Combinatorics on Words formalized
Basics

Štěpán Holub
Martin Raška
Štěpán Starosta

February 6, 2026

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

Contents

0.1 Arithmetical hints 5

1 Reverse symmetry **9**
1.1 Quantifications and maps 9
1.1.1 Quantifications and reverse 11
1.2 Attributes 11
1.2.1 Cons reversion 11
1.2.2 Final corrections 11
1.2.3 Declaration attribute *reversal-rule* 11
1.2.4 Tracing attribute 12
1.2.5 Rule attribute *reversed* 12
1.3 Declaration of basic reversal rules 18
1.3.1 Pure 18
1.3.2 *HOL.HOL* 18
1.3.3 *HOL.Set* 19
1.3.4 *HOL.List* 19
1.3.5 Reverse Symmetry 21
1.4 21
1.5 Examples 22
1.5.1 Cons and append 22
1.5.2 Induction rules 22
1.5.3 hd, tl, last, butlast 23
1.5.4 set 23
1.5.5 rotate 24

2 Basics of Combinatorics on Words **25**
2.1 Definitions and notations 25
2.1.1 Empty and nonempty word 26
2.1.2 Prefix 26
2.1.3 Suffix 28
2.1.4 Factor 29
2.2 Various elementary lemmas 29
2.2.1 General facts 34
2.2.2 Map injective function 35

2.2.3 Orderings on lists: prefix, suffix, factor . . . 36
2.2.4 On the empty word . . . 36
2.2.5 Counting letters . . . 37
2.2.6 Set inspection method . . . 37
2.3 Length and its properties . . . 37
2.4 List inspection method . . . 40
2.5 Prefix and prefix comparability properties . . . 42
2.5.1 Prefix comparability . . . 45
2.5.2 Minimal and maximal prefix with a given property . . . 50
2.6 Suffix and suffix comparability properties . . . 51
2.6.1 Suffix comparability . . . 52
2.7 Left and Right Quotient . . . 54
2.7.1 Left Quotient . . . 55
2.7.2 Right quotient . . . 58
2.7.3 Left and right quotients combined . . . 58
2.8 Equidivisibility . . . 59
2.9 Longest common prefix . . . 61
2.9.1 Longest common prefix and prefix comparability . . . 66
2.9.2 Longest common suffix . . . 68
2.10 Mismatch . . . 69
2.11 Factor properties . . . 71
2.12 Power and its properties . . . 74
2.12.1 Comparison . . . 82
2.13 Rotation . . . 83
2.14 Lists of words and their concatenation . . . 85
2.15 Commutation . . . 89
2.16 Periods . . . 91
2.16.1 periodic root . . . 91
2.16.2 Maximal root-prefix . . . 96
2.16.3 Period - numeric . . . 97
2.16.4 Period: overview . . . 104
2.16.5 Singleton and its power . . . 104
2.17 Border . . . 109
2.17.1 The shortest border . . . 110
2.17.2 Relation to period and conjugation . . . 112
2.18 The longest border and the shortest period . . . 113
2.18.1 The longest border . . . 113
2.18.2 The shortest period . . . 117
2.19 Primitive words . . . 119
2.20 Primitive root . . . 123
2.20.1 Primitivity and the shortest period . . . 132
2.21 Conjugation . . . 133
2.21.1 Enumerating conjugates . . . 152
2.21.2 General lemmas using conjugation . . . 152

2.22 Element of lists: a method for testing if a word is in lists A . 154
2.23 Reversed mappings . . . 156
2.24 Overlapping powers, periods, prefixes and suffixes . . . 157
2.25 Testing primitivity . . . 174
2.26 Factor interpretation . . . 175
2.26.1 Auxiliary lemmas on suffix and border extension . . . 188
2.27 Computing the Border Array . . . 191
2.27.1 Verification of the computation . . . 193
2.28 Border array . . . 200

3 Submonoids of a free monoid 203
3.1 Generated monoid (also called hull) . . . 203
3.2 Factorization into generators . . . 209
3.2.1 Refinement into a specific decomposition . . . 210
3.3 Basis . . . 212
3.4 Code . . . 219
3.5 Prefix code . . . 225
3.5.1 Suffix code . . . 227
3.5.2 Bifix code . . . 228
3.6 Marked code . . . 229
3.7 Non-overlapping code . . . 230
3.8 Binary code . . . 234
3.8.1 Binary code locale . . . 235
3.9 Two words hull (not necessarily a code) . . . 250
3.9.1 Binary Mismatch tools . . . 252
3.9.2 Applied mismatch . . . 260
3.10 Free hull . . . 261
3.11 Reversing hulls and decompositions . . . 266
3.12 Lists as the free hull of singletons . . . 267
3.13 Various additional lemmas . . . 270
3.13.1 Roots of binary set . . . 270
3.13.2 Other . . . 271

4 Morphisms 273
4.1 One morphism . . . 273
4.1.1 Morphism, core map and extension . . . 273
4.1.2 periodic morphism . . . 278
4.1.3 Non-erasing morphism . . . 280
4.1.4 Code morphism . . . 282
4.1.5 Prefix code morphism . . . 284
4.1.6 Marked morphism . . . 285
4.1.7 Image length . . . 286
4.1.8 Endomorphism . . . 288
4.1.9 Decomposition into primitive roots . . . 290

4.2 Primitivity preserving morphisms . . . 294
4.3 Two morphisms . . . 294
4.3.1 Solutions . . . 294
4.3.2 Coincidence pairs . . . 296
4.3.3 Basics . . . 297
4.3.4 Factor intepretation of morphic images . . . 302
4.3.5 Two nonerasing morphisms . . . 304
4.3.6 Two marked morphisms . . . 311

5 The periodicity Lemma 322
5.1 Main claim . . . 322
5.2 Optimality of the bound by construction of the Fine and Wilf word. . . . 325
5.3 Optimality of the Fine and Wilf word. . . . 333
5.4 Other variants of the periodicity lemma . . . 338

6 Lyndon-Schützenberger Equation 342
6.1 The original result . . . 342
6.2 The original result . . . 342
6.2.1 Some alternative formulations. . . . 349
6.3 Parametric solution of the equation $x \,@\, j \cdot y \,@\, k = z \,@\, l$ . . . 350
6.3.1 Auxiliary lemmas . . . 350
6.3.2 x is longer . . . 351
6.3.3 Putting things together . . . 356
6.3.4 Uniqueness of the imprimitivity witness . . . 362
6.4 The bound on the exponent in Lyndon-Schützenberger equation 366

7 Binary alphabet and binary morphisms 370
7.1 Datatype of a binary alphabet . . . 370
7.2 Binary morphisms . . . 376
7.2.1 Binary periodic morphisms . . . 377
7.3 From two words to a binary morphism . . . 378
7.4 Two binary morphism . . . 379
7.5 Binary code morphism . . . 381
7.5.1 Locale - binary code morphism . . . 381
7.5.2 More translations . . . 391
7.6 Marked binary morphism . . . 392
7.7 Marked version . . . 394
7.8 Two binary code morphisms . . . 398
7.9 Two marked binary morphisms with blocks . . . 406
7.10 Binary primitivity preserving morphism given by a pair of words . . . 408
7.10.1 Translating to to list concatenation . . . 408
7.10.2 Basic properties of *bin-prim* . . . 410

8 Equations on words - basics **411**
8.1 Miscellanea 411
8.1.1 Mismatch additions 411
8.1.2 Conjugate words with conjugate periods 412
8.1.3 Covering uvvu 414
8.1.4 Conjugate words 418
8.1.5 Predicate “commutes” 422
8.1.6 Strong elementary lemmas 424
8.1.7 Binary words without a letter square 424
8.1.8 Three covers 426
8.1.9 Binary Equality Words 439

References **448**

theory *Arithmetical-Hints*
imports *Main*
begin

0.1 Arithmetical hints

In this section we give some specific auxiliary lemmas on natural numbers.

lemma *zero-diff-eq*: $i \leq j \Longrightarrow (0::nat) = j - i \Longrightarrow j = i$
by *simp*

lemma *zero-less-diff'*: $i < j \Longrightarrow j - i \neq (0::nat)$
by *simp*

lemma *nat-prod-le*: $m \neq (0 :: nat) \Longrightarrow m*n \leq k \Longrightarrow n \leq k$
using *le-trans*[*of n m*∗*n k*] **by** *auto*

lemma *get-div*: $(p :: nat) < a \Longrightarrow m = (m * a + p)\ div\ a$
by *simp*

lemma *get-mod*: $(p :: nat) < a \Longrightarrow p = (m * a + p)\ mod\ a$
by *simp*

lemma *plus-one-between*: $(a :: nat) < b \Longrightarrow \neg\ b < a + 1$
by *auto*

lemma *quotient-smaller*: $k \neq (0 :: nat) \Longrightarrow\ b \leq k * b$
by *simp*

thm *mult-right-le-imp-le*

lemma *add-lessD2*: $k + m < (n::nat) \Longrightarrow m < n$
unfolding *add.commute*[*of k*] **using** *add-lessD1*.

lemma *mod-offset*: **assumes** $M \neq (0 :: nat)$
obtains k **where** $n \ mod \ M = (l + k) \ mod \ M$
proof−
have $(l + (M - l \ mod \ M)) \ mod \ M = 0$
using *mod-add-left-eq*[*of l M* $(M - l \ mod \ M)$, *unfolded le-add-diff-inverse*[*OF mod-le-divisor*[*OF assms*[*unfolded neq0-conv*]], *of l*] *mod-self*, *symmetric*].
from *mod-add-left-eq*[*of* $(l + (M - l \ mod \ M))$ *M n*, *symmetric*, *unfolded this add.commute*[*of 0*] *add.comm-neutral*]
have $((l + (M - l \ mod \ M)) + n) \ mod \ M = n \ mod \ M$.
from *that*[*OF this*[*unfolded add.assoc*, *symmetric*]]
show *thesis*.
qed

lemma assumes $q \neq (0::nat)$ **shows** $p \leq p + q - gcd \ p \ q$
using *gcd-le2-nat*[*OF* ‹$q \neq 0$›, *of p*]
by *linarith*

lemma *pos-div-gcd-pos* [*intro*]: **assumes** $(0 :: nat) < p$ **shows** $0 < p \ div \ gcd \ p \ q$
using ‹$0 < p$› *dvd-div-eq-0-iff*[*OF gcd-dvd1*] *gr0I* **by** *metis*

lemma *less-mult-one*: **assumes** $(m-1)*k < k$ **obtains** $m = 0 \mid m = (1::nat)$
using *assms* **by** *fastforce*

lemmas *gcd-le2-pos* = *gcd-le2-nat*[*folded zero-order*(*4*)] **and**
gcd-le1-pos = *gcd-le1-nat*[*folded zero-order*(*4*)]

lemma *ge1-pos-conv*: $1 \leq k \longleftrightarrow 0 < (k::nat)$
by *linarith*

lemma *per-lemma-len-le*: **assumes** *le*: $p + q - gcd \ p \ q \leq (n :: nat)$ **and** $0 < q$ **shows** $p \leq n$
using *le* **unfolding** *add-diff-assoc*[*OF gcd-le2-pos*[*OF* ‹$0 < q$›], *symmetric*] **by** (*rule add-leD1*)

lemma *Suc-less-iff-Suc-le*: $Suc \ n < k \longleftrightarrow Suc \ n \leq k - 1$
by *auto*

lemma *nat-induct-pair*: $P \ 0 \ 0 \Longrightarrow (\bigwedge m \ n. \ P \ m \ n \Longrightarrow P \ m \ (Suc \ n)) \Longrightarrow (\bigwedge m \ n. \ P \ m \ n \Longrightarrow P \ (Suc \ m) \ n) \Longrightarrow P \ m \ n$
by (*induction m arbitrary*: *n*) (*metis nat-induct*, *simp*)

lemma *One-less-Two-le-iff*: $1 < k \longleftrightarrow 2 \leq (k :: nat)$
by *fastforce*

lemma *at-least2-Suc*: **assumes** $2 \leq k$
obtains k' **where** $k = Suc(Suc \ k')$

using *Suc3-eq-add-3* *less-eqE*[*OF assms*] **by** *auto*

lemma *at-least3-Suc*: **assumes** $3 \leq k$
obtains k' **where** $k = Suc(Suc(Suc\ k'))$
using *Suc3-eq-add-3* *less-eqE*[*OF assms*] **by** *auto*

lemmas *not0-SucE*[*elim*] = *not0-implies-Suc*[*THEN exE*]

lemma *le1-SucE*: **assumes** $1 \leq n$
obtains k **where** $n = Suc\ k$ **using** *Suc-le-D*[*OF assms*[*unfolded One-nat-def*]]
by *blast*

lemma *Suc-minus*: $k \neq 0 \Longrightarrow Suc\ (k - 1) = k$
by *simp*

lemma *Suc-minus'*: $1 \leq k \Longrightarrow Suc(k - 1) = k$
by *simp*

lemmas *Suc-minus-pos* = *Suc-diff-1*

lemma *Suc-minus2*: $2 \leq k \Longrightarrow Suc\ (Suc(k - 2)) = k$
by *auto*

lemma *Suc-leE*: **assumes** $Suc\ k \leq n$ **obtains** m **where** $n = Suc\ m$ **and** $k \leq m$
using *Suc-le-D assms* **by** *blast*

lemma *two-three-add-le-mult*: $2 \leq (l::nat) \Longrightarrow 3 \leq k \Longrightarrow k + l + 1 \leq k*l$
unfolding *numeral-nat*
by (*elim Suc-leE*) *simp*

lemma *almost-equal-equal*: **assumes** $(a::\ nat) \neq 0$ **and** $b \neq 0$ **and** *eq*: $k*(a+b) + a = m*(a+b) + b$
shows $k = m$ **and** $a = b$
proof−
show $k = m$
proof (*rule linorder-cases*[*of k m*])
assume $k < m$
from *add-le-mono1*[*OF mult-le-mono1*[*OF Suc-leI*[*OF this*]]]
have $(Suc\ k)*(a + b) + b \leq m*(a+b) + b$.
hence *False*
using ‹$b \neq 0$› **unfolding** *mult-Suc eq*[*symmetric*] **by** *force*
thus *?thesis* **by** *blast*
next
assume $m < k$
from *add-le-mono1*[*OF mult-le-mono1*[*OF Suc-leI*[*OF this*]]]
have $(Suc\ m)*(a + b) + a \leq k*(a+b) + a$.
hence *False*
using ‹$a \neq 0$› **unfolding** *mult-Suc eq* **by** *force*
thus *?thesis* **by** *blast*

qed (*simp*)
thus $a = b$
using *eq* **by** *auto*
qed

lemma *add-le-cancel-ge-right*: $a + b \leq c + d \Longrightarrow d \leq b \Longrightarrow a \leq (c :: nat)$ **for** *a b c d*
by *force*

lemma *add-less-cancel-ge-right*: $a + b \leq c + d \Longrightarrow d < b \Longrightarrow a < (c :: nat)$ **for** *a b c d*
by *force*

lemma *add-le-cancel-ge-left*: $a + b \leq c + d \Longrightarrow c \leq a \Longrightarrow b \leq (d :: nat)$ **for** *a b c d*
by *simp*

lemma *add-less-cancel-ge-left*: $a + b \leq c + d \Longrightarrow c < a \Longrightarrow b < (d :: nat)$ **for** *a b c d*
by *simp*

lemma *crossproduct-le*: **assumes** $(a::nat) \leq b$ **and** $c \leq d$
shows $a*d + b*c \leq a*c + b*d$
proof−
have $b * c \leq b * d + a * c$
using *assms* **by** (*simp add*: *trans-le-add1*)
note *mult-le-mono*[*OF assms*]
have $a * (d - c) \leq b * (d - c)$
using *mult-le-mono1*[*OF* ‹$a \leq b$›]**.**
hence $a * d - a * c \leq b * d - b * c$
using *diff-mult-distrib2* **by** *metis*
hence $a * d \leq b * d - b * c + a * c$
using *le-diff-conv* **by** *blast*
hence $a * d \leq (b * d + a * c) - b * c$
by (*simp add*: ‹$c \leq d$›)
hence $a * d + b * c \leq (b * d + a * c) - b * c + b * c$
by *simp*
thus *?thesis*
using ‹$b * c \leq b * d + a * c$› **by** *force*
qed

lemma (**in** *linorder*) *le-less-cases*: $(a \leq b \Longrightarrow P) \Longrightarrow (b < a \Longrightarrow P) \Longrightarrow P$
by (*metis local.not-less*)

end

theory *Reverse-Symmetry*
imports *Main*

begin

Chapter 1

Reverse symmetry

This theory deals with a mechanism which produces new facts on lists from already known facts by the reverse symmetry of lists, induced by the mapping *rev*. It constructs the rule attribute "reversed" which produces the symmetrical fact using so-called reversal rules, which are rewriting rules that may be applied to obtain the symmetrical fact. An example of such a reversal rule is the already existing *rev ys* @ *rev xs* = *rev* (*xs* @ *ys*). Some additional reversal rules are given in this theory.

The symmetrical fact 'A[reversed]' is constructed from the fact 'A' in the following manner: 1. each schematic variable *xs* of type *'a list* is instantiated by *rev xs*; 2. constant Nil is substituted by *rev* []; 3. each quantification of *'a list* type variable $\bigwedge xs.\ P\ xs$ is substituted by (logically equivalent) quantification $\bigwedge xs.\ P\ (rev\ xs)$, similarly for $\forall$, $\exists$ and $\exists!$ quantifiers; each bounded quantification of *'a list* type variable $\forall\, xs{\in}A.\ P\ xs$ is substituted by (logically equivalent) quantification $\forall\, xs{\in}rev\ `\ A.\ P\ (rev\ xs)$, similarly for bounded $\exists$ quantifier; 4. simultaneous rewrites according to a the current list of reversal rules are performed; 5. final correctional rewrites related to reversion of (#) are performed.

List of reversal rules is maintained by declaration attribute "reversal_rule" with standard "add" and "del" options.

See examples at the end of the file.

1.1 Quantifications and maps

lemma *all-surj-conv*: **assumes** *surj f* **shows** $(\bigwedge x.\ PROP\ P\ (f\ x)) \equiv (\bigwedge y.\ PROP\ P\ y)$
proof
  fix *y* **assume** $\bigwedge x.\ PROP\ P\ (f\ x)$
  then have *PROP P* (*f* (*inv f y*))**.**
  then show *PROP P y* **unfolding** *surj-f-inv-f*[*OF assms*]**.**
qed

lemma *All-surj-conv*: **assumes** *surj f* **shows** $(\forall x.\ P\ (f\ x)) \longleftrightarrow (\forall y.\ P\ y)$
proof (*intro iffI allI*)
fix y **assume** $\forall x.\ P\ (f\ x)$
then have $P\ (f\ (inv\ f\ y))$**..**
then show $P\ y$ **unfolding** *surj-f-inv-f*[*OF assms*]**.**
qed *simp*

lemma *Ex-surj-conv*: **assumes** *surj f* **shows** $(\exists x.\ P\ (f\ x)) \longleftrightarrow (\exists y.\ P\ y)$
proof
assume $\exists x.\ P\ (f\ x)$
then obtain x **where** $P\ (f\ x)$**..**
then show $\exists x.\ P\ x$**..**
next
assume $\exists y.\ P\ y$
then obtain y **where** $P\ y$**..**
then have $P\ (f\ (inv\ f\ y))$ **unfolding** *surj-f-inv-f*[*OF assms*]**.**
then show $\exists x.\ P\ (f\ x)$**..**
qed

lemma *Ex1-bij-conv*: **assumes** *bij f* **shows** $(\exists!x.\ P\ (f\ x)) \longleftrightarrow (\exists!y.\ P\ y)$
proof
have *imp*: $\exists!y.\ Q\ y$ **if** *bij*: *bij g* **and** *ex1*: $\exists!x.\ Q\ (g\ x)$ **for** $g\ Q$
proof −
from *ex1E*[*OF ex1*, *rule-format*]
obtain x **where** *ex*: $Q\ (g\ x)$ **and** *uniq*: $\bigwedge x'.\ Q\ (g\ x') \Longrightarrow x' = x$ **by** *blast*
{
fix y **assume** $Q\ y$
then have $Q\ (g\ (inv\ g\ y))$ **unfolding** *surj-f-inv-f*[*OF bij-is-surj*[*OF bij*]]**.**
from *uniq*[*OF this*] **have** $x = inv\ g\ y$**..**
then have $y = g\ x$ **unfolding** *bij-inv-eq-iff*[*OF bij*]**..**
}
with *ex* **show** $\exists!y.\ Q\ y$**..**
qed
show $\exists!x.\ P\ (f\ x) \Longrightarrow \exists!y.\ P\ y$ **using** *imp*[*OF assms*]**.**
assume $\exists!y.\ P\ y$
then have $\exists!y.\ P\ (f\ (inv\ f\ y))$ **unfolding** *surj-f-inv-f*[*OF bij-is-surj*[*OF assms*]]**.**
from *imp*[*OF bij-imp-bij-inv*[*OF assms*] *this*]
show $\exists!x.\ P\ (f\ x)$**.**
qed

lemma *Ball-inj-conv*: **assumes** *inj f* **shows** $(\forall y{\in}f\ `\ A.\ P\ (inv\ f\ y)) \longleftrightarrow (\forall x{\in}A.\ P\ x)$
using *ball-simps*(*9*)[*of f A* $\lambda y.\ P\ (inv\ f\ y)$] **unfolding** *inv-f-f*[*OF assms*]**.**

lemma *Bex-inj-conv*: **assumes** *inj f* **shows** $(\exists y{\in}f\ `\ A.\ P\ (inv\ f\ y)) \longleftrightarrow (\exists x{\in}A.\ P\ x)$
using *bex-simps*(*7*)[*of f A* $\lambda y.\ P\ (inv\ f\ y)$] **unfolding** *inv-f-f*[*OF assms*]**.**

1.1.1 Quantifications and reverse

lemma *rev-involution'*: *rev* ∘ *rev* = *id*
by *auto*

lemma *rev-inv*: *inv rev* = *rev*
using *inv-unique-comp*[*OF rev-involution' rev-involution'*].

1.2 Attributes

context
begin

1.2.1 Cons reversion

definition *snocs* :: *'a list* ⇒ *'a list* ⇒ *'a list*
where *snocs xs ys* = *xs* @ *ys*

1.2.2 Final corrections

lemma *snocs-snocs*: *snocs* (*snocs xs* (*y* # *ys*)) *zs* = *snocs xs* (*y* # *snocs ys zs*)
unfolding *snocs-def* **by** *simp*

lemma *snocs-Nil*: *snocs* [] *xs* = *xs*
unfolding *snocs-def* **by** *simp*

lemma *snocs-is-append*: *snocs xs ys* = *xs* @ *ys*
unfolding *snocs-def*..

private lemmas *final-correct1* =
snocs-snocs

private lemmas *final-correct2* =
snocs-Nil

private lemmas *final-correct3* =
snocs-is-append

1.2.3 Declaration attribute *reversal-rule*

ML ‹
structure Reversal-Rules =
Named-Thms(
val name = @{*binding reversal-rule*}
val description = *Rules performing reverse*−*symmetry transformation on theorems on lists*
)
›

attribute-setup *reversal-rule* =

‹*Attrib.add-del*
(*Thm.declaration-attribute Reversal-Rules.add-thm*)
(*Thm.declaration-attribute Reversal-Rules.del-thm*)›
maintaining a list of reversal rules

1.2.4 Tracing attribute

ML ‹
*val reversed-trace = Config.declare-bool (reversed-trace, **here**) (K false);*
val enable-tracing = Config.put-generic reversed-trace true
val tracing-attr = Thm.declaration-attribute (K enable-tracing)
val tracing-parser : attribute context-parser = Scan.lift (Scan.succeed tracing-attr)
›

attribute-setup *reversed-trace = tracing-parser reversed trace configuration*

1.2.5 Rule attribute *reversed*

private lemma *rev-Nil*: *rev [] ≡ []*
by *simp*

private lemma *map-Nil*: *map f [] ≡ []*
by *simp*

private lemma *image-empty*: *f ‘ Set.empty ≡ Set.empty*
by *simp*

definition *COMP* :: *('b ⇒ prop) ⇒ ('a ⇒ 'b) ⇒ 'a ⇒ prop* (**infixl** *oo 55*)
where *F oo g ≡ (λx. F (g x))*

lemma *COMP-assoc*: *F oo (f o g) ≡ (F oo f) oo g*
unfolding *COMP-def o-def*.

private lemma *image-comp-image*: *(‘) f ∘ (‘) g ≡ (‘) (f ∘ g)*
unfolding *comp-def image-comp*.

private lemma *rev-involution*: *rev ∘ rev ≡ id*
unfolding *comp-def rev-rev-ident id-def*.

private lemma *map-involution*: **assumes** *f ∘ f ≡ id* **shows** *(map f) ∘ (map f) ≡ id*
unfolding *map-comp-map ‹f ∘ f ≡ id› List.map.id*.

private lemma *image-involution*: **assumes** *f ∘ f ≡ id* **shows** *(image f) ∘ (image f) ≡ id*
unfolding *image-comp-image ‹f ∘ f ≡ id› image-id*.

private lemma *rev-map-comm*: *rev ∘ map f ≡ map f ∘ rev*
unfolding *comp-def rev-map*.

private lemma *involut-comm-comp*: **assumes** *f o f* ≡ *id* **and** *g o g* ≡ *id* **and** *f o g* ≡ *g o f*
shows (*f* ∘ *g*) ∘ (*f* ∘ *g*) ≡ *id*
by (*simp add*: *comp-assoc comp-assoc*[*symmetric*] *assms*)

private lemma *rev-map-involution*: **assumes** *g o g* ≡ *id*
shows (*rev* ∘ *map g*) ∘ (*rev* ∘ *map g*) ≡ *id*
using *involut-comm-comp*[*OF rev-involution map-involution*[*OF* ‹*g o g* ≡ *id*›] *rev-map-comm*].

private lemma *prop-abs-subst*: **assumes** *f o f* ≡ *id* **shows** (λ*x*. *F* (*f x*)) *oo f* ≡ (λ*x*. *F x*)
unfolding *COMP-def o-apply*[*symmetric*] **unfolding** ‹*f o f* ≡ *id*› *id-def*.

private lemma *prop-abs-subst-comm*: **assumes** *f o f* ≡ *id* **and** *g o g* ≡ *id* **and** *f o g* ≡ *g o f*
shows (λ*x*. *F* (*f* (*g x*))) *oo* (*f o g*) ≡ (λ*x*. *F x*)
unfolding ‹*f o g* ≡ *g o f*› **unfolding** *COMP-assoc*
unfolding *prop-abs-subst*[*OF* ‹*g o g* ≡ *id*›, *of* λ*x*. *F* (*f x*)] *prop-abs-subst*[*OF* ‹*f o f* ≡ *id*›].

private lemma *prop-abs-subst-rev-map*: **assumes** *g o g* ≡ *id*
shows (λ*x*. *F* (*rev* (*map g x*))) *oo* (*rev o map g*) ≡ (λ*x*. *F x*)
using *prop-abs-subst-comm*[*OF rev-involution map-involution*[*OF* ‹*g o g* ≡ *id*›] *rev-map-comm*].

private lemma *obj-abs-subst*: **assumes** *f o f* ≡ *id* **shows** (λ*x*. *F* (*f x*)) *o f* ≡ (λ*x*. *F x*)
unfolding *comp-def* **unfolding** *o-apply*[*of f*, *symmetric*] ‹*f o f* ≡ *id*› *id-def*.

private lemma *obj-abs-subst-comm*: **assumes** *f o f* ≡ *id* **and** *g o g* ≡ *id* **and** *f o g* ≡ *g o f*
shows (λ*x*. *F* (*f* (*g x*))) *o* (*f o g*) ≡ (λ*x*. *F x*)
unfolding ‹*f o g* ≡ *g o f*› **unfolding** *comp-assoc*[*symmetric*]
unfolding *obj-abs-subst*[*OF* ‹*g o g* ≡ *id*›, *of* λ*x*. *F* (*f x*)] *obj-abs-subst*[*OF* ‹*f o f* ≡ *id*›].

private lemma *obj-abs-subst-rev-map*: **assumes** *g o g* ≡ *id*
shows (λ*x*. *F* (*rev* (*map g x*))) *o* (*rev o map g*) ≡ (λ*x*. *F x*)
using *obj-abs-subst-comm*[*OF rev-involution map-involution*[*OF* ‹*g o g* ≡ *id*›] *rev-map-comm*].

ML ‹

local
fun comp-const T = *Const*(***const-name***‹*comp*›, (*T* −−> *T*) −−> (*T* −−> *T*) −−> *T* −−> *T*)
fun rev-const T = *Const*(***const-name***‹*rev*›, *T* −−> *T*)
fun map-const S T = *Const*(***const-name***‹*map*›, (*S* −−> *S*) −−> *T* −−> *T*)

```
  fun image-const S T = Const(const-name‹image›, (S --> S) --> T -->
T)

  val rev-Nil-thm = @{thm rev-Nil}
  val map-Nil-thm = @{thm map-Nil}
  val image-empty-thm = @{thm image-empty}
  val rev-involut-thm = @{thm rev-involution}
  val map-involut-thm = @{thm map-involution}
  val image-involut-thm = @{thm image-involution}
  val rev-map-comm-thm = @{thm rev-map-comm}
  val involut-comm-comp-thm = @{thm involut-comm-comp}
  fun abs-subst-thm T =
    if T = propT then @{thm prop-abs-subst} else @{thm obj-abs-subst}
  fun abs-subst-rev-map-thm T =
   if T = propT then @{thm prop-abs-subst-rev-map} else @{thm obj-abs-subst-rev-map}

  fun comp T f gs = fold (fn f => fn g =>
        (comp-const T $ f $ g)) gs f
  fun app ctxt gs ct = fold-rev (fn g => fn ct' =>
        Thm.apply (Thm.cterm-of ctxt g) ct') gs ct
in

  fun subst ctxt T ct =
    let
      fun tr (T as Type(type-name‹list›, [S])) = [rev-const T] @ (
          case tr S of
            [] => []
          | (g :: gs') => [map-const S T $ comp S g gs'])
        | tr (T as Type(type-name‹set›, [S])) = (
          case tr S of
            [] => []
          | (g :: gs') => [image-const S T $ comp S g gs'])
        | tr - = []
    in app ctxt (tr T) ct end

  fun abs-cv T U ct =
    let
      fun tr-eq (T as Type(type-name‹list›, [S])) =
            rev-involut-thm :: (
            case tr-eq S of
              [] => []
            | [f-eq] => [f-eq RS map-involut-thm]
            | [f-eq, g-eq] =>
                [([f-eq, g-eq, rev-map-comm-thm] MRS involut-comm-comp-thm) RS
map-involut-thm])
        | tr-eq (T as Type(type-name‹set›, [S])) = (
            case tr-eq S of
              [] => []
            | [f-eq] => [f-eq RS image-involut-thm]
```

```
        | [f-eq, g-eq] =>
            [([f-eq, g-eq, rev-map-comm-thm] MRS involut-comm-comp-thm) RS
image-involut-thm])
      | tr-eq - = []
  in case tr-eq T of
    [] => Thm.reflexive ct
  | [f-inv] =>
      [Thm.reflexive ct, Thm.symmetric (f-inv RS abs-subst-thm U)] MRS transi-
tive-thm
  | [f-inv, g-inv] =>
      [Thm.reflexive ct, Thm.symmetric (g-inv RS abs-subst-rev-map-thm U)] MRS
transitive-thm
  end;

 fun Nil-cv ctxt T ct =
  ((Conv.try-conv o Conv.arg-conv o Conv.rewr-conv) map-Nil-thm
  then-conv (Conv.try-conv o Conv.rewr-conv) rev-Nil-thm) (subst ctxt T ct)
    |> Thm.symmetric

 fun empty-cv ctxt T ct =
  (Conv.try-conv o Conv.rewr-conv) image-empty-thm (subst ctxt T ct)
    |> Thm.symmetric

end

fun initiate-cv ctxt ct =
 case Thm.term-of ct of
  - $ - => Conv.comb-conv (initiate-cv ctxt) ct
 | Abs(-, T, b) => (Conv.abs-conv (initiate-cv o snd) ctxt then-conv abs-cv T
(fastype-of b)) ct
 | Const(const-name‹Nil›, T) => Nil-cv ctxt T ct
 | Const(const-name‹bot›, T as Type(type-name‹set›, -)) => empty-cv ctxt T
ct
 | - => Thm.reflexive ct
›

ML ‹

fun trace-rule-prems-proof ctxt rule goals rule-prems rule' =
 if not (Config.get ctxt reversed-trace) then () else
  let
    val ctxt-prems = Raw-Simplifier.prems-of ctxt
    val np = Thm.nprems-of rule
    val np' = Thm.nprems-of rule'
    val pretty-term = Syntax.pretty-term ctxt
    val pretty-thm = pretty-term o Thm.prop-of
    val success = rule-prems |> List.all is-some
    val sendback = Active.make-markup Markup.sendbackN
      {implicit = false, properties = [Markup.padding-command]}
```

```
    val - = [
      [
        Trying to use conditional reverse rule: |> Pretty.para,
        rule |> pretty-thm
      ] |> Pretty.fbreaks |> Pretty.block,
      [(if null ctxt-prems
        then No context premises.
        else Context premises:
       ) |> Pretty.para
      ] @ (
        ctxt-prems |> map (Pretty.item o single o pretty-thm)
      ) |> Pretty.fbreaks |> Pretty.block,
      ( if success then [
        Successfully derived unconditional reverse rule: |> Pretty.para,
        rule' |> pretty-thm
      ] else [
         Unable to prove " ^ string-of-int np ^ " out of " ^ string-of-int np' ^ " rule
premises:\n
         |> Pretty.para
      ] @ (
        (goals ~~ rule-prems) |> map-filter (
          fn (goal, NONE) => SOME ([
              lemma |> Pretty.str, Pretty.brk 1,
              goal |> pretty-term |> Pretty.quote, Pretty.fbrk,
              sorry |> Pretty.str
            ] |> curry Pretty.blk 0 |> Pretty.mark sendback |> single |> Pretty.item)
            | - => NONE
        )
      )) |> Pretty.fbreaks |> Pretty.block
    ] |> Pretty.chunks |> Pretty.string-of |> tracing
  in () end

fun full-resolve ctxt prem i =
  let
    val tac = resolve-tac ctxt [prem] THEN-ALL-NEW blast-tac ctxt
  in rule-by-tactic ctxt (tac i) end

fun prover method ss ctxt rule =
  let
    val ctxt-prems = Raw-Simplifier.prems-of ctxt
    val rule-prems' = Logic.strip-imp-prems (Thm.prop-of rule)
    val goals = rule-prems' |> map (fn prem =>
      Logic.list-implies (map Thm.prop-of ctxt-prems, prem));
    val ctxt' = ctxt |> put-simpset ss
    fun prove t = SOME (Goal.prove ctxt' [] [] t
        (fn {context = goal-ctxt, prems} => NO-CONTEXT-TACTIC goal-ctxt
(method goal-ctxt prems)))
      handle ERROR - => NONE
    val ths = goals |> map prove
```

```
    val gen-ctxt-prems = map (Variable.gen-all ctxt) ctxt-prems
    fun full-resolve1 prem = full-resolve ctxt prem 1
    val rule-prems = ths |> (map o Option.map) (fold full-resolve1 gen-ctxt-prems)
    val rule' = (fold o curry) (
      fn (SOME th, rule') => rule' |> full-resolve1 th
      | (NONE, rule') => Drule.rotate-prems 1 rule'
      ) rule-prems rule
    val nprems = Thm.nprems-of rule'
    val - = trace-rule-prems-proof ctxt rule goals rule-prems rule'
  in if nprems = 0 then SOME rule' else NONE end

fun rewrite - - [] = Thm.reflexive
  | rewrite method ctxt thms =
      let
        val p = prover method (simpset-of ctxt)
        val ctxt' = Raw-Simplifier.init-simpset thms ctxt
      in
        Raw-Simplifier.rewrite-cterm (true, true, true) p ctxt'
      end

fun rewrite-rule method ctxt = Conv.fconv-rule o rewrite method ctxt;

fun meta-reversal-rules ctxt extra =
  map (Local-Defs.meta-rewrite-rule ctxt) (extra @ Reversal-Rules.get ctxt)

fun reverse method extra-rules context th =
  let
    val ctxt = Context.proof-of context
   val final-correct1 = map (Local-Defs.meta-rewrite-rule ctxt) @{thms final-correct1}
   val final-correct2 = map (Local-Defs.meta-rewrite-rule ctxt) @{thms final-correct2}
   val final-correct3 = map (Local-Defs.meta-rewrite-rule ctxt) @{thms final-correct3}
    val rules = meta-reversal-rules ctxt extra-rules
    val cvars = Thm.add-vars th Vars.empty
    val cvars' = Vars.map ((subst ctxt o snd)) cvars
    val th-subst = Thm.instantiate (TVars.empty, cvars') th
    val ((-, [th-import]), ctxt') = Variable.import true [th-subst] ctxt
    val th-init = th-import |> Conv.fconv-rule (initiate-cv ctxt')
    val th-rev = th-init |> rewrite-rule method ctxt' rules
    val th-corr = th-rev
          |> Raw-Simplifier.rewrite-rule ctxt' final-correct1
          |> Raw-Simplifier.rewrite-rule ctxt' final-correct2
          |> Raw-Simplifier.rewrite-rule ctxt' final-correct3
    val th-export = th-corr |> singleton (Variable.export ctxt' ctxt)
  in
    Drule.zero-var-indexes th-export
  end

val default-method = SIMPLE-METHOD o CHANGED-PROP o auto-tac
```

val solve-arg = Args.$$$ solve

val extra-rules-parser = Scan.optional (Scan.lift (Args.add −− Args.colon) |−− Attrib.thms) []

val solve-parser = Scan.lift (Scan.optional
(solve-arg −− Args.colon |−− Method.parse >> (fst #> Method.evaluate))
default-method
)

val reversed = extra-rules-parser −− solve-parser
>> (fn (ths, method) => Thm.rule-attribute [] (reverse method ths))
›

attribute-setup *reversed =*
‹reversed›
Transforms the theorem on lists to reverse−symmetric version

end

1.3 Declaration of basic reversal rules

1.3.1 Pure

lemma *all-surj-conv' [reversal-rule]*: **assumes** *surj f* **shows** *Pure.all (P oo f) ≡ Pure.all P*
unfolding *COMP-def* **using** *all-surj-conv[OF assms]*.

