y \cdot x$› **by** *blast*
  obtain *j1 j2* **where** $[x]^{@}j1 \cdot [y] \cdot [x]^{@}j2 = ws$
    using *bin-count-one-decompose*[*OF* ‹*ws* $\in$ *lists* $\{x,y\}$› ‹$x \neq y$› ‹*count-list ws y* = *1*›]**.**
  have *1* $\leq$ *j2* + *j1*
    using ‹$[x] \,^{@}\, j1 \cdot [y] \cdot [x] \,^{@}\, j2 = ws$› ‹*2* $\leq$ $|ws|$› *not-less-eq-eq* **by** *fastforce*
  have $ws \sim [x]^{@}(j2+j1) \cdot [y]^{@}1$
    using *conjugI'*[*of* $[x] \,^{@}\, j1 \cdot [y]\ [x] \,^{@}\, j2$]
    unfolding ‹$[x] \,^{@}\, j1 \cdot [y] \cdot [x] \,^{@}\, j2 = ws$›[*symmetric*] *add-exps rassoc pow-1***.**
  from *that*[*OF* ‹*1* $\leq$ *j2* + *j1*› *- - this*]
  show *?thesis*
    by *blast*
qed

lemma *bin-imprim-expsE*: **assumes** $x \cdot y \neq y \cdot x$ **and**
  $ws \in lists\ \{x,y\}$ **and**
  2 $\leq$ $|ws|$ **and**
  primitive ws **and**
  $\neg$ *primitive* (*concat ws*)
obtains *j k* **where** *1* $\leq$ *j 1* $\leq$ *k j* = *1* $\vee$ *k* = *1*
  $ws \sim [x]^{@}j \cdot [y]^{@}k$
proof–
  note ‹*ws* $\in$ *lists* $\{x,y\}$›[*unfolded insert-commute*[*of x*]]

  from *bin-lists-count-zero*[*OF* ‹*ws* $\in$ *lists* $\{x,y\}$›]
    sing-lists-exp-len[*of ws y*]
    prim-exp-one[*OF* ‹*primitive ws*›, *of* $[y]$ $|ws|$]
  have *count-list ws x* $\neq$ *0*
    using ‹*2* $\leq$ $|ws|$› **by** *fastforce*

  from *bin-lists-count-zero*[*OF* ‹*ws* $\in$ *lists* $\{y,x\}$›]
    sing-lists-exp-len[*of ws x*]
    prim-exp-one[*OF* ‹*primitive ws*›, *of* $[x]$ $|ws|$]
  have *count-list ws y* $\neq$ *0*

using ‹*2* ≤ |*ws*|› **by** *fastforce*

consider *count-list ws x = 1* | *count-list ws y = 1*
using *bin-imprim-both-twice*[*OF* ‹*x* · *y* ≠ *y* · *x*› ‹*ws* ∈ *lists* {*x*,*y*}› - -
‹*primitive ws*› ‹¬ *primitive* (*concat ws*)›]
‹*count-list ws x* ≠ *0*› ‹*count-list ws y* ≠ *0*›
unfolding *One-less-Two-le-iff*[*symmetric*] *less-one*[*symmetric*] **by** *fastforce*
thus *thesis*
proof(*cases*)
assume ‹*count-list ws x = 1*›
from *bin-imprim-expsE-y*[*reversed*, *OF* ‹*x* · *y* ≠ *y* · *x*› ‹*ws* ∈ *lists* {*y*, *x*}› ‹*2* ≤ |*ws*|›
‹*primitive ws*› ‹¬ *primitive* (*concat ws*)› ‹*count-list ws x = 1*›]
show *thesis*
using *that* **by** *metis*
next
assume ‹*count-list ws y = 1*›
from *bin-imprim-expsE-y*[*OF* ‹*x* · *y* ≠ *y* · *x*› ‹*ws* ∈ *lists* {*x*, *y*}› ‹*2* ≤ |*ws*|›
‹*primitive ws*› ‹¬ *primitive* (*concat ws*)› ‹*count-list ws y = 1*›]
show *?thesis*
using *that*.
qed
qed

0.7.1 The target theorem

Given a binary code $\{x, y\}$ such that there is a primitive factorisation ws over it whose concatenation is imprimitive, we finally show that there are integers j and k (depending only on $\{x, y\}$) such that any other such factorisation ws' is conjugate to $[x]^{@}j \cdot [y]^{@}k$.