1.3.2 *HOL.HOL*

lemmas *[reversal-rule] = rev-is-rev-conv inj-eq*

— All
lemma *All-surj-conv' [reversal-rule]*: **assumes** *surj f* **shows** *All (P o f) = All P*
unfolding *comp-def* **using** *All-surj-conv[OF assms]*.

— Ex
lemma *Ex-surj-conv' [reversal-rule]*: **assumes** *surj f* **shows** *Ex (P o f) ⟷ Ex P*
unfolding *comp-def* **using** *Ex-surj-conv[OF assms]*.

— Ex1
lemma *Ex1-bij-conv' [reversal-rule]*: **assumes** *bij f* **shows** *Ex1 (P o f) ⟷ Ex1 P*
unfolding *comp-def* **using** *Ex1-bij-conv[OF assms]*.

— If
lemma *if-image [reversal-rule]*: *(if P then f u else f v) = f (if P then u else v)*
by *simp*

1.3.3 *HOL.Set*

lemma *collect-image*: *Collect* $(P \circ f) = f -` (Collect\ P)$
by *fastforce*

lemma *collect-image'* [*reversal-rule*]: **assumes** $f \circ f = id$ **shows** *Collect* $(P \circ f)$ $= f\ `\ (Collect\ P)$
unfolding *collect-image*
unfolding *bij-vimage-eq-inv-image*[*OF o-bij*[*OF assms assms*]]
unfolding *inv-unique-comp*[*OF assms assms*]**..**

— *Ball*
lemma *Ball-image* [*reversal-rule*]: **assumes** $(g \circ f)\ `\ A = A$ **shows** *Ball* $(f\ `\ A)$ $(P \circ g) = Ball\ A\ P$
unfolding *Ball-image-comp*[*symmetric*] *image-comp* ‹$(g \circ f)\ `\ A = A$›**..**

— *Bex*
lemma *Bex-image-comp*: *Bex* $(f\ `\ A)\ g = Bex\ A\ (g \circ f)$
by *simp*

lemma *Bex-image* [*reversal-rule*]: **assumes** $(g \circ f)\ `\ A = A$ **shows** *Bex* $(f\ `\ A)$ $(P$ $\circ\ g) = Bex\ A\ P$
unfolding *Bex-image-comp*[*symmetric*] *image-comp* ‹$(g \circ f)\ `\ A = A$›**..**

— *insert*
lemma *insert-image* [*reversal-rule*]: *insert* $(f\ x)\ (f\ `\ X) = f\ `\ (insert\ x\ X)$
by *blast*

— *List.member*
lemmas [*reversal-rule*] = *inj-image-mem-iff*

— $(\subseteq)$
lemmas [*reversal-rule*] = *inj-image-subset-iff*

1.3.4 *HOL.List*

lemmas [*reversal-rule*] = *set-rev set-map*

— $(\#)$
lemma *Cons-rev*: $a\ \#\ rev\ u = rev\ (snocs\ u\ [a])$
unfolding *snocs-def* **by** *simp*

lemma *Cons-map*: $(f\ x)\ \#\ (map\ f\ xs) = map\ f\ (x\ \#\ xs)$
using *list.simps*(*9*)[*symmetric*]**.**

lemmas [*reversal-rule*] = *Cons-rev Cons-map*

— *hd*
lemmas [*reversal-rule*] = *hd-rev hd-map*

— *tl*
lemma *tl-rev*: *tl* (*rev xs*) = *rev* (*butlast xs*)
using *butlast-rev*[*of rev xs*, *symmetric*] **unfolding** *rev-swap rev-rev-ident*.

lemmas [*reversal-rule*] = *tl-rev map-tl*[*symmetric*]

— *last*
lemmas [*reversal-rule*] = *last-rev last-map*

— *butlast*
lemmas [*reversal-rule*] = *butlast-rev map-butlast*[*symmetric*]

— *List.coset*
lemma *coset-rev*: *List.coset* (*rev xs*) = *List.coset xs*
by *simp*

lemma *coset-map*: **assumes** *bij f* **shows** *List.coset* (*map f xs*) = *f* ‘ *List.coset xs*
using *bij-image-Compl-eq*[*OF* ‹*bij f*›, *symmetric*] **unfolding** *coset-def set-map*.

lemmas [*reversal-rule*] = *coset-rev coset-map*

— (@)
lemmas [*reversal-rule*] = *rev-append*[*symmetric*] *map-append*[*symmetric*]

— *concat*
lemma *concat-rev-map-rev*: *concat* (*rev* (*map rev xs*)) = *rev* (*concat xs*)
using *rev-concat*[*symmetric*] **unfolding** *rev-map*.

lemma *concat-rev-map-rev′*: *concat* (*rev* (*map* (*rev* ∘ *f*) *xs*)) = *rev* (*concat* (*map f xs*))
unfolding *map-comp-map*[*symmetric*] *o-apply* **using** *concat-rev-map-rev*.

lemmas [*reversal-rule*] = *concat-rev-map-rev concat-rev-map-rev′*

— *drop*
lemmas [*reversal-rule*] = *drop-rev drop-map*

— *take*
lemmas [*reversal-rule*] = *take-rev take-map*

— (!)
lemmas [*reversal-rule*] = *rev-nth nth-map*

— *List.insert*
lemma *list-insert-map* [*reversal-rule*]:
assumes *inj f* **shows** *List.insert* (*f x*) (*map f xs*) = *map f* (*List.insert x xs*)
unfolding *List.insert-def set-map inj-image-mem-iff*[*OF* ‹*inj f*›] *Cons-map if-image*..

— *List.union*

lemma *list-union-map* [*reversal-rule*]:
  assumes *inj f* **shows** *List.union* (*map f xs*) (*map f ys*) = *map f* (*List.union xs ys*)
proof (*induction xs arbitrary*: *ys*)
  case (*Cons a xs*)
   show *?case* **using** *Cons.IH* **unfolding** *List.union-def Cons-map*[*symmetric*] *fold.simps*(*2*) *o-apply*
    unfolding *list-insert-map*[*OF* ‹*inj f*›]**.**
qed (*simp add*: *List.union-def*)

  — *length*
lemmas [*reversal-rule*] = *length-rev length-map*

  — *rotate*
lemmas [*reversal-rule*] = *rotate-rev rotate-map*

  — *lists*
lemma *rev-in-lists*: *rev u* ∈ *lists A* ⟷ *u* ∈ *lists A*
  by *auto*

lemma *map-in-lists*: *inj f* ⟹ *map f u* ∈ *lists* (*f* ‘ *A*) ⟷ *u* ∈ *lists A*
  by (*simp add*: *lists-image inj-image-mem-iff inj-mapI*)

lemmas [*reversal-rule*] = *rev-in-lists map-in-lists*

  — *list-all*
lemmas [*reversal-rule*] = *list-all-rev*

  — *list-ex*
lemmas [*reversal-rule*] = *list-ex-rev*

1.3.5 Reverse Symmetry

lemma *snocs-map* [*reversal-rule*]: *snocs* (*map f u*) [*f a*] = *map f* (*snocs u* [*a*])
  unfolding *snocs-def* **by** *simp*

1.4

lemma *bij-rev*: *bij rev*
  using *o-bij*[*OF rev-involution*′ *rev-involution*′]**.**

lemma *bij-map*: *bij f* ⟹ *bij* (*map f*)
  using *bij-betw-def inj-mapI lists-UNIV lists-image* **by** *metis*

lemma *surj-map*: *surj f* ⟹ *surj* (*map f*)
  using *lists-UNIV lists-image* **by** *metis*

lemma *bij-image*: *bij f* ⟹ *bij* (*image f*)
  using *bij-betw-Pow* **by** *force*

lemma *inj-image*: *inj f* $\Longrightarrow$ *inj* (*image f*)
by (*simp add*: *inj-on-image*)

lemma *surj-image*: *surj f* $\Longrightarrow$ *surj* (*image f*)
using *Pow-UNIV image-Pow-surj* **by** *metis*

lemmas [*simp*] =
bij-rev
bij-is-inj
bij-is-surj
bij-comp
inj-compose
comp-surj
bij-map
inj-mapI
surj-map
bij-image
inj-image
surj-image

1.5 Examples

context
begin

1.5.1 Cons and append

private lemma *example-Cons-append*:
assumes *xs* = [*a*, *b*] **and** *ys* = [*b*, *a*, *b*]
shows *xs* @ *xs* @ *xs* = *a* # *b* # *a* # *ys*
using *assms* **by** *simp*

thm
example-Cons-append
example-Cons-append[*reversed*]
example-Cons-append[*reversed*, *reversed*]

thm
not-Cons-self
not-Cons-self[*reversed*]

thm
neq-Nil-conv
neq-Nil-conv[*reversed*]

1.5.2 Induction rules

thm

list-nonempty-induct
list-nonempty-induct[*reversed*] *list-nonempty-induct*[*reversed*, **where** $P{=}\lambda x.\ P\ (rev\ x)$ **for** P, *unfolded rev-rev-ident*]

thm
list-induct2
list-induct2[*reversed*] *list-induct2*[*reversed*, **where** $P{=}\lambda x\ y.\ P\ (rev\ x)\ (rev\ y)$ **for** P, *unfolded rev-rev-ident*]

1.5.3 hd, tl, last, butlast

thm
hd-append
hd-append[*reversed*]
last-append

thm
length-tl
length-tl[*reversed*]
length-butlast

thm
hd-Cons-tl
hd-Cons-tl[*reversed*]
append-butlast-last-id
append-butlast-last-id[*reversed*]

1.5.4 set

thm
hd-in-set
hd-in-set[*reversed*]
last-in-set

thm
set-ConsD
set-ConsD[*reversed*]

thm
split-list-first
split-list-first[*reversed*]

thm
split-list-first-prop
split-list-first-prop[*reversed*]
split-list-first-prop[*reversed*, *unfolded append-assoc append-Cons append-Nil*]
split-list-last-prop

thm
split-list-first-propE

split-list-first-propE[*reversed*]
split-list-first-propE[*reversed*, *unfolded append-assoc append-Cons append-Nil*]
split-list-last-propE

1.5.5 rotate

private lemma *rotate1-hd-tl*: $xs \neq [] \Longrightarrow rotate\ 1\ xs = tl\ xs\ @\ [hd\ xs]$
by (*cases xs*) *simp-all*

thm
rotate1-hd-tl
rotate1-hd-tl[*reversed*]

end

end

theory *CoWBasic*
imports *HOL−Library.Sublist Arithmetical-Hints Reverse-Symmetry HOL−Eisbach.Eisbach-Tools List-Power.List-Power*
begin

Chapter 2

Basics of Combinatorics on Words

Combinatorics on Words, as the name suggests, studies words, finite or infinite sequences of elements from a, usually finite, alphabet. The essential operation on finite words is the concatenation of two words, which is associative and noncommutative. This operation yields many simply formulated problems, often in terms of *equations on words*, that are mathematically challenging.

See for instance [1] for an introduction to Combinatorics on Words, and [**?**, 5, 6] as another reference for Combinatorics on Words. This theory deals exclusively with finite words and provides basic facts of the field which can be considered as folklore.

The most natural way to represent finite words is by the type *'a list*. From an algebraic viewpoint, lists are free monoids. On the other hand, any free monoid is isomoporphic to a monoid of lists of its generators. The algebraic point of view and the combinatorial point of view therefore overlap significantly in Combinatorics on Words.

2.1 Definitions and notations

First, we introduce elementary definitions and notations.

The concatenation (@) of two finite lists/words is the very basic operation in Combinatorics on Words, its notation is usually omitted. In this field, a common notation for this operation is ·, which we use and add here.

notation *append* (**infixr** · *65*)

lemmas *rassoc* = *append-assoc*
lemmas *lassoc* = *append-assoc*[*symmetric*]

We add a common notation for the length of a given word $|w|$.

notation
length (|-| *1000*) — note that it's bold |
notation (*latex* **output**)
length (|-|)
notation *longest-common-prefix* (**infixr** $\wedge_p$ *61*) — provided by Sublist.thy

2.1.1 Empty and nonempty word

As the word of length zero [] or [] will be used often, we adopt its frequent notation ε in this formalization.

notation *Nil* (ε)

named-theorems *emp-simps pow-list-Nil*
lemmas [*emp-simps*] = *append-Nil2 append-Nil list.map*(*1*) *list.size*(*3*)

2.1.2 Prefix

The property of being a prefix shall be frequently used, and we give it yet another frequent shorthand notation. Analogously, we introduce shorthand notations for non-empty prefix and strict prefix, and continue with suffixes and factors.

notation *prefix* (**infixl** $\leq p$ *50*)

lemmas *prefI'*[*intro*] = *prefixI*

lemma *prefI*[*intro*]: $p \cdot s = w \Longrightarrow p \leq p\ w$
by *auto*

lemma *prefD*: $u \leq p\ v \Longrightarrow \exists\ z.\ v = u \cdot z$
unfolding *prefix-def***.**

definition *prefix-comparable* :: *'a list* ⇒ *'a list* ⇒ *bool* (**infixl** $\bowtie$ *50*)
where (*prefix-comparable u v*) ≡ $u \leq p\ v \vee v \leq p\ u$

lemma *pref-compI1*: $u \leq p\ v \implies u \bowtie v$
unfolding *prefix-comparable-def***..**

lemma *pref-compI2*: $v \leq p\ u \Longrightarrow u \bowtie v$
unfolding *prefix-comparable-def***..**

lemma *pref-compE* [*elim*]: **assumes** $u \bowtie v$ **obtains** $u \leq p\ v \mid v \leq p\ u$
using *assms* **unfolding** *prefix-comparable-def***..**

lemma *pref-compI*[*intro*]: $u \leq p\ v \vee v \leq p\ u \Longrightarrow u \bowtie v$

unfolding *prefix-comparable-def*
by *simp*

notation *strict-prefix* (**infixl** $<_p$ *50*)
notation (*latex* **output**) *strict-prefix* (**infixl** $<_p$ *50*)

lemmas [*simp*] = *strict-prefix-def*

interpretation *lcp*: *semilattice-order* ($\wedge_p$) *prefix strict-prefix*
proof
fix *a b c* :: *'a list*
show $(a \wedge_p b) \wedge_p c = a \wedge_p b \wedge_p c$
by (*rule prefix-order.antisym*)
(*meson longest-common-prefix-max-prefix longest-common-prefix-prefix1 longest-common-prefix-prefix2 prefix-order.trans*)+
show $a \wedge_p b = b \wedge_p a$
by (*simp add*: *longest-common-prefix-max-prefix longest-common-prefix-prefix1 longest-common-prefix-prefix2 prefix-order.antisym*)
show $a \wedge_p a = a$
by (*simp add*: *longest-common-prefix-max-prefix longest-common-prefix-prefix1 prefix-order.eq-iff*)
show $a \leq p\ b = (a = a \wedge_p b)$
by (*metis longest-common-prefix-max-prefix longest-common-prefix-prefix1 longest-common-prefix-prefix2 prefix-order.dual-order.eq-iff*)
thus $(a <p\ b) = (a = a \wedge_p b \wedge a \neq b)$
by *simp*
qed

lemma *sprefI1*[*intro*]: $v = u \cdot z \Longrightarrow z \neq \varepsilon \Longrightarrow u <p\ v$
by *simp*

lemma *sprefI1'*[*intro*]: $u \cdot z = v \Longrightarrow z \neq \varepsilon \Longrightarrow u <p\ v$
by *force*

lemma *sprefI2*[*intro*]: $u \leq p\ v \Longrightarrow |u| < |v| \Longrightarrow u <p\ v$
by *force*

lemmas *sprefI*[*intro*] = *strict-prefixI*

lemma *sprefD*: $u <p\ v \Longrightarrow u \leq p\ v \wedge u \neq v$
by *auto*

lemmas *sprefD1*[*dest*] = *prefix-order.strict-implies-order* **and**
sprefD2 = *prefix-order.less-imp-neq*

lemmas *sprefE*[*elim?*] = *strict-prefixE*

lemma *spref-exE*[*elim*]: **assumes** $u <p\ v$ **obtains** z **where** $u \cdot z = v$ **and** $z \neq \varepsilon$

using *assms* **unfolding** *strict-prefix-def prefix-def* **by** *blast*

lemma *pref-comp-sprefI*: $u \bowtie v \implies \neg\ u \leq_p v \implies v <_p u$
by *auto*

2.1.3 Suffix

notation *suffix* (**infixl** $\leq_s$ *50*)
notation (*latex* **output**) *suffix* ($\leq_s$)

lemma *sufI*[*intro*]: $p \cdot s = w \implies s \leq_s w$
by (*auto simp add*: *suffix-def*)

lemma *sufD*[*elim*]: $u \leq_s v \implies \exists\ z.\ z \cdot u = v$
by (*auto simp add*: *suffix-def*)

notation *strict-suffix* (**infixl** $<_s$ *50*)
notation (*latex* **output**) *strict-suffix* ($<_s$)
lemmas [*simp*] = *strict-suffix-def*

lemmas [*intro*] = *suffix-order.le-neq-trans*

lemmas *ssufI* = *suffix-order.le-neq-trans*

lemma *ssufI1*[*intro*]: $u \cdot v = w \implies u \neq \varepsilon \implies v <_s w$
by *blast*

lemma *ssufI2*[*intro*]: $u \leq_s v \implies length\ u < length\ v \implies u <_s v$
by *blast*

lemma *ssufE*: $u <_s v \implies (u \leq_s v \implies u \neq v \implies thesis) \implies thesis$
by *auto*

lemma *ssufD*[*elim*]: $u <_s v \implies u \leq_s v \wedge u \neq v$
by *auto*

lemmas *ssufD1*[*elim*] = *suffix-order.strict-implies-order* **and**
ssufD2[*elim*] = *suffix-order.less-imp-neq*

definition *suffix-comparable* :: *'a list* ⇒ *'a list* ⇒ *bool* (**infixl** $\bowtie_s$ *50*)
where (*suffix-comparable u v*) ⟷ (*rev u*) ⋈ (*rev v*)

lemma *suf-compI1*[*intro*]: $u \leq_s v \implies u \bowtie_s v$
by (*simp add*: *pref-compI suffix-comparable-def suffix-to-prefix*)

lemma *suf-compI2*[*intro*]: $v \leq_s u \implies u \bowtie_s v$
by (*simp add*: *pref-compI suffix-comparable-def suffix-to-prefix*)

2.1.4 Factor

A *sublist* of some word is in Combinatorics of Words called a factor. We adopt a common shorthand notation for the property of being a factor, strict factor and nonempty factor (the latter we also define).

notation *sublist* (**infixl** $\leq_f$ *50*)
notation (*latex* **output**) *sublist* ($\leq_f$)
lemmas *fac-def* = *sublist-def*

notation *strict-sublist* (**infixl** $<_f$ *50*)
notation (*latex* **output**) *strict-sublist* ($<_f$)
lemmas *strict-factor-def*[*simp*] = *strict-sublist-def*

definition *nonempty-factor* (**infixl** $\leq_{nf}$ *60*) **where** *nonempty-factor-def*[*simp*]: $u \leq_{nf} v \equiv u \neq \varepsilon \wedge (\exists\ p\ s.\ p \cdot u \cdot s = v)$
notation (*latex* **output**) *nonempty-factor* ($\leq_{nf}$)

lemmas *facI* = *sublist-appendI*

lemma *facI'*: $a \cdot u \cdot b = w \Longrightarrow u \leq_f w$
by *auto*

lemma *facE*[*elim*]: **assumes** $u \leq_f v$
obtains $p\ s$ **where** $v = p \cdot u \cdot s$
using *assms* **unfolding** *fac-def*
by *blast*

lemma *facE'*[*elim*]: **assumes** $u \leq_f v$
obtains $p\ s$ **where** $p \cdot u \cdot s = v$
using *assms* **unfolding** *fac-def*
by *blast*

2.2 Various elementary lemmas

thm *drop-eq-Nil*

lemma *exE2*: $\exists\ x\ y.\ P\ x\ y \Longrightarrow (\bigwedge x\ y.\ P\ x\ y \Longrightarrow thesis) \Longrightarrow thesis$
by *auto*

lemmas *concat-morph* = *concat-append*

lemmas *cancel* = *same-append-eq* **and**
cancel-right = *append-same-eq*

lemmas *disjI* = *verit-and-neg*(*3*)

lemma *rev-in-conv*: $rev\ u \in A \longleftrightarrow u \in rev\ `\ A$

by *force*

lemmas *map-rev-involution*[*simp*] = *list.map-comp*[*of rev rev*, *unfolded rev-involution'* *list.map-id*]

lemma *map-rev-lists-rev*: *map rev ‘ (lists (rev ‘ A)) = lists A*
unfolding *lists-image*[*of rev*] *image-comp*
by *simp*

lemma *inj-on-map-lists*: **assumes** *inj-on f A*
shows *inj-on (map f) (lists A)*
proof
fix *xs ys*
assume *xs ∈ lists A* **and** *ys ∈ lists A* **and** *map f xs = map f ys*
have *x = y* **if** *x ∈ set xs* **and** *y ∈ set ys* **and** *f x = f y* **for** *x y*
using *in-listsD*[*OF ‹xs ∈ lists A›*, *rule-format*, *OF ‹x ∈ set xs›*]
in-listsD[*OF ‹ys ∈ lists A›*, *rule-format*, *OF ‹y ∈ set ys›*]
‹inj-on f A›[*unfolded inj-on-def*, *rule-format*, *OF - - ‹f x = f y›*] **by** *blast*
from *list.inj-map-strong*[*OF this ‹map f xs = map f ys›*]
show *xs = ys*.
qed

lemma *bij-lists*: *bij-betw f X Y ⟹ bij-betw (map f) (lists X) (lists Y)*
unfolding *bij-betw-def* **using** *inj-on-map-lists lists-image* **by** *metis*

lemma *concat-sing'*: *concat [r] = r*
by *simp*

lemma *concat-sing*: **assumes** *s = [a]* **shows** *concat s = a*
using *concat-sing'* **unfolding** *‹s = [a]›*.

lemma *rev-sing*: *rev [x] = [x]*
by *simp*

lemma *hd-word*: *a#ws = [a] · ws*
by *simp*

lemma *hd-Cons-append*[*intro*,*simp*]: *hd ((a#v) · u) = a*
by *simp*

lemma *pref-singE*: **assumes** *p ≤p [a]* **obtains** *p = ε | p = [a]*
using *assms* **unfolding** *prefix-Cons* **by** *fastforce*

lemma *map-hd*: *map f (a#v) = [f a] · (map f v)*
by *simp*

lemma *hd-tl*: *w ≠ ε ⟹ [hd w] · tl w = w*
by *simp*

lemma *hd-tlE*: **assumes** $w \neq \varepsilon$
obtains $a\ w'$ **where** $w = a\#w'$
using *exE2*[*OF assms*[*unfolded neq-Nil-conv*]].

lemma *hd-tl-lenE*: **assumes** $0 < |w|$
obtains $a\ w'$ **where** $w = a\#w'$
using *exE2*[*OF assms*[*unfolded length-greater-0-conv neq-Nil-conv*]].

lemma *long-list-tl*[*intro*]: **assumes** $1 < |w|$
shows $tl\ w \neq \varepsilon$
using *list.size*(3) *zero-less-diff'*[*OF assms*, *folded length-tl*[*of w*]] **by** *force*

lemma *long-last-tl*: **assumes** $1 < |w|$ **shows** $last\ (tl\ w) = last\ w$
using *last-tl*[*OF disjI2*[*OF long-list-tl*[*OF assms*]]].

lemma *hd-middle-last*: **assumes** $1 < |w|$
shows $[hd\ w] \cdot butlast\ (tl\ w) \cdot [last\ w] = w$
unfolding *long-last-tl*[*OF assms*, *symmetric*]
append-butlast-last-id[*OF long-list-tl*[*OF assms*]]
using *hd-tl*[*of w*] *list.sel*(2) *long-list-tl*[*OF assms*] **by** *blast*

lemma *hd-pref*[*intro*]: $w \neq \varepsilon \Longrightarrow [hd\ w] \leq p\ w$
using *hd-tl* **by** *blast*

lemma *pref-hd-pref*[*intro*]: $u \cdot v = w \Longrightarrow u \neq w \Longrightarrow\ u \cdot [hd\ v] \leq p\ w$
by *force*

lemma *pref-hd-pref'*[*intro*]: $u \cdot v = w \Longrightarrow v \neq \varepsilon \Longrightarrow\ u \cdot [hd\ v] \leq p\ w$
by *force*

lemma *Nil-take-Nil* [*simp*]: $u = \varepsilon \Longrightarrow take\ p\ u = \varepsilon$
using *take-Nil*[*of p*] **by** *blast*

lemma *add-nth*: **assumes** $n < |w|$ **shows** $(take\ n\ w) \cdot [w!n] \leq p\ w$
using *take-is-prefix*[*of Suc n w*, *unfolded take-Suc-conv-app-nth*[*OF assms*]].

lemma *hd-pref'*: **assumes** $w \neq \varepsilon$ **shows** $[w!0] \leq p\ w$
using *hd-pref*[*OF* ‹$w \neq \varepsilon$›, *folded hd-conv-nth*[*OF* ‹$w \neq \varepsilon$›, *symmetric*]].

lemma *sub-lists-mono*: $A \subseteq B \Longrightarrow x \in lists\ A \Longrightarrow x \in lists\ B$
by *auto*

lemma *lists-hd-in-set*[*simp*]: $us \neq \varepsilon \Longrightarrow us \in lists\ Q \Longrightarrow hd\ us \in Q$
by *fastforce*

lemma *lists-last-in-set*[*simp*]: $us \neq \varepsilon \Longrightarrow us \in lists\ Q \Longrightarrow last\ us \in Q$
by *fastforce*

lemma *replicate-in-lists*: *replicate k z* ∈ *lists* {*z*}
by (*induction k*) *auto*

lemma *tl-in-lists*: **assumes** *us* ∈ *lists A* **shows** *tl us* ∈ *lists A*
using *suffix-lists*[*OF suffix-tl assms*].

lemmas *lists-butlast* = *tl-in-lists*[*reversed*]

lemma *tl-set*: *x* ∈ *set* (*tl a*) ⟹ *x* ∈ *set a*
using *list.sel*(*2*) *list.set-sel*(*2*) **by** *metis*

lemma *drop-take-inv*: *n* ≤ |*u*| ⟹ *drop n* (*take n u* · *w*) = *w*
by *simp*

lemma *split-list-long*: **assumes** *1* < |*us*| **and** *u* ∈ *set us*
obtains *xs ys* **where** *us* = *xs* · [*u*] · *ys* **and** *xs*·*ys*≠ε
proof−
obtain *xs ys* **where** *us* = *xs* · [*u*] · *ys*
using *split-list-first*[*OF* ‹*u* ∈ *set us*›] **by** *auto*
hence *xs*·*ys*≠ε
using ‹*1* < |*us*|› **by** *auto*
from *that*[*OF* ‹*us* = *xs* · [*u*] · *ys*› *this*]
show *thesis*.
qed

lemma *flatten-lists*: *G* ⊆ *lists B* ⟹ *xs* ∈ *lists G* ⟹ *concat xs* ∈ *lists B*
by (*induct xs*, *simp-all add*: *subset-iff*)

lemma *concat-map-sing-ident*: *concat* (*map* (λ *x*. [*x*]) *xs*) = *xs*
by *auto*

lemma *hd-concat-tl*: **assumes** *ws* ≠ ε **shows** *hd ws* · *concat* (*tl ws*) = *concat ws*
using *concat.simps*(*2*)[*of hd ws tl ws*, *unfolded list.collapse*[*OF* ‹*ws* ≠ ε›], *symmetric*].

lemma *concat-butlast-last*: **assumes** *nemp*: *ws* ≠ ε **shows** *concat* (*butlast ws*) · *last ws* = *concat ws*
using *arg-cong*[*OF append-butlast-last-id*[*OF assms*], *of concat*, *unfolded concat-morph concat-sing*[*symmetric*]].

lemma *spref-butlast-pref*: **assumes** *u* ≤*p v* **and** *u* ≠ *v* **shows** *u* ≤*p butlast v*
using *butlast-append prefixE*[*OF* ‹*u* ≤*p v*›] ‹*u* ≠ *v*› *append-Nil2 prefixI* **by** *metis*

lemma *last-concat*: *xs* ≠ ε ⟹ *last xs* ≠ ε ⟹ *last* (*concat xs*) = *last* (*last xs*)
using *concat-butlast-last last-appendR* **by** *metis*

lemma *concat-last-suf*: *ws* ≠ ε ⟹ *last ws* ≤*s concat ws*
using *concat-butlast-last* **by** *blast*

lemma *concat-hd-pref*: $ws \neq \varepsilon \Longrightarrow hd\ ws \leq p\ concat\ ws$
using *hd-concat-tl* **by** *blast*

lemma *set-nemp-concat-nemp*: **assumes** $ws \neq \varepsilon$ **and** $\varepsilon \notin set\ ws$ **shows** $concat\ ws \neq \varepsilon$
using ‹$\varepsilon \notin set\ ws$› *last-in-set*[*OF* ‹$ws \neq \varepsilon$›] *concat-butlast-last*[*OF* ‹$ws \neq \varepsilon$›] **by** *fastforce*

lemmas *takedrop* = *append-take-drop-id*

lemma *drop-neq* [*intro*]: $w \neq \varepsilon \Longrightarrow 0 < n \Longrightarrow drop\ n\ w \neq w$
using *takedrop*[*of n w*] **by** *force*

lemma *suf-drop-conv*: $u \leq s\ w \longleftrightarrow drop\ (|w| - |u|)\ w = u$
using *suffix-take append-take-drop-id same-append-eq suffix-drop* **by** *metis*

lemma *comm-rev-iff*: $rev\ u \cdot rev\ v = rev\ v \cdot rev\ u \longleftrightarrow u \cdot v = v \cdot u$
unfolding *rev-append*[*symmetric*] *rev-is-rev-conv eq-ac*(*1*)[*of* $u \cdot v$] **by** *blast*

lemma *rev-induct2*:
$[\![\ P\ []\ []; \bigwedge x\ xs.\ P\ (xs \cdot [x])\ []; \bigwedge y\ ys.\ P\ []\ (ys \cdot [y]); \bigwedge x\ xs\ y\ ys.\ P\ xs\ ys \Longrightarrow P\ (xs \cdot [x])\ (ys \cdot [y])\ ]\!] \Longrightarrow P\ xs\ ys$
proof (*induct xs arbitrary*: *ys rule*: *rev-induct*)
case *Nil*
then show *?case*
using *rev-induct*[*of P* ε]
by *presburger*
next
case (*snoc x xs*)
hence *P xs ys'* **for** *ys'*
by *simp*
then show *?case*
by (*simp add*: *rev-induct snoc.prems*(*2*) *snoc.prems*(*4*))
qed

lemma *fin-bin*: *finite* {*x*,*y*}
by *simp*

lemma *rev-rev-image-eq* [*reversal-rule*]: *rev* ‘ *rev* ‘ *X* = *X*
by (*simp add*: *image-comp*)

lemma *last-take-conv-nth*: **assumes** $n < length\ xs$ **shows** $last\ (take\ (Suc\ n)\ xs) = xs!n$

unfolding *take-Suc-conv-app-nth*[*OF assms*] **by** *simp*

lemma *inj-map-inv*: *inj-on f A* $\Longrightarrow$ *u* $\in$ *lists A* $\Longrightarrow$ *u* = *map* (*the-inv-into A f*) (*map f u*)
by (*induct u*, *simp*, *simp add*: *the-inv-into-f-f*)

lemma *last-sing*[*simp*]: *last* [*c*] = *c*
by *simp*

lemma *hd-hdE*: (*u* = ε $\Longrightarrow$ *thesis*) $\Longrightarrow$ (*u* = [*hd u*] $\Longrightarrow$ *thesis*) $\Longrightarrow$ (*u* = [*hd u*, *hd* (*tl u*)] · *tl* (*tl u*) $\Longrightarrow$ *thesis*) $\Longrightarrow$ *thesis*
using *Cons-eq-appendI*[*of hd u* [*hd* (*tl u*)] - *tl u tl* (*tl u*)] *hd-tl*[*of u*] *hd-tl*[*of tl u*] *hd-word*
by *fastforce*

lemma *same-sing-pref*: *u* · [*a*] $\leq p$ *v* $\Longrightarrow$ *u* · [*b*] $\leq p$ *v* $\Longrightarrow$ *a* = *b*
using *prefix-same-cases* **by** *fastforce*

lemma *compow-Suc*: (*f*^^(*Suc k*)) *w* = *f* ((*f*^^*k*) *w*)
by *simp*

lemma *compow-Suc'*: (*f*^^(*Suc k*)) *w* = (*f*^^*k*) (*f w*)
by (*simp add*: *funpow-swap1*)

2.2.1 General facts

lemma *doubleton-neq-eq-iff* [*simp*]: *x* $\neq$ *y* $\Longrightarrow$ {*a*,*b*} = {*x*,*y*} $\longleftrightarrow$ *a* $\neq$ *b* $\wedge$ *a* $\in$ {*x*,*y*} $\wedge$ *b* $\in$ {*x*,*y*}
unfolding *doubleton-eq-iff* **by** *force*

lemma *two-elem-sub*: *x* $\in$ *A* $\Longrightarrow$ *y* $\in$ *A* $\Longrightarrow$ {*x*,*y*} $\subseteq$ *A*
by *simp*

thm *fun.inj-map*[*THEN injD*]

lemma *inj-comp*: **assumes** *inj* (*f* :: *'a list* $\Rightarrow$ *'b list*) **shows** (*g* *w* = *h w* $\longleftrightarrow$ (*f* $\circ$ *g*) *w* = (*f* $\circ$ *h*) *w*)
by (*rule iffI*, *simp*) (*use injD*[*OF assms*] **in** *fastforce*)

lemma *inj-comp-eq*: **assumes** *inj* (*f* :: *'a list* $\Rightarrow$ *'b list*) **shows** (*g* = *h* $\longleftrightarrow$ *f* $\circ$ *g* = *f* $\circ$ *h*)
by (*rule*, *fast*) (*use fun.inj-map*[*OF assms*, *unfolded inj-on-def*] **in** *fast*)

lemma *two-elem-cases*[*elim*!]: **assumes** *u* $\in$ {*x*, *y*} **obtains** (*fst*) *u* = *x* | (*snd*) *u* = *y*
using *assms* **by** *blast*

lemma *two-elem-cases2*[*elim*]: **assumes** *u* $\in$ {*x*, *y*} *v* $\in$ {*x*,*y*} *u* $\neq$ *v*
shows (*u* = *x* $\Longrightarrow$ *v* = *y* $\Longrightarrow$ *thesis*) $\Longrightarrow$ (*u* = *y* $\Longrightarrow$ *v* = *x* $\Longrightarrow$ *thesis*) $\Longrightarrow$ *thesis*

using *assms* **by** *blast*

lemma *two-elemP*: $u \in \{x, y\} \Longrightarrow P\ x \Longrightarrow P\ y \Longrightarrow P\ u$
by *blast*

lemma *pairs-extensional*: $(\bigwedge r\ s.\ P\ r\ s \longleftrightarrow (\exists\ a\ b.\ Q\ a\ b \wedge r = fa\ a \wedge s = fb\ b)) \Longrightarrow \{(r,s).\ P\ r\ s\} = \{(fa\ a, fb\ b) \mid a\ b.\ Q\ a\ b\}$
by *auto*

lemma *pairs-extensional'*: $(\bigwedge r\ s.\ P\ r\ s \longleftrightarrow (\exists\ t.\ Q\ t \wedge r = fa\ t \wedge s = fb\ t)) \Longrightarrow \{(r,s).\ P\ r\ s\} = \{(fa\ t, fb\ t) \mid t.\ Q\ t\}$
by *auto*

lemma *doubleton-subset-cases*: **assumes** $A \subseteq \{x,y\}$
obtains $A = \{\} \mid A = \{x\} \mid A = \{y\} \mid A = \{x,y\}$
using *assms* **by** *blast*

2.2.2 Map injective function

lemma *map-pref-conv* [*reversal-rule*]: **assumes** *inj f* **shows** $map\ f\ u \leq p\ map\ f\ v \longleftrightarrow u \leq p\ v$
using *map-mono-prefix*[*of map f u map f v inv f*] *map-mono-prefix*
unfolding *map-map inv-o-cancel*[*OF* ‹*inj f*›] *list.map-id*..

lemma *map-suf-conv* [*reversal-rule*]: **assumes** *inj f* **shows** $map\ f\ u \leq s\ map\ f\ v \longleftrightarrow u \leq s\ v$
using *map-mono-suffix*[*of map f u map f v inv f*] *map-mono-suffix*
unfolding *map-map inv-o-cancel*[*OF* ‹*inj f*›] *list.map-id*..

lemma *map-fac-conv* [*reversal-rule*]: **assumes** *inj f* **shows** $map\ f\ u \leq f\ map\ f\ v \longleftrightarrow u \leq f\ v$
using *map-mono-sublist*[*of map f u map f v inv f*] *map-mono-sublist*
unfolding *map-map inv-o-cancel*[*OF* ‹*inj f*›] *list.map-id*..

lemma *map-lcp-conv*: **assumes** *inj f* **shows** $(map\ f\ xs) \wedge_p (map\ f\ ys) = map\ f\ (xs \wedge_p ys)$
proof (*induct xs ys rule*: *list-induct2'*)
case (*4 x xs y ys*)
then show *?case*
proof (*cases x = y*)
assume $x = y$
thus *?case*
using *4.hyps* **by** *simp*
next
assume $x \neq y$
hence $f\ x \neq f\ y$
using *inj-eq*[*OF* ‹*inj f*›] **by** *simp*
thus *?case* **using** ‹$x \neq y$› **by** *simp*
qed

qed *simp-all*

2.2.3 Orderings on lists: prefix, suffix, factor

lemmas *self-pref* = *prefix-order.refl* **and**
pref-antisym = *prefix-order.antisym* **and**
pref-trans = *prefix-order.trans* **and**
spref-trans = *prefix-order.less-trans* **and**
suf-trans = *suffix-order.trans* **and**
fac-trans[*intro*] = *sublist-order.order.trans*

2.2.4 On the empty word

lemma *nemp-elem-setI*[*intro*]: $u \in S \Longrightarrow u \neq \varepsilon \Longrightarrow u \in S - \{\varepsilon\}$
by *simp*

lemma *emp-concat-emp*: $us \in lists\ (S - \{\varepsilon\}) \Longrightarrow concat\ us = \varepsilon \Longrightarrow us = \varepsilon$
using *DiffD2* **by** *auto*

lemma *take-nemp*: $w \neq \varepsilon \Longrightarrow 0 < n \Longrightarrow take\ n\ w \neq \varepsilon$
by *simp*

lemma *pref-nemp* [*intro*]: $u \neq \varepsilon \Longrightarrow u \cdot v \neq \varepsilon$
unfolding *append-is-Nil-conv* **by** *simp*

lemma *suf-nemp* [*intro*]: $v \neq \varepsilon \Longrightarrow u \cdot v \neq \varepsilon$
unfolding *append-is-Nil-conv* **by** *simp*

lemma *pref-of-emp*: $u \cdot v = \varepsilon \Longrightarrow u = \varepsilon$
using *append-is-Nil-conv* **by** *simp*

lemma *suf-of-emp*: $u \cdot v = \varepsilon \Longrightarrow v = \varepsilon$
using *append-is-Nil-conv* **by** *simp*

lemma *nemp-comm*: $(u \neq \varepsilon \Longrightarrow v \neq \varepsilon \Longrightarrow u \neq v \Longrightarrow u \cdot v = v \cdot u) \Longrightarrow u \cdot v = v \cdot u$
by *force*

lemma *non-triv-comm* [*intro*]: $(u \neq \varepsilon \Longrightarrow v \neq \varepsilon \Longrightarrow u \neq v \Longrightarrow u \cdot v = v \cdot u) \Longrightarrow u \cdot v = v \cdot u$
by *force*

lemma *split-list'*: $a \in set\ ws \Longrightarrow \exists\, p\ s.\ ws = p \cdot [a] \cdot s$
using *split-list* **by** *fastforce*

lemma *split-listE*: **assumes** $a \in set\ w$
obtains $p\ s$ **where** $w = p \cdot [a] \cdot s$
using *exE2*[*OF split-list'*[*OF assms*]].

2.2.5 Counting letters

declare *count-list-rev* [*reversal-rule*]

lemma *count-list-map-conv* [*reversal-rule*]:
assumes *inj f* **shows** *count-list* (*map f ws*) (*f a*) = *count-list ws a*
by (*induction ws*) (*simp-all add*: *inj-eq*[*OF assms*])

2.2.6 Set inspection method

This section defines a simple method that splits a goal into subgoals by enumerating all possibilites for x in a premise such as $x \in \{a, b, c\}$. See the demonstrations below.

method *set-inspection* = (
(*unfold insert-iff*),
(*elim disjE emptyE*),
(*simp-all only*: *singleton-iff refl True-implies-equals*)
)

lemma $u \in \{x,y\} \Longrightarrow P\ u$
apply(*set-inspection*)
oops

lemma $\bigwedge u.\ u \in \{x,y\} \Longrightarrow u = x \vee u = y$
by(*set-inspection*, *simp-all*)

2.3 Length and its properties

lemmas *lenarg* = *arg-cong*[*of* - - *length*] **and**
lenmorph = *length-append*

lemma *lenarg-not*: $|u| \neq |v| \Longrightarrow u \neq v$
using *size-neq-size-imp-neq*.

lemma *len-less-neq*: $|u| < |v| \Longrightarrow u \neq v$
by *blast*

lemmas *len-nemp-conv* = *length-greater-0-conv*

lemma *npos-len*: $|u| \leq 0 \Longrightarrow u = \varepsilon$
by *simp*

lemma *nemp-len*[*intro*]: $w \neq \varepsilon \Longrightarrow 0 < |w|$
by *blast*

lemma *nemp-len-not0* [*intro*]: $w \neq \varepsilon \Longrightarrow |w| \neq 0$
by *blast*

lemma *nemp-le-len*: $r \neq \varepsilon \Longrightarrow 1 \leq |r|$
by (*simp add*: *leI*)

lemma *nemp-spref-len*: **assumes** $u \neq \varepsilon$ $u <_p v$ **shows** $1 < |v|$
using *le-less-trans*[*OF nemp-le-len*[*OF* ‹$u \neq \varepsilon$›] *prefix-length-less*[*OF* ‹$u <_p v$›]].

lemma *nemp-ssuf-len*: **assumes** $u \neq \varepsilon$ $u <_s v$ **shows** $1 < |v|$
using *le-less-trans*[*OF nemp-le-len*[*OF* ‹$u \neq \varepsilon$›] *suffix-length-less*[*OF* ‹$u <_s v$›]].

lemma *tl-emp* [*intro*]: $|w| \leq 1 \Longrightarrow tl\ w = \varepsilon$
using *length-tl nemp-le-len* **by** *force*

lemma *butlast-emp* [*intro*]: $|w| \leq 1 \Longrightarrow butlast\ w = \varepsilon$
using *length-butlast nemp-le-len* **by** *force*

lemma *butlast-tl-emp*[*intro*]: $|w| \leq 2 \Longrightarrow butlast(tl\ w) = \varepsilon$
by (*intro npos-len*) *simp*

lemma *tl-butlast-emp*[*intro*]: $|w| \leq 2 \Longrightarrow tl(butlast\ w) = \varepsilon$
using *butlast-tl-emp* **unfolding** *butlast-tl*.

lemma *swap-len*: $|u \cdot v| = |v \cdot u|$
by *simp*

lemma *len-after-drop*: $p + q \leq |w| \Longrightarrow\ \ q \leq |drop\ p\ w|$
by *simp*

lemma *short-take-append*: $n \leq |u| \Longrightarrow take\ n\ (u \cdot v) = take\ n\ u$
by *simp*

lemma *sing-word*: $|us| = 1 \Longrightarrow [hd\ us] = us$
by (*cases us*) *simp+*

lemma *sing-word-concat*: **assumes** $|us| = 1$ **shows** $[concat\ us] = us$
unfolding *concat-sing*[*OF sing-word*[*OF* ‹$|us| = 1$›, *symmetric*]] **using** *sing-word*[*OF* ‹$|us| = 1$›].

lemma *len-one-concat-in*: $ws \in lists\ A \Longrightarrow |ws| = 1 \Longrightarrow concat\ ws \in A$
using *Cons-in-lists-iff sing-word-concat* **by** *metis*

lemma *concat-nemp*: $concat\ us \neq \varepsilon \Longrightarrow us \neq \varepsilon$
using *concat.simps*(*1*) **by** *blast*

lemma *sing-len*: $|[a]| = 1$
by *simp*

lemmas *pref-len* = *prefix-length-le* **and**
suf-len = *suffix-length-le*

lemmas *spref-len* = *prefix-length-less* **and**
ssuf-len = *suffix-length-less*

lemma *pref-len'*: $|u| \leq |u \cdot z|$
by *auto*

lemma *suf-len'*: $|u| \leq |z \cdot u|$
by *auto*

lemma *fac-len*: $u \leq f\ v \Longrightarrow |u| \leq |v|$
by *auto*

lemma *fac-len'*: $|w| \leq |u \cdot w \cdot v|$
by *simp*

lemma *fac-len-eq*: $u \leq f\ v \Longrightarrow |u| = |v| \Longrightarrow u = v$
unfolding *fac-def* **using** *lenmorph npos-len* **by** *fastforce*

thm *length-take*

lemma *len-take1*: $|take\ p\ w| \leq p$
by *simp*

lemma *len-take2*: $|take\ p\ w| \leq |w|$
by *simp*

lemma *drop-len*: $|u \cdot w| \leq |u \cdot v \cdot w|$
by *simp*

lemma *drop-pref*: $drop\ |u|\ (u \cdot w) = w$
by *simp*

lemma *take-len*: $p \leq |w| \Longrightarrow |take\ p\ w| = p$
using *min-absorb2*[*of p* $|w|$, *folded length-take*[*of p w*]].

lemma *eq-conjug-len*: $p \cdot x = x \cdot s \Longrightarrow |p| = |s|$
using *lenmorph*[*of p x*] *lenmorph*[*of x s*] *add.commute add-left-imp-eq*
by *auto*

lemma *take-nemp-len*: $u \neq \varepsilon \Longrightarrow r \neq \varepsilon \Longrightarrow take\ |r|\ u \neq \varepsilon$
by *simp*

lemma $r \neq \varepsilon \Longrightarrow w \neq \varepsilon \Longrightarrow drop\ |r|\ w \neq w$
using *drop-neq* **by** *blast*

lemma *emp-len*: $w = \varepsilon \Longrightarrow |w| = 0$

by *simp*

lemma *take-self*: *take* $|w|$ $w = w$
using *take-all*[*of* w $|w|$, *OF order.refl*]**.**

lemma *len-le-concat*: $\varepsilon \notin$ *set ws* $\Longrightarrow$ $|ws| \leq |$*concat ws*$|$
proof (*induct ws*)
case (*Cons a ws*)
hence $1 \leq |a|$
using *list.set-intros*(*1*)[*of a ws*] *nemp-le-len*[*of a*] **by** *blast*
then show *?case*
unfolding *concat.simps*(*2*) **unfolding** *lenmorph hd-word*[*of a ws*] *sing-len*
using *Cons.hyps Cons.prems* **by** *simp*
qed *simp*

lemma *eq-len-iff*: **assumes** *eq*: $x \cdot y = u \cdot v$ **shows** $|x| \leq |u| \longleftrightarrow |v| \leq |y|$
using *lenarg*[*OF eq*] **unfolding** *lenmorph* **by** *auto*

lemma *eq-len-iff-less*: **assumes** *eq*: $x \cdot y = u \cdot v$ **shows** $|x| < |u| \longleftrightarrow |v| < |y|$
using *lenarg*[*OF eq*] **unfolding** *lenmorph* **by** *auto*

lemma *Suc-len-nemp*: $|w| =$ *Suc n* $\Longrightarrow w \neq \varepsilon$
by *force*

lemma *same-sufix-nil*: **assumes** $z \cdot u \leq p\ u$ **shows** $z = \varepsilon$
using *prefix-length-le*[*OF assms*] **unfolding** *lenmorph* **by** *simp*

lemma *count-list-gr-0-iff*: *0* $<$ *count-list u a* $\longleftrightarrow$ $a \in$ *set u*
by (*intro iffI*, *use count-notin*[*folded not-gr0*, *of a u*] **in** *blast*) (*induction u*, *auto*)

lemma *bin-len-concat*: **assumes** $u \neq v$ *ws* $\in$ *lists* $\{u,v\}$
shows $|$*concat ws*$|$ = *count-list ws u* $*$ $|u|$ + *count-list ws v* $*$ $|v|$
proof–
from *sum-list-map-eq-sum-count2*[*of ws* $\{u,v\}$ *length*, *folded length-concat*]
have $|$*concat ws*$|$ = $(\sum x{\in}\{u,\ v\}.$ *count-list ws x* $*$ $|x|)$
using ‹*ws* $\in$ *lists* $\{u,v\}$› **by** *fast*
thus *?thesis*
using ‹$u \neq v$› **by** *auto*
qed

2.4 List inspection method

In this section we define a proof method, named list_inspection, which splits the goal into subgoals which enumerate possibilities on lists with fixed length and given alphabet. More specifically, it looks for a premise of the form such as $|w| = 2 \wedge w \in$ *lists* $\{x, y, z\}$ or $|w| \leq 2 \wedge w \in$ *lists* $\{x, y, z\}$ and substitutes the goal by the goals listing all possibilities for the word w, see

demonstrations after the method definition.

context
begin

First, we define an elementary lemma used for unfolding the premise. Since it is very specific, we keep it private.

private lemma *hd-tl-len-list-iff*: $|w| = Suc\ n \wedge w \in lists\ A \longleftrightarrow hd\ w \in A \wedge w = hd\ w \# tl\ w \wedge |tl\ w| = n \wedge tl\ w \in lists\ A$ (**is** *?L = ?R*)
proof
  show *?L* $\Longrightarrow$ *?R*
  proof (*elim conjE*)
    assume $|w| = Suc\ n$ **and** $w \in lists\ A$
    note *Suc-len-nemp*[*OF* ‹$|w| = Suc\ n$›]
      from *lists-hd-in-set*[*OF* ‹$w \neq \varepsilon$› ‹$w \in lists\ A$›] *list.collapse*[*OF* ‹$w \neq \varepsilon$›] *tl-in-lists*[*OF* ‹$w \in lists\ A$›]
    show $hd\ w \in A \wedge w = hd\ w \# tl\ w \wedge |tl\ w| = n \wedge tl\ w \in lists\ A$
      using ‹$|w| = Suc\ n$› **by** *simp*
  qed
next
  show *?R* $\Longrightarrow$ *?L*
    using *Cons-in-lists-iff*[*of hd w tl w*] *length-Cons*[*of hd w tl w*] **by** *presburger*
qed

We define a list of lemmas used for the main unfolding step.

private lemmas *len-list-word-dec* =
    numeral-nat hd-tl-len-list-iff
    insert-iff empty-iff simp-thms length-0-conv

The method itself accepts an argument called ‘add‘, which is supplied to the method simp_all to solve some simple cases, and thus lower the number of produced goals on the fly.

method *list-inspection0* **uses** *add* =
    ((*match* **premises in** *len*[*thin*]: $|w| \leq k$ **and** *list*[*thin*]: $w \in lists\ A$ **for** *w k A* ⇒
        ‹*insert conjI*[*OF len list*]›)+)*?*,
    (*unfold numeral-nat le-Suc-eq le-0-eq*), — unfold numeral and e.g. $k \leq (2::'a)$
    (*unfold conj-ac*(*1*)[*of* $w \in lists\ A\ |w| \leq k$ **for** *w A k*]
        conj-disj-distribR[**where** *?R* = $w \in lists\ A$ **for** *w A*])*?*,
    ((*match* **premises in** *len*[*thin*]: $|w| = k$ **and** *list*[*thin*]: $w \in lists\ A$ **for** *w k A* ⇒
        ‹*insert conjI*[*OF len list*]›)+)*?*,
        — transform into the conjuction such as $|w| = 2 \wedge w \in lists\ \{x, y, z\}$
    (*unfold conj-ac*(*1*)[*of* $w \in lists\ A\ |w| = k$ **for** *w A k*] *len-list-word-dec*), — unfold w
    (*elim disjE conjE*), — split into all cases
    (*simp-all only*: *singleton-iff lists.Nil list.sel refl True-implies-equals*)*?*, — replace w everywhere
    (*simp-all only*: *add empty-iff lists.Nil bool-simps*)*?* — solve simple cases

method *list-inspection* = (*insert method-facts*, *use nothing* **in** *list-inspection0*)

List inspection demonstrations

The required premise in the form of conjuction. First, inequality bound on the length, second, equality bound.

lemma $|w| = 2 \wedge w \in lists\ \{x,y,z\} \Longrightarrow P\ w$
apply(*list-inspection*)
oops

The required premise as 2 separate premises.

lemma $|w| \leq 2 \Longrightarrow w \in lists\ \{x,y,z\} \Longrightarrow P\ w$
apply(*list-inspection*)
oops

The required premise as 2 separate assumptions.

lemma assumes $|w| \leq 2$ **and** $w \in lists\ \{x,y,z\}$ **shows** $P\ w$
using *assms*
apply(*list-inspection*)
oops

lemma $w \leq p\ w \Longrightarrow |w| \leq 2 \Longrightarrow w \in lists\ \{a,b\} \Longrightarrow hd\ w = a \Longrightarrow w \neq \varepsilon \Longrightarrow w = [a,\ b] \vee w = [a,\ a] \vee w = [a]$
by *list-inspection*

lemma $w \leq p\ w \Longrightarrow |w| = 2 \Longrightarrow w \in lists\ \{a,b\} \Longrightarrow hd\ w = a \Longrightarrow w = [a,\ b] \vee w = [a,\ a]$
by *list-inspection*

lemma $w \leq p\ w \Longrightarrow |w| = 2 \wedge w \in lists\ \{a,b\} \Longrightarrow hd\ w = a \Longrightarrow w = [a,\ b] \vee w = [a,\ a]$
by *list-inspection*

lemma $w \leq p\ w \Longrightarrow w \in lists\ \{a,b\} \wedge |w| = 2 \Longrightarrow hd\ w = a \Longrightarrow w = [a,\ b] \vee w = [a,\ a]$
by *list-inspection*

end

2.5 Prefix and prefix comparability properties

lemmas *pref-emp* = *prefix-bot.extremum-uniqueI*

lemma *triv-pref*: $r \leq p\ r \cdot s$
using *prefI*[*OF refl*]**.**

lemma *triv-spref*: $s \neq \varepsilon \Longrightarrow r <p\ r \cdot s$
by *simp*

lemma *pref-cancel*: $z \cdot u \leq p\ z \cdot v \Longrightarrow u \leq p\ v$
by *simp*

lemma *pref-cancel'* [*intro*]: $u \leq p\ v \Longrightarrow z \cdot u \leq p\ z \cdot v$
by *simp*

lemma *spref-cancel*: $z \cdot u <p\ z \cdot v \Longrightarrow u <p\ v$
by *simp*

lemma *spref-of-nemp* [*intro*]: $u <p\ w \Longrightarrow w \neq \varepsilon$
by *blast*

lemma *spref-cancel'* [*intro*]: $u <p\ v \Longrightarrow z \cdot u <p\ z \cdot v$
by *simp*

lemmas *pref-cancel-conv* = *same-prefix-prefix* **and**
suf-cancel-conv = *same-suffix-suffix* — provided by Sublist.thy

lemma *pref-cancel-hd-conv*: $a\ \#\ u \leq p\ a\ \#\ v \longleftrightarrow u \leq p\ v$
by *simp*

lemma *spref-cancel-conv*: $z \cdot x <p\ z \cdot y \longleftrightarrow x <p\ y$
by *auto*

lemma *spref-snoc-iff* [*simp*]: $u <p\ v \cdot [a] \longleftrightarrow u \leq p\ v$
by (*auto simp only*: *strict-prefix-def prefix-snoc*) *simp*

lemma *butlast-not-pref*[*intro*, *simp*] : **assumes** $u \neq \varepsilon$ **shows** $\neg\ u \leq p\ butlast\ u$
unfolding *spref-snoc-iff*[*of u butlast u last u*, *symmetric*]
append-butlast-last-id[*OF assms*] **by** *blast*

lemma *spref-two-lettersE*: **assumes** $p <p\ [a,b]$ **obtains** $p = \varepsilon \mid p = [a]$
using *assms pref-singE*[*of p a thesis*]
unfolding *hd-word*[*of a* [*b*]] *spref-snoc-iff* **by** *fastforce*

lemmas *pref-ext*[*intro*] = *prefix-prefix* — provided by Sublist.thy

lemmas *pref-extD* = *append-prefixD* **and**
suf-extD = *suffix-appendD*

lemma *spref-extD*: $x \cdot y <p\ z \Longrightarrow x <p\ z$
using *prefix-order.le-less-trans*[*OF triv-pref*]**.**

lemma *spref-ext*: $r <p\ u \Longrightarrow r <p\ u \cdot v$
by *force*

lemma *pref-ext-nemp*: $r \le_p u \Longrightarrow v \neq \varepsilon \Longrightarrow r <_p u \cdot v$
by *auto*

lemma *pref-take*: $p \le_p w \Longrightarrow take\ |p|\ w = p$
unfolding *prefix-def* **by** *force*

lemma *pref-take-conv*: $take\ (|r|)\ w = r \longleftrightarrow r \le_p w$
using *pref-take*[*of r w*] *take-is-prefix*[*of* $|r|$ *w*] **by** *argo*

lemma *le-suf-drop*: **assumes** $i \le j$ **shows** $drop\ j\ w \le_s drop\ i\ w$
using *suffix-drop*[*of* $j - i$ *drop i w*, *unfolded drop-drop le-add-diff-inverse2*[*OF* ‹$i \le j$›]]**.**

lemma *spref-take*: $p <_p w \Longrightarrow take\ |p|\ w = p$
by (*elim spref-exE*) *force*

lemma *pref-same-len*: $u \le_p v \Longrightarrow |u| = |v| \Longrightarrow u = v$
by (*fastforce elim*: *prefixE*)

lemma *pref-same-len'*: $u \cdot z \le_p v \cdot w \Longrightarrow |u| = |v| \Longrightarrow u = v$
by (*fastforce elim*: *prefixE*)

lemma *pref-comp-eq*: $u \bowtie v \Longrightarrow |u| = |v| \Longrightarrow u = v$
using *pref-same-len* **by** *fastforce*

lemma *ruler-eq-len*: $u \le_p w \Longrightarrow v \le_p w \Longrightarrow |u| = |v| \Longrightarrow u = v$
by (*fastforce simp add*: *prefix-def*)

lemma *pref-prod-eq*: $u \le_p v \cdot z \Longrightarrow |u| = |v| \Longrightarrow u = v$
by (*fastforce simp add*: *prefix-def*)

lemmas *pref-comm-eq* = *pref-same-len*[*OF - swap-len*] **and**
pref-comm-eq' = *pref-prod-eq*[*OF - swap-len*, *unfolded rassoc*]

lemma *pref-comm-eq-conv*: $u \cdot v \le_p v \cdot u \longleftrightarrow u \cdot v = v \cdot u$
using *pref-comm-eq self-pref* **by** *metis*

lemma *add-nth-pref*: **assumes** $u <_p w$ **shows** $u \cdot [w!|u|] \le_p w$
using *add-nth*[*OF prefix-length-less*[*OF* ‹$u <_p w$›], *unfolded spref-take*[*OF* ‹$u <_p w$›]]**.**

lemma *index-pref*: $|u| \le |w| \Longrightarrow (\forall\ i < |u|.\ u!i = w!i) \Longrightarrow u \le_p w$
using *trans*[*OF sym*[*OF take-all*[*OF order-refl*]] *nth-take-lemma*[*OF order-refl*], *of u w*]
take-is-prefix[*of* $|u|$ *w*] **by** *auto*

lemma *pref-index*: **assumes** $u \le_p w$ $i < |u|$ **shows** $u!i = w!i$
using *nth-take*[*OF* ‹$i < |u|$›, *of w*, *unfolded pref-take*[*OF* ‹$u \le_p w$›]]**.**

lemma *pref-drop*: $u \leq_p v \Longrightarrow drop\ p\ u \leq_p drop\ p\ v$
using *prefI*[*OF sym*[*OF drop-append*]] **unfolding** *prefix-def* **by** *blast*

lemma *pref-prod-short*: $u \leq_p v \cdot w \Longrightarrow |u| < |v| \Longrightarrow u <_p v$
using *sprefI2 prefix-same-cases*[*of u*] *order.asym prefix-length-less prefix-order.less-le-not-le prefix-same-cases*
triv-pref **by** *metis*

2.5.1 Prefix comparability

lemma *pref-comp-sym*[*sym*]: $u \bowtie v \Longrightarrow v \bowtie u$
by *blast*

lemma *not-pref-comp-sym*[*sym*]: $\neg\ u \bowtie v \Longrightarrow \neg\ v \bowtie u$
by *blast*

lemma *pref-comp-sym-iff*: $u \bowtie v \longleftrightarrow v \bowtie u$
by *blast*

lemmas *ruler-le* = *prefix-length-prefix* **and**
ruler = *prefix-same-cases* **and**
ruler' = *prefix-same-cases*[*folded prefix-comparable-def*]

lemma *ruler-prefs*: **assumes** $L = R$ $|u| \leq |v|$ $u \leq_p L$ $v \leq_p R$
shows $u \leq_p v$
using *ruler-le*[*OF* ‹$u \leq_p L$› ‹$v \leq_p R$›[*folded* ‹$L = R$›] ‹$|u| \leq |v|$›].

lemmas *ruler-sufs* = *ruler-prefs*[*reversed*]

lemma *ruler-eq*: $u \cdot x = v \cdot y \Longrightarrow u \leq_p v \vee v \leq_p u$
by (*metis prefI prefix-same-cases*)

lemma *ruler-eq'*: $u \cdot x = v \cdot y \Longrightarrow u \leq_p v \vee v <_p u$
using *ruler-eq prefix-order.le-less* **by** *blast*

lemmas *ruler-eqE* = *ruler-eq*[*THEN disjE*] **and**
ruler-eqE' = *ruler-eq'*[*THEN disjE*] **and**
ruler-pref = *ruler*[*OF append-prefixD triv-pref*] **and**
ruler'[*THEN pref-comp-eq*]
lemmas *ruler-prefE* = *ruler-pref*[*THEN disjE*]

lemma *ruler-comp*: $u \leq_p v \Longrightarrow u' \leq_p v' \Longrightarrow v \bowtie v' \Longrightarrow u \bowtie u'$
unfolding *prefix-comparable-def*
using *disjE*[*OF - ruler*[*OF pref-trans*] *ruler*[*OF - pref-trans*]].

lemma *ruler-pref'*: $w \leq_p v \cdot z \Longrightarrow w \leq_p v \vee v \leq_p w$
using *ruler* **by** *blast*

lemma *ruler-pref''*: $w \le_p v{\cdot}z \Longrightarrow w \bowtie v$
unfolding *prefix-comparable-def* **using** *ruler-pref'*.

lemma *pref-prod-pref-short*: $u \le_p z \cdot w \Longrightarrow v \le_p w \Longrightarrow |u| \le |z \cdot v| \Longrightarrow u \le_p z \cdot v$
using *ruler-le*[*OF - pref-cancel'*].

lemma *pref-prod-pref*: $u \le_p z \cdot w \Longrightarrow u \le_p w \Longrightarrow u \le_p z \cdot u$
using *pref-prod-pref-short*[*OF - - suf-len'*].

lemma *pref-prod-pref'*: **assumes** $u \le_p z{\cdot}u{\cdot}w$ **shows** $u \le_p z{\cdot}u$
using *pref-prod-pref*[*of u z u · w*, *OF* ‹$u \le_p z{\cdot}u{\cdot}w$› *triv-pref*].

lemma *pref-prod-long*: $u \le_p v \cdot w \Longrightarrow |v| \le |u| \Longrightarrow v \le_p u$
using *ruler-le*[*OF triv-pref*].

lemmas *pref-prod-long-ext* = *pref-prod-long*[*OF append-prefixD*]

lemma *pref-prod-long-less*: **assumes** $u \le_p v \cdot w$ **and** $|v| < |u|$ **shows** $v <_p u$
using *sprefI2*[*OF pref-prod-long*[*OF* ‹$u \le_p v \cdot w$› *less-imp-le*[*OF* ‹$|v| < |u|$›]] ‹$|v| < |u|$›].

lemma *ruler-less*: $ps \le_p xs \Longrightarrow qs \le_p xs \Longrightarrow |ps| < |qs| \Longrightarrow ps <_p qs$
using *prefD pref-prod-long-less* **by** *metis*

lemma *ruler-le-neq*: $ps \le_p xs \Longrightarrow qs \le_p xs \Longrightarrow |ps| \le |qs| \Longrightarrow ps \neq qs \Longrightarrow ps <_p qs$
using *pref-prod-long prefix-def prefix-order.less-le* **by** *metis*

lemma *pref-keeps-per-root*: $u \le_p r \cdot u \Longrightarrow v \le_p u \Longrightarrow v \le_p r \cdot v$
using *pref-prod-pref*[*of v r u*] *pref-trans*[*of v u r·u*] **by** *blast*

lemma *pref-keeps-per-root'*: $u <_p r \cdot u \Longrightarrow v \le_p u \Longrightarrow v <_p r \cdot v$
using *pref-keeps-per-root* **by** *auto*

lemma *per-root-pref-sing*: $w <_p r \cdot w \Longrightarrow u \cdot [a] \le_p w \Longrightarrow u \cdot [a] \le_p r \cdot u$
using *append-assoc pref-keeps-per-root' spref-snoc-iff* **by** *metis*

lemma *pref-prolong*: $w \le_p z \cdot r \Longrightarrow r \le_p s \Longrightarrow w \le_p z \cdot s$
using *pref-trans*[*OF - pref-cancel'*].

lemma *spref--pref-prolong*: $w <_p z \cdot r \Longrightarrow r \le_p s \Longrightarrow w <_p z \cdot s$
using *prefix-order.less-le-trans*[*OF - pref-cancel'*].

lemma *pref-spref-prolong*: $w \le_p z \cdot r \Longrightarrow r <_p s \Longrightarrow w <_p z \cdot s$
using *prefix-order.le-less-trans*[*OF - spref-cancel'*].

lemma *spref-spref-prolong*: $w <_p z \cdot r \Longrightarrow r <_p s \Longrightarrow w <_p z \cdot s$
using *prefix-order.less-trans*[*OF - spref-cancel'*].

lemmas *pref-shorten* = *pref-trans*[*OF pref-cancel'*]

lemma *pref-prolong'*: $u \le_p w \cdot z \Longrightarrow v \cdot u \le_p z \Longrightarrow u \le_p w \cdot v \cdot u$
using *ruler-le*[*OF - pref-cancel' le-trans*[*OF suf-len' suf-len'*]].

lemma *pref-prolong-per-root*: $u \le_p r \cdot s \Longrightarrow s \le_p r \cdot s \Longrightarrow u \le_p r \cdot u$
using *pref-prolong*[*of u r s r · s, THEN pref-prod-pref*].

lemma *pref-prod-le*[*intro*]: $u \le_p v \cdot w \Longrightarrow |u| \le |v| \Longrightarrow u \le_p v$
using *ruler-le*[*OF - triv-pref*].

lemma *prod-pref-prod-le*: $u{\cdot}v \le_p x{\cdot}y \Longrightarrow |u| \le |x| \Longrightarrow u \le_p x$
using *pref-prod-le*[*OF append-prefixD*].