theorem *bin-imprim-code*: **assumes** *x* · *y* ≠ *y* · *x* **and** *ws* ∈ *lists* {*x*,*y*} **and**
2 ≤ |*ws*| **and** *primitive ws* **and** ¬ *primitive* (*concat ws*)
obtains *j k* **where** *1* ≤ *j* **and** *1* ≤ *k* **and** *j = 1* ∨ *k = 1*
⋀*ws*. *ws* ∈ *lists* {*x*,*y*} ⟹ *2* ≤ |*ws*| ⟹
(*primitive ws* ∧ ¬ *primitive* (*concat ws*) ⟷ *ws* ∼ [*x*]^@*j* · [*y*]^@*k*) **and**
|*y*| ≤ |*x*| ⟹ *2* ≤ *j* ⟹ *j = 2* ∧ *primitive x* ∧ *primitive y* **and**
|*y*| ≤ |*x*| ⟹ *2* ≤ *k* ⟹ *j = 1* ∧ *primitive x*
proof−
obtain *j k* **where** *1* ≤ *j 1* ≤ *k j = 1* ∨ *k = 1*
ws ∼ [*x*]^@*j* · [*y*]^@*k*
using *bin-imprim-expsE*[*OF* ‹*x* · *y* ≠ *y* · *x*›]
using *assms* **by** *metis*

have ¬ *primitive* (*x*^@*j* · *y*^@*k*)
using ‹¬ *primitive* (*concat ws*)›
unfolding *concat-morph concat-sing-pow*
conjug-prim-iff[*OF conjug-concat-conjug*[*OF* ‹*ws* ∼ [*x*] @ *j* · [*y*] @ *k*›]].

from *not-prim-primroot-expE*[*OF this*]
obtain $z\ l$ **where** [*symmetric*]: $z^{@}l = x^{@}j \cdot y^{@}k$ **and** $2 \le l$**.**

show *thesis*
proof (*rule that*[*of j k*])
  show $1 \le j\ 1 \le k\ j = 1 \vee k = 1$ **by** *fact+*

  fix ws'
  assume *hyps*: $ws' \in$ *lists* $\{x,y\}$ $2 \le |ws'|$
  show *primitive* $ws' \wedge \neg$ *primitive* (*concat* ws') $\longleftrightarrow$ $ws' \sim [x]^{@}j \cdot [y]^{@}k$
  proof
    assume *primitive* $ws' \wedge \neg$ *primitive* (*concat* ws')
    hence *prems*: *primitive* ws' $\neg$ *primitive* (*concat* ws') **by** *blast+*
    obtain $j'\ k'$ **where** $1 \le j'\ 1 \le k'\ j' = 1 \vee k' = 1$
      $ws' \sim [x]^{@}j' \cdot [y]^{@}k'$
      using *bin-imprim-expsE*[*OF* ‹$x \cdot y \neq y \cdot x$› *hyps prems*]**.**

    have $\neg$ *primitive* ($x\ ^{@}\ j' \cdot\ y\ ^{@}\ k'$)
      using ‹$\neg$ *primitive* (*concat* ws')›
      unfolding *concat-morph concat-sing-pow*
        conjug-prim-iff[*OF conjug-concat-conjug*[*OF* ‹$ws' \sim [x]\ ^{@}\ j' \cdot [y]\ ^{@}\ k'$›]]**.**

    have $j = j'\ k = k'$
      using *LS-unique*[*OF* ‹$x \cdot y \neq y \cdot x$›
          ‹$1 \le j$› ‹$1 \le k$› ‹$\neg$ *primitive* ($x\ ^{@}\ j \cdot y\ ^{@}\ k$)›
          ‹$1 \le j'$› ‹$1 \le k'$› ‹$\neg$ *primitive* ($x\ ^{@}\ j' \cdot\ y\ ^{@}\ k'$)›]**.**