lemma *pref-cancel-right-len*: $u \cdot z \le_p v \cdot z' \Longrightarrow |z| = |z'| \Longrightarrow u \le_p v$
by (*rule prod-pref-prod-le, blast*) (*use pref-len*[*of u · z v · z', unfolded lenmorph*]
in *force*)

lemma *spref-cancel-right-len*: **assumes** $u \cdot z <_p v \cdot z'$ $|z| = |z'|$
shows $u <_p v$
by (*rule sprefI*[*OF pref-cancel-right-len*[*OF sprefD1*[*OF* ‹$u \cdot z <_p v \cdot z'$›] ‹$|z| = |z'|$›]]])
(*use spref-len*[*OF spref-cancel, of u z z'*] *assms* **in** *force*)

lemmas *pref-cancel-right* = *pref-cancel-right-len*[*OF - refl*] **and**
spref-cancel-right = *spref-cancel-right-len*[*OF - refl*]

lemma *eq-le-pref*[*elim*]: $x \cdot y = u \cdot v \Longrightarrow |x| \le |u| \Longrightarrow x \le_p u$
using *pref-prod-le*[*OF prefI*].

lemma *pref-less-spref*: $u \cdot w \le_p v \cdot z \Longrightarrow |u| < |v| \Longrightarrow u <_p v$
using *prod-pref-prod-le*[*OF - less-imp-le, THEN sprefI2*].

lemma *pref-prod-cancel*: **assumes** $u \le_p p{\cdot}w{\cdot}q$ **and** $|p| \le |u|$ **and** $|u| \le |p{\cdot}w|$
obtains r **where** $p \cdot r = u$ **and** $r \le_p w$
proof–
obtain r **where** [*symmetric*]: $u = p \cdot r$ **using** *pref-prod-long*[*OF* ‹$u \le_p p{\cdot}w{\cdot}q$›
‹$|p| \le |u|$›]..
moreover have $r \le_p w$
using *pref-prod-le*[*OF* ‹$u \le_p p{\cdot}w{\cdot}q$›[*unfolded lassoc*] ‹$|u| \le |p{\cdot}w|$›]
unfolding ‹$p \cdot r = u$›[*symmetric*] **by** *simp*
ultimately show *thesis*..
qed

lemma *pref-prod-cancel′*: **assumes** $u \leq p\ p{\cdot}w{\cdot}q$ **and** $|p| < |u|$ **and** $|u| \leq |p{\cdot}w|$
obtains r **where** $p \cdot r = u$ **and** $r \leq p\ w$ **and** $r \neq \varepsilon$
proof−
obtain r **where** $p \cdot r = u$ **and** $r \leq p\ w$
using *pref-prod-cancel*[*OF* ‹$u \leq p\ p{\cdot}w{\cdot}q$› *less-imp-le*[*OF* ‹$|p| < |u|$›] ‹$|u| \leq |p{\cdot}w|$›].
moreover have $r \neq \varepsilon$ **using** ‹$p \cdot r = u$› *less-imp-neq*[*OF* ‹$|p| < |u|$›] **by** *fastforce*
ultimately show *thesis*..
qed

lemma *non-comp-parallel*: $\neg\ u \bowtie v \longleftrightarrow u \parallel v$
unfolding *prefix-comparable-def parallel-def de-Morgan-disj*..

lemma *comp-refl*: $u \bowtie u$
unfolding *prefix-comparable-def*
by *simp*

lemma *incomp-cancel*: $\neg\ p{\cdot}u \bowtie p{\cdot}v \Longrightarrow \neg\ u \bowtie v$
unfolding *prefix-comparable-def*
by *simp*

lemma *comm-ruler*: $r \cdot s \leq p\ w1 \Longrightarrow s \cdot r \leq p\ w2 \Longrightarrow w1 \bowtie w2 \Longrightarrow r \cdot s = s \cdot r$
using *pref-comp-eq*[*OF ruler-comp swap-len*].

lemma *comm-comp-eq*: $r \cdot s \bowtie s \cdot r \Longrightarrow r \cdot s = s \cdot r$
using *comm-ruler* **by** *blast*

lemma *pref-share-take*: $u \leq p\ v \Longrightarrow q \leq |u| \Longrightarrow take\ q\ u = take\ q\ v$
by (*auto simp add*: *prefix-def*)

lemma *pref-prod-longer*: $u \leq p\ z \cdot w \Longrightarrow v \leq p\ w \Longrightarrow |z \cdot v| \leq |u| \Longrightarrow z \cdot v \leq p\ u$
using *ruler-le*[*OF pref-cancel′*].

lemma *pref-comp-not-pref*: $u \bowtie v \Longrightarrow \neg\ v \leq p\ u \Longrightarrow u <p\ v$
by *auto*

lemma *pref-comp-not-spref*: $u \bowtie v \Longrightarrow \neg\ u <p\ v \Longrightarrow v \leq p\ u$
using *contrapos-np*[*OF* - *pref-comp-not-pref*].

lemma *hd-prod*: $u \neq \varepsilon \Longrightarrow (u \cdot v)!0 = u!0$
by (*cases u*) (*blast*, *simp*)

lemma *distinct-first*: **assumes** $w \neq \varepsilon\ z \neq \varepsilon\ w!0 \neq z!0$ **shows** $w \cdot w' \neq z \cdot z'$
using *hd-prod*[*of w w′*, *OF* ‹$w \neq \varepsilon$›] *hd-prod*[*of z z′*, *OF* ‹$z \neq \varepsilon$›] ‹$w!0 \neq z!0$› **by** *auto*

lemmas *last-no-split* = *prefix-snoc*

lemma *last-no-split'*: $u <p\ w \Longrightarrow w \leq p\ u \cdot [a] \Longrightarrow w = u \cdot [a]$
unfolding *prefix-order.less-le-not-le last-no-split* **by** *blast*

lemma *comp-shorter*: $v \bowtie w \Longrightarrow |v| \leq |w| \Longrightarrow v \leq p\ w$
unfolding *prefix-comparable-def*
by (*auto simp add*: *prefix-def*)

lemma *comp-shorter-conv*: $|u| \leq |v| \Longrightarrow u \bowtie v \longleftrightarrow u \leq p\ v$
using *comp-shorter* **by** *auto*

lemma *pref-comp-len-trans*: $w \leq p\ v \Longrightarrow u \bowtie v \Longrightarrow |w| \leq |u| \Longrightarrow w \leq p\ u$
using *ruler-le pref-trans* **by** (*elim pref-compE*)

lemma *comp-cancel*: $z \cdot w1 \bowtie z \cdot w2 \longleftrightarrow w1 \bowtie w2$
unfolding *prefix-comparable-def*
using *pref-cancel* **by** *auto*

lemma *emp-pref*: $\varepsilon \leq p\ u$
by *simp*

lemma *emp-spref*: $u \neq \varepsilon \Longrightarrow \varepsilon <p\ u$
by *simp*

lemma *long-pref*: $u \leq p\ v \Longrightarrow |v| \leq |u| \Longrightarrow u = v$
by (*auto simp add*: *prefix-def*)

lemma *not-comp-ext*: $\neg\ w1 \bowtie\ w2 \Longrightarrow \neg\ w1 \cdot z \bowtie w2 \cdot z'$
using *contrapos-nn*[*OF - ruler-comp*[*OF triv-pref triv-pref*]]**.**

lemma *mismatch-incopm*: $|u| = |v| \Longrightarrow x \neq y \Longrightarrow \neg\ u \cdot [x] \bowtie v \cdot [y]$
by (*auto simp add*: *prefix-def*)

lemma *comp-prefs-comp*: $u \cdot z \bowtie v \cdot w \Longrightarrow u \bowtie v$
using *ruler-comp*[*OF triv-pref triv-pref*]**.**

lemma *comp-hd-eq*: $u \bowtie v \Longrightarrow u \neq \varepsilon \Longrightarrow v \neq \varepsilon \Longrightarrow hd\ u = hd\ v$
unfolding *prefix-comparable-def*
by (*auto simp add*: *prefix-def*)

lemma *pref-hd-eq'*: $p \leq p\ u \Longrightarrow p \leq p\ v \Longrightarrow p \neq \varepsilon \Longrightarrow\ hd\ u = hd\ v$
by (*auto simp add*: *prefix-def*)

lemma *pref-hd-eq*: $u \leq p\ v \Longrightarrow u \neq \varepsilon \Longrightarrow\ hd\ u = hd\ v$
by (*auto simp add*: *prefix-def*)

lemma *sing-pref-hd*: $[a] \leq p\ v \Longrightarrow hd\ v = a$
by (*auto simp add*: *prefix-def*)

lemma *suf-last-eq*: $p \leq s\ u \Longrightarrow p \leq s\ v \Longrightarrow p \neq \varepsilon \Longrightarrow\ last\ u = last\ v$

by (*auto simp add*: *suffix-def*)

lemma *comp-hd-eq'*: $u \cdot r \bowtie v \cdot s \Longrightarrow u \neq \varepsilon \Longrightarrow v \neq \varepsilon \Longrightarrow hd\ u = hd\ v$
using *comp-hd-eq*[*OF comp-prefs-comp*]**.**

2.5.2 Minimal and maximal prefix with a given property

lemma *le-take-pref*: **assumes** $k \leq n$ **shows** *take k ws* $\leq p$ *take n ws*
using *take-add*[*of k* $(n-k)$ *ws*, *unfolded le-add-diff-inverse*[*OF* ‹$k \leq n$›]]
by *force*

lemma *min-pref*: **assumes** $u \leq p\ w$ **and** $P\ u$
obtains v **where** $v \leq p\ w$ **and** $P\ v$ **and** $\bigwedge y.\ y \leq p\ w \Longrightarrow P\ y \Longrightarrow v \leq p\ y$
using *assms*
proof(*induction* $|u|$ *arbitrary*: *u rule*: *less-induct*)
case (*less u'*)
then show *?case*
proof (*cases* $\forall\ y.\ y \leq p\ w \longrightarrow P\ y \longrightarrow u' \leq p\ y$, *blast*)
assume $\neg\ (\forall\ y.\ y \leq p\ w \longrightarrow P\ y \longrightarrow u' \leq p\ y)$
then obtain x **where** $x \leq p\ w$ **and** $P\ x$ **and** $\neg\ u' \leq p\ x$
by *blast*
have $|x| < |u'|$
using *prefix-length-less*[*OF pref-comp-not-pref*[*OF ruler'*[*OF* ‹$x \leq p\ w$› ‹$u' \leq p\ w$›]‹$\neg\ u' \leq p\ x$›]]**.**
from *less.hyps*[*OF this* - ‹$x \leq p\ w$› ‹$P\ x$›] *that*
show *thesis* **by** *blast*
qed
qed

lemma *min-pref'*: **assumes** $u \leq p\ w$ **and** $P\ u$
obtains v **where** $v \leq p\ w$ **and** $P\ v$ **and** $\bigwedge y.\ y \leq p\ v \Longrightarrow P\ y \Longrightarrow y = v$
proof−
from *min-pref*[*of* - - P, *OF assms*]
obtain v **where** $v \leq p\ w$ **and** $P\ v$ **and** *min*: $\bigwedge y.\ y \leq p\ w \Longrightarrow P\ y \Longrightarrow v \leq p\ y$
by *blast*
have $y = v$ **if** $y \leq p\ v$ **and** $P\ y$ **for** y
using *min*[*OF pref-trans*[*OF* ‹$y \leq p\ v$› ‹$v \leq p\ w$›] ‹$P\ y$›] ‹$y \leq p\ v$› **by** *force*
from *that*[*OF* ‹$v \leq p\ w$› ‹$P\ v$› *this*]
show *thesis***.**
qed

lemma *max-pref*: **assumes** $u \leq p\ w$ **and** $P\ u$
obtains v **where** $v \leq p\ w$ **and** $P\ v$ **and** $\bigwedge y.\ y \leq p\ w \Longrightarrow P\ y \Longrightarrow y \leq p\ v$
using *assms*
proof(*induction* $|w|-|u|$ *arbitrary*: *u rule*: *less-induct*)
case (*less u'*)
then show *?case*
proof (*cases* $\forall\ y.\ y \leq p\ w \longrightarrow P\ y \longrightarrow y \leq p\ u'$, *blast*)

assume $\neg\ (\forall\, y.\ y \leq_p w \longrightarrow P\ y \longrightarrow\ y \leq_p\ u')$
then obtain x **where** $x \leq_p w$ **and** $P\ x$ **and** $\neg\ x \leq_p u'$ **and** $u' \neq w$
by *blast*
from *ruler'*[*OF* ‹$x \leq_p w$› ‹$u' \leq_p w$›]
have $|u'| < |x|$
using *comp-shorter*[*OF* ‹$x \bowtie u'$›] ‹$\neg\ x \leq_p u'$› **by** *fastforce*
hence $|w| - |x| < |w| - |u'|$
using ‹$x \leq_p w$› ‹$u' \neq w$› *diff-less-mono2 leI*[*THEN long-pref*[*OF* ‹$u' \leq_p w$›]] **by** *blast*
from *less.hyps*[*OF this* - ‹$x \leq_p w$› ‹$P\ x$›] *that*
show *thesis* **by** *blast*
qed
qed

2.6 Suffix and suffix comparability properties

lemmas *suf-emp* = *suffix-bot.extremum-uniqueI*

lemma *triv-suf*: $u \leq_s v \cdot u$
by (*simp add*: *suffix-def*)

lemma *emp-ssuf*: $u \neq \varepsilon \Longrightarrow \varepsilon <_s u$
by *simp*

lemma *suf-cancel*: $u{\cdot}v \leq_s w{\cdot}v \Longrightarrow u \leq_s w$
by *simp*

lemma *suf-cancel'* [*intro*]: $u \leq_s w \Longrightarrow u{\cdot}v \leq_s w{\cdot}v$
by *simp*

lemma *ssuf-cancel'* [*intro*]: $u <_s w \Longrightarrow u{\cdot}v <_s w{\cdot}v$
by *simp*

lemma *ssuf-cancel-conv*: $x \cdot z <_s y \cdot z \longleftrightarrow x <_s y$
by *auto*

Straightforward relations of suffix and prefix follow.

lemmas *suf-rev-pref-iff* = *suffix-to-prefix* — provided by Sublist.thy

lemmas *ssuf-rev-pref-iff* = *strict-suffix-to-prefix* — provided by Sublist.thy

lemma *pref-rev-suf-iff*: $u \leq_p v \longleftrightarrow rev\ u \leq_s rev\ v$
using *suffix-to-prefix*[*of rev u rev v*] **unfolding** *rev-rev-ident*
by *blast*

lemma *spref-rev-suf-iff*: $s <_p w \longleftrightarrow rev\ s <_s rev\ w$
using *strict-suffix-to-prefix*[*of rev s rev w*, *unfolded rev-rev-ident*, *symmetric*].

lemmas [*reversal-rule*] =

suf-rev-pref-iff[*symmetric*]
pref-rev-suf-iff[*symmetric*]
ssuf-rev-pref-iff[*symmetric*]
spref-rev-suf-iff[*symmetric*]

lemmas *sufE* = *prefixE*[*reversed*] **and**
prefE = *prefixE* **and**
ssuf-exE[*elim*] = *spref-exE*[*reversed*]

lemmas *suf-prod-long-ext* = *pref-prod-long-ext*[*reversed*]

lemmas *suf-prolong-per-root* = *pref-prolong-per-root*[*reversed*]

lemmas *suf-ext* = *suffix-appendI* — provided by Sublist.thy

lemmas *ssuf-ext* = *spref-ext*[*reversed*] **and**
ssuf-extD = *spref-extD*[*reversed*] **and**
suf-ext-nem = *pref-ext-nemp*[*reversed*] **and**
suf-same-len = *pref-same-len*[*reversed*] **and**
suf-take = *pref-drop*[*reversed*] **and**
suf-share-take = *pref-share-take*[*reversed*] **and**
long-suf = *long-pref*[*reversed*] **and**
strict-suffixE' = *strict-prefixE'*[*reversed*] **and**
ssuf-tl-suf = *spref-butlast-pref*[*reversed*]

lemma *ssuf-Cons-iff* [*simp*]: $u <_s a \# v \longleftrightarrow u \leq_s v$
by (*auto simp only*: *strict-suffix-def suffix-Cons*) (*simp add*: *suffix-def*)

lemma *ssuf-induct* [*case-names ssuf*]:
assumes $\bigwedge u.\ (\bigwedge v.\ v <_s u \Longrightarrow P\ v) \Longrightarrow P\ u$
shows $P\ u$
proof (*induction u rule*: *list.induct*[*of* $\lambda u.\ \forall v.\ v \leq_s u \longrightarrow P\ v$, *rule-format*, *OF* -- *triv-suf*],
use assms suffix-bot.extremum-strict **in** *blast*)
qed (*metis assms ssuf-Cons-iff suffix-Cons*)

2.6.1 Suffix comparability

lemma *eq-le-suf*[*elim*]: **assumes** $x \cdot y = u \cdot v$ $|x| \leq |u|$ **shows** $v \leq_s y$
using *eq-le-pref*[*reversed*, *OF assms(1)*[*symmetric*]]
lenarg[*OF* ‹$x \cdot y = u \cdot v$›, *unfolded lenmorph*] ‹$|x| \leq |u|$› **by** *linarith*

lemmas *eq-le-suf'*[*elim*] = *eq-le-pref*[*reversed*]

lemma *eq-le-suf''*[*elim*]: **assumes** $v \cdot u = y \cdot x$ $|x| \leq |u|$ **shows** $x \leq_s u$
using *eq-le-suf'*[*OF assms(1)*[*symmetric*] *assms(2)*].

lemma *pref-comp-rev-suf-comp*[*reversal-rule*]: $(rev\ w) \bowtie_s (rev\ v) \longleftrightarrow w \bowtie v$
unfolding *suffix-comparable-def* **by** *simp*

lemma *suf-comp-rev-pref-comp*[*reversal-rule*]: $(rev\ w) \bowtie (rev\ v) \longleftrightarrow w \bowtie_s v$
unfolding *suffix-comparable-def* **by** *simp*

lemmas *suf-ruler-le* = *suffix-length-suffix* — provided by Sublist.thy, same as ruler_le[reversed]

lemmas *suf-ruler* = *suffix-same-cases* — provided by Sublist.thy, same as ruler[reversed]

lemmas *suf-ruler-eq-len* = *ruler-eq-len*[*reversed*] **and**
suf-ruler-comp = *ruler-comp*[*reversed*] **and**
ruler-suf = *ruler-pref*[*reversed*] **and**
ruler-suf' = *ruler-pref'*[*reversed*] **and**
ruler-suf'' = *ruler-pref''*[*reversed*] **and**
suf-prod-le = *pref-prod-le*[*reversed*] **and**
prod-suf-prod-le = *prod-pref-prod-le*[*reversed*] **and**
suf-prod-eq = *pref-prod-eq*[*reversed*] **and**
suf-prod-cancel = *pref-prod-cancel*[*reversed*] **and**
suf-prod-cancel' = *pref-prod-cancel'*[*reversed*] **and**
suf-prod-suf-short = *pref-prod-pref-short*[*reversed*] **and**
suf-prod-suf = *pref-prod-pref*[*reversed*] **and**
suf-prod-suf' = *pref-prod-pref'*[*reversed*, *unfolded rassoc*] **and**
suf-prolong = *pref-prolong*[*reversed*] **and**
suf-prolong' = *pref-prolong'*[*reversed*, *unfolded rassoc*] **and**
suf-prod-long = *pref-prod-long*[*reversed*] **and**
suf-prod-long-less = *pref-prod-long-less*[*reversed*] **and**
suf-prod-longer = *pref-prod-longer*[*reversed*] **and**
suf-keeps-root = *pref-keeps-per-root*[*reversed*] **and**
comm-suf-ruler = *comm-ruler*[*reversed*] **and**
suf-ruler-less = *ruler-less*[*reversed*] **and**
suf-ruler-le-neq = *ruler-le-neq*[*reversed*]

lemmas *comp-sufs-comp* = *comp-prefs-comp*[*reversed*] **and**
suf-comp-not-suf = *pref-comp-not-pref*[*reversed*] **and**
suf-comp-not-ssuf = *pref-comp-not-spref*[*reversed*] **and**
suf-comp-cancel = *comp-cancel*[*reversed*] **and**
suf-not-comp-ext = *not-comp-ext*[*reversed*] **and**
mismatch-suf-incopm = *mismatch-incopm*[*reversed*] **and**
suf-comp-sym[*sym*] = *pref-comp-sym*[*reversed*] **and**
suf-comp-refl = *comp-refl*[*reversed*]

lemma *suf-comp-or*: $u \bowtie_s v \longleftrightarrow u \leq s\ v \vee v \leq s\ u$
unfolding *suffix-comparable-def prefix-comparable-def suf-rev-pref-iff* **..**

lemma *comm-comp-eq-conv*: $r \cdot s \bowtie s \cdot r \longleftrightarrow r \cdot s = s \cdot r$
using *pref-comp-eq*[*OF - swap-len*] *comp-refl* **by** *metis*

lemma *comm-comp-eq-conv-suf*: $r \cdot s \bowtie_s s \cdot r \longleftrightarrow r \cdot s = s \cdot r$

using *pref-comp-eq*[*reversed*, *OF* - *swap-len*, *of r s*] *suf-comp-refl*[*of* $r \cdot s$] **by** *argo*

lemma *suf-comp-last-eq*: **assumes** $u \bowtie_s v$ $u \neq \varepsilon$ $v \neq \varepsilon$
shows *last* u = *last* v
using *comp-hd-eq*[*reversed*, *OF assms*] **unfolding** *hd-rev hd-rev*.

lemma *suf-comp-last-eq'*: $r \cdot u \bowtie_s s \cdot v \Longrightarrow u \neq \varepsilon \Longrightarrow v \neq \varepsilon \Longrightarrow$ *last* u = *last* v
using *comp-sufs-comp suf-comp-last-eq* **by** *blast*

2.7 Left and Right Quotient

A useful function of left quotient is given. Note that the function is sometimes undefined.

definition *left-quotient*:: *'a list* ⇒ *'a list* ⇒ *'a list* ((-$^{-1>}$)(-) [*75*,*74*] *74*)
where *left-quotient u v* = *drop* |*u*| *v*
notation (*latex* **output**) *left-quotient* (- $^{-1}\cdot$ -)

Analogously, we define the right quotient.

definition *right-quotient* :: *'a list* ⇒ *'a list* ⇒ *'a list* ((-)($^{<-1}$-) [*76*,*77*] *76*)
where *right-quotient u v* = *rev* ((*rev v*)$^{-1>}$(*rev u*))
notation (*latex* **output**) *right-quotient* (- · - $^{-1}$)

lemmas *lq-def* = *left-quotient-def* **and**
rq-def = *right-quotient-def*

Priorities of these operations are as follows:

lemma $u^{<-1}v^{<-1}w = (u^{<-1}v)^{<-1}w$
by *simp*

lemma $u^{-1>}v^{-1>}w = u^{-1>}(v^{-1>}w)$
by *simp*

lemma $u^{-1>}v^{<-1}w = u^{-1>}(v^{<-1}w)$
by *simp*

lemma $r \cdot u^{-1>}w^{<-1}v \cdot s = r \cdot (u^{-1>}w^{<-1}v) \cdot s$
by *simp*

lemma *rq-rev-lq*[*reversal-rule*]: $(rev\ v)^{<-1}(rev\ u) = rev\ (u^{-1>}v)$
unfolding *right-quotient-def*
by *simp*

lemma *lq-rev-rq*[*reversal-rule*]: $(rev\ v)^{-1>}rev\ u = rev\ (u^{<-1}v)$
unfolding *right-quotient-def*
by *simp*

2.7.1 Left Quotient

lemma *lqI*: $u \cdot z = v \Longrightarrow u^{-1>}v = z$
unfolding *left-quotient-def*
by *force*

lemma *lq-triv*[*simp*]: $u^{-1>}(u \cdot z) = z$
using *lqI*[*OF refl*]**.**

lemma *lq-triv'*[*simp*]: $u \cdot u^{-1>}(u \cdot z) = u \cdot z$
by *simp*

lemma *append-lq*: **assumes** $u{\cdot}v \leq p\ w$ **shows** $(u{\cdot}v)^{-1>}w = v^{-1>}(u^{-1>}w)$
using *lq-triv*[*of* $u{\cdot}v$] *lq-triv*[*of* v] *lq-triv*[*of* $u\ v{\cdot}$-] *assms*[*unfolded prefix-def*]
by *force*

lemma *lq-self*[*simp*]: $u^{-1>}u = \varepsilon$
unfolding *left-quotient-def*
by *simp*

lemma *lq-emp*[*simp*]: $\varepsilon^{-1>}u = u$
unfolding *left-quotient-def*
by *simp*

lemma *lq-pref*[*simp*]: $u \leq p\ v \Longrightarrow u \cdot (u^{-1>}v) = v$
unfolding *left-quotient-def prefix-def*
by *fastforce*

lemmas *lq-spref* = *lq-pref*[*OF sprefD1*]

lemma *lq-pref-conv*: $|u| \leq |v| \Longrightarrow u \leq p\ v \longleftrightarrow u \cdot u^{-1>}v = v$
using *lq-pref* **by** *blast*

lemma *lq-len*: $|u^{-1>}v| = |v| - |u|$
unfolding *left-quotient-def* **using** *length-drop***.**

lemmas *lcp-lq* = *lq-pref*[*OF longest-common-prefix-prefix1*] *lq-pref*[*OF longest-common-prefix-prefix2*]

lemma *lq-pref-cancel*: $u \leq p\ v \Longrightarrow v \cdot r = u \cdot s \Longrightarrow (u^{-1>}v) \cdot r = s$
unfolding *prefix-def*
by *force*

lemma *lq-the*: **assumes** $u \leq p\ v$
shows (*THE z*. $u \cdot z = v$) $= (u^{-1>}v)$
proof–
have $u{\cdot}z = v \Longrightarrow z = (u^{-1>}v)$ **for** z
by *fastforce*
from *the-equality*[*of* $\lambda z.\ u{\cdot}z{=}v$, *OF lq-pref this*, *OF assms*]
show *?thesis***.**
qed

lemma *lq-same-len*: $|u| = |v| \Longrightarrow u^{-1>}v = \varepsilon$
unfolding *left-quotient-def* **by** *simp*

lemma *lq-assoc*: $|u| \leq |v| \Longrightarrow (u^{-1>}v)^{-1>}w = v^{-1>}(u \cdot w)$
unfolding *left-quotient-def* **using** *prefix-length-le* **by** *auto*

lemma *lq-assoc'*: $(u \cdot w)^{-1>}v = w^{-1>}(u^{-1>}v)$
unfolding *left-quotient-def lenmorph*
by (*simp add*: *add.commute*)

lemma *lq-reassoc*: $u \leq_p v \Longrightarrow (u^{-1>}v)\cdot w = u^{-1>}(v\cdot w)$
unfolding *prefix-def*
by *force*

lemma *lq-trans*: $u \leq_p v \Longrightarrow v \leq_p w \Longrightarrow (u^{-1>}v) \cdot (v^{-1>}w) = u^{-1>}w$
by (*simp add*: *lq-reassoc*)

lemma *lq-rq-reassoc-suf*: **assumes** $u \leq_p z$ $u \leq_s w$ **shows** $w\cdot u^{-1>}z = w^{<-1}u \cdot z$
using *rassoc*[*of* $w^{<-1}u$ u $u^{-1>}z$, *unfolded lq-pref*[*OF* ‹$u \leq_p z$›] *lq-pref*[*reversed*, *OF* ‹$u \leq_s w$›]].

lemma *lq-ne*: **assumes** $u \neq \varepsilon$ **shows** $p^{-1>}(u\cdot p) \neq \varepsilon$
using *drop-eq-Nil2*[*of* $|p|$ $u\cdot p$, *folded left-quotient-def*]
unfolding *lenmorph* **using** *nemp-len*[*OF assms*] **by** *force*

lemma *lq-spref-nemp*: $u <_p v \Longrightarrow u^{-1>}v \neq \varepsilon$
using *lq-pref* **by** (*auto simp add*: *prefix-def*)

lemma *lq-is-suf*[*simp*]: $r^{-1>}s \leq_s s$
unfolding *left-quotient-def* **using** *suffix-drop* **by** *auto*

lemma *lq-short-len*: $r \leq_p s \Longrightarrow |r| + |r^{-1>}s| = |s|$
by (*auto simp add*: *prefix-def*)

lemma *pref-lq*: $v \leq_p w \Longrightarrow u^{-1>}v \leq_p u^{-1>}w$
unfolding *left-quotient-def prefix-def*
using *drop-append* **by** *blast*

lemma *spref-lq*: $u \leq_p v \Longrightarrow v <_p w \Longrightarrow u^{-1>}v <_p u^{-1>}w$
by (*auto simp add*: *prefix-def*)

lemma *pref-gcd-lq*: **assumes** $u \leq_p v$ **shows** $(gcd\ |u|\ |u^{-1>}v|) = gcd\ |u|\ |v|$
using *gcd-add2*[*of* $|u|$ $|u^{-1>}v|$, *unfolded lq-short-len*[*OF assms*], *symmetric*].

lemma *conjug-lq*: $x \cdot z = z \cdot y \Longrightarrow y = z^{-1>}(x \cdot z)$
by *simp*

lemma *conjug-emp-emp*: $p \leq_p u \cdot p \Longrightarrow p^{-1>}(u \cdot p) = \varepsilon \Longrightarrow u = \varepsilon$

using *lq-ne* **by** *blast*

lemma *hd-lq-conv-nth*: **assumes** $u <p v$ **shows** $hd(u^{-1>}v) = v!|u|$
using *prefix-length-less*[*OF assms*, *THEN hd-drop-conv-nth*] **unfolding** *lq-def*.

lemma *concat-morph-lq*: $us \leq p\ ws \Longrightarrow concat\ (us^{-1>}ws) = (concat\ us)^{-1>}(concat\ ws)$
by (*auto simp add*: *prefix-def*)

lemma *pref-suf-len-split*: **assumes** $p \leq p\ u\ s \leq s\ u\ |p \cdot s| = |u|$
shows $p \cdot s = u$
proof–
have $|p| = |u| - |s|$
using ‹$|p \cdot s| = |u|$› **unfolding** *lenmorph* **by** *force*
from *lq-pref*[*OF* ‹$p \leq p\ u$›, *unfolded lq-def*[*of p u*] *this*
‹$s \leq s\ u$›[*unfolded suf-drop-conv*]]
show *?thesis*.
qed

lemma *pref-cancel-lq*: **assumes** $u \leq p\ x \cdot y$
shows $x^{-1>}u \leq p\ y$
using *pref-lq*[*OF* ‹$u \leq p\ x \cdot y$›, *of x*, *unfolded lq-triv*].

lemma *pref-cancel-lq-ext*: **assumes** $u \cdot v \leq p\ x \cdot y$ **and** $|x| \leq |u|$ **shows** $x^{-1>}u \cdot v \leq p\ y$
proof–
note *pref-prod-long*[*OF append-prefixD*, *OF* ‹$u \cdot v \leq p\ x \cdot y$› ‹$|x| \leq |u|$›]
from *pref-cancel-lq*[*OF* ‹$u \cdot v \leq p\ x \cdot y$›]
show $x^{-1>}u \cdot v \leq p\ y$
unfolding *lq-reassoc*[*OF* ‹$x \leq p\ u$›] **using** ‹$|x| \leq |u|$› **by** *force*
qed

lemma *pref-cancel-lq-ext'*: **assumes** $u \cdot v \leq p\ x \cdot y$ **and** $|u| \leq |x|$ **shows** $v \leq p\ u^{-1>}x \cdot y$
using *pref-lq*[*OF* ‹$u \cdot v \leq p\ x \cdot y$›, *of u*]
unfolding *lq-triv lq-reassoc*[*OF pref-prod-le*[*OF append-prefixD*[*OF* ‹$u \cdot v \leq p\ x \cdot y$›] ‹$|u| \leq |x|$›]].

lemmas *pref-cancel-lq-ext-pref*[*intro*] = *pref-cancel-lq-ext*[*OF - prefix-length-le*] **and**
pref-cancel-lq-ext-pref'[*intro*] = *pref-cancel-lq-ext'*[*OF - prefix-length-le*]

lemma *empty-lq-eq*: $r \leq p\ z \Longrightarrow r^{-1>}z = \varepsilon \Longrightarrow r = z$
unfolding *prefix-def* **by** *force*

lemma *le-if-then-lq*: $|u| \leq |v| \Longrightarrow (if\ |v| \leq |u|\ then\ v^{-1>}u\ \ else\ u^{-1>}v) = u^{-1>}v$
by (*cases* $|u| = |v|$, *simp-all add*: *lq-same-len*)

lemma *append-comp-lq*: $u \cdot v \bowtie w \Longrightarrow v \bowtie u^{-1>}w$
proof (*elim pref-compE*)

assume $u \cdot v \leq_p w$
from *pref-drop*[*OF this*, *of* $|u|$, *unfolded drop-pref*]
show $v \bowtie u^{-1>} w$
unfolding *left-quotient-def* **by** (*rule pref-compI1*)
next
assume $w \leq_p u \cdot v$
from *pref-drop*[*OF this*, *of* $|u|$, *unfolded drop-pref*]
show $v \bowtie u^{-1>} w$
unfolding *left-quotient-def* **by** (*rule pref-compI2*)
qed

2.7.2 Right quotient

lemmas *rqI* = *lqI*[*reversed*] **and**
rq-triv[*simp*] = *lq-triv*[*reversed*] **and**
rq-triv'[*simp*] = *lq-triv'*[*reversed*] **and**
rq-self[*simp*] = *lq-self*[*reversed*] **and**
rq-emp[*simp*] = *lq-emp*[*reversed*] **and**
rq-suf[*simp*] = *lq-pref*[*reversed*] **and**
rq-ssuf[*simp*] = *lq-spref*[*reversed*] **and**
rq-ssuf-nemp = *lq-spref-nemp*[*reversed*] **and**
rq-reassoc = *lq-reassoc*[*reversed*] **and**
rq-len = *lq-len*[*reversed*] **and**
rq-trans = *lq-trans*[*reversed*] **and**
rq-lq-reassoc-suf = *lq-rq-reassoc-suf*[*reversed*] **and**
rq-ne = *lq-ne*[*reversed*] **and**
rq-is-pref[*simp*] = *lq-is-suf*[*reversed*] **and**
suf-rq = *pref-lq*[*reversed*] **and**
ssuf-rq = *spref-lq*[*reversed*] **and**
conjug-rq = *conjug-lq*[*reversed*] **and**
conjug-emp-emp' = *conjug-emp-emp*[*reversed*] **and**
rq-take = *lq-def*[*reversed*] **and**
empty-rq-eq = *empty-lq-eq*[*reversed*] **and**
append-rq = *append-lq*[*reversed*] **and**
rq-same-len = *lq-same-len*[*reversed*] **and**
rq-assoc = *lq-assoc*[*reversed*] **and**
rq-assoc' = *lq-assoc'*[*reversed*] **and**
le-if-then-rq = *le-if-then-lq*[*reversed*] **and**
append-comp-rq = *append-comp-lq*[*reversed*]

2.7.3 Left and right quotients combined

lemma *pref-lq-rq-id*: $p \leq_p w \Longrightarrow w^{<-1}(p^{-1>} w) = p$
unfolding *prefix-def*
using *rq-triv*[*of p* $p^{-1>} w$] **by** *force*

lemmas *suf-rq-lq-id* = *pref-lq-rq-id*[*reversed*]

lemma *rev-lq'*: $r \leq_p s \Longrightarrow rev\ (r^{-1>} s) = (rev\ s)^{<-1}(rev\ r)$
by (*simp add*: *rq-rev-lq*)

lemma *pref-rq-suf-lq*: $s \leq s\ u \Longrightarrow r \leq p\ (u^{<-1}s) \Longrightarrow s \leq s\ (r^{-1>}u)$
using *lq-reassoc*[*of r* $u^{<-1}s$ *s*] *rq-suf*[*of s u*] *triv-suf*[*of s* $r^{-1>}u^{<-1}s$]
by *presburger*

lemmas *suf-lq-pref-rq* = *pref-rq-suf-lq*[*reversed*]

lemma $w{\cdot}s = v \Longrightarrow v^{<-1}s = w$ **using** *rqI*.

lemma *lq-rq-assoc*: $s \leq s\ u \Longrightarrow r \leq p\ (u^{<-1}s) \Longrightarrow (r^{-1>}u)^{<-1}s = r^{-1>}(u^{<-1}s)$
using *lq-reassoc*[*of r* $u^{<-1}s$ *s*] *rq-suf*[*of s u*] *rqI*[*of* $r^{-1>}u^{<-1}s$ *s* $r^{-1>}u$]
by *argo*

lemmas *rq-lq-assoc* = *lq-rq-assoc*[*reversed*]

lemma *lq-prod*: $u \leq p\ v{\cdot}u \Longrightarrow u \leq p\ w \Longrightarrow\ u^{-1>}(v{\cdot}u){\cdot}u^{-1>}w = u^{-1>}(v{\cdot}w)$
using *lq-reassoc*[*of u v* · *u* $u^{-1>}w$] *lq-rq-reassoc-suf*[*of u w v* · *u*, *unfolded rq-triv*[*of v u*]]
by (*simp add*: *suffix-def*)

lemmas *rq-prod* = *lq-prod*[*reversed*]

lemma *pref-suf-mid*: **assumes** $p{\cdot}w{\cdot}s = p'{\cdot}v{\cdot}s'$ **and** $p \leq p\ p'$ **and** $s \leq s\ s'$
shows $v \leq f\ w$
proof–
have $p{\cdot}w{\cdot}s\ = (p \cdot p^{-1>}p') \cdot v \cdot (s'^{<-1}s\ \cdot s)$
using $\langle p{\cdot}w{\cdot}s = p'{\cdot}v{\cdot}s' \rangle$
unfolding *lq-pref*[*OF* $\langle p \leq p\ p' \rangle$] *rq-suf*[*OF* $\langle s \leq s\ s' \rangle$].
thus *?thesis*
by *simp*
qed

2.8 Equidivisibility

Equidivisibility is the following property: if

$$xy = uv,$$

then there exists a word t such that $xt = u$ and $ty = v$, or $ut = x$ and $y = tv$. For monoids over words, this property is equivalent to the freeness of the monoid. As the monoid of all words is free, we can prove that it is equidivisible. Related lemmas based on this property follow.

thm *append-eq-conv-conj*[*folded left-quotient-def*]

lemma *eqd-pref*: $x \cdot y = u \cdot v \Longrightarrow |x| \leq |u| \Longrightarrow x \cdot (x^{-1>}u) = u \wedge (x^{-1>}u) \cdot v = y$
unfolding *append-eq-conv-conj left-quotient-def* **by** *force*

lemma *eqd*: $x \cdot y = u \cdot v \Longrightarrow |x| \leq |u| \Longrightarrow \exists\ t.\ x \cdot t = u \wedge t \cdot v = y$
by (*simp add*: *append-eq-conv-conj*)

lemma *eqd-or*: $x \cdot y = u \cdot v \Longrightarrow \exists\ t.\ x \cdot t = u \wedge t \cdot v = y \vee u \cdot t = x \wedge t \cdot y = v$
unfolding *append-eq-append-conv2* **by** *blast*

lemma *eqdE*: **assumes** $x \cdot y = u \cdot v$ **and** $|x| \leq |u|$
obtains t **where** $x \cdot t = u$ **and** $t \cdot v = y$
using *eqd*[*OF assms*] **by** *blast*

lemma *eqd-less*: **assumes** $x \cdot y = u \cdot v$ **and** $|x| < |u|$
shows $\exists\ t.\ x \cdot t = u \wedge t \cdot v = y \wedge t \neq \varepsilon$
using *eqdE*[*OF assms*(*1*) *less-imp-le*[*OF assms*(*2*)]] *assms*(*2*)
using *append.right-neutral less-not-refl* **by** *metis*

lemma *eqd-lessE*: **assumes** $x \cdot y = u \cdot v$ **and** $|x| < |u|$
obtains t **where** $x \cdot t = u$ **and** $t \cdot v = y$ **and** $t \neq \varepsilon$
using *eqd-less*[*OF assms*] **by** *blast*

lemma *eqdE'*: **assumes** $x \cdot y = u \cdot v$ **and** $|v| \leq |y|$
obtains t **where** $x \cdot t = u$ **and** $t \cdot v = y$
using *eqdE*[*OF assms*(*1*)] *lenarg*[*OF assms*(*1*), *unfolded lenmorph*] *assms*(*2*)
by *auto*

thm *long-pref*

lemma *eqd-pref-suf-iff*: **assumes** $x \cdot y = u \cdot v$ **shows** $x \leq p\ u \longleftrightarrow v \leq s\ y$
by (*rule linorder-le-cases*[*of* $|x|$ $|u|$], *use eqd*[*OF assms*] **in** *blast*)
(*use eqd*[*OF assms*[*symmetric*]] **in** *fastforce*)

lemma *eqd-spref-ssuf-iff*: **assumes** $x \cdot y = u \cdot v$ **shows** $x <p\ u \longleftrightarrow v <s\ y$
using *eqd-pref-suf-iff*[*OF assms*] *assms* **by** *force*

lemma *eqd-pref1*: $x \cdot y = u \cdot v \Longrightarrow |x| \leq |u| \Longrightarrow x \cdot (x^{-1>}u) = u$
using *eqd-pref* **by** *blast*

lemma *eqd-pref2*: $x \cdot y = u \cdot v \Longrightarrow |x| \leq |u| \Longrightarrow (x^{-1>}u) \cdot v = y$
using *eqd-pref* **by** *blast*

lemma *eqd-eq*: **assumes** $x \cdot y = u \cdot v$ $|x| = |u|$ **shows** $x = u$ $y = v$
using *assms* **by** *simp-all*

lemma *eqd-eq-suf*: $x \cdot y = u \cdot v \Longrightarrow |y| = |v| \Longrightarrow x = u \wedge y = v$
by *simp*

lemma *eqd-comp*: **assumes** $x \cdot y = u \cdot v$ **shows** $x \bowtie u$
using *le-cases*[*of* $|x|$ $|u|$ $x \bowtie u$]
eqd-pref1[*of x y u v*, *THEN prefI*[*of x* $x^{-1>}u$ *u*], *OF assms*]
eqd-pref1[*of u v x y*, *THEN prefI*[*of u* $u^{-1>}x$ *x*], *OF assms*[*symmetric*]] **by** *auto*

lemma *eqd-linear-cases* [*elim*]: **assumes** $x \cdot y = u \cdot v$ **and**
$\bigwedge t.\ t \neq \varepsilon \Longrightarrow x \cdot t = u \Longrightarrow t \cdot v = y \Longrightarrow$ *thesis* **and**
$\bigwedge t.\ t \neq \varepsilon \Longrightarrow u \cdot t = x \Longrightarrow t \cdot y = v \Longrightarrow$ *thesis* **and**
$x = u \Longrightarrow y = v \Longrightarrow$ *thesis*
shows *thesis*
using *eqd-or*[*OF assms*(*1*)] *assms*(*2*−) **by** *blast*

lemmas *eqd-comp'* = *eqd-comp*[*reversed*]

— not equal to eqd_pref1[reversed]
lemma *eqd-suf1*: $x \cdot y = u \cdot v \Longrightarrow |x| \leq |u| \Longrightarrow (y^{<-1}v)\cdot v = y$
using *eqd-pref2 rq-triv* **by** *blast*

— not equal to eqd_pref2[reversed]
lemma *eqd-suf2*: **assumes** $x \cdot y = u \cdot v$ $|x| \leq |u|$ **shows** $x \cdot (y^{<-1}v) = u$
using *rq-reassoc*[*OF sufI*[*OF eqd-suf1*[*OF* ‹$x \cdot y = u \cdot v$› ‹$|x| \leq |u|$›]], *of x*, *unfolded* ‹$x \cdot y = u \cdot v$› *rq-triv*[*of u v*]]**.**

— not equal to eqd_pref[reversed]
lemma *eqd-suf*: **assumes** $x \cdot y = u \cdot v$ **and** $|x| \leq |u|$
shows $(y^{<-1}v)\cdot v = y \wedge x \cdot (y^{<-1}v) = u$
using *eqd-suf1*[*OF assms*] *eqd-suf2*[*OF assms*] **by** *blast*

lemma *eqd-exchange-aux*:
assumes $u \cdot v = x \cdot y$ **and** $u \cdot v' = x \cdot y'$ **and** $u' \cdot v = x' \cdot y$ **and** $|u| \leq |x|$
shows $u' \cdot v' = x' \cdot y'$
using *eqd*[*OF* ‹$u \cdot v = x \cdot y$› ‹$|u| \leq |x|$›] *eqd*[*OF* ‹$u \cdot v' = x \cdot y'$› ‹$|u| \leq |x|$›] ‹$u' \cdot v = x' \cdot y$› **by** *force*

lemma *eqd-exchange*:
assumes $u \cdot v = x \cdot y$ **and** $u \cdot v' = x \cdot y'$ **and** $u' \cdot v = x' \cdot y$
shows $u' \cdot v' = x' \cdot y'$
using *eqd-exchange-aux*[*OF assms*] *eqd-exchange-aux*[*OF assms*[*symmetric*], *symmetric*] **by** *force*

hide-fact *eqd-exchange-aux*

2.9 Longest common prefix

lemmas *lcp-simps* = *longest-common-prefix.simps* — provided by Sublist.thy

lemmas *lcp-sym* = *lcp.commute*

— provided by Sublist.thy
lemmas *lcp-pref* = *longest-common-prefix-prefix1*
lemmas *lcp-pref'* = *longest-common-prefix-prefix2*
lemmas *pref-pref-lcp*[*intro*] = *longest-common-prefix-max-prefix*

lemma *lcp-pref-ext*: $u \leq p\ v \Longrightarrow u \leq p\ (u \cdot w) \wedge_p (v \cdot z)$
using *longest-common-prefix-max-prefix prefix-prefix triv-pref* **by** *metis*

lemma *pref-non-pref-lcp-pref*: **assumes** $u \leq p\ w$ **and** $\neg\ u \leq p\ z$ **shows** $w \wedge_p z < p$ u
proof−
note *ruler'*[*OF* ‹$u \leq p\ w$› *lcp-pref*, *of z*, *unfolded prefix-comparable-def*]
with *pref-trans*[*of u w* $\wedge_p$ *z*, *OF - lcp-pref'*] ‹$\neg\ u \leq p\ z$›
show $w \wedge_p z < p\ u$
by *auto*
qed

lemmas *lcp-take* = *pref-take*[*OF lcp-pref*] **and**
lcp-take' = *pref-take*[*OF lcp-pref'*]

lemma *lcp-take-eq*: *take* (|$u \wedge_p v$|) u = *take* (|$u \wedge_p v$|) v
unfolding *lcp-take lcp-take'*..

lemma *lcp-pref-conv*: $u \wedge_p v = u \longleftrightarrow u \leq p\ v$
unfolding *prefix-order.eq-iff*[*of* $u \wedge_p v$ *u*]
using *lcp-pref'*[*of u v*]
lcp-pref[*of u v*] *longest-common-prefix-max-prefix*[*OF self-pref*[*of u*], *of v*]
by *auto*

lemma *lcp-pref-conv'*: $u \wedge_p v = v \longleftrightarrow v \leq p\ u$
using *lcp-pref-conv*[*of v u*, *unfolded lcp-sym*[*of v*]].

lemmas *lcp-left-idemp*[*simp*] = *lcp-pref*[*folded lcp-pref-conv'*] **and**
lcp-right-idemp[*simp*] = *lcp-pref'*[*folded lcp-pref-conv*] **and**
lcp-left-idemp'[*simp*] = *lcp-pref'*[*folded lcp-pref-conv'*] **and**
lcp-right-idemp'[*simp*] = *lcp-pref*[*folded lcp-pref-conv*]

lemma *lcp-per-root*: $r \cdot s \wedge_p s \cdot r \leq p\ r \cdot (r \cdot s \wedge_p s \cdot r)$
using *pref-prod-pref*[*OF pref-prolong*[*OF lcp-pref triv-pref*] *lcp-pref'*].

lemma *lcp-per-root'*: $r \cdot s \wedge_p s \cdot r \leq p\ s \cdot (r \cdot s \wedge_p s \cdot r)$
using *lcp-per-root*[*of s r*, *unfolded lcp-sym*[*of* $s \cdot r$]].

lemma *pref-lcp-pref*: $w \leq p\ u \wedge_p v \Longrightarrow w \leq p\ u$
using *lcp-pref pref-trans* **by** *blast*

lemma *pref-lcp-pref'*: $w \leq p\ u \wedge_p v \Longrightarrow w \leq p\ v$
using *pref-lcp-pref*[*of w v u*, *unfolded lcp-sym*[*of v u*]].

lemmas *lcp-self* = *lcp.idem*

lemma *lcp-eq-len*: |u| = |$u \wedge_p v$|$\Longrightarrow u = u \wedge_p v$
using *long-pref*[*OF lcp-pref*, *of u v*] **by** *auto*

lemma *lcp-len*: $|u| \leq |u \wedge_p v| \Longrightarrow u \leq p\ v$
using *long-pref*[*OF lcp-pref*, *of u v*] **unfolding** *lcp-pref-conv*[*symmetric*].

lemma *lcp-len'*: $\neg\ u \leq p\ v \Longrightarrow |u \wedge_p v| < |u|$
using *not-le-imp-less*[*OF contrapos-nn*[*OF - lcp-len*]].

lemma *incomp-lcp-len*: $\neg\ u \bowtie v \Longrightarrow |u \wedge_p v| < min\ |u|\ |v|$
using *lcp-len'*[*of u v*] *lcp-len'*[*of v u*] **unfolding** *lcp-sym*[*of v*] *min-less-iff-conj*
by *blast*

lemma *lcp-ext-right-conv*: $\neg\ r \bowtie r' \Longrightarrow (r \cdot u) \wedge_p (r' \cdot v) = r \wedge_p r'$
unfolding *prefix-comparable-def*
by (*induct r r' rule*: *list-induct2'*) *simp-all*

lemma *lcp-ext-right* [*case-names comp non-comp*]: **obtains** $r \bowtie r' \mid (r \cdot u) \wedge_p (r' \cdot v) = r \wedge_p r'$
using *lcp-ext-right-conv* **by** *blast*

lemma *lcp-same-len*: $|u| = |v| \Longrightarrow u \neq v \Longrightarrow u \cdot w \wedge_p v \cdot w' = u \wedge_p v$
using *pref-comp-eq* **by** (*cases rule*: *lcp-ext-right*) (*elim notE*)

lemma *lcp-mismatch*: $|u \wedge_p v| < |u| \Longrightarrow |u \wedge_p v| < |v| \Longrightarrow u!\ |u \wedge_p v| \neq v!\ |u \wedge_p v|$
by (*induct u v rule*: *list-induct2'*) *auto*

lemma *lcp-mismatch'*: $\neg\ u \bowtie v \Longrightarrow u!\ |u \wedge_p v| \neq v!\ |u \wedge_p v|$
using *incomp-lcp-len lcp-mismatch* **unfolding** *min-less-iff-conj*..

lemma *lcp-mismatchE*: **assumes** $\neg\ us \bowtie vs$
obtains *us' vs'*
where $(us \wedge_p vs) \cdot us' = us$ **and** $(us \wedge_p vs) \cdot vs' = vs$ **and**
$us' \neq \varepsilon$ **and** $vs' \neq \varepsilon$ **and** $hd\ us' \neq hd\ vs'$
proof −
obtain *us' vs'* **where** *us*: $(us \wedge_p vs) \cdot us' = us$ **and** *vs*: $(us \wedge_p vs) \cdot vs' = vs$
using *prefixE*[*OF lcp-pref prefixE*[*OF lcp-pref'*]]
unfolding *eq-commute*[*of x·y* **for** *x y*].
with ‹$\neg\ us \bowtie vs$› **have** $us' \neq \varepsilon$ **and** $vs' \neq \varepsilon$
unfolding *prefix-comparable-def lcp-pref-conv*[*symmetric*] *lcp-sym*[*of vs*]
by *fastforce*+
hence *us*! $|us \wedge_p vs| = hd\ us'$ **and** *vs*! $|us \wedge_p vs| = hd\ vs'$
using *hd-lq-conv-nth*[*OF triv-spref*, *symmetric*] **unfolding** *lq-triv*
unfolding *arg-cong*[*OF us*[*symmetric*], *of nth*] *arg-cong*[*OF vs*[*symmetric*], *of nth*]
by *blast*+
from *lcp-mismatch'*[*OF* ‹$\neg\ us \bowtie vs$›, *unfolded this*]
have $hd\ us' \neq hd\ vs'$.
from *that*[*OF us vs* ‹$us' \neq \varepsilon$› ‹$vs' \neq \varepsilon$› *this*]
show *thesis*.
qed

lemma *lcp-mismatch-lq*: **assumes** $\neg\ u \bowtie v$
shows
$(u \wedge_p v)^{-1>} u \neq \varepsilon$ **and**
$(u \wedge_p v)^{-1>} v \neq \varepsilon$ **and**
$hd\ ((u \wedge_p v)^{-1>} u) \neq hd\ ((u \wedge_p v)^{-1>} v)$
proof−
from *lcp-mismatchE*[*OF assms*]
obtain *su sv* **where** $(u \wedge_p v) \cdot su = u$ **and**
$(u \wedge_p v) \cdot sv = v$ **and** $su \neq \varepsilon$ **and** $sv \neq \varepsilon$ **and** $hd\ su \neq hd\ sv$**.**
thus $(u \wedge_p v)^{-1>} u \neq \varepsilon$ **and** $(u \wedge_p v)^{-1>} v \neq \varepsilon$ **and** $hd\ ((u \wedge_p v)^{-1>} u) \neq hd\ ((u \wedge_p v)^{-1>} v)$
using *lqI*[*OF* ‹$(u \wedge_p v) \cdot su = u$›] *lqI*[*OF* ‹$(u \wedge_p v) \cdot sv = v$›] **by** *blast*+
qed

lemma *lcp-ext-left*: $(z \cdot u) \wedge_p (z \cdot v) = z \cdot (u \wedge_p v)$
by (*induct z*) *auto*

lemma *lcp-first-letters*: $u!0 \neq v!0 \Longrightarrow u \wedge_p v = \varepsilon$
by (*induct u v rule*: *list-induct2'*) *auto*

lemma *lcp-first-mismatch*: $a \neq b \Longrightarrow w \cdot [a] \cdot u \wedge_p w \cdot [b] \cdot v\ = w$
by (*simp add*: *lcp-ext-left*)

lemma *lcp-first-mismatch'*: $a \neq b \Longrightarrow u \cdot [a] \wedge_p u \cdot [b] = u$
using *lcp-first-mismatch*[*of a b u ε ε*] **by** *simp*

lemma *lcp-mismatch-eq-len*: **assumes** $|u| = |v|$ $x \neq y$ **shows** $u \cdot [x] \wedge_p v \cdot [y] = u \wedge_p v$
using *lcp-self lcp-first-mismatch'*[*OF* ‹$x \neq y$›] *lcp-same-len*[*OF* ‹$|u| = |v|$›]
by (*cases* $u = v$) *auto*

lemma *lcp-first-mismatch-pref*: **assumes** $p \cdot [a] \leq p\ u$ **and** $p \cdot [b] \leq p\ v$ **and** $a \neq b$
shows $u \wedge_p v = p$
using *assms*(*1*−*2*) *lcp-first-mismatch*[*OF* ‹$a \neq b$›]
unfolding *prefix-def rassoc* **by** *blast*

lemma *lcp-append-monotone*: $u \wedge_p x \leq p\ (u \cdot v) \wedge_p (x \cdot y)$
by (*simp add*: *lcp.mono*)

lemma *lcp-distinct-hd*: $hd\ u \neq hd\ v \Longrightarrow u \wedge_p v = \varepsilon$
using *pref-hd-eq'*[*OF lcp-pref lcp-pref'*] **by** *blast*

lemma *nemp-lcp-distinct-hd*: **assumes** $u \neq \varepsilon$ **and** $v \neq \varepsilon$ **and** $u \wedge_p v = \varepsilon$
shows $hd\ u \neq hd\ v$
proof
assume $hd\ u = hd\ v$
from *lcp-ext-left*[*of* [*hd u*] *tl u tl v*,
unfolded hd-tl[*OF* ‹$u \neq \varepsilon$›] *hd-tl*[*OF* ‹$v \neq \varepsilon$›, *folded this*]]

show *False*
using ‹$u \wedge_p v = \varepsilon$› **by** *simp*
qed

lemma *lcp-lenI*: **assumes** $i < min\ |u|\ |v|$ **and** $take\ i\ u = take\ i\ v$ **and** $u!i \neq v!i$
shows $i = |u \wedge_p v|$
proof–
have *u*: $take\ i\ u \cdot [u\ !\ i] \cdot drop\ (Suc\ i)\ u = u$
using ‹$i < min\ |u|\ |v|$› *id-take-nth-drop*[*of i u*] **by** *simp*
have *v*: $take\ i\ u \cdot [v\ !\ i] \cdot drop\ (Suc\ i)\ v = v$
using ‹$i < min\ |u|\ |v|$›
unfolding ‹$take\ i\ u = take\ i\ v$› **using** *id-take-nth-drop*[*of i v*] **by** *force*
from *lcp-first-mismatch*[*OF* ‹$u!i \neq v!i$›, *of take i u drop* (*Suc i*) *u drop* (*Suc i*) *v*, *unfolded u v*]
have $u \wedge_p v = take\ i\ u$.
thus *?thesis*
using ‹$i < min\ |u|\ |v|$› **by** *auto*
qed

lemma *lcp-prefs*: $|u \cdot w \wedge_p v \cdot w'| < |u| \Longrightarrow |u \cdot w \wedge_p v \cdot w'| < |v| \Longrightarrow u \wedge_p v = u \cdot w \wedge_p v \cdot w'$
by (*induct u v rule*: *list-induct2'*) *auto*

lemma *lcp-extend-eq*: **assumes** $u \leq p\ v$ **and** $u' \leq p\ v'$ **and**
$|v \wedge_p v'| \leq |u|$ **and** $|v \wedge_p v'| \leq |u'|$
shows $u \wedge_p u' = v \wedge_p v'$
proof–
consider $|v \wedge_p v'| = |u| \mid |v \wedge_p v'| = |u'| \mid |v \wedge_p v'| < |u| \wedge |v \wedge_p v'| < |u'|$
using *assms*(*3*–*4*) **by** *force*
thus *?thesis*
proof (*cases*)
assume $|v \wedge_p v'| = |u|$
from *ruler-eq-len*[*OF longest-common-prefix-prefix1* ‹$u \leq p\ v$› *this*]
have $u \leq p\ u'$
using *prefix-length-prefix*[*OF longest-common-prefix-prefix2 assms*(*2,4*)] **by** *blast*
thus *?thesis*
unfolding ‹$v \wedge_p v' = u$› *lcp-pref-conv*.
next
assume $|v \wedge_p v'| = |u'|$
from *ruler-eq-len*[*OF longest-common-prefix-prefix2* ‹$u' \leq p\ v'$› *this*]
have $u' \leq p\ u$
using *prefix-length-prefix*[*OF longest-common-prefix-prefix1 assms*(*1,3*)] **by** *blast*
thus *?thesis*
unfolding ‹$v \wedge_p v' = u'$› *lcp-pref-conv'*.
next
assume $|v \wedge_p v'| < |u| \wedge |v \wedge_p v'| < |u'|$
thus *?thesis*

using *lcp-prefs*[*of u* $u^{-1>}$ *v u' u'*$^{-1>}$ *v'*, *unfolded lq-pref*[*OF* ‹*u* ≤*p v*›] *lq-pref*[*OF* ‹*u'* ≤*p v'*›]]
by *blast*
qed
qed

lemma *long-lcp-same*: **assumes** ¬ (*u* ∧*p v* ≤*p w*) **shows** *u* ∧*p w* = *v* ∧*p w*
proof–
have *v* ∧*p w* ≤*p u*
using *ruler*[*OF lcp-pref' lcp-pref'*, *of u v w*] *assms* **unfolding** *lcp-sym*[*of v*] **by** *force*
have *u* ∧*p w* ≤*p v*
using *ruler*[*OF lcp-pref lcp-pref*, *of u v w*] *assms* **by** *force*
show *?thesis*
unfolding *prefix-order.eq-iff*
using ‹*v* ∧*p w* ≤*p u*› ‹*u* ∧*p w* ≤*p v*› **by** *force*
qed

lemma *long-lcp-sameE*: **obtains** *u* ∧*p v* ≤*p w* | *u* ∧*p w* = *v* ∧*p w*
using *long-lcp-same* **by** *blast*

lemma *ruler-spref-lcp*: **assumes** *u* ∧*p w* <*p v* ∧*p w*
shows *u* ∧*p v* = *u* ∧*p w*
proof–
have ¬ *v* ∧*p w* ≤*p u*
using *prefix-order.leD*[*of v* ∧*p w u* ∧*p w*] *assms* **by** *force*
from *long-lcp-same*[*OF this*]
show *?thesis*
unfolding *lcp-sym*[*of u*].
qed

2.9.1 Longest common prefix and prefix comparability

lemma *lexord-cancel-right*: (*u* · *z*, *v* · *w*) ∈ *lexord r* ⟹ ¬ *u* ⋈ *v* ⟹ (*u*,*v*) ∈ *lexord r*
unfolding *prefix-comparable-def*
by (*induction rule*: *list-induct2'*) *auto*

lemma *lcp-rulersE*: **assumes** *r* ≤*p s* **and** *r'* ≤*p s'* **obtains** *r* ⋈ *r'* | *s* ∧*p s'* = *r* ∧*p r'*
by (*cases rule*: *lcp-ext-right*[*of* - - - *r*$^{-1>}$*s r'*$^{-1>}$*s'*]) (*assumption*, *simp only*: *assms lq-pref*)

lemma *lcp-rulers*: *r* ≤*p s* ⟹ *r'* ≤*p s'* ⟹ (*r* ⋈ *r'* ∨ *s* ∧*p s'* = *r* ∧*p r'*)
by (*cases rule*: *lcp-ext-right*[*of* - - - *r*$^{-1>}$*s r'*$^{-1>}$*s'*], *blast*) (*meson lcp-rulersE*)

lemma *lcp-rulers'*: *w* ≤*p r* ⟹ *w'* ≤*p s* ⟹ ¬ *w* ⋈ *w'* ⟹ (*r* ∧*p s*) = *w* ∧*p w'*
using *lcp-rulers* **by** *blast*

lemma *lcp-ruler*: $r \bowtie w1 \Longrightarrow r \bowtie w2 \Longrightarrow \neg\ w1 \bowtie w2 \Longrightarrow r \leq p\ w1 \wedge_p w2$
unfolding *prefix-comparable-def* **by** (*meson pref-pref-lcp pref-trans ruler*)

lemma *comp-monotone*: $w \bowtie r \Longrightarrow u \leq p\ w \Longrightarrow u \bowtie r$
using *pref-compI1*[*OF pref-trans*] *ruler'* **by** (*elim pref-compE*)

lemma *comp-monotone'*: $w \bowtie r \Longrightarrow w \wedge_p w' \bowtie r$
using *comp-monotone*[*OF - lcp-pref*]**.**

lemma *double-ruler-aux*: **assumes** $w \bowtie r$ **and** $w' \bowtie r'$ **and** $\neg\ r \bowtie r'$ **and** $|w| \leq |w'|$
shows $w \wedge_p w' = take\ |w|\ (r \wedge_p r')$
proof−
have *pref1*: $w \wedge_p w' \leq p\ r \wedge_p r'$
using *comp-monotone'*[*OF* ‹$w' \bowtie r'$›] *lcp-ruler*[*OF comp-monotone'*[*OF* ‹$w \bowtie r$›] - ‹$\neg\ r \bowtie r'$›]
unfolding *lcp-sym*[*of w'*] **by** *simp*
show *?thesis*
proof (*cases*)
assume $w \bowtie w'$
hence $w \wedge_p w' = w$
using ‹$|w| \leq |w'|$›
by (*simp add*: *comp-shorter lcp.absorb1*)
show *?thesis*
using *pref-take*[*OF pref1*, *symmetric*] **unfolding** ‹$w \wedge_p w' = w$›**.**
next
assume $\neg\ w \bowtie w'$
hence *pref2*: $r \wedge_p r' \leq p\ w \wedge_p w'$
using *comp-monotone'*[*OF* ‹$w' \bowtie r'$›[*symmetric*]] *lcp-ruler*[*OF comp-monotone'*[*OF* ‹$w \bowtie r$›[*symmetric*]] - ‹$\neg\ w \bowtie w'$›]
unfolding *lcp-sym*[*of r'*] **by** *simp*
hence $w \wedge_p w' = r \wedge_p r'$
using *pref1 pref-antisym* **by** *blast*
then show *?thesis*
using *lcp-take len-take2 take-all-iff* **by** *metis*
qed
qed

lemma *double-ruler*: **assumes** $w \bowtie r$ **and** $w' \bowtie r'$ **and** $\neg\ r \bowtie r'$
shows $w \wedge_p w' = take\ (min\ |w|\ |w'|)\ (r \wedge_p r')$
by (*cases* $|w|$ $|w'|$ *rule*: *le-cases*)
(*use double-ruler-aux*[*OF assms*] *double-ruler-aux*[*OF assms*(*2*,*1*) *assms*(*3*)[*symmetric*], *unfolded lcp-sym*[*of r'*] *lcp-sym*[*of w'*]]
in *linarith*)+

hide-fact *double-ruler-aux*

lemmas *pref-lcp-iff* = *lcp.bounded-iff*

lemma *pref-comp-ruler*: **assumes** $w \bowtie u \cdot [x]$ **and** $w \bowtie v \cdot [y]$ **and** $x \neq y$ **and** $|u| = |v|$
shows $w \leq p\ u \wedge w \leq p\ v$
using *double-ruler*[*OF* ‹$w \bowtie u \cdot [x]$› ‹$w \bowtie v \cdot [y]$› *mismatch-incopm*[*OF* ‹$|u| = |v|$› ‹$x \neq y$›]]
take-is-prefix lcp-self lcp-mismatch-eq-len[*OF* ‹$|u| = |v|$› ‹$x \neq y$›] *pref-lcp-iff* **by** *metis*

lemma *comp-per-partes*:
shows $(u \bowtie w \wedge v \bowtie u^{-1>} w) \longleftrightarrow u \cdot v \bowtie w$
proof
assume $u \cdot v \bowtie w$
from *comp-monotone*[*OF - triv-pref*, *OF this*] *append-comp-lq*[*OF this*]
show $u \bowtie w \wedge v \bowtie u^{-1>} w$
by *blast*
next
assume *c2*: $u \bowtie w \wedge v \bowtie u^{-1>} w$
hence $u \cdot v \bowtie u \cdot u^{-1>} w$
unfolding *comp-cancel* **by** *blast*
show $u \cdot v \bowtie w$
by (*rule pref-compE*[*OF conjunct1*[*OF c2*]], *use* ‹$u \cdot v \bowtie u \cdot u^{-1>} w$› **in** *force*,*blast*)
qed

lemmas *scomp-per-partes* = *comp-per-partes*[*reversed*]

2.9.2 Longest common suffix

definition *longest-common-suffix* (- $\wedge_s$ - [*61*,*62*] *64*)
where
longest-common-suffix u v ≡ *rev* (*rev u* $\wedge_p$ *rev v*)

lemma *lcs-lcp* [*reversal-rule*]: *rev u* $\wedge_p$ *rev v* = *rev* (*u* $\wedge_s$ *v*)
unfolding *longest-common-suffix-def rev-rev-ident*..

lemmas *lcs-simp* = *lcp-simps*[*reversed*] **and**
lcs-sym = *lcp-sym*[*reversed*] **and**
lcs-suf = *lcp-pref*[*reversed*] **and**
lcs-suf′ = *lcp-pref′*[*reversed*] **and**
suf-suf-lcs = *pref-pref-lcp*[*reversed*] **and**
suf-non-suf-lcs-suf = *pref-non-pref-lcp-pref*[*reversed*] **and**
lcs-drop-eq = *lcp-take-eq*[*reversed*] **and**
lcs-take = *lcp-take*[*reversed*] **and**
lcs-take′ = *lcp-take′*[*reversed*] **and**
lcs-suf-conv = *lcp-pref-conv*[*reversed*] **and**
lcs-suf-conv′ = *lcp-pref-conv′*[*reversed*] **and**
lcs-per-root = *lcp-per-root*[*reversed*] **and**
lcs-per-root′ = *lcp-per-root′*[*reversed*] **and**
suf-lcs-suf = *pref-lcp-pref*[*reversed*] **and**

suf-lcs-suf′ = *pref-lcp-pref′*[*reversed*] **and**
lcs-self[*simp*] = *lcp-self*[*reversed*] **and**
lcs-eq-len = *lcp-eq-len*[*reversed*] **and**
lcs-len = *lcp-len*[*reversed*] **and**
lcs-len′ = *lcp-len′*[*reversed*] **and**
suf-incomp-lcs-len = *incomp-lcp-len*[*reversed*] **and**
lcs-ext-left-conv = *lcp-ext-right-conv*[*reversed*] **and**
lcs-ext-left [*case-names comp non-comp*] = *lcp-ext-right*[*reversed*] **and**
lcs-same-len = *lcp-same-len*[*reversed*] **and**
lcs-mismatch = *lcp-mismatch*[*reversed*] **and**
lcs-mismatch′ = *lcp-mismatch′*[*reversed*] **and**
lcs-mismatchE = *lcp-mismatchE*[*reversed*] **and**
lcs-mismatch-rq = *lcp-mismatch-lq*[*reversed*] **and**
lcs-ext-right = *lcp-ext-left*[*reversed*] **and**
lcs-first-mismatch = *lcp-first-mismatch*[*reversed*, *unfolded rassoc*] **and**
lcs-first-mismatch′ = *lcp-first-mismatch′*[*reversed*, *unfolded rassoc*] **and**
lcs-mismatch-eq-len = *lcp-mismatch-eq-len*[*reversed*] **and**
lcs-first-mismatch-suf = *lcp-first-mismatch-pref*[*reversed*] **and**
lcs-rulers = *lcp-rulers*[*reversed*] **and**
lcs-rulers′ = *lcp-rulers′*[*reversed*] **and**
suf-suf-lcs′ = *lcp.mono*[*reversed*] **and**
lcs-distinct-last = *lcp-distinct-hd*[*reversed*] **and**
lcs-lenI = *lcp-lenI*[*reversed*] **and**
lcs-sufs = *lcp-prefs*[*reversed*]

lemmas *lcs-ruler* = *lcp-ruler*[*reversed*] **and**
suf-comp-monotone = *comp-monotone*[*reversed*] **and**
suf-comp-monotone′ = *comp-monotone′*[*reversed*] **and**
double-ruler-suf = *double-ruler*[*reversed*] **and**
suf-lcs-iff = *pref-lcp-iff*[*reversed*] **and**
suf-comp-ruler = *pref-comp-ruler*[*reversed*]