    show $ws' \sim [x]\ ^{@}\ j \cdot [y]\ ^{@}\ k$
      unfolding ‹$j = j'$› ‹$k = k'$› **by** *fact*
  next
    assume $ws' \sim [x]^{@}j \cdot [y]^{@}k$
    note *conjug-trans*[*OF* ‹$ws \sim [x]^{@}j \cdot [y]^{@}k$› *conjug-sym*[*OF this*]]
    from *prim-conjug*[*OF* ‹*primitive ws*› *this*]
        ‹$\neg$ *primitive* (*concat ws*)›[*unfolded conjug-concat-prim-iff*[*OF* ‹$ws \sim ws'$›]]
    show *primitive* $ws' \wedge \neg$ *primitive* (*concat* ws') **by** *blast*
  qed
next
  assume $|y| \le |x|$
  interpret *LS-len-le x y j k l z*
  by *unfold-locales fact+*

  assume $2 \le j$
  with *jk-small*
  have $k = 1$ **by** *fastforce*
  from *case-j2k1*[*OF* ‹$2 \le j$› *this*]
  show $j = 2 \wedge$ *primitive* $x \wedge$ *primitive* y
    by *blast*
next
  assume $|y| \le |x|$

interpret *LS-len-le x y j k l z*
by *unfold-locales fact+*

assume $2 \leq k$
show $j = 1 \wedge$ *primitive x*
using ‹$2 \leq k$› ‹$j = 1 \vee k = 1$› *case-j1k2-primitive* **by** *auto*
qed
qed

— Formulation in terms of (binary) primitive morphism

definition *bin-imprim-code* **where** *bin-imprim-code x y* $\equiv x \cdot y \neq y \cdot x \wedge (\neg$ *bin-prim x y*)

theorem *bin-imprim-code'*: **assumes** *bin-imprim-code x y*
obtains $j\ k$ **where** $1 \leq j$ **and** $1 \leq k$ **and** $j = 1 \vee k = 1$
$\bigwedge ws.\ ws \in lists\ \{x,y\} \Longrightarrow\ 2 \leq |ws| \Longrightarrow$
(*primitive ws* $\wedge \neg$ *primitive* (*concat ws*) $\longleftrightarrow$ *ws* $\sim [x]^{@}j \cdot [y]^{@}k$) **and**
$|y| \leq |x| \Longrightarrow 2 \leq j \Longrightarrow j = 2 \wedge$ *primitive x* $\wedge$ *primitive y* **and**
$|y| \leq |x| \Longrightarrow 2 \leq k \Longrightarrow j = 1 \wedge$ *primitive x*
proof–
thm *bin-imprim-code*
obtain *ws* **where** $x \cdot y \neq y \cdot x$
and *ws* $\in$ *lists* $\{x,y\}$ **and** $2 \leq |ws|$ **and** *primitive ws* **and** $\neg$ *primitive* (*concat ws*)
using *assms* **unfolding** *bin-imprim-code-def bin-prim-altdef2* **by** *blast*
from *bin-imprim-code*[*OF this*] *that*
show *thesis*
by *blast*
qed

end

theory *Binary-Imprimitive-Decision*
imports
Binary-Code-Imprimitive.Binary-Code-Imprimitive

begin

0.8 Upper bound of the power exponent in the canonical imprimitivity witness

lemma *LS-power-len-ge*:
assumes $y^{@}\ k \cdot x = z^{@}\ l$
and $k * |y| \geq |z| + |y| - 1$
shows $x \cdot y = y \cdot x$

proof (*rule nemp-comm*)
assume $y \neq \varepsilon$
have $y \,^{@}\, k \leq_p z \cdot y \,^{@}\, k$
using ‹$y \,^{@}\, k \cdot x = z \,^{@}\, l$›
by (*blast intro*!: *pref-prod-root*)
moreover have $y \,^{@}\, k \leq_p y \cdot y \,^{@}\, k$
by (*blast intro*!: *pref-pow-ext'*)
moreover have $1 \leq gcd\ |z|\ |y|$
using ‹$y \neq \varepsilon$›
by (*simp flip*: *less-eq-Suc-le*)
from *this* ‹$k * |y| \geq |z| + |y| - 1$›
have $|z| + |y| - (gcd\ |z|\ |y|) \leq k * |y|$
by (*rule le-trans*[*OF diff-le-mono2*])
ultimately have $z \cdot y = y \cdot z$
unfolding *pow-len*[*symmetric*] **by** (*fact per-lemma-comm*)
with ‹$y \,^{@}\, k \cdot x = z \,^{@}\, l$›
show $x \cdot y = y \cdot x$
by (*fact LS-comm*)
qed