2.10 Mismatch

The first pair of letters on which two words/lists disagree

In the following definition, $\varepsilon!0$ has a role of an end marker of each word. It also allows the alternative definition which follows

function *mismatch-pair* :: *'a list* ⇒ *'a list* ⇒ (*'a* × *'a*) **where**
mismatch-pair ε *v* = (ε!0, *v*!0) |
mismatch-pair v ε = (*v*!0, ε!0) |
mismatch-pair (*a*#*u*) (*b*#*v*) = (*if a*=*b then mismatch-pair u v else* (*a*,*b*))
using *shuffles.cases* **by** *blast*+
termination
by (*relation measure* (λ (*t*,*s*). *length t*), *simp-all*)

Alternatively, mismatch pair may be defined using the longest common prefix as follows.

lemma *mismatch-pair-lcp*: *mismatch-pair u v* = (*u*!|*u*∧$_p$*v*|,*v*!|*u*∧$_p$*v*|)
by (*induction u v rule*: *mismatch-pair.induct*) *simp-all*

For incomparable words the pair is out of diagonal.

lemma *incomp-neq*: ¬ *u* ⋈ *v* ⟹ (*mismatch-pair u v*) ∉ *Id*
unfolding *mismatch-pair-lcp* **by** (*simp add*: *lcp-mismatch'*)

lemma *mismatch-ext-left*: ¬ *u* ⋈ *v* ⟹ *mismatch-pair u v* = *mismatch-pair* (*p*·*u*) (*p*·*v*)
unfolding *mismatch-pair-lcp* **by** (*simp add*: *lcp-ext-left*)

lemma *mismatch-ext-right*: **assumes** ¬ *u* ⋈ *v*
shows *mismatch-pair u v* = *mismatch-pair* (*u*·*z*) (*v*·*w*)
proof−
have *less1*: |*u* ∧$_p$ *v*| < |*u*| **and** *less2*: |*v* ∧$_p$ *u*| < |*v*|
using *lcp-len'*[*of u v*] *lcp-len'*[*of v u*] *assms* **by** *auto*
show *?thesis*
unfolding *mismatch-pair-lcp* **unfolding** *pref-index*[*OF triv-pref less1*, *of z*] *pref-index*[*OF triv-pref less2*, *of w*, *unfolded lcp-sym*[*of v*]]
using *assms lcp-ext-right*[*of u v - z w*] **by** *metis*
qed

lemma *mismatchI*: ¬ *u* ⋈ *v* ⟹ *i* < *min* |*u*| |*v*| ⟹ *take i u* = *take i v* ⟹ *u*!*i* ≠ *v*!*i*
⟹ *mismatch-pair u v* = (*u*!*i*,*v*!*i*)
unfolding *mismatch-pair-lcp* **using** *lcp-lenI* **by** *blast*

For incomparable words, the mismatch letters work in a similar way as the lexicographic order

lemma *mismatch-lexord*: **assumes** ¬ *u* ⋈ *v* **and** *mismatch-pair u v* ∈ *r*
shows (*u*,*v*) ∈ *lexord r*
unfolding *lexord-take-index-conv mismatch-pair-lcp*
using ‹*mismatch-pair u v* ∈ *r*›[*unfolded mismatch-pair-lcp*]
incomp-lcp-len[*OF assms*(*1*)] *lcp-take-eq* **by** *blast*

However, the equivalence requires r to be irreflexive. (Due to the definition of lexord which is designed for irreflexive relations.)

lemma *lexord-mismatch*: **assumes** ¬ *u* ⋈ *v* **and** *irrefl r*
shows *mismatch-pair u v* ∈ *r* ⟷ (*u*,*v*) ∈ *lexord r*
proof
assume (*u*,*v*) ∈ *lexord r*
obtain *i* **where** *i* < *min* |*u*| |*v*| **and** *take i u* = *take i v* **and** (*u* ! *i*, *v* ! *i*) ∈ *r*
using ‹(*u*,*v*) ∈ *lexord r*›[*unfolded lexord-take-index-conv*] ‹¬ *u* ⋈ *v*› *pref-take-conv* **by** *blast*
have *u*!*i* ≠ *v*!*i*
using ‹*irrefl r*›[*unfolded irrefl-def*] ‹(*u* ! *i*, *v* ! *i*) ∈ *r*› **by** *fastforce*
from ‹(*u* ! *i*, *v* ! *i*) ∈ *r*›[*folded mismatchI*[*OF* ‹¬ *u* ⋈ *v*› ‹*i* < *min* |*u*| |*v*|› ‹*take i u* = *take i v*› ‹*u*!*i* ≠ *v*!*i*›]]

show *mismatch-pair u v* ∈ *r*.
next
from *mismatch-lexord*[*OF* ‹¬ *u* ⋈ *v*›]
show *mismatch-pair u v* ∈ *r* ⟹ (*u*, *v*) ∈ *lexord r*.
qed

2.11 Factor properties

lemmas [*simp*] = *sublist-Cons-right*

lemma *rev-fac*[*reversal-rule*]: *rev u* ≤*f rev v* ⟷ *u* ≤*f v*
using *Sublist.sublist-rev*.

lemma *fac-pref*: *u* ≤*f v* ≡ ∃ *p*. *p* · *u* ≤*p v*
by (*simp add*: *prefix-def fac-def*)

lemma *fac-pref-suf*: *u* ≤*f v* ⟹ ∃ *p*. *p* ≤*p v* ∧ *u* ≤*s p*
using *sublist-altdef* **by** *blast*

lemma *pref-suf-fac*: *r* ≤*p v* ⟹ *u* ≤*s r* ⟹ *u* ≤*f v*
using *sublist-altdef* **by** *blast*

lemmas
fac-suf = *fac-pref*[*reversed*] **and**
fac-suf-pref = *fac-pref-suf*[*reversed*] **and**
suf-pref-fac = *pref-suf-fac*[*reversed*]

lemma *suf-pref-eq*: *s* ≤*s p* ⟹ *p* ≤*p s* ⟹ *p* = *s*
using *sublist-order.order.eq-iff* **by** *blast*

lemma *fac-triv*: *p*·*x*·*q* = *x* ⟹ *p* = *ε*
using *long-pref*[*OF prefI suf-len'*] **unfolding** *append-self-conv2 rassoc*.

lemma *fac-triv'*: *p*·*x*·*q* = *x* ⟹ *q* = *ε*
using *fac-triv*[*reversed*] **unfolding** *rassoc*.

lemmas
suf-fac = *suffix-imp-sublist* **and**
pref-fac = *prefix-imp-sublist*

lemma *fac-ConsE*: **assumes** *u* ≤*f* (*a*#*v*)
obtains *u* ≤*p* (*a*#*v*) | *u* ≤*f v*
using *assms* **unfolding** *sublist-Cons-right*
by *blast*

lemmas
fac-snocE = *fac-ConsE*[*reversed*]

lemma *fac-elim-suf*: **assumes** *f* ≤*f m*·*s* ¬ *f* ≤*f s*

```
  shows f ≤f m·(take (|f|−1) s)
  using assms
proof(induction s rule:rev-induct)
  case (snoc s ss)
  have ¬ f ≤f ss
    using ‹¬ f ≤f ss · [s]›[unfolded sublist-append] by blast

  show ?case
  proof(cases)
    assume  f ≤f m · ss
    hence f ≤f m · take (|f| − 1) ss
      using ‹¬ f ≤f ss› snoc.IH by blast
    then show ?thesis
      unfolding take-append lassoc using append-assoc sublist-append by metis
  next
    assume ¬ f ≤f m · ss
    hence f ≤s m · ss · [s]
      using  snoc.prems(1)[unfolded lassoc sublist-snoc, unfolded rassoc] by blast
    from suf-prod-le[OF this, THEN suffix-imp-sublist] ‹¬ f ≤f ss · [s]›
    have |ss · [s]| < |f|
      by linarith
    from this Suc-less-iff-Suc-le length-append-singleton[of ss s]
    show ?thesis
      using snoc.prems(1) take-all-iff by metis
  qed
qed auto

lemmas fac-elim-pref = fac-elim-suf[reversed]

lemma fac-elim: assumes f ≤f p·m·s and  ¬ f ≤f p and ¬ f ≤f s
  shows f ≤f (drop (|p| − (|f| − 1)) p) · m · (take (|f|−1) s)
  using  fac-elim-suf[OF fac-elim-pref[OF ‹f ≤f p·m·s›, unfolded lassoc], unfolded
rassoc, OF assms(2−3)].

lemma fac-ext-pref: u ≤f w ⟹ u ≤f p · w
  by (meson sublist-append)

lemma fac-ext-suf: u ≤f w ⟹ u ≤f w · s
  by (meson sublist-append)

lemma fac-ext: u ≤f w ⟹ u ≤f p · w · s
  by (meson fac-ext-pref fac-ext-suf)

lemma fac-ext-hd:u ≤f w ⟹ u ≤f a#w
  by (metis sublist-Cons-right)

lemma card-switch-fac: assumes 2 ≤ card (set ws)
  obtains c d where c ≠ d and  [c,d] ≤f ws
  using assms
```

proof (*induct ws*, *force*)
case (*Cons a ws*)
then show *?case*
proof (*cases*)
assume *2* ≤ *card* (*set ws*)
from *Cons.hyps*[*OF - this*] *Cons.prems*(*1*) *fac-ext-hd*
show *thesis* **by** *metis*
next
assume ¬ *2* ≤ *card* (*set ws*)
have *ws* ≠ *ε*
using ‹*2* ≤ *card* (*set* (*a* # *ws*))› **by** *force*
hence *a* = *hd ws* ⟹ *set* (*a* # *ws*) = *set ws*
using *hd-Cons-tl*[*OF* ‹*ws* ≠ *ε*›] **by** *force*
hence *a* ≠ *hd ws*
using ‹*2* ≤ *card* (*set* (*a* # *ws*))› ‹¬ *2* ≤ *card* (*set ws*)› **by** *force*
from *Cons.prems*(*1*)[*OF this*]
show *thesis*
using *Cons-eq-appendI*[*OF - hd-tl*[*OF* ‹*ws* ≠ *ε*›, *symmetric*]] *sublist-append-rightI*
by *blast*
qed
qed

lemma *fac-overlap-len*: **assumes** *u* ≤*f x* · *y* · *z* **and** |*u*| ≤ |*y*|
shows *u* ≤*f x* · *y* ∨ *u* ≤*f y* · *z*
proof−
obtain *s p* **where** *eq*: *x* · *y* · *z* = *p* · *u* · *s*
using ‹*u* ≤*f x* · *y* · *z*› **unfolding** *fac-def* **by** *blast*
show *?thesis*
proof (*rule le-cases*)
assume |*p*| ≤ |*x*|
from *add-le-mono*[*OF this* ‹|*u*| ≤ |*y*|›]
have |*p* · *u*| ≤ |*x* · *y*|
unfolding *lenmorph*.
from *eq-le-pref*[*OF eq*[*symmetric*, *unfolded lassoc*] *this*]
have *u* ≤*f x* · *y*
using *fac-pref* **by** *blast*
thus *?thesis* **by** *blast*
next
assume |*x*| ≤ |*p*|
from *eqd*[*OF eq this*]
show *u* ≤*f x* · *y* ∨ *u* ≤*f y* · *z*
unfolding *fac-def* **by** *metis*
qed
qed

2.12 Power and its properties

The core facts about iterated concatenation of a list are given in the AFP theory *List-Power.List-Power*. Word powers are an important research topic in Combinatorics on Words. We adopt a specific notation for the word power.

abbreviation *listpow* :: *'a list* ⇒ *nat* ⇒ *'a list* (**infixr** ‹$^{@}$› *80*)
  where $f\ ^{@}\ n \equiv compow\ n\ f$

some simplified names

lemmas
pow-zero = *pow-list-zero* **and**
pow-Suc = *pow-list-Suc* **and**
pow-Suc2 = *pow-list-Suc2* **and**
pow-comm = *pow-list-comm* **and**
pow-add = *pow-list-add* **and**
pow-mult = *pow-list-mult* **and**
comm-pow-comm = *pow-list-commuting-commutes* **and**
rev-pow = *rev-pow-list* **and**
pow-len = *length-pow-list* **and**
eq-pow-exp = *eq-pow-list-iff-eq-exp*

lemma *pow-pos*: $0 < k \Longrightarrow a^{@}k = a \cdot a^{@}(k-1)$
  by (*simp add*: *pow-list-eq-if*)

lemma *pow-pos2*:$0 < k \Longrightarrow a^{@}k = a^{@}(k-1) \cdot a$
  unfolding *pow-list-Suc2*[*symmetric*] **by** *force*

lemma *pow-diff*: $k < n \Longrightarrow a^{@}(n - k) = a \cdot a^{@}(n-k-1)$
  by (*rule pow-pos*) *simp*

lemma *pow-list-3*: $u^{@}3 = u{\cdot}u{\cdot}u$
  by (*simp add*: *pow-pos*)

named-theorems *exp-simps*
lemmas [*exp-simps*] = *pow-list-zero pow-list-one pow-list-Nil numeral-nat less-eq-Suc-le neq0-conv pow-list-mult*[*symmetric*]

named-theorems *cow-simps*
lemmas [*cow-simps*] = *emp-simps exp-simps*

— more power properties

lemma *nemp-pref-nemp*: $u \leq p\ v \Longrightarrow u \neq \varepsilon \Longrightarrow v \neq \varepsilon$

by *force*

lemma *pref-concat-emp*[*intro*]: $us \leq p\ vs \Longrightarrow concat\ vs\ = \varepsilon \Longrightarrow concat\ us = \varepsilon$
using *prefD* **by** *fastforce*

lemma *nemp-exp-pos*: $w \neq \varepsilon \Longrightarrow r^{@}k = w \Longrightarrow 0 < k$
using *pow-list-not-NilD* **by** *blast*

lemma *nemp-pow-nemp*: $t^{@}m \neq \varepsilon \Longrightarrow t \neq \varepsilon$
by *auto*

lemma *emp-pow-emp*: $t = \varepsilon \Longrightarrow t^{@}m = \varepsilon$
by *auto*

lemma *list-pow-emp* [*intro*]: $t = \varepsilon \vee m = 0 \Longrightarrow t^{@}m = \varepsilon$
using *nemp-pow-nemp* **by** *auto*

lemma *list-pow-emp-iff* [*simp*]: $t^{@}m = \varepsilon \longleftrightarrow t = \varepsilon \vee m = 0$
using *list-pow-emp pow-list-Nil-iff-0*[*of t m*] **by** *blast*

lemma *hd-nemp-pow*[*intro*]: $v^{@}k \neq \varepsilon \Longrightarrow hd\ (v^{@}k) = hd\ v$
by (*simp add*: *hd-pow-list*)

lemma *hd-pref-pow*[*intro*]: **assumes** $u \leq p\ v^{@}k$ **and** $u \neq \varepsilon$
shows $hd\ u = hd\ v$
using *nemp-pref-nemp*[*OF assms*, *THEN hd-nemp-pow*, *folded pref-hd-eq*[*OF assms*]]**.**

lemma *pop-pow*: $m \leq k \Longrightarrow u^{@}m \cdot u^{@}(k-m) = u^{@}k$
using *le-add-diff-inverse pow-add* **by** *metis*

lemma *pop-pow-2*: $1 < k \Longrightarrow u \cdot u \cdot u^{@}(k-2) = u^{@}k$
using *pop-pow*[*of 2 k*] **by** *auto*

lemma *pop-pow-cancel*: $u^{@}k \cdot v = u^{@}m \cdot w \Longrightarrow m \leq k \Longrightarrow u^{@}(k-m) \cdot v = w$
using *lassoc pop-pow*[*of m k u*] *same-append-eq*[*of* $u^{@}m$ $u^{@}(k-m) \cdot v$ *w*, *unfolded lassoc*] **by** *argo*

lemma *pows-comm*: $t^{@}k \cdot t^{@}m = t^{@}m \cdot t^{@}k$
unfolding *pow-add*[*symmetric*] *add.commute*[*of k*]**..**

lemma *comm-pows-comm*: **assumes** $r \cdot u = u \cdot r$ **shows** $r^{@}m \cdot u^{@}k = u^{@}k \cdot r^{@}m$
using *comm-pow-comm*[*OF comm-pow-comm*[*OF assms*, *symmetric*], *symmetric*]**.**

lemma *pows-comp*: $x^{@}i \bowtie x^{@}j$
unfolding *prefix-comparable-def* **using** *ruler-eqE*[*OF pows-comm*] **by** *metis*

lemmas *pows-suf-comp* = *pows-comp*[*reversed*, *folded rev-pow suffix-comparable-def*]

lemmas [*reversal-rule*] = *rev-pow*[*symmetric*]

lemmas *pow-eq-if-list'* = *pow-list-eq-if*[*reversed*] **and**
pop-pow-list-one' = *pow-pos*[*reversed*] **and**
pop-pow' = *pop-pow*[*reversed*] **and**
pop-pow-cancel' = *pop-pow-cancel*[*reversed*]

lemma *emp-pow-pos-emp* [*intro*]: **assumes** $v^{@}j = \varepsilon$ $0 < j$ **shows** $v = \varepsilon$
using *pow-pos*[*OF* ‹$0 < j$›, *of v*, *unfolded* ‹$v^{@}j = \varepsilon$›] **by** *blast*

lemma *pow-nemp-pos-nemp*[*intro*]: $w = u\ ^{@}\ k \Longrightarrow\ u \neq \varepsilon \Longrightarrow 0 < k \Longrightarrow w \neq \varepsilon$
by *fast*

lemma *nemp-emp-pow*: **assumes** $u \neq \varepsilon$ **shows** $u^{@}m = \varepsilon \longleftrightarrow m = 0$
using *eq-pow-exp*[*OF assms*, *of m 0*, *unfolded pow-zero*]**.**

lemma *nemp-pow-nemp-pos-conv*: **assumes** $u \neq \varepsilon$ **shows** $u^{@}m \neq \varepsilon \longleftrightarrow 0 < m$
unfolding *nemp-emp-pow*[*OF assms*] **by** *blast*

lemma *nemp-Suc-pow-nemp*: $u \neq \varepsilon \Longrightarrow u^{@}Suc\ k \neq \varepsilon$
by *simp*

lemma *Suc-pow-eq-eq*[*elim*]: $u^{@}Suc\ k = v^{@}Suc\ k \Longrightarrow u = v$
using *pow-eq-eq* **by** *blast*

lemmas [*reversal-rule*] = *map-pow-list*[*symmetric*]

named-theorems *concat-simps*
lemmas [*concat-simps*] = *concat-morph concat-sing' concat-pow-list concat.simps emp-simps*

lemma *pow-sing-nemp* [*elim*]: $w = [a]\ ^{@}\ k \Longrightarrow 0 < k \Longrightarrow w \neq \varepsilon$
by *force*

lemma *nemp-pow-emp*[*intro*]: $u \neq \varepsilon \Longrightarrow u^{@}n = \varepsilon \Longrightarrow n = 0$
by *blast*

lemma *long-pow*: $r \neq \varepsilon \Longrightarrow m \leq |r^{@}m|$
unfolding *pow-len* **using** *nemp-le-len*[*of r*] **by** *simp*

lemma *long-pow-exp'*: $r \neq \varepsilon \Longrightarrow m < |r^{@}(Suc\ m)|$
using *Suc-le-lessD long-pow* **by** *blast*

lemma *long-pow-expE*: **assumes** $r \neq \varepsilon$ **obtains** n **where** $m \leq |r^{@}Suc\ n|$
using *long-pow-exp'*[*OF* ‹$r \neq \varepsilon$›] *nat-less-le* **by** *blast*

lemma *pref-pow-ext*: $x \leq p\ r^{@}k \Longrightarrow x \leq p\ r^{@}Suc\ k$
using *pref-trans*[*OF - prefI*[*OF pow-Suc2*[*symmetric*]]].

lemma *pref-pow-ext'*: $u \leq p\ r^{@}k \Longrightarrow u \leq p\ r \cdot r^{@}k$
using *pref-pow-ext*[*unfolded pow-Suc*].

lemma *pref-pow-root-ext*: $x \leq p\ r^{@}k \Longrightarrow r \cdot x \leq p\ r^{@}Suc\ k$
by *simp*

lemma *pref-pow-root*: $u \leq p\ r^{@}k \Longrightarrow u \leq p\ r \cdot u$
using *pref-pow-ext'*[*THEN pref-prod-pref*].

lemma *le-exps-pref* [*intro*]: $k \leq l \Longrightarrow r^{@}k \leq p\ r^{@}l$
using *leI pop-pow*[*of k l r*] **by** *blast*

lemmas *less-exps-pref* = *le-exps-pref*[*OF less-imp-le*]

lemma *pref-exp-le*: **assumes** $u \neq \varepsilon$ $u^{@}m \leq p\ u^{@}n$ **shows** $m \leq n$
using *mult-right-le-imp-le*[*OF - nemp-len*[*OF* ‹$u \neq \varepsilon$›]]
prefix-length-le[*OF* ‹$u^{@}m \leq p\ u^{@}n$›, *unfolded pow-len*]
by *blast*

lemma *sing-exp-pref-iff*: **assumes** $a \neq b$
shows $[a]^{@}i \leq p\ [a]^{@}k \cdot [b] \cdot w \longleftrightarrow i \leq k$
proof
assume $i \leq k$
thus $[a]^{@}i \leq p\ [a]^{@}k \cdot [b] \cdot w$
using *pref-ext*[*OF le-exps-pref*[*OF* ‹$i \leq k$›]] **by** *blast*
next
have $\neg\ [a]^{@}i \leq p\ [a]^{@}k \cdot [b] \cdot w$ **if** $\neg\ i \leq k$
proof (*rule notI*)
assume $[a]^{@}i \leq p\ [a]^{@}k \cdot [b] \cdot w$
hence $k < i$ **and** $0 < i - k$ **using** ‹$\neg\ i \leq k$› **by** *force+*
from *pop-pow*[*OF less-imp-le*, *OF this*(*1*)]
have $[a]^{@}k \cdot [a]^{@}(i - k) = [a]^{@}i$.
from ‹$[a]^{@}i \leq p\ [a]^{@}k \cdot [b] \cdot w$›[*folded this*, *unfolded pref-cancel-conv*
pow-pos[*OF* ‹$0 < i - k$›]]
show *False*
using ‹$a \neq b$› **by** *simp*

qed
thus $[a]^{@}\ i \leq p\ [a]^{@}\ k \cdot [b] \cdot w \Longrightarrow i \leq k$
by *blast*
qed

lemmas
suf-pow-ext = *pref-pow-ext*[*reversed*] **and**
suf-pow-ext'= *pref-pow-ext'*[*reversed*] **and**
suf-pow-root-ext = *pref-pow-root-ext*[*reversed*] **and**
suf-prod-root = *pref-pow-root*[*reversed*] **and**
suf-exps-pow = *le-exps-pref*[*reversed*] **and**
suf-exp-le = *pref-exp-le*[*reversed*] **and**
sing-exp-suf-iff = *sing-exp-pref-iff*[*reversed*]

lemma *comm-common-pow-list-iff*: **assumes** $r \cdot u = u \cdot r$ **shows** $r^{@}|u| = u^{@}|r|$
using *eqd-eq*[*OF comm-pows-comm*[*OF* ‹$r \cdot u = u \cdot r$›], *of* $|u|$ $|r|$]
unfolding *pow-len* **by** *fastforce*

lemma *concat-morph-power*: $xs \in lists\ B \Longrightarrow xs = ts^{@}k \Longrightarrow concat\ ts^{@}k = concat\ xs$
by (*induct k arbitrary*: *xs ts*) *simp-all*

lemma *per-exp-pref*: $u \leq p\ r \cdot u \Longrightarrow u \leq p\ r^{@}k \cdot u$
proof(*induct k*)
case (*Suc k*) **show** *?case*
unfolding *pow-Suc rassoc*
using *Suc.hyps Suc.prems pref-prolong* **by** *blast*
qed *simp*

lemmas
per-exp-suf = *per-exp-pref*[*reversed*]

lemma *hd-sing-pow*: $k \neq 0 \Longrightarrow hd\ ([a]^{@}k) = a$
by (*induction k*) *simp+*

lemma *sing-pref-comp-mismatch*:
assumes $b \neq a$ **and** $c \neq a$ **and** $[a]^{@}k \cdot [b] \bowtie [a]^{@}l \cdot [c]$
shows $k = l \wedge b = c$
proof
show $k = l$
using *assms*
proof (*induction k l rule*: *diff-induct*)
show $b \neq a \Longrightarrow c \neq a \Longrightarrow [a]^{@}\ x \cdot [b] \bowtie [a]^{@}\ 0 \cdot [c] \Longrightarrow x = 0$ **for** x
by (*rule ccontr*, *elim not0-SucE*) *fastforce*
qed (*simp add:prefix-comparable-def*)+
show $b = c$

using *assms*(*3*) **unfolding** ‹$k = l$› **by** *auto*
qed

lemma *sing-pref-comp-lcp*: **assumes** $r \neq s$ **and** $a \neq b$ **and** $a \neq c$
shows $[a]^{@}r \cdot [b] \cdot u \wedge_p [a]^{@}s \cdot [c] \cdot v = [a]^{@}(min\ r\ s)$
proof−
have $r \neq s \longrightarrow [a]^{@}r \cdot [b] \cdot u \wedge_p [a]^{@}s \cdot [c] \cdot v = [a]^{@}(min\ r\ s)$
proof (*rule diff-induct*[*of* $\lambda\ r\ s.\ r \neq s \longrightarrow [a]^{@}r \cdot [b] \cdot u \wedge_p [a]^{@}s \cdot [c] \cdot v = [a]^{@}(min\ r\ s)$])
have $[a]^{@}\ Suc\ (x - 1) \cdot [b] \cdot u \wedge_p [c] \cdot v = [a]^{@}\ min\ x\ 0$ **if** $x \neq 0$ **for** x
unfolding *pow-Suc min-0R exp-simps rassoc* **by** (*simp add*: ‹$a \neq c$›)
thus $x \neq 0 \longrightarrow [a]^{@}\ x \cdot [b] \cdot u \wedge_p [a]^{@}\ 0 \cdot [c] \cdot v = [a]^{@}\ min\ x\ 0$ **for** x **by** *force*
show $0 \neq Suc\ y \longrightarrow [a]^{@}\ 0 \cdot [b] \cdot u \wedge_p [a]^{@}\ Suc\ y \cdot [c] \cdot v = [a]^{@}\ min\ 0\ (Suc\ y)$ **for** y
unfolding *pow-Suc min-0L exp-simps rassoc* **using** ‹$a \neq b$› **by** *auto*
show $x \neq y \longrightarrow [a]^{@}\ x \cdot [b] \cdot u \wedge_p [a]^{@}\ y \cdot [c] \cdot v = [a]^{@}\ min\ x\ y \Longrightarrow$
$Suc\ x \neq Suc\ y \longrightarrow [a]^{@}\ Suc\ x \cdot [b] \cdot u \wedge_p [a]^{@}\ Suc\ y \cdot [c] \cdot v = [a]^{@}\ min\ (Suc\ x)\ (Suc\ y)$ **for** $x\ y$
unfolding *rassoc min-Suc-Suc* **by** *simp*
qed
with *assms*
show *?thesis* **by** *blast*
qed

lemmas *sing-suf-comp-mismatch* = *sing-pref-comp-mismatch*[*reversed*]

lemma *exp-pref-cancel*: **assumes** $t^{@}m \cdot y = t^{@}k$ **shows** $y = t^{@}(k - m)$
using *lqI*[*of* $t^{@}m$ $t^{@}(k-m)$ $t^{@}k$] **unfolding** *lqI*[*OF* ‹$t^{@}m \cdot y = t^{@}k$›]
using *nat-le-linear*[*of m k*] *pop-pow*[*of m k t*] *diff-is-0-eq*[*of k m*] *append.right-neutral*[*of* $t^{@}k$] *pow-zero*[*of t*]
pref-antisym[*of* $t^{@}m$ $t^{@}k$, *OF prefI*[*OF* ‹$t^{@}m \cdot y = t^{@}k$›] *le-exps-pref*[*of k m t*]]
by *presburger*

lemmas *exp-suf-cancel* = *exp-pref-cancel*[*reversed*]

lemma *index-pow-mod*: $i < |r^{@}k| \Longrightarrow (r^{@}k)!i = r!(i\ mod\ |r|)$
proof(*induction k*)
have *aux*: $|r^{@}(Suc\ l)| = |r^{@}l| + |r|$ **for** l
by *simp*
have *aux1*: $|(r^{@}l)| \leq i \Longrightarrow i < |r^{@}l| + |r| \Longrightarrow i\ mod\ |r| = i - |r^{@}l|$ **for** l
unfolding *pow-len* **using** *less-diff-conv2*[*of* $l * |r|$ i $|r|$, *unfolded add.commute*[*of* $|r|$ $l * |r|$]]
get-mod[*of* $i - l * |r|$ $|r|$ l] *le-add-diff-inverse*[*of* $l*|r|$ i] **by** *argo*
case (*Suc k*)
show *?case*
unfolding *aux sym*[*OF pow-Suc2*[*symmetric*]] *nth-append le-mod-geq*
using *aux1*[*OF - Suc.prems*[*unfolded aux*]]
Suc.IH pow-Suc2[*symmetric*] *Suc.prems*[*unfolded aux*] *leI*[*of i* $|r^{@}k|$] **by**

presburger
qed *auto*

lemma *sing-pow-len* [*simp*]: $|[r]^{@}l| = l$
by (*induct l*) *auto*

lemma *take-sing-pow*: $k \leq l \Longrightarrow take\ k\ ([r]^{@}l) = [r]^{@}k$
proof (*induct k*)
case (*Suc k*)
have $k < |[r]^{@}l|$ **using** *Suc-le-lessD*[*OF* ‹*Suc k* ≤ *l*›] **unfolding** *sing-pow-len*.
from *take-Suc-conv-app-nth*[*OF this*]
show *?case*
unfolding *Suc.hyps*[*OF Suc-leD*[*OF* ‹*Suc k* ≤ *l*›]] *pow-Suc2*
unfolding *nth-pow-list-single*[*OF Suc-le-lessD*[*OF* ‹*Suc k* ≤ *l*›]].
qed *simp*

lemma *concat-take-sing*: **assumes** $k \leq l$
shows $concat\ (take\ k\ ([r]^{@}l)) = r^{@}k$
unfolding *take-sing-pow*[*OF* ‹*k* ≤ *l*›] **using** *concat-pow-list-single*.

lemma *sing-set-word*: $set\ w \subseteq \{a\} \Longrightarrow w = [a]^{@}|w|$
by (*induction w*, *auto*)

lemma *sing-set-concat*: $set\ w \subseteq \{a\} \Longrightarrow concat\ w = a^{@}|w|$
using *sing-set-word*[*of w a*] *concat-pow-list-single*[*of* |*w*| *a*] **by** *presburger*

lemma *sing-set-word-hd*: $set\ w \subseteq \{a\} \Longrightarrow w = [hd\ w]^{@}|w|$
by (*cases* $w = \varepsilon$, *force*) (*use sing-set-word hd-sing-pow* [*OF nemp-len-not0*, *of w a*] **in** *fastforce*)

lemma *sing-set-wordE*[*elim*]: **assumes** $set\ w \subseteq \{a\}$ **obtains** k **where** $w = [a]^{@}k$
using *sing-set-word assms* **by** *metis*

lemma *sing-set-wordE'*[*elim*]: **assumes** $set\ w = \{a\}$ **obtains** k **where** $w = [a]^{@}k$ **and** $0 < k$
proof−
obtain l **where** $w = [a]^{@}l$
using *sing-set-wordE*[*of w a*] *assms* **by** *force*
have $w \neq \varepsilon$
using *assms* **by** *force*
hence $0 < l$
unfolding ‹$w = [a]^{@}l$› **by** *blast*
show *thesis*
using *that*[*OF* ‹$w = [a]^{@}l$› ‹$0 < l$›].
qed

lemma *conjug-pow*: $x \cdot z = z \cdot y \Longrightarrow x^{@}k \cdot z = z \cdot y^{@}k$
by (*induct k*) *fastforce*+

lemma *lq-conjug-pow*: **assumes** $p \leq_p x \cdot p$ **shows** $p^{-1>}(x^{@}k \cdot p) = (p^{-1>}(x \cdot p))^{@}k$
using *lqI*[*OF sym*[*OF conjug-pow*[*of x p* $p^{-1>}(x \cdot p)$, *OF sym*[*OF lq-pref*[*OF* ‹$p \leq_p x \cdot p$›]], *of k*]]]**.**

lemmas *rq-conjug-pow* = *lq-conjug-pow*[*reversed*]

lemma *pow-pref-root-one*: **assumes** $0 < k$ **and** $r \neq \varepsilon$ **and** $r^{@}k \leq_p r$
shows $k = 1$
unfolding *eq-pow-exp*[*OF* ‹$r \neq \varepsilon$›, *of k 1*, *symmetric*] *pow-list-1*
using ‹$r^{@}k \leq_p r$› *triv-pref*[*of r* $r^{@}(k-1)$, *folded pow-pos*[*OF* ‹$0 < k$›]] **by** *auto*

lemma *comp-pows-pref*: **assumes** $v \neq \varepsilon$ **and** $(u \cdot v)^{@}k \cdot u \leq_p (u \cdot v)^{@}m$ **shows** $k \leq m$
using *pref-exp-le*[*OF* - *pref-extD*[*OF assms*(*2*)]] *assms*(*1*) **by** *blast*

lemma *comp-pows-pref'*: **assumes** $v \neq \varepsilon$ **and** $(u \cdot v)^{@}k \leq_p (u \cdot v)^{@}m \cdot u$ **shows** $k \leq m$
proof(*rule ccontr*)
assume $\neg\ k \leq m$
hence *Suc m* $\leq k$ **by** *simp*
from *le-exps-pref*[*OF this*, *unfolded pow-Suc2*]
have $(u \cdot v)^{@}m \cdot (u \cdot v) \leq_p (u \cdot v)^{@}k$**.**
from *pref-trans*[*OF this assms*(*2*)] ‹$v \neq \varepsilon$›
show *False* **by** *auto*
qed

lemma *comp-pows-not-pref*: $\neg\ (u \cdot v)^{@}k \cdot u \leq_p (u \cdot v)^{@}m \Longrightarrow m \leq k$
by (*induction k m rule*: *diff-induct*) *auto*

lemma *comp-pows-spref*: $u^{@}k <_p u^{@}m \Longrightarrow k < m$
by (*induction k m rule*: *diff-induct*) *auto*

lemma *comp-pows-spref-ext*: $(u \cdot v)^{@}k \cdot u <_p (u \cdot v)^{@}m \Longrightarrow k < m$
by (*induction k m rule*: *diff-induct*) *auto*

lemma *comp-pows-pref-zero*:$(u \cdot v)^{@}k <_p u \Longrightarrow k = 0$
by (*induct k*) *auto*

lemma *comp-pows-spref'*: $(u \cdot v)^{@}k <_p (u \cdot v)^{@}m \cdot u \Longrightarrow k < Suc\ m$
by (*induction k m rule*: *diff-induct*, *simp-all add*: *comp-pows-pref-zero*)

lemmas *comp-pows-suf* = *comp-pows-pref*[*reversed*] **and**
comp-pows-suf' = *comp-pows-pref'*[*reversed*] **and**
comp-pows-not-suf = *comp-pows-not-pref*[*reversed*] **and**
comp-pows-ssuf = *comp-pows-spref*[*reversed*] **and**
comp-pows-ssuf-ext = *comp-pows-spref-ext*[*reversed*] **and**

comp-pows-suf-zero = *comp-pows-pref-zero*[*reversed*] **and**
comp-pows-ssuf' = *comp-pows-spref'*[*reversed*]