lemma *LS-root-len-ge*:
assumes $y \,^{@}\, k \cdot x = z \,^{@}\, l$
and $1 \leq k$ **and** $2 \leq l$
and $x \cdot y \neq y \cdot x$
shows $(k - 1) * |y| + 2 \leq |z|$
proof (*intro leI notI*)
assume $|z| < (k - 1) * |y| + 2$
then have $|z| + |y| \leq Suc\ (k - 1) * |y| + 1$
by *simp*
also have $\ldots = k * |y| + 1$
using ‹$1 \leq k$› **by** *simp*
finally have $k * |y| \geq |z| + |y| - 1$
unfolding *le-diff-conv*.
from ‹$x \cdot y \neq y \cdot x$› *LS-power-len-ge*[*OF* ‹$y \,^{@}\, k \cdot x = z \,^{@}\, l$› *this*]
show *False*..
qed

lemma *LS-root-len-le*:
assumes $y \,^{@}\, k \cdot x = z \,^{@}\, l$
and $1 \leq k$ **and** $2 \leq l$
and $x \cdot y \neq y \cdot x$
shows $|z| \leq |x| + |y| - 2$
proof −
have $|x| + k * |y| = l * |z|$
using *lenarg*[*OF* ‹$y \,^{@}\, k \cdot x = z \,^{@}\, l$›]
by (*simp only*: *pow-len lenmorph add.commute*[*of* $|x|$])
have $|z| \leq (l - 1) * |z|$
using *diff-le-mono*[*OF* ‹$2 \leq l$›, *of 1*] **by** *simp*
also have $\ldots = |x| + k * |y| - |z|$

unfolding *diff-mult-distrib* ‹$|x| + k * |y| = l * |z|$›[*symmetric*] **by** *simp*
also have $\ldots \leq |x| + k * |y| - ((k - 1) * |y| + 2)$
using *LS-root-len-ge*[*OF assms*]
by (*rule diff-le-mono2*)
also have $\ldots \leq |x| + |y| - 2$
using ‹$1 \leq k$› **unfolding** *diff-diff-eq*[*symmetric*]
by (*intro diff-le-mono*) (*simp add*: *le-diff-conv add.assoc diff-mult-distrib*)
finally show $|z| \leq |x| + |y| - 2$.
qed

lemma *LS-exp-le'*:
assumes $y \ ^{@} \ k \cdot x = z \ ^{@} \ l$
and $2 \leq l$
and $x \cdot y \neq y \cdot x$
shows $k \leq (|x| - 4) \ div \ |y| + 2$
proof (*cases* $1 \leq k$)
assume $1 \leq k$
have $|y| > 0$
using ‹$x \cdot y \neq y \cdot x$› **by** *blast*
have $(k - 1) * |y| + 2 \leq |z|$
using *LS-root-len-ge* ‹$y \ ^{@} \ k \cdot x = z \ ^{@} \ l$› ‹$1 \leq k$› ‹$2 \leq l$› ‹$x \cdot y \neq y \cdot x$›.
also have $|z| \leq |x| + |y| - 2$
using *LS-root-len-le* ‹$y \ ^{@} \ k \cdot x = z \ ^{@} \ l$› ‹$1 \leq k$› ‹$2 \leq l$› ‹$x \cdot y \neq y \cdot x$›.
finally have $(k - 1) * |y| + 2 \leq |x| + |y| - 2$.
then have $(k - 1) * |y| + 2 - |y| - 2 \leq |x| + |y| - 2 - |y| - 2$
by (*intro diff-le-mono*)
then have $(k - 1) * |y| + 2 - 2 - |y| \leq |x| - (2 + 2)$
unfolding *diff-commute*[*of - 2* $|y|$] **unfolding** *diff-add-inverse2 diff-diff-eq*.
then have $(k - (1 + 1)) * |y| \leq |x| - 4$
unfolding *diff-add-inverse2 nat-distrib diff-diff-eq mult-1*
by *presburger*
then show $k \leq (|x| - 4) \ div \ |y| + 2$
using ‹$|y| > 0$›
by (*simp only*: *less-eq-div-iff-mult-less-eq one-add-one flip*: *le-diff-conv*)
qed (*simp add*: *trans-le-add2*)

lemma *LS-exp-le*:
assumes $x \cdot y \ ^{@} \ k = z \ ^{@} \ l$
and $2 \leq l$
and $x \cdot y \neq y \cdot x$
shows $k \leq (|x| - 4) \ div \ |y| + 2$
using *LS-exp-le'*[*reversed*, *OF* ‹$x \cdot y \ ^{@} \ k = z \ ^{@} \ l$› ‹$2 \leq l$› ‹$x \cdot y \neq y \cdot x$›[*symmetric*]].