2.12.1 Comparison

named-theorems *shifts*
lemma *shift-pow*[*shifts*]: $(u{\cdot}v)^{@}k{\cdot}u = u{\cdot}(v{\cdot}u)^{@}k$
using *conjug-pow*[*OF rassoc*]**.**
lemma[*shifts*]: $(u \cdot v)^{@}k \cdot u \cdot z = u \cdot (v \cdot u)^{@}k \cdot z$
by (*simp add*: *shift-pow*)
lemma[*shifts*]: $u^{@}k \cdot u \cdot z = u \cdot u^{@}k \cdot z$
by (*simp add*: *conjug-pow*)
lemma[*shifts*]: $r^{@}k \leq_p r \cdot r^{@}k$
by (*simp add*: *pow-comm*[*symmetric*])
lemma [*shifts*]: $r^{@}k \leq_p r \cdot r^{@}k \cdot z$
unfolding *lassoc pow-comm*[*symmetric*] **unfolding** *rassoc* **by** *blast*
lemma [*shifts*]: $(r \cdot q)^{@}k \leq_p r \cdot q \cdot \ (r \cdot q)^{@}k \cdot z$
unfolding *lassoc pow-comm*[*symmetric*] **unfolding** *rassoc* **by** *simp*
lemma [*shifts*]: $(r \cdot q)^{@}k \leq_p r \cdot q \cdot \ (r \cdot q)^{@}k$
unfolding *lassoc pow-comm*[*symmetric*] **unfolding** *rassoc* **by** *simp*
lemma[*shifts*]: $r^{@}k \cdot u \leq_p r \cdot r^{@}k \cdot v \longleftrightarrow u \leq_p r \cdot v$
unfolding *lassoc pow-comm*[*symmetric*] **unfolding** *rassoc pref-cancel-conv***..**
lemma[*shifts*]: $u \cdot u^{@}k \cdot z = u^{@}k \cdot w \longleftrightarrow u \cdot z = w$
unfolding *lassoc pow-comm*[*symmetric*] **unfolding** *rassoc cancel***..**
lemma[*shifts*]: $(r \cdot q)^{@}k \cdot u \leq_p r \cdot q \ \cdot (r \cdot q)^{@}k \cdot v \longleftrightarrow u \leq_p r \cdot q \cdot v$
unfolding *lassoc pow-comm*[*symmetric*] **unfolding** *rassoc pref-cancel-conv***..**
lemma[*shifts*]: $(r \cdot q)^{@}k \cdot u = r \cdot q \ \cdot (r \cdot q)^{@}k \cdot v \longleftrightarrow u = r \cdot q \cdot v$
unfolding *lassoc pow-comm*[*symmetric*] **unfolding** *rassoc cancel***..**
lemma[*shifts*]: $r \cdot q \ \cdot (r \cdot q)^{@}k \cdot v = (r \cdot q)^{@}k \cdot u \longleftrightarrow r \cdot q \cdot v = u$
unfolding *lassoc pow-comm*[*symmetric*] **unfolding** *rassoc cancel***..**
lemma *shifts-spec* [*shifts*]: $(u^{@}k \cdot v)^{@}l \cdot u \cdot u^{@}k \cdot z = u^{@}k \cdot (v \cdot u^{@}k)^{@}l \cdot u \cdot z$
unfolding *lassoc cancel-right* **unfolding** *rassoc pow-comm*[*symmetric*]
unfolding *lassoc cancel-right shift-pow***..**
lemmas [*shifts*] = *shifts-spec*[*of* - - $r \cdot q$, *unfolded rassoc*] **for** r q
lemmas [*shifts*] = *shifts-spec*[*of* - - $r \cdot q$ - ε , *unfolded rassoc emp-simps*] **for** r q
lemmas [*shifts*] = *shifts-spec*[*of* - - $r \cdot q$ $r \cdot q$, *unfolded rassoc*] **for** r q
lemmas [*shifts*] = *shifts-spec*[*of* - - $r \cdot q$ $r \cdot q$ ε , *unfolded rassoc emp-simps*] **for** r q
lemma[*shifts*]: $(u \cdot (v \cdot u)^{@}k)^{@}j \cdot (u \cdot v)^{@}k = (u \cdot v)^{@}k \cdot (u \cdot (u \cdot v)^{@}k)^{@}j$
by (*metis shift-pow*)
lemma[*shifts*]: $(u \cdot (v \cdot u)^{@}k \cdot z)^{@}j \cdot (u \cdot v)^{@}k = (u \cdot v)^{@}k \cdot (u \cdot z \cdot (u \cdot v)^{@}k)^{@}j$
by (*simp add*: *conjug-pow*)
lemmas[*shifts*] = *pow-comm cancel rassoc pow-Suc pref-cancel-conv suf-cancel-conv pow-add cancel-right numeral-nat pow-zero emp-simps*
lemmas[*shifts*] = *less-eq-Suc-le*
lemmas[*shifts*] = *neq0-conv*
lemma *shifts-hd-hd* [*shifts*]: $a\#b\#v = [a] \cdot b\#v$
using *hd-word***.**
lemmas [*shifts*] = *shifts-hd-hd*[*of* - - ε]

lemma[*shifts*]: $n \leq k \Longrightarrow x^{@}k = x^{@}(n + (k - n))$
by *simp*
lemma[*shifts*]: $n < k \Longrightarrow x^{@}k = x^{@}(n + (k - n))$
by *simp*
lemmas[*shifts*] = *cancel cancel-right pref-cancel-conv suf-cancel-conv triv-pref*
lemmas[*shifts*] = *pow-diff*
lemmas[*shifts*] = *pows-comm*
lemma[*shifts*]: $m < l \Longrightarrow x^{@}m \cdot w \leq p\ x^{@}l \cdot w' \longleftrightarrow w \leq p\ x^{@}(l-m) \cdot w'$
by (*metis append-assoc nless-le pop-pow same-prefix-prefix*)
lemma[*shifts*]: $x \cdot x^{@}m \cdot w \leq p\ x^{@}m \cdot w' \longleftrightarrow x \cdot w \leq p\ \ w'$
unfolding *lassoc pow-comm*[*symmetric*] **by** *force*

lemmas *shifts-rev* = *shifts*[*reversed*]

lemmas *shift-simps* = *shifts shifts*[*reversed*]

method *comparison* = ((*simp only*: *shifts*; *fail*) | (*simp only*: *shifts-rev*; *fail*))

lemma *take-root*: $0 < k \Longrightarrow$ *take* $|ws|$ $(ws\ ^{@}\ k) = ws$
using *pow-pos*[*of k ws*] **by** *force*

2.13 Rotation

lemma *rotate-root-self*: *rotate* $|r|$ $(r^{@}k) = r^{@}k$
proof (*cases* $r = \varepsilon$)
assume $r \neq \varepsilon$
show *?thesis*
proof (*cases k*)
fix *pred*
assume *k*: $k =$ *Suc pred*
show *?thesis*
unfolding *k pow-Suc rotate-append pow-comm*..
qed *simp*
qed *simp*

lemma *rotate-pow-self*: *rotate* $(l*|u|)$ $(u^{@}k) = u^{@}k$
proof(*induct l*)
case (*Suc l*)
show *?case*
unfolding *mult-Suc rotate-rotate*[*symmetric*] *Suc.hyps*
using *rotate-root-self*.
qed *simp*

lemma *rotate-pow-mod*: *rotate n* $(u^{@}k)$ = *rotate* $(n \bmod |u|)$ $(u^{@}k)$
using *rotate-rotate*[*of n mod* $|u|$ *n div* $|u| * |u|$ $u^{@}k$, *symmetric*]
unfolding *rotate-pow-self*[*of n div* $|u|$ *u k*] *div-mult-mod-eq*[*of n* $|u|$, *unfolded add.commute*[*of n div* $|u| * |u|$ *n mod* $|u|$]].

lemma *rotate-conj-pow*: *rotate* $|u|$ $((u \cdot v)^{@}k) = (v \cdot u)^{@}k$

by (*induct k*, *simp*, *simp add*: *rotate-append shift-pow*)

lemmas *rotate-pow-comm-two* = *rotate-pow-list-swap*[*of* - *2*, *unfolded pow-list-2*]

lemma *rotate-back*: *rotate* (|*u*| − *n mod* |*u*|) (*rotate n u*) = *u*
proof (*cases u* = *ε*)
  assume *u* ≠ *ε*
  show *?thesis*
  unfolding *rotate-conv-mod*[*of n u*] *rotate-rotate*[*of* |*u*| − *n mod* |*u*| *n mod* |*u*| *u*]
  le-add-diff-inverse2[*OF mod-le-divisor*, *OF nemp-len*[*OF* ‹*u* ≠ *ε*›]]
  by *simp*
qed *simp*

lemma *rotate-backE*: **obtains** *m* **where** *rotate m* (*rotate n u*) = *u*
  using *rotate-back* **by** *blast*

lemma *rotate-back′*: **assumes** *rotate m w* = *rotate n w*
  shows *rotate* (*m*−*n*) *w* = *w*
proof (*cases*)
  assume *n* ≤ *m*
  from *rotate-backE* **obtain** *k* **where** *rotate k* (*rotate n w*) = *w*.
  hence *nk*: *rotate n* (*rotate k w*) = *w*
    unfolding *rotate-rotate add.commute*[*of* - *k*].
  have *mn*: *rotate m* (*rotate k w*) = (*rotate n* (*rotate k w*))
   unfolding *rotate-rotate add.commute*[*of* - *k*] **unfolding** *rotate-rotate*[*symmetric*]
assms..
  have *rotate* (*m* − *n*) (*rotate n* (*rotate k w*)) = *rotate m* (*rotate k w*)
    unfolding *rotate-rotate* **using** ‹*n* ≤ *m*› **by** *simp*
  from *this*[*unfolded mn nk*]
  show *?thesis*.
qed *simp*

lemma *rotate-class-rotate′*: (∃ *n*. *rotate n w* = *u*) ⟷ (∃ *n*. *rotate n* (*rotate l w*)
= *u*)
proof
  obtain *m* **where** *rot-m*: *rotate m* (*rotate l w*) = *w* **using** *rotate-backE*.
  assume ∃ *n*. *rotate n w* = *u*
  then obtain *n* **where** *rot-n*: *rotate n w* = *u* **by** *blast*
  show ∃ *n*. *rotate n* (*rotate l w*) = *u*
    using *exI*[*of* λ *x*. *rotate x* (*rotate l w*) = *u n*+*m*, *OF*
      rotate-rotate[*symmetric*, *of n m rotate l w*, *unfolded rot-m rot-n*]].
next
  show ∃ *n*. *rotate n* (*rotate l w*) = *u* ⟹ ∃ *n*. *rotate n w* = *u*
    using *rotate-rotate*[*symmetric*] **by** *blast*
qed

lemma *rotate-class-rotate*: {*u* . ∃ *n*. *rotate n w* = *u*} = {*u* . ∃ *n*. *rotate n* (*rotate l*

$w) = u\}$
using *rotate-class-rotate'* **by** *blast*

lemma *rotate-comp-eq*:$w \bowtie$ *rotate n w* $\Longrightarrow$ *rotate n w* $= w$
using *pref-same-len*[*OF - length-rotate*[*of n w*]] *pref-same-len*[*OF - length-rotate*[*of n w*, *symmetric*], *symmetric*]
by *blast*

corollary *mismatch-iff-lexord*: **assumes** *rotate n w* $\neq w$ **and** *irrefl r*
shows *mismatch-pair w* (*rotate n w*) $\in r \longleftrightarrow$ (*w*,*rotate n w*) $\in$ *lexord r*
proof−
have $\neg$ *w* $\bowtie$ *rotate n w*
using *rotate-comp-eq* ‹*rotate n w* $\neq$ *w*›
unfolding *prefix-comparable-def* **by** *blast*
from *lexord-mismatch*[*OF this* ‹*irrefl r*›]
show *?thesis*.
qed

2.14 Lists of words and their concatenation

The helpful lemmas of this section deal with concatenation of a list of words *concat*. The main objective is to cover elementary facts needed to study factorizations of words.

lemma *concat-take-is-prefix*: *concat*(*take n ws*) $\leq p$ *concat ws*
using *concat-morph*[*of take n ws drop n ws*,*symmetric*, *unfolded append-take-drop-id*[*of n ws*], *THEN prefI*].

lemma *concat-take-Suc*: **assumes** *j* $<$ |*ws*| **shows** *concat*(*take j ws*) $\cdot$ *ws*!*j* $=$ *concat*(*take* (*Suc j*) *ws*)
unfolding *take-Suc-conv-app-nth*[*OF* ‹*j* $<$ |*ws*|›]
using *sym*[*OF concat-append*[*of* (*take j ws*) [*ws* ! *j*],
unfolded concat.simps(*2*)[*of ws*!*j* ε, *unfolded concat.simps*(*1*) *append-Nil2*]]].

lemma *pref-mod-list'*: **assumes** *u* $<p$ *concat ws*
obtains *j r* **where** *j* $<$ |*ws*| **and** *r* $<p$ *ws*!*j* **and** *concat* (*take j ws*) $\cdot$ *r* $=$ *u*
proof−
have |*ws*| $\neq$ *0*
using *assms* **by** *auto*
then obtain *l* **where** *Suc l* $=$ |*ws*|
using *Suc-pred* **by** *blast*
let *?P* $= \lambda$ *j*. *u* $<p$ *concat*(*take* (*Suc j*) *ws*)
have *?P l*
using *assms* ‹*Suc l* $=$ |*ws*|› **by** *auto*
define *j* **where** *j* $=$ (*LEAST j*. *?P j*) — smallest j such that concat (take (Suc j) ws) <p u
have *u* $<p$ *concat*(*take* (*Suc j*) *ws*)
using *LeastI*[*of ?P*, *OF* ‹*?P l*›] **unfolding** *sym*[*OF j-def*].
have *j* $<$ |*ws*|

```
    using Least-le[of ?P, OF ‹?P l›] ‹Suc l = |ws|› unfolding sym[OF j-def]
    by auto
  have concat(take j ws) ≤p u
    using Least-le[of ?P (j − Suc 0), unfolded sym[OF j-def]]
      ruler[OF concat-take-is-prefix sprefD1[OF assms], of j]
    by (cases j = 0, simp) force
  from prefixE[OF this]
  obtain r where u = concat(take j ws) · r.
  from ‹u <p concat (take (Suc j) ws)›[unfolded this]
  have r <p ws!j
    unfolding concat-take-Suc[OF ‹j < |ws|›, symmetric]  spref-cancel-conv.
  show thesis
    using that[OF ‹j < |ws|› ‹r <p ws!j› ‹u = concat(take j ws) · r›[symmetric]].
qed

lemma pref-mod-list: assumes u ≤p concat ws u ≠ ε
  obtains ps p s where ps ≤p ws and p ≠ ε and p · s = last ps
   concat ps = u · s and ⋀ y. y ≤p ws ⟹ u ≤p concat y ⟹ ps ≤p y
proof−
  from min-pref[OF self-pref, of λ x. u ≤p concat x, OF ‹u ≤p concat ws›]
  obtain v where v ≤p ws u ≤p concat v and
  min: (⋀y. y ≤p ws ⟹ u ≤p concat y ⟹ v ≤p y)
    by blast
  have v ≠ ε
    using ‹u ≤p concat v› ‹u ≠ ε› by auto
  hence ‹v = butlast v · [last v]› concat v = concat (butlast v) · last v
    using concat-butlast-last[OF ‹v ≠ ε›] by simp-all

  from min[of butlast v, OF pref-trans, OF prefixeq-butlast ‹v ≤p ws›]
  have ¬ u ≤p concat (butlast v)
    using ‹v = butlast v · [last v]› pref-antisym by force

  from prefE[OF ‹u ≤p concat v›]
   obtain s where concat v = u · s
     by blast
   have s <s last v
     using ‹¬ u ≤p concat (butlast v)›
     ruler-eq[reversed, OF ‹concat v = u · s›[unfolded ‹concat v = concat (butlast
v) · last v›]]
     unfolding eqd-pref-suf-iff[OF ‹concat v = concat (butlast v) · last v›[unfolded
‹concat v = u · s›]]
     by blast
   from ssuf-exE[OF this]
   obtain p where p · s = last v p ≠ ε
     by blast
   show thesis
     using that[OF ‹v ≤p ws› ‹p ≠ ε› ‹p · s = last v› ‹concat v = u · s›] min by
simp
 qed
```

lemma *pref-mod-pow*: **assumes** $u \leq p\ w^{@} l$ **and** $w \neq \varepsilon$
obtains $k\ z$ **where** $k \leq l$ **and** $z <p\ w$ **and** $w^{@} k \cdot z = u$
proof (*cases* $u = w^{@} l$)
assume $u \neq w^{@} l$
from *sprefI*[*OF* ‹$u \leq p\ w^{@} l$› *this*]
have $u <p\ w\ ^{@}\ l$.
have $w^{@} l = concat\ ([w]^{@} l)$
by *simp*
from *pref-mod-list'*[*of* u $[w]^{@} l$, *unfolded sing-pow-len concat-pow-list-single*, *OF* ‹$u <p\ w^{@} l$›]
obtain $j\ r$ **where** $j < l\ r <p\ ([w]\ ^{@}\ l)\ !\ j\ concat\ (take\ j\ ([w]\ ^{@}\ l)) \cdot r = u$.
hence $j \leq l$ **and** $r <p\ w$ **and** $w^{@} j \cdot r = u$
unfolding *nth-pow-list-single*[*OF* ‹$j < l$›] *concat-take-sing*[*OF less-imp-le*[*OF* ‹$j < l$›]] **by** *auto*
from *that*[*OF this*]
show *thesis*.
qed (*use emp-spref assms* **in** *blast*)

lemma *pref-mod-pow'*: **assumes** $u <p\ w^{@} l$
obtains $k\ z$ **where** $k < l$ **and** $z <p\ w$ **and** $w^{@} k \cdot z = u$
proof−
have $w \neq \varepsilon$
by (*rule nemp-pow-nemp*[*of* l]) (*use assms* **in** *force*)
from *pref-mod-pow*[*OF sprefD1*[*OF assms*] *this*]
obtain $k\ z$ **where** $k \leq l\ z <p\ w\ w\ ^{@}\ k \cdot z = u$.
note *spref-extD*[*OF* ‹$u <p\ w^{@} l$›[*folded* ‹$w\ ^{@}\ k \cdot z = u$›]]
have $k < l$
using *comp-pows-spref*[*OF* ‹$w\ ^{@}\ k <p\ w\ ^{@}\ l$›].
from *that*[*OF this* ‹$z <p\ w$› ‹$w\ ^{@}\ k \cdot z = u$›]
show *thesis*.
qed

lemma *split-pow*: **assumes** $u \cdot v = w^{@} k\ 0 < k\ v \neq \varepsilon$
obtains $p\ s\ i\ j$ **where** $w = p \cdot s\ s \neq \varepsilon\ u = (p \cdot s)^{@} i \cdot p\ v = (s \cdot p)^{@} j \cdot s\ k = i + j + 1$
proof−
have $u <p\ w^{@} k$
using *assms*(*1*,*3*) **by** *blast*
from *pref-mod-pow'*[*OF this*]
obtain $ku\ p$ **where** $ku < k\ p <p\ w\ w\ ^{@}\ ku \cdot p = u$.
from *spref-exE*[*OF this*(*2*)]
obtain s **where** $p \cdot s = w\ s \neq \varepsilon$.
obtain kv **where** $k = Suc(ku + kv)$
using *less-imp-Suc-add*[*OF* ‹$ku < k$›] **by** *blast*
from ‹$u \cdot v = w^{@} k$›[*folded this*[*symmetric*] ‹$p \cdot s = w$› ‹$w\ ^{@}\ ku \cdot p = u$›, *unfolded rassoc pow-Suc2*]
have $v = s \cdot w^{@} kv$
unfolding *shifts* **unfolding** *lassoc shift-pow*[*symmetric*] **unfolding** *rassoc cancel*

‹$p \cdot s = w$›.
show *thesis*
using *that*[*OF* ‹$p \cdot s = w$›[*symmetric*] ‹$s \neq \varepsilon$› ‹$w \ @ \ ku \cdot p = u$›[*folded* ‹$p \cdot s = w$›, *symmetric*]
‹$v = s \cdot w \ @ \ kv$›[*folded* ‹$p \cdot s = w$›,*folded shift-pow*] ‹$k = Suc(ku + kv)$›[*unfolded Suc-eq-plus1*]].
qed

lemma *del-emp-concat* [*simp*]: *concat* (*filter* ($\lambda x.\ x \neq \varepsilon$) *us*) = *concat us*
by (*induct us*) *simp*+

lemma *lists-minus*: *us* ∈ *lists* (*C* − *A*) ⟹ *us* ∈ *lists C*
by *blast*

lemma *lists-minus'*: *us* ∈ *lists C* ⟹ (*filter* ($\lambda x.\ x \neq \varepsilon$) *us*) ∈ *lists* (*C* − {ε})
by (*simp add*: *in-lists-conv-set*)

lemma *pref-concat-pref*[*intro*]: *us* ≤*p ws* ⟹ *concat us* ≤*p concat ws*
by (*auto simp add*: *prefix-def*)

lemmas *suf-concat-suf* = *pref-concat-pref*[*reversed*]

lemma *concat-mono-fac*: *us* ≤*f ws* ⟹ *concat us* ≤*f concat ws*
using *concat-morph facE facI'* **by** *metis*

lemma *ruler-concat-less*: **assumes** *us* ≤*p ws* **and** *vs* ≤*p ws* **and** |*concat us*| < |*concat vs*|
shows *us* <*p vs*
using *ruler*[*OF* ‹*us* ≤*p ws*› ‹*vs* ≤*p ws*›] *pref-concat-pref*[*of vs us*, *THEN prefix-length-le*] ‹|*concat us*| < |*concat vs*|›
by *force*

lemma *concat-take-mono-strict*: **assumes** *concat* (*take i ws*) <*p concat* (*take j ws*)
shows *take i ws* <*p take j ws*
using *ruler-concat-less*[*OF - - prefix-length-less*, *OF take-is-prefix take-is-prefix assms*].

lemma *take-pp-less*: **assumes** *take k ws* <*p take n ws* **shows** *k* < *n*
using *conjunct2*[*OF sprefD*[*OF assms*]]
leI[*of k n*, *THEN*[*2*] *le-take-pref*[*of n k ws*, *THEN*[*2*] *pref-antisym*[*of take k ws take n ws*]], *OF conjunct1*[*OF sprefD*[*OF assms*]]]
by *blast*

lemma *concat-pp-less*: **assumes** *concat* (*take k ws*) <*p concat* (*take n ws*) **shows** *k* < *n*
using *le-take-pref*[*of n k ws*, *THEN pref-concat-pref*] *conjunct1*[*OF sprefD*[*OF assms*]]
conjunct2[*OF sprefD*[*OF assms*]] *pref-antisym*[*of concat*(*take k ws*) *concat*(*take n ws*)]

by *fastforce*

lemma *take-le-take*: $j \leq k \Longrightarrow$ *take* j (*take* k *xs*) = *take* j *xs*
proof (*rule disjE*[*OF le-less-linear*, *of* k |*xs*|])
assume $j \leq k$ **and** $k \leq$ |*xs*|
show *?thesis*
using *pref-share-take*[*OF take-is-prefix*, *of* j k *xs*, *unfolded take-len*[*OF* ‹$k \leq$ |*xs*|›], *OF* ‹$j \leq k$›].
qed *simp*

lemma *count-in*: *count-list ws a* $\neq 0 \Longrightarrow a \in$ *set ws*
using *count-notin*[*of a ws*] **by** *fast*

lemma *count-in-conv*: *count-list w a* $\neq 0 \longleftrightarrow$ $a \in$ *set w*
by (*induct w*, *auto*)

lemma *two-in-set-concat-len*: **assumes** $u \neq v$ **and** $\{u,v\} \subseteq$ *set ws*
shows |u| + |v| ≤ |*concat ws*|
proof–
let *?ws* = *filter* (λ x. $x \in \{u,v\}$) *ws*
have *set*: *set ?ws* = $\{u,v\}$
using ‹$\{u,v\} \subseteq$ *set ws*› **by** *auto*
have |*concat ?ws*| ≤ |*concat ws*|
unfolding *length-concat* **using** *sum-list-filter-le-nat* **by** *blast*
have *sum*: *sum* (λ x. *count-list ?ws x* ∗ |x|) $\{u,v\}$ = (*count-list ?ws u*) ∗ |u| + (*count-list ?ws v*)∗|v|
using *assms* **by** *simp*
have *count-list ?ws u* $\neq 0$ **and** *count-list ?ws v* $\neq 0$
unfolding *count-in-conv* **using** *assms* **by** *simp-all*
hence |u| + |v| ≤ |*concat ?ws*|
unfolding *length-concat sum-list-map-eq-sum-count set sum*
using *add-le-mono quotient-smaller* **by** *presburger*
thus *?thesis*
using ‹|*concat ?ws*| ≤ |*concat ws*|› **by** *linarith*
qed

2.15 Commutation

The solution of the easiest nontrivial word equation, $x \cdot y = y \cdot x$, is contained AFP theory *List-Power.List-Power* as $(u \cdot v = v \cdot u) = (\exists\, r\ k\ m.\ u = r\ ^{@}\ k \wedge v = r\ ^{@}\ m)$.

lemmas *comm* = *comm-append-pow-list-iff*

lemma *commE*[*elim*]: **assumes** $x \cdot y = y \cdot x$
obtains $t\ m\ k$ **where** $x = t^{@}k$ **and** $y = t^{@}m$ **and** $t \neq \varepsilon$
proof–
from *assms*[*unfolded comm*]
obtain $t\ k\ m$ **where** *pows*: $x = t^{@}k$ $y = t^{@}m$

by *blast*
show *thesis*
by (*cases* $t = \varepsilon$)
(*use pows that*[*of 0* [*undefined*] *0*] *that*[*OF pows*] **in** *auto*)
qed

lemma *comm-nemp-eqE*: **assumes** $u \cdot v = v \cdot u$ $u \neq \varepsilon$ $v \neq \varepsilon$
obtains k m **where** $u^{@}k = v^{@}m$ $0 < k$ $0 < m$
proof−
from *commE*[*OF* ‹$u \cdot v = v \cdot u$›]
obtain t m k **where** $u = t^{@}k$ **and** $v = t^{@}m$**.**
hence $0 < m$ $0 < k$
using ‹$u \neq \varepsilon$› ‹$v \neq \varepsilon$› **by** *blast*+
have $u^{@}m = v^{@}k$
unfolding ‹$u = t^{@}k$› ‹$v = t^{@}m$› *pow-mult*[*symmetric*]
by (*simp add*: *mult.commute*)
from *that*[*OF this* ‹$0 < m$› ‹$0 < k$›]
show *thesis***.**
qed

lemma *comm-prod*[*intro*]: **assumes** $r{\cdot}u = u{\cdot}r$ **and** $r{\cdot}v = v{\cdot}r$
shows $r{\cdot}(u{\cdot}v) = (u{\cdot}v){\cdot}r$
unfolding *lassoc* ‹$r{\cdot}u = u{\cdot}r$› **unfolding** *rassoc* ‹$r{\cdot}v = v{\cdot}r$›**..**

lemma *comm-cancel-pref*: **assumes** $r \cdot u = u \cdot r$ $r \cdot u \cdot v = u \cdot v \cdot r$
shows $r \cdot v = v \cdot r$
using ‹$r \cdot u = u \cdot r$›
unfolding *cancel-right*[*of* $r \cdot u$ v, *symmetric*] *rassoc*
unfolding ‹$r \cdot u \cdot v = u \cdot v \cdot r$›
unfolding *cancel* **by** *simp*

lemma *comm-cancel-suf*: **assumes** $r \cdot v = v \cdot r$ $r \cdot u \cdot v = u \cdot v \cdot r$
shows $r \cdot u = u \cdot r$
using *comm-cancel-pref*[*reversed*, *unfolded rassoc*, *OF assms*[*symmetric*], *symmetric*]**.**

lemma *LS-comm*:
assumes $y \,^{@}\, k \cdot x = z \,^{@}\, l$
and $z \cdot y = y \cdot z$
shows $x \cdot y = y \cdot x$
proof −
from ‹$z \cdot y = y \cdot z$›
have $(y \,^{@}\, k \cdot x) \cdot y = y \cdot (y \,^{@}\, k \cdot x)$
unfolding ‹$y \,^{@}\, k \cdot x = z \,^{@}\, l$› **by** (*fact comm-pow-comm*)
then show $x \cdot y = y \cdot x$
unfolding *lassoc pow-comm*[*symmetric*] **unfolding** *rassoc cancel***.**
qed

2.16 Periods

periodicity is probably the most studied property of words. It captures the fact that a word overlaps with itself. Another possible point of view is that the periodic word is a prefix of an (infinite) power of some nonempty word, which can be called its period word. Both these points of view are expressed by the following definition.

2.16.1 periodic root

lemma $u <p\ r \cdot u \longleftrightarrow u \leq p\ r \cdot u \wedge r \neq \varepsilon$
by *simp*

lemma *per-rootI*[*intro*]: $u \leq p\ r \cdot u \Longrightarrow r \neq \varepsilon \Longrightarrow u <p\ r \cdot u$
by *simp*

lemma *per-rootI'*[*intro*]: **assumes** $u \leq p\ r^{@}k$ **and** $r \neq \varepsilon$ **shows** $u <p\ r \cdot u$
using *per-rootI*[*OF pref-prod-pref*[*OF pref-pow-ext'*[*OF* ‹$u \leq p\ r^{@}k$›] ‹$u \leq p\ r^{@}k$›] ‹$r \neq \varepsilon$›].

lemma *per-root-nemp*[*dest*]: $u <p\ r \cdot u \Longrightarrow r \neq \varepsilon$
by *simp*

Empty word is not a periodic root but it has all nonempty periodic roots.

Any nonempty word is its own periodic root.

lemmas *root-self* = *triv-spref*

"Short roots are prefixes"

lemma $w <p\ r \cdot u \Longrightarrow |r| \leq |w| \Longrightarrow r \leq p\ w$
using *pref-prod-long*[*OF sprefD1*].

periodic words are prefixes of the power of the root, which motivates the notation

lemma *pref-pow-ext-take*: **assumes** $u \leq p\ r^{@}k$ **shows** $u \leq p$ *take* $|r|\ u \cdot r^{@}k$
proof (*rule le-cases*[*of* $|u|\ |r|$])
assume $|r| \leq |u|$
show $u \leq p$ *take* $|r|\ u \cdot r\ ^{@}\ k$
unfolding *pref-take*[*OF pref-prod-long*[*OF pref-pow-ext'*[*OF* ‹$u \leq p\ r^{@}k$›] ‹$|r| \leq |u|$›]] **using** *pref-pow-ext'*[*OF* ‹$u \leq p\ r^{@}k$›].
qed *simp*

lemma *pref-pow-take*: **assumes** $u \leq p\ r^{@}k$ **shows** $u \leq p$ *take* $|r|\ u \cdot u$
using *pref-prod-pref*[*of u take* $|r|\ u\ r^{@}k$, *OF pref-pow-ext-take* ‹$u \leq p\ r^{@}k$›, *OF* ‹$u \leq p\ r^{@}k$›].