thm *bin-imprim-expsE*
lemma *bin-imprim-code-witnessE*:
assumes $x \cdot y \neq y \cdot x$ **and** $|y| \leq |x|$
and *ws* $\in$ *lists* {*x*,*y*} **and** $2 \leq |ws|$
and *primitive ws* **and** $\neg$ *primitive* (*concat ws*)

```
  obtains   ws ∼ [x, x, y]
  | k where 1 ≤ k and k ≤ (|x| − 4) div |y| + 2
    and ws ∼ [x] · [y] @ k
proof −
  obtain j k
    where 1 ≤ j and 1 ≤ k and j = 1 ∨ k = 1
      and witness-iff: ⋀ws. ws ∈ lists {x,y} ⟹  2 ≤ |ws| ⟹
          (primitive ws ∧ ¬ primitive (concat ws) ⟷ ws ∼ [x] @ j · [y] @ k)
      and
      j-ge: |y| ≤ |x| ⟹ 2 ≤ j ⟹ j = 2 ∧ primitive x ∧ primitive y and
      k-ge: |y| ≤ |x| ⟹ 2 ≤ k ⟹ j = 1 ∧ primitive x
    by (fact bin-imprim-code[OF assms(1, 3 − 6)])
  have ws ∼ [x] @ j · [y] @ k
    using witness-iff[OF ‹ws ∈ lists {x, y}› ‹2 ≤ |ws|›] ‹primitive ws› ‹¬ primitive
(concat ws)›
    by simp
  show thesis
  proof (cases)
    assume 2 ≤ j
    from j-ge[OF ‹|y| ≤ |x|› this] ‹j = 1 ∨ k = 1›
    have j = 2 and k = 1
      by simp-all
    then have ws ∼ [x, x, y]
      using ‹ws ∼ [x] @ j · [y] @ k› by simp
    then show thesis..
  next
    assume ¬ 2 ≤ j
    with ‹1 ≤ j› have j = 1
      by simp
    then have ws ∼ [x] · [y] @ k
      using ‹ws ∼ [x] @ j · [y] @ k› by simp
    then have ¬ primitive (x · y @ k)
      using ‹¬ primitive (concat ws)› unfolding conjug-concat-prim-iff[OF ‹ws ∼
[x] · [y] @ k›]
      by simp
    moreover have x · y @ k ≠ ε
      using ‹x · y ≠ y · x› by (intro notI) simp
    ultimately obtain z l
      where 2 ≤ l and z @ l = x · y @ k
      by (elim not-prim-expE)
    have k ≤ (|x| − 4) div |y| + 2
      using LS-exp-le[OF ‹z @ l = x · y @ k›[symmetric] ‹2 ≤ l› ‹x · y ≠ y · x›].
    from ‹1 ≤ k› this ‹ws ∼ [x] · [y] @ k›
    show thesis..
  qed
qed
```

0.8.1 Optimality of the exponent upper bound

lemma *examples-bound-optimality*:
fixes m k **and** x y z :: *binA list*
assumes $1 \leq m$ **and** $k' = 0 \Longrightarrow m = 1$
defines $x \equiv \mathfrak{a} \cdot \mathfrak{b} \cdot (\mathfrak{b} \cdot (\mathfrak{a} \cdot \mathfrak{b}) \ ^{@} \ m) \ ^{@} \ k' \cdot \mathfrak{b} \cdot \mathfrak{a}$
and $y \equiv \mathfrak{b} \cdot (\mathfrak{a} \cdot \mathfrak{b}) \ ^{@} \ m$
and $z \equiv \mathfrak{a} \cdot \mathfrak{b} \cdot (\mathfrak{b} \cdot (\mathfrak{a} \cdot \mathfrak{b}) \ ^{@} \ m) \ ^{@} \ (k' + 1)$
and $k \equiv k' + 2$
shows $|y| \leq |x|$ **and** $x \cdot y \ ^{@} \ k = z \cdot z$ **and** $k = (|x| - 4) \ div \ |y| + 2$
proof −
obtain m' **where** m: $m = m' + 1$
using ‹$1 \leq m$› **using** *add.commute le-Suc-ex* **by** *blast*
have *x-len*: $|x| = k' * (2 * m + 1) + 4$
unfolding *x-def* **by** (*simp add*: *pow-len*)
have *y-len*: $|y| = 2 * m + 1$
unfolding *y-def* **by** (*simp add*: *pow-len*)
have *z-len*: $|z| = (k' + 1) * (2 * m + 1) + 2$
unfolding *z-def* **by** (*simp add*: *pow-len*)
show $|y| \leq |x|$
using ‹$k' = 0 \Longrightarrow m = 1$› **unfolding** *x-len y-len*
by (*cases* k') (*simp-all add*: *pow-len*)
show $x \cdot y \ ^{@} \ k = z \cdot z$
unfolding *x-def y-def z-def k-def*
by *comparison*
show $k = (|x| - 4) \ div \ |y| + 2$
proof −
have $|x| - 4 = k' * |y|$
unfolding *x-len y-len* **by** *simp*
have $|y| \neq 0$
unfolding *y-def* **by** *blast*
have $k' = (|x| - 4) \ div \ |y|$
unfolding ‹$|x| - 4 = k' * |y|$› *nonzero-mult-div-cancel-right*[*OF* ‹$|y| \neq 0$›]..
then show $k = (|x| - 4) \ div \ |y| + 2$
unfolding *k-def* **by** *blast*
qed
qed

0.9 Characterization of binary primitivity preserving morphisms given by a pair of words

lemma *len-le-not-bin-primE*:
assumes $|y| \leq |x|$
and ¬ *bin-prim x y*
obtains ¬ *primitive* $(x \cdot x \cdot y)$
| k **where** $1 \leq k$ **and** $k \leq (|x| - 4) \ div \ |y| + 2$
and ¬ *primitive* $(x \cdot y \ ^{@} \ k)$
proof (*cases*)
assume $x \cdot y = y \cdot x$

```
  have ¬ primitive (x · x · y)
    using ‹x · y = y · x› ‹|y| ≤ |x|›
    by (cases x ≠ ε) (intro comm-not-prim, simp-all)
  then show thesis..
next
  assume x · y ≠ y · x
  then have x ≠ y
    by blast
  obtain ws where
      ws ∈ lists {x, y} and 2 ≤ |ws| and primitive ws and ¬ primitive (concat ws)
    using ‹¬ bin-prim x y› unfolding bin-prim-concat-prim-pres-conv[OF ‹x ≠ y›]
    by blast
  then consider ws ∼ [x, x, y]
    | k where 1 ≤ k and k ≤ (|x| − 4) div |y| + 2
      and   ws ∼ [x] · [y] @ k
    by (rule bin-imprim-code-witnessE[OF ‹x · y ≠ y · x› ‹|y| ≤ |x|›])
  then show thesis
  proof (cases)
    assume ws ∼ [x, x, y]
    then have ¬ primitive (x · x · y)
      using ‹¬ primitive (concat ws)›
      by (simp add: conjug-concat-prim-iff)
    then show thesis..
  next
    fix k
    assume 1 ≤ k and k ≤ (|x| − 4) div |y| + 2 and ws ∼ [x] · [y] @ k
    have ¬ primitive (x · y @ k)
      using ‹ws ∼ [x] · [y] @ k› ‹¬ primitive (concat ws)›
      by (simp add: conjug-concat-prim-iff)
    from ‹1 ≤ k› ‹k ≤ (|x| − 4) div |y| + 2› this
    show thesis..
  qed
qed

lemma bin-prim-xyk:
  assumes bin-prim x y and 0 < k
  shows primitive (x · y @ k)
proof −
  have primitive ([x] · [y] @ k)
    using bin-prim-code[OF ‹bin-prim x y›]
    by (intro prim-abk) blast
  from bin-prim-concat-prim-pres[OF ‹bin-prim x y› - - this] ‹0 < k›
  show primitive (x · y @ k)
    by (simp add: pow-in-lists)
qed

lemma len-le-bin-prim-iff:
  assumes |y| ≤ |x|
  shows
```