lemma *per-root-powE*: **assumes** $u \leq p\ r \cdot u\ r \neq \varepsilon$
obtains k **where** $u <p\ r^{@}k$ **and** $0 < k$

proof–
show *thesis*
proof (*rule that*[*of Suc* |*u*|])
show $u <p\ r\ ^{@}\ Suc\ |u|$
using *per-exp-pref*[*OF* ‹$u \leq p\ r \cdot u$›, *of Suc* |*u*|]
long-pow-exp'[*OF* ‹$r \neq \varepsilon$›, *of* |*u*|] *nless-le* **by** *blast*
qed *simp*
qed

thm *per-rootI per-rootI'*

lemma *per-root-modE'* [*elim*]: **assumes** $u \leq p\ r \cdot u\ r \neq \varepsilon$
obtains p **where** $p <p\ r$ **and** $r^{@}(|u|\ div\ |r|) \cdot p = u$
proof–
have $r \neq \varepsilon$
using *assms* **by** *blast*
obtain m **where** $u <p\ r^{@}m$
using *per-root-powE*[*OF assms*].
from *pref-mod-pow*[*OF sprefD1*[*OF this*] ‹$r \neq \varepsilon$›]
obtain $k\ z$ **where** $k \leq m$ **and** $z <p\ r$ **and** $r\ ^{@}\ k \cdot z = u$.
have $k = (|u|\ div\ |r|)$
using *lenarg*[*OF* ‹$r\ ^{@}\ k \cdot z = u$›, *unfolded lenmorph pow-len*]
get-div[*OF prefix-length-less*[*OF* ‹$z <p\ r$›]] **by** *metis*
thus *?thesis*
using *that* ‹$r\ ^{@}\ k \cdot z = u$› ‹$z <p\ r$› **by** *blast*
qed

lemma *per-root-modE* [*elim*]: **assumes** $u \leq p\ r \cdot u\ r \neq \varepsilon$
obtains $n\ p\ s$ **where** $p \cdot s = r$ **and** $r^{@}n \cdot p = u$ **and** $s \neq \varepsilon$
using *per-root-modE'*[*OF assms*] *spref-exE* **by** *metis*

lemma *nemp-per-root-conv*: $r \neq \varepsilon \Longrightarrow u <p\ r \cdot u \longleftrightarrow u \leq p\ r \cdot u$
by *force*

lemma *root-ruler*: **assumes** $w \leq p\ u \cdot w\ v \leq p\ u \cdot v\ u \neq \varepsilon\ v \neq \varepsilon$
shows $w \bowtie v$
proof–
obtain $k\ l$ **where** $w <p\ u^{@}k\ v <p\ u^{@}l$
using *assms per-root-powE* **by** *meson*
moreover have $u^{@}k \bowtie u^{@}l$
using *conjug-pow eqd-comp* **by** *metis*
ultimately show *?thesis*
by (*rule ruler-comp*[*OF sprefD1 sprefD1*])
qed

lemmas *same-len-nemp-root-eq* = *root-ruler*[*THEN pref-comp-eq*]

lemma *per-root-add-exp*: **assumes** $u <p\ r \cdot u\ 0 < m$ **shows** $u <p\ r^{@}m \cdot u$

using ‹*0* < *m*›
proof (*induct m*)
case (*Suc m*)
then show *?case*
unfolding *pow-Suc rassoc*
using *spref-trans*[*OF* ‹*u* <*p r* · *u*›, *of r* · *r* ^@ *m* · *u*] ‹*u* <*p r* · *u*›
unfolding *spref-cancel-conv* **by** (*cases m* = *0*) *simp-all*
qed *simp*

theorem *per-root-pow-conv*: *x* <*p r* · *x* ⟷ (∃ *k*. *x* ≤*p r*^@*k*) ∧ *r* ≠ *ε*
by (*rule iffI*) (*metis per-root-powE self-append-conv2 sprefD*, *blast*)

lemma [*intro*]: *r* = *ε* ⟹ *u* ≤*p r* ⟹ *u* = *ε*
by *simp*

lemma [*intro*]: *u* = *ε* ⟹ *u* ≤*p w*
by *simp*

lemma *per-root-exp′*: **assumes** *x* ≤*p r*^@*k* **shows** *x* ≤*p r*^@|*x*|
proof (*cases r* = *ε*)
assume *r* ≠ *ε*
have |*x*| ≤ |*r*^@|*x*||
unfolding *pow-len* **using** *nemp-le-len*[*OF* ‹*r* ≠ *ε*›] **by** *force*
with *pref-ext*[*OF* ‹*x* ≤*p r*^@*k*›, *of r*^@|*x*|, *unfolded pows-comm*[*of k r*]]
show *?thesis*
by *blast*
qed (*use assms* **in** *blast*)

lemma *per-root-exp*: **assumes** *x* <*p r* · *x* **shows** *x* ≤*p r*^@|*x*|
proof–
obtain *k* **where** *x* ≤*p r*^@*k*
using ‹*x* <*p r* · *x*› **unfolding** *per-root-pow-conv* **by** *blast*
from *per-root-exp′*[*OF this*]
show *x* ≤*p r*^@|*x*|.
qed

lemma *per-root-drop-exp*: *u* <*p* (*r*^@*m*) · *u* ⟹ *u* <*p r* · *u*
unfolding *per-root-pow-conv* **unfolding** *pow-mult*[*symmetric*]
by *blast*

lemma *per-root-exp-conv*: *u* <*p* (*r*^@*Suc m*) · *u* ⟷ *u* <*p r* · *u*
by (*rule iffI*) (*use per-root-drop-exp* **in** *blast*, *use per-root-add-exp* **in** *blast*)

lemma *per-root-sq-drop*[*simp*]: *x* ≤*p y* · *y* · *x* ⟹ *x* ≤*p y* · *x*
by (*cases y* = *ε*, *force*)
(*use per-root-drop-exp*[*of x 2 y*, *THEN sprefD1*, *unfolded pow-list-2 rassoc*] **in** *force*)

lemmas *per-root-sq-drop-suf* = *per-root-sq-drop*[*reversed*, *unfolded rassoc*]

lemma *per-drop-exp*: $0 < k \Longrightarrow x \leq p\ r^{@}k \cdot x \Longrightarrow x \leq p\ \ r \cdot x$
using *pow-list-Nil-iff-Nil per-rootI per-root-drop-exp sprefD* **by** *metis*

lemmas *per-drop-exp-rev* = *per-drop-exp*[*reversed*]

corollary *comm-drop-exp*: **assumes** $0 < m$ **and** $u \cdot r^{@}m = r^{@}m' \cdot u$ **shows** $r \cdot u = u \cdot r$
proof
assume $r \neq \varepsilon\ u \neq \varepsilon$
hence $m = m'$
using *lenarg*[*OF* ‹$u \cdot r^{@}m = r^{@}m' \cdot u$›] **unfolding** *lenmorph pow-len*
by *auto*
have $u{\cdot}r \leq p\ u{\cdot}r^{@}m$
unfolding *pow-pos*[*OF* ‹$0 < m$›] **by** *simp*
have $u{\cdot}r \leq p\ r^{@}m' \cdot u \cdot r$
using *pref-ext*[*of* $u \cdot r\ r^{@}m \cdot u\ r$, *unfolded rassoc* ‹$m = m'$›, *OF* ‹$u{\cdot}r \leq p\ u{\cdot}r^{@}m$›[*unfolded* ‹$u \cdot r^{@}m = r^{@}m' \cdot u$›]].
hence $u{\cdot}r \leq p\ r{\cdot}(u{\cdot}r)$
using *per-root-drop-exp*[*of* $u{\cdot}r\ m'\ r$] ‹$0 < m$›[*unfolded* ‹$m = m'$›] *per-drop-exp*
by *blast*
from *comm-ruler*[*OF self-pref*[*of* $r \cdot u$], *of* $r \cdot u \cdot r$, *OF this*]
show $r \cdot u = u \cdot r$
unfolding *prefix-comparable-def*
by *force*
qed

lemma *comm-drop-exp'*: **assumes** $u^{@}k \cdot v = v \cdot u^{@}k'\ 0 < k'$ **shows** $u \cdot v = v \cdot u$
using *comm-drop-exp*[*OF* ‹$0 < k'$› *assms*(*1*)[*symmetric*]].

lemma *comm-drop-exps*: $0 < k \Longrightarrow$
$0 < m \Longrightarrow u\ ^{@}\ k \cdot v\ ^{@}\ m = v\ ^{@}\ m \cdot u\ ^{@}\ k \Longrightarrow u \cdot v = v \cdot u$
using *comm-pows-list-comm* **by** *blast*

lemma *comm-pow-roots*:
assumes $0 < m$ **and** $0 < k$
shows $u^{@}m \cdot v^{@}k = v^{@}k \cdot u^{@}m \longleftrightarrow u \cdot v = v \cdot u$
by (*rule*, *use comm-drop-exps*[*OF assms*] **in** *blast*)
(*use comm-pows-comm* **in** *blast*)

lemma *pow-comm-comm'*: **assumes** *comm*: $u^{@}(Suc\ k) = v^{@}(Suc\ l)$ **shows** $u \cdot v = v \cdot u$
using *comm pow-list-comm-comm* **by** *blast*

lemma *comm-trans*: **assumes** *uv*: $u{\cdot}v = \ v{\cdot}u$ **and** *vw*: $w{\cdot}v = v{\cdot}w$ **and** *nemp*: $v \neq \varepsilon$ **shows** $u \cdot w = w \cdot u$
proof −
consider (*u-emp*) $u = \varepsilon$ | (*w-emp*) $w = \varepsilon$ | (*nemp'*) $u \neq \varepsilon$ **and** $w \neq \varepsilon$ **by** *blast*

```
  then show ?thesis proof (cases)
    case nemp'
    have eq: u^@(|v| * |w|) = w^@(|v| * |u|)
    unfolding pow-mult comm-common-pow-list-iff[OF uv] comm-common-pow-list-iff[OF
vw]
      unfolding pow-mult[symmetric] mult.commute[of |u|]..
    obtain k l where k: |v| * |w| = Suc k and l: |v| * |u| = Suc l
      using nemp nemp' unfolding length-0-conv[symmetric]
      using not0-implies-Suc[OF no-zero-divisors]
      by presburger
    show ?thesis
      using pow-comm-comm'[OF eq[unfolded k l]].
  qed simp+
qed

lemma drop-per-pref: assumes w ≤p u · w shows drop |u| w ≤p w
  using assms drop-pref pref-drop by metis
```

Note that $[\![w <_p u \cdot w;\ u <_p t \cdot u]\!] \Longrightarrow w <_p t \cdot w$ does not hold.

```
lemma per-root-same-prefix:w <p r · w ⟹ w' ≤p r · w' ⟹  w ⋈ w'
  using root-ruler per-root-nemp pref-comp-sym ruler' sprefD by metis

lemma take-after-drop:  |u| + q ≤ |w| ⟹ w ≤p u · w ⟹ take q (drop |u| w) =
take q w
  using pref-share-take[OF drop-per-pref[of w u] len-after-drop[of |u| q w]].
```

The following lemmas are a weak version of the periodicity lemma

```
lemma two-pers:
  assumes pu: w ≤p u · w and pv: w ≤p v · w and len: |u| + |v| ≤ |w|
  shows u · v = v · u
proof–
  have uv: w ≤p (u · v) · w
    using pref-prolong[OF pu pv] unfolding lassoc.
  have vu: w ≤p (v · u) · w
    using pref-prolong[OF pv pu] unfolding lassoc.
  have u · v ≤p w
    using len pref-prod-long[OF uv] by simp
  moreover have v · u ≤p w
    using len pref-prod-long[OF vu] by simp
  ultimately show u · v = v · u
    using pref-comp-eq[OF - swap-len]
    unfolding prefix-comparable-def
    using ruler
    by meson
qed
```

2.16.2 Maximal root-prefix

lemma *max-root-mismatch*: **assumes** $u \cdot [a] <p\ r \cdot u \cdot [a]$ **and** $u \cdot [b] \leq p\ w$ **and** $a \neq b$
shows $w \wedge_p r \cdot w = u$
proof (*rule lcp-first-mismatch-pref*[*OF* ‹$u \cdot [b] \leq p\ w$› - ‹$a \neq b$›[*symmetric*]])
have $u \cdot [a] \leq p\ r \cdot u$
using *assms*(*1*)[*unfolded lassoc spref-snoc-iff*].
thus $u \cdot [a] \leq p\ r \cdot w$
using *append-prefixD*[*OF* ‹$u \cdot [b] \leq p\ w$›] *pref-prolong* **by** *blast*
qed

lemma *max-pref-per-root*: $u \wedge_p r \cdot u \leq p\ r \cdot (u \wedge_p r \cdot u)$
by (*rule pref-prod-pref*[*of* - - *u*]) *force*+

lemma *max-pref-pref*:
assumes $r \neq \varepsilon$
shows $u \wedge_p r \cdot u \leq p\ r^{@}|u \wedge_p r \cdot u|$
proof−
have $u \wedge_p r \cdot u <p\ r \cdot (u \wedge_p r \cdot u)$
using *assms max-pref-per-root* **by** *auto*
from *per-root-exp*[*OF this*]
show *?thesis*.
qed

lemma *max-pref-lcp-root-pow*: **assumes** $r \neq \varepsilon$ **and** $|u \wedge_p r \cdot u| \leq k$
shows $u \wedge_p r \cdot u = u \wedge_p r^{@}k$ (**is** *?max* $= u \wedge_p r^{@}k$)
proof (*rule pref-antisym*)
from *max-pref-pref*[*OF assms*(*1*)] *le-exps-pref*[*OF assms*(*2*)]
have *?max* $\leq p\ r^{@}k$
using *pref-trans* **by** *metis*
thus *?max* $\leq p\ u \wedge_p r^{@}k$
by *force*
show $u \wedge_p r^{@}k \leq p$ *?max*
proof (*rule lcp.boundedI*, *force*)
show $u \wedge_p r\ ^{@}\ k \leq p\ r \cdot u$
proof (*rule pref-prolong*)
show $u \wedge_p r\ ^{@}\ k \leq p\ r \cdot (u \wedge_p r\ ^{@}\ k)$
using *lcp.cobounded2* **by** (*rule pref-pow-root*[*of* $u \wedge_p r^{@}k$])
show $u \wedge_p r\ ^{@}\ k \leq p\ u$
using *lcp.cobounded1*.
qed
qed
qed

lemma *max-pref-shorter-lcp*: **assumes** $u \wedge_p r \cdot u <p\ v \wedge_p r \cdot v$
shows $u \wedge_p v = u \wedge_p r \cdot u$
proof (*cases*)

assume $r = \varepsilon$
then show *?thesis*
using *assms* **by** (*clarify*, *unfold emp-simps lcp.idem*) (*use lcp.absorb3* **in** *blast*)
next
let $?u = u \wedge_p r \cdot u$ **and** $?v = v \wedge_p r \cdot v$
assume $r \neq \varepsilon$
from *max-pref-lcp-root-pow*[*OF this*]
obtain k **where** $?u = u \wedge_p r^@k$ **and** $?v = v \wedge_p r^@k$
using *pref-len' suf-len'* **by** *meson*
from *ruler-spref-lcp*[*OF assms*[*unfolded this*], *folded* ‹$?u = u \wedge_p r^@k$›]
show $u \wedge_p v = u \wedge_p r \cdot u$**.**
qed

find-theorems $?u \wedge_p ?r \cdot ?u$

2.16.3 Period - numeric

Definition of a period as the length of the periodic root is often offered as the basic one. From our point of view, it is secondary, and less convenient for reasoning.

definition *period* :: *'a list* ⇒ *nat* ⇒ *bool*
where [*simp*]: *period w n* ≡ $w \leq p$ (*take n w*) · *w*

lemma *periodI*: $w \leq p$ (*take n w*) · $w \Longrightarrow$ *period w n*
unfolding *period-def* **by** *blast*

lemma *periodI-pref*: **assumes** $|r| = n$ $w \leq p$ $r \cdot w$
shows *period w n*
proof (*rule periodI*)
show $w \leq p$ *take n w* · *w*
proof (*rule disjE*[*OF linear*, *of* $|w|$ *n*], *force*)
assume $n \leq |w|$
hence *take n w* = *r*
using ‹$w \leq p$ $r \cdot w$› ‹$|r| = n$› *pref-prod-long*[*THEN pref-take*] **by** *blast*
thus $w \leq p$ *take n w* · *w*
using ‹$w \leq p$ $r \cdot w$› **by** *blast*
qed
qed

lemma *periodI-pref'*: **assumes** $w \leq p$ $r \cdot w$
shows *period w* $|r|$
using *periodI-pref*[*OF refl*] ‹$w \leq p$ $r \cdot w$› **by** *auto*

The numeric definition respects the following convention about empty words and empty periods.

lemma *emp-all-periods*: *period* ε *n*
by *simp*

lemma *zero-period*: *period w 0*
by *simp*

lemma *periodD*: *period w n* $\Longrightarrow$ *w* $\leq p$ *take n w* $\cdot$ *w*
by *simp*

A nonempty word has all "long" periods

lemma *all-long-pers*: $|w| \leq n \Longrightarrow$ *period w n*
by *simp*

lemma *len-is-per*: *period w* $|w|$
by *simp*

Period is preserved by reversal and taking factors

lemma *period-rev*: **assumes** *period w p* **shows** *period* (*rev w*) *p*
proof (*rule disjE*[*OF linear*, *of* $|w|$ *p*], *use assms* **in** *force*)
assume $p \leq |w|$
obtain *s* **where** *s*: *take p w* $\cdot$ *w* = *w* $\cdot$ *s*
using *prefixE*[*OF periodD*[*OF assms*]]**.**
have $|rev\ s| = p$
using *lenarg*[*OF s*, *unfolded lenmorph*] ‹$p \leq |w|$› **by** *force*
have *rev w* $\leq p$ *rev s* $\cdot$ *rev w*
using *s* **using** *prefI rev-append* **by** *metis*
from *periodI-pref*[*OF refl this*]
show *period* (*rev w*) *p*
using ‹$|rev\ s| = p$› **by** *blast*
qed

lemma *period-rev-conv* [*reversal-rule*]: *period* (*rev w*) *n* $\longleftrightarrow$ *period w n*
using *period-rev period-rev*[*of rev w*] **unfolding** *rev-rev-ident* **by** (*intro iffI*)

lemma *period-pref-period*: **assumes** *period w n v* $\leq p$ *w*
shows *period v n*
proof (*rule disjE*[*OF linear*, *of* $|v|$ *n*], *use all-long-pers* **in** *blast*)
assume $n \leq |v|$
hence *same-take*: *take n v* = *take n w*
using *pref-share-take*[*OF assms*(*2*)] **by** *metis*
hence *v* $\leq p$ *take n v* $\cdot$ *v*
using *pref-keeps-per-root*[*OF periodD* ‹*v* $\leq p$ *w*›, *OF* ‹*period w n*›] **by** *argo*
show *?thesis*
by (*rule periodI*) *fact*
qed

lemmas *period-suf-period* = *period-pref-period*[*reversed*] **and**
periodI-suf = *periodI-pref*[*reversed*] **and**

periodI-suf' = *periodI-pref'*[*reversed*]

lemma *period-fac-period*: **assumes** *period* (*u*·*w*·*v*) *p*
shows *period w p*
using *period-suf-period*[*OF assms*(*1*), *of w* · *v*, *THEN period-pref-period*] **by** *blast*

lemma *period-fac-period'*: *period v p* ⟹ *u* ≤*f v* ⟹ *period u p*
by (*elim facE*, *hypsubst*, *rule period-fac-period*)

The standard numeric definition of a period uses indeces.

lemma *period-indeces*: **assumes** *period w n* **and** *i* + *n* < |*w*| **shows** *w*!*i* = *w*!(*i*+*n*)
proof−
have *w* ! *i* = (*take n w* · *w*) ! (*n* + *i*)
using *nth-append-length-plus*[*of take n w w i*, *symmetric*]
unfolding *take-len*[*OF less-imp-le*[*OF add-lessD2*[*OF* ‹*i* + *n* < |*w*|›]]].
also have ... = *w* ! (*i* + *n*)
using *pref-index*[*OF periodD*[*OF* ‹*period w n*›] ‹*i* + *n* < |*w*|›, *symmetric*]
unfolding *add.commute*[*of n i*].
finally show *?thesis*.
qed

lemma *period-mod*: *period w n* ⟹ *i* < |*w*| ⟹ *w*!*i* = *w*!(*i mod n*)
proof (*cases n* = *0*, *force*)
assume *n* ≠ *0*
show *period w n* ⟹ *i* < |*w*| ⟹ *w*!*i* = *w*!(*i mod n*)
proof (*induct i rule*: *less-induct*)
case (*less i*)
then show *?case*
proof (*cases i* < *n*, *unfold mod-less*[*of i n*], *blast*)
assume ¬ *i* < *n*
hence *i* − *n* + *n* = *i i* − *n* < *i i* − *n* < |*w*| (*i* − *n*) *mod n* = *i mod n*
using ‹*i* < |*w*|› ‹*n* ≠ *0*› *mod-add-self2*[*of i*−*n n*] **by** *force*+
from *less.hyps*[*OF* ‹*i* − *n* < *i*› ‹*period w n*› ‹*i* − *n* < |*w*|›]
show *w* ! *i* = *w* ! (*i mod n*)
unfolding *period-indeces*[*OF* ‹*period w n*›, *of i*−*n*, *unfolded* ‹*i* − *n* + *n* = *i*›, *OF* ‹*i* < |*w*|›] ‹(*i* − *n*) *mod n* = *i mod n*›.
qed
qed
qed

lemma *indeces-period*:
assumes *forall*: ⋀ *i*. *i* + *n* < |*w*| ⟹ *w*!*i* = *w*!(*i*+*n*)
shows *period w n*
proof−
have |*w*| ≤ |*take n w* · *w*|
by *auto*
have *w* ! *j* = (*take n w* · *w*) ! *j* **if** *j* < |*w*| **for** *j*
proof (*cases j* < |*take n w*|)
assume *j* < |*take n w*| **show** *w* ! *j* = (*take n w* · *w*) ! *j*

```
    using pref-index[OF take-is-prefix ‹j < |take n w|›, symmetric]
    unfolding pref-index[OF triv-pref ‹j < |take n w|›, of w].
  next
    assume ¬ j < |take n w|
    from leI[OF this] ‹j < |w|›
    have |take n w| = n
      by force
    hence j = (j − n) + n and (j − n) + n < |w|
      using  leI[OF ‹¬ j < |take n w|›] ‹j < |w|› by simp+
    hence w!j = w!(j − n)
      using forall by simp
   from this[folded nth-append-length-plus[of take n w w j−n, unfolded ‹|take n w|
= n›]]
    show w ! j = (take n w · w) ! j
      using ‹j = (j − n) + n› by simp
  qed
  with index-pref[OF ‹|w| ≤ |take n w · w|›]
  have w ≤p take n w · w by blast
  thus ?thesis
    using assms by force
qed
```

Alternative proof of $period\ ?w\ ?p \Longrightarrow period\ (rev\ ?w)\ ?p$ using indeces

```
lemma assumes period w p shows period (rev w) p
proof (rule indeces-period[of p rev w])
  fix i assume i + p < |rev w|
  from this[unfolded length-rev] add-lessD1
  have i < |w| and i + p < |w| by blast+
  have e: |w| − Suc (i + p) + p = |w| − Suc i using ‹i + p < |rev w|› by simp
  have |w| − Suc (i + p) + p < |w|
    using ‹i + p < |w|› Suc-diff-Suc ‹i < |w|›
      diff-less-Suc e less-irrefl-nat not-less-less-Suc-eq by metis
  from period-indeces[OF assms this] rev-nth[OF ‹i  < |w|›, folded e] rev-nth[OF
‹i + p < |w|›]
  show rev w ! i = rev w !(i+p) by presburger
qed

lemma pow-per[intro]: period (y@k) |y|
  using periodI[OF pref-pow-ext-take, OF self-pref] by blast

lemma per-fac: assumes w ≤f y@k shows period w |y|
   using pow-per period-fac-period  facE[OF ‹w ≤f y@k›] by metis



lemma per-drop [intro]: assumes period w n shows drop n w ≤p w
proof−
  show drop n w ≤p w
```

using *drop-per-pref*[*OF* ‹*period w n*›[*unfolded period-def*]]
append-take-drop-id[*of n w*, *unfolded append-eq-conv-conj*] **by** *argo*
qed

Two more characterizations of a period

theorem *per-shift*:
shows *period w n* ⟷ *drop n w* ≤*p w*
proof
assume *drop n w* ≤*p w*
then show *period w n*
unfolding *period-def* **using** *append-take-drop-id pref-cancel′* **by** *metis*
qed (*simp only*: *per-drop*)

lemma *rotate-per-root*: **assumes** *w* = *rotate n w*
shows *period w n*
proof (*cases* |*w*| ≤ *n*)
assume |*w*| ≤ *n*
from *all-long-pers*[*OF this*]
show *period w n*.
next
assume *not*: ¬ |*w*| ≤ *n*
have *drop* (*n mod* |*w*|) *w* ≤*p w*
using *prefI*[*OF rotate-drop-take*[*symmetric*, *of n w*]]
unfolding ‹*w* = *rotate n w*›[*symmetric*].
from *this*[*unfolded mod-less*[*OF not*[*unfolded not-le*]]]
show *period w n*
unfolding *per-shift strict-prefix-def drop-neq* **by** *blast*
qed

Various lemmas on periods

lemma *ext-per-left*: **assumes** *period w p* **and** *p* ≤ |*w*|
shows *period* (*take p w* · *w*) *p*
proof–
have *f*: *take p* (*take p w* · *w*) = *take p w*
using ‹*p* ≤ |*w*|› **by** *simp*
show *?thesis*
using ‹*period w p*› *pref-cancel′*[*of w take p w* · *w take p w*]
unfolding *f period-def*
by *blast*
qed

lemma *ext-per-left-power*: *period w p* ⟹ *p* ≤ |*w*| ⟹ *period* ((*take p w*)^@*k* · *w*) *p*
proof (*induction k*)
case (*Suc k*)
show *?case*
using *ext-per-left*[*OF Suc.IH*[*OF* ‹*period w p*› ‹*p* ≤ |*w*|›]] ‹*p* ≤ |*w*|›

unfolding *pref-share-take*[*OF per-exp-pref*[*OF periodD*[*OF* ‹*period w p*›]] ‹$p \leq |w|$›,*symmetric*]
lassoc pow-Suc[*symmetric*] **by** *fastforce*
qed *auto*

lemma *take-several-pers*: **assumes** *period w n* **and** $m*n \leq |w|$
shows $(take\ n\ w)^{@}m = take\ (m*n)\ w$
proof (*cases* $m = 0$)
assume $m \neq 0$
have $|(take\ n\ w)^{@}m| = m*n$
unfolding *pow-len nat-prod-le*[*OF* ‹$m \neq 0$› ‹$m*n \leq |w|$›, *THEN take-len*] **by** *blast*
have $(take\ n\ w)^{@}m \leq p\ w$
using ‹*period w n*›[*unfolded period-def*]
ruler-le[*of take n* $w^{@}m$ *take n* $w^{@}m \cdot w\ w$, *OF triv-pref*] ‹$m * n \leq |w|$›[*folded* ‹$|take\ n\ w^{@}m| = m * n$›]
per-exp-pref **by** *metis*
show *?thesis*
using *pref-take*[*OF* ‹$take\ n\ w^{@}m \leq p\ w$›, *unfolded* ‹$|take\ n\ w^{@}m| = m * n$›, *symmetric*]**.**
qed *simp*

lemma *per-div*: **assumes** *n dvd* $|w|$ **and** *period w n*
shows $(take\ n\ w)^{@}(|w|\ div\ n) = w$
using *take-several-pers*[*OF* ‹*period w n*› *div-times-less-eq-dividend*] **unfolding** *dvd-div-mult-self*[*OF* ‹*n dvd* $|w|$›] *take-self***.**

lemma *per-mult*: **assumes** *period w n* **shows** *period w* $(m*n)$
proof (*cases* $m*n \leq |w|$)
assume $\neg\ m * n \leq |w|$ **thus** *period w* $(m*n)$
using *all-long-pers*[*of* $w\ m * n$] **by** *linarith*
next
assume $m * n \leq |w|$
show *period w* $(m*n)$
using ‹*period w n*›
unfolding *period-def*
using *per-exp-pref*[*of w take n w*]
take-several-pers[*OF* ‹*period w n*› ‹$m*n \leq |w|$›, *symmetric*]
by *presburger*
qed

theorem *two-periods*:
assumes *period w p period w q* $p + q \leq |w|$
shows *period w* $(gcd\ p\ q)$
proof (*cases* $p = 0 \vee q = 0 \vee w = \varepsilon$, *use assms*(*1*,*2*) **in** *fastforce*)
assume $\neg\ (p = 0 \vee q = 0 \vee w = \varepsilon)$
hence $p \neq 0\ q \neq 0\ w \neq \varepsilon$
by *blast*+
have *p1*: $w \leq p\ take\ p\ w \cdot w$ **and** *p2*: $w \leq p\ take\ q\ w \cdot w$

using ‹*period w p*›[*unfolded period-def*] ‹*period w q*›[*unfolded period-def*].
have |*take p w*| + |*take q w*| ≤ |*w*|
using ‹*p* + *q* ≤ |*w*|› *add-leD1 add-leD2 take-len* **by** *metis*
from *two-pers*[*OF p1 p2 this*, *unfolded comm*]
obtain *t k m* **where** *take p w* = $t^{@}k$ *take q w* = $t^{@}m$
by *blast*
hence *0* < *k m* ≠ *0*
using ‹*p* ≠ *0*› ‹*q* ≠ *0*› ‹*w* ≠ *ε*› ‹*take p w* = *t* $^{@}$ *k*› **by** *force*+
have *w* ≤*p t* · *w*
using *per-drop-exp*[*OF* ‹*0* < *k*› ‹*w* ≤*p take p w* · *w*›[*unfolded* ‹*take p w* = $t^{@}k$›]].
hence *period w* |*t*|
using *periodI-pref* **by** *blast*
have |*t*| *dvd* (*gcd p q*)
using *lenarg*[*OF* ‹*take p w* = $t^{@}k$›] *lenarg*[*OF* ‹*take q w* = $t^{@}m$›]
unfolding *lenmorph pow-len*
take-len[*OF add-leD1*[*OF* ‹*p* + *q* ≤ |*w*|›]] *take-len*[*OF add-leD2*[*OF* ‹*p* + *q* ≤ |*w*|›]]
using *dvd-triv-right gcd-nat.boundedI* **by** *meson*
from *per-mult*[*OF* ‹*period w* |*t*|›, *of gcd p q div* |*t*|]
show *?thesis*
unfolding *dvd-div-mult-self*[*OF* ‹|*t*| *dvd* (*gcd p q*)›].
qed

lemma *index-mod-per-root*: **assumes** *r* ≠ *ε* **and** *i*: ∀ *i* < |*w*|. *w*!*i* = *r*!(*i mod* |*r*|)
shows *w* <*p r* · *w*
proof−
have *i* < |*w*| ⟹ (*r* · *w*) ! *i* = *r* ! (*i mod* |*r*|) **for** *i*
by (*simp add*: *i mod-if nth-append*)
hence *w* ≤*p r* · *w*
using *index-pref*[*of w r* · *w*] *i*
by *simp*
thus *?thesis* **using** ‹*r* ≠ *ε*› **by** *auto*
qed

lemma *index-pref-pow-mod*: *w* ≤*p* $r^{@}k$ ⟹ *i* < |*w*| ⟹ *w*!*i* = *r*!(*i mod* |*r*|)
using *index-pow-mod*[*of i k r*] *less-le-trans*[*of i* |*w*| |$r^{@}k$|] *prefix-length-le*[*of w* $r^{@}k$] *pref-index*[*of w* $r^{@}k$ *i*] **by** *argo*

lemma *index-per-root-mod*: **assumes** *w* ≤*p r* · *w r* ≠ *ε i* < |*w*|
shows *w*!*i* = *r*!(*i mod* |*r*|)
using *index-pref-pow-mod*[*OF - assms*(*3*)] *per-root-powE*[*OF assms*(*1,2*)] *sprefD1*
by *meson*

lemma *prepend-periods*: **assumes** *p dvd n n* ≤ |*w*| *period w p*
shows *period* ((*take n w*) · *w*) *p*
proof (*cases n* = *0* ∨ *w* = *ε*, *use* ‹*period w p*› **in** *force*)
assume *ntriv*: ¬ (*n* = *0* ∨ *w* = *ε*)

have *len*: $|take\ n\ w| = n$
using ‹$n \leq |w|$› **by** *simp*
have $p - n = 0$
using ‹p *dvd* n› *ntriv* **by** *auto*
have $take\ n\ w \neq \varepsilon$
using *ntriv* **by** *force*
define r **where** $r = take\ p\ (take\ n\ w)$
define e **where** $e = n\ div\ p$
have r: $take\ p\ w = r$
unfolding *r-def* **using** ‹p *dvd* n› *ntriv* **by** *auto*
from *period-pref-period*[*OF* ‹*period w p*› *take-is-prefix*]
have *r-e*: $take\ n\ w = r^{@} e$
using *per-div*[*of p take n w*, *symmetric*] **unfolding** *e-def r-def len* **using** ‹p *dvd* n› **by** *blast*
have r': $take\ p\ (take\ n\ w \cdot w) = r$
unfolding *take-append r-def len* ‹$p - n = 0$› **by** *simp*
show *?thesis*
using ‹*period w p*› **unfolding** *period-def* r'
unfolding *r r-e lassoc pow-Suc*[*symmetric*] *pow-Suc2* **by** *force*
qed

2.16.4 Period: overview

notepad
begin
fix *w r*::$'a$ *list*
fix *n*::*nat*
assume $w \neq \varepsilon$ $r \neq \varepsilon$ $0 < n$
have $\neg\ w <p\ \varepsilon \cdot w$
by *simp*
have $\neg\ \varepsilon <p\ \varepsilon \cdot \varepsilon$
by *simp*
have $\varepsilon <p\ r \cdot \varepsilon$
using ‹$r \neq \varepsilon$› **by** *blast*
have *period w 0*
by *simp*
have *period* ε *0*
by *simp*
have *period* ε *n*
by *simp*
end

2.16.5 Singleton and its power

lemma *concat-len-one*: **assumes** $|us| = 1$ **shows** *concat us* = *hd us*
using *concat-sing*[*OF sing-word*[*OF* ‹$|us| = 1$›, *symmetric*]].

lemma *sing-pow-hd-tl*: $set\ (c\ \#\ w) \subseteq \{a\} \longleftrightarrow c = a \wedge set\ w \subseteq \{a\}$
by *auto*

lemma *pref-sing-pow*: **assumes** $w \leq p\ [a]^{@} m$ **shows** $w = [a]^{@}|w|$
using *pref-same-len*[*OF per-root-exp'*[*OF assms*], *unfolded sing-pow-len*, *OF refl*]**.**

lemmas *suf-sing-pow* = *pref-sing-pow*[*reversed*]

lemma *sing-pow-palindrom*: **assumes** $w = [a]^{@} k$ **shows** *rev* $w = w$
by (*simp add*: *assms rev-pow*)

lemma *sing-pow-palindrom'*[*simp*]: *rev* $[a]^{@} k = [a]^{@} k$
using *sing-pow-palindrom* **by** *simp*

lemma *sing-fac-pow*: **assumes** *set* $w \subseteq \{a\}$ **and** $v \leq f\ w$ **shows** *set* $v \subseteq \{a\}$
using ‹*set* $w \subseteq \{a\}$› *set-mono-sublist*[*OF* ‹$v \leq f\ w$›] **by** *order*

lemma *sing-pow-fac*: **assumes** $a \neq b$ **and** *set* $w \subseteq \{a\}$ **shows** $\neg\ ([b] \leq f\ w)$
using *sing-fac-pow*[*OF* ‹*set* $w \subseteq \{a\}$›, *of* $[b]$] **unfolding** *sing-pow-hd-tl*
using ‹$a \neq b$› **by** *auto*

lemma *all-set-sing-pow*: $(\forall\ b.\ b \in set\ w \longrightarrow b = a) \longleftrightarrow set\ w \subseteq \{a\}$
by *blast*

lemma *sing-fac-set*: $[a] \leq f\ x \Longrightarrow a \in set\ x$
by *fastforce*

lemma *set-sing-pow-hd* [*simp*]: **assumes** $0 < k$ **shows** $a \in set\ ([a]^{@} k)$
using *assms gr0-conv-Suc* **by** *force*

lemma *neq-set-not-root*: $a \neq b \Longrightarrow b \in set\ w \Longrightarrow \neg\ set\ w \subseteq \{a\}$
by *blast*

lemma *sing-pow-set-Suc*[*simp*]: *set* $([a]^{@} Suc\ k) = \{a\}$
by (*induct k*, *simp-all*)

lemma *per-one*: *set* $w \subseteq \{a\} \longleftrightarrow w <p\ [a] \cdot w$
by (*rule iffI*) (*induct w*, *simp-all*)+

lemma *per-one'*: **assumes** *set* $r \subseteq \{a\}$ $w <p\ r \cdot w$
shows *set* $w \subseteq \{a\}$
using *assms* **by** (*induct w*, *force*)
(*metis per-one per-root-drop-exp sing-set-word*)

thm *pref-sing-pow*

lemma *pref-sing-pow'*: $w \leq p\ [a]^{@} m \Longrightarrow set\ w \subseteq \{a\}$
unfolding *per-one* **by** *blast*

lemma *sing-pow-set'*: *set* $([a]\ ^{@}\ k) \subseteq \{a\}$
by (*induction k*) *auto*

lemma *sing-pow-set-sub* [*elim*]: $x = [a] \ ^{@} \ k \Longrightarrow set\ x \subseteq \{a\}$
using *sing-pow-set'*[*of k a*] **by** *blast*

lemma *set-sing-nemp-eq* [*intro*]: $set\ w \subseteq \{a\} \Longrightarrow w \neq \varepsilon \Longrightarrow set\ w = \{a\}$
using *set-empty* **unfolding** *subset-singleton-iff* **by** *blast*

lemma *sing-pow-set-pos-eq* [*intro*]: $w = [a] \ ^{@} \ k \Longrightarrow 0 < k \Longrightarrow set\ w = \{a\}$
by (*rule set-sing-nemp-eq*[*OF sing-pow-set-sub*]) *force+*

lemma *unique-letter-fac-expE*: **assumes** $w \leq f\ [a]^{@}k$
obtains m **where** $w = [a]^{@}m$
using *sing-set-wordE*[*OF subset-trans*[*OF set-mono-sublist*[*OF assms*]], *OF sing-pow-set'*].

lemma *neq-in-set-not-pow*: **assumes** $a \neq b\ b \in set\ x$ **shows** $x \neq [a]^{@}k$
by (*rule notI*) (*use assms sing-pow-set-sub* **in** *fast*)

lemma *sing-pow-card-set-Suc*: **assumes** $c = [a]^{@}Suc\ k$ **shows** $card(set\ c) = 1$
proof−
have $card\ \{a\} = 1$ **by** *simp*
from *this*[*folded sing-pow-set-Suc*[*of k a*]]
show $card(set\ c) = 1$
unfolding *assms*.
qed

lemma *sing-pow-card-set*: **assumes** $k \neq 0$ **and** $c = [a]^{@}k$ **shows** $card(set\ c) = 1$
using *sing-pow