bin-prim x y $\longleftrightarrow$ *primitive* $(x \cdot x \cdot y) \wedge (\forall k.\ 1 \leq k \wedge k \leq (|x| - 4)\ div\ |y| + 2 \longrightarrow primitive\ (x \cdot y\ ^{@}\ k))$
(**is** *bin-prim x y* $\longleftrightarrow$ (*?xxy* $\wedge$ *?xyk*))
proof (*intro iffI*[*OF - contrapos-pp*])
assume *bin-prim x y*
show *?xxy* $\wedge$ *?xyk*
proof (*intro conjI allI impI*)
show *primitive* $(x \cdot x \cdot y)$
using *bin-prim-xyk*[*OF* ‹*bin-prim x y*›[*symmetric*], *of 2*] *conjug-prim-iff'*
by (*simp add*: *conjug-prim-iff'*[*of y*])
show *primitive* $(x \cdot y\ ^{@}\ k)$ **if** $1 \leq k \wedge k \leq (|x| - 4)\ div\ |y| + 2$ **for** *k*
using *bin-prim-xyk*[*OF* ‹*bin-prim x y*›, *of k*] *conjunct1*[*OF that*]
by *fastforce*
qed
next
assume ¬ *bin-prim x y*
then consider ¬ *primitive* $(x \cdot x \cdot y)$
| *k* **where** $1 \leq k$ **and** $k \leq (|x| - 4)\ div\ |y| + 2$
and ¬ *primitive* $(x \cdot y\ ^{@}\ k)$
by (*elim len-le-not-bin-primE*[*OF* ‹$|y| \leq |x|$›])
then show ¬ (*?xxy* $\wedge$ *?xyk*)
by (*cases*) *blast+*
qed

lemma *len-eq-bin-prim-iff*:
assumes $|x| = |y|$
shows *bin-prim x y* $\longleftrightarrow$ *primitive* $(x \cdot y)$
proof
show *bin-prim x y* $\Longrightarrow$ *primitive* $(x \cdot y)$
using *bin-prim-xyk*[*of - - 1*]
by *simp*
assume *primitive* $(x \cdot y)$
then have $x \cdot y \neq y \cdot x$
using *assms eq-append-not-prim* **by** *auto*
from *this bin-uniform-prim-morph*[*OF this* ‹$|x| = |y|$› ‹*primitive* $(x \cdot y)$›]
show *bin-prim x y*
unfolding *bin-prim-altdef2*
by *simp*
qed

theorem *bin-prim-iff*:
bin-prim x y $\longleftrightarrow$
(*if* $|y| < |x|$
then primitive $(x \cdot x \cdot y) \wedge (\forall k.\ 1 \leq k \wedge k \leq (|x| - 4)\ div\ |y| + 2 \longrightarrow primitive\ (x \cdot y\ ^{@}\ k))$
else if $|x| < |y|$
then primitive $(y \cdot y \cdot x) \wedge (\forall k.\ 1 \leq k \wedge k \leq (|y| - 4)\ div\ |x| + 2 \longrightarrow primitive\ (y \cdot x\ ^{@}\ k))$
else primitive $(x \cdot y)$

)
proof (*cases rule*: *linorder-cases*)
assume $|x| < |y|$
then show *?thesis*
unfolding *bin-prim-commutes*[*of x y*]
unfolding *len-le-bin-prim-iff*[*OF less-imp-le*[*OF* ‹$|x| < |y|$›]]
by (*simp only*: *if-not-P*[*OF less-not-sym*] *if-P*)
next
assume $|y| < |x|$
then show *?thesis*
unfolding *len-le-bin-prim-iff*[*OF less-imp-le*[*OF* ‹$|y| < |x|$›]]
by (*simp only*: *if-P*)
next
assume $|x| = |y|$
then show *?thesis*
unfolding *len-eq-bin-prim-iff*[*OF* ‹$|x| = |y|$›]
by *simp*
qed

0.9.1 Code equation for *bin-prim* predicate

context
begin

private lemma *all-less-Suc-conv*: $(\forall\, k < n.\ P\ (Suc\ k)) \longleftrightarrow (\forall\, k \leq n.\ k \geq 1 \longrightarrow P\ k)$
proof (*intro iffI allI impI*)
fix *k*
assume $\forall\, k{<}n.\ P\ (Suc\ k)$ **and** $k \leq n$ **and** $1 \leq k$
then show $P\ k$
by (*elim allE*[*of - k − 1*]) (*simp only*: *Suc-diff-1*)
qed *simp*

lemma *bin-prim-iff'* [*code*]:
bin-prim x y $\longleftrightarrow$
(*if* $|y| < |x|$
then primitive $(x \cdot x \cdot y) \wedge (\forall\, k < (|x| - 4)\ div\ |y| + 2.\ primitive\ (x \cdot y\ ^{@}\ (Suc\ k)))$
else if $|x| < |y|$
then bin-prim y x
else primitive $(x \cdot y)$
)
proof (*cases rule*: *linorder-cases*)
show $|x| < |y| \Longrightarrow$ *?thesis*
unfolding *bin-prim-commutes*[*of x y*]
by *simp*
next
assume $|y| < |x|$
then show *?thesis*

using *len-le-bin-prim-iff*[*OF less-imp-le*[*OF* ‹|*y*| < |*x*|›]]
unfolding *all-less-Suc-conv*[**where** $P = \lambda k.$ *primitive* (- · - @ *k*)]
unfolding *conj-comms*[*of* 1 ≤ -] *imp-conjL*
by (*simp only*: *if-P*)
next
assume |*x*| = |*y*|
then show *?thesis*
unfolding *len-eq-bin-prim-iff*[*OF* ‹|*x*| = |*y*|›]
by *simp*
qed

end
value *bin-prim* (𝔞·𝔟·𝔟·𝔞·𝔞) 𝔟 — True
value *bin-prim* (𝔞·𝔟·𝔟·𝔞) 𝔟 — False
value *bin-prim* (𝔞·𝔟·𝔟·𝔞) (𝔟·𝔞·𝔟·𝔞·𝔟) — False
value *bin-prim* (𝔞·𝔟) (𝔞·𝔟) — False
value *bin-prim* (𝔞·𝔟) (𝔞·𝔟·𝔞·𝔟) — False
value *bin-prim* (𝔞·𝔟·𝔟·𝔞·𝔞) (𝔟·𝔟·𝔟·𝔟·𝔟·𝔟) — True

0.10 Characterization of binary imprimitivity codes

theorem *bin-imprim-code-iff*:
bin-imprim-code x y ⟷ *x* · *y* ≠ *y* · *x* ∧
(*if* |*y*| < |*x*|
then ¬ *primitive* (*x* · *x* · *y*) ∨ (∃ *k*. 1 ≤ *k* ∧ *k* ≤ (|*x*| − 4) *div* |*y*| + 2 ∧ ¬ *primitive* (*x* · *y* @ *k*))
else if |*x*| < |*y*|
then ¬ *primitive* (*y* · *y* · *x*) ∨ (∃ *k*. 1 ≤ *k* ∧ *k* ≤ (|*y*| − 4) *div* |*x*| + 2 ∧ ¬ *primitive* (*y* · *x* @ *k*))
else ¬ *primitive* (*x* · *y*)
)
unfolding *bin-imprim-code-def bin-prim-iff*
by (*simp only*: *de-Morgan-conj not-all not-imp conj-assoc flip*: *if-image*[*of* - *Not*])

value *bin-imprim-code* (𝔞·𝔟·𝔟·𝔞·𝔞) 𝔟 — False
value *bin-imprim-code* (𝔞·𝔟·𝔟·𝔞) 𝔟 — True
value *bin-imprim-code* (𝔞·𝔟·𝔟·𝔞) (𝔟·𝔞·𝔟·𝔞·𝔟) — True
value *bin-imprim-code* (𝔞·𝔟) (𝔞·𝔟) — False
value *bin-imprim-code* (𝔞·𝔟) (𝔞·𝔟·𝔞·𝔟) — False
value *bin-imprim-code* (𝔞·𝔟·𝔟·𝔞·𝔞) (𝔟·𝔟·𝔟·𝔟·𝔟·𝔟) — False

end

References

[1] E. Barbin-Le Rest and M. Le Rest. Sur la combinatoire des codes à deux mots. *Theor. Comput. Sci.*, 41:61–80, 1985.

[2] J. Maňuch. Defect effect of bi-infinite words in the two-element case. *Discret. Math. Theor. Comput. Sci.*, 4(2):273–290, 2001.

[3] J.-C. Spehner. *Quelques problèmes d'extension, de conjugaison et de présentation des sous-monoïdes d'un monoïde libre.* PhD thesis, Université Paris VII, Paris, 1976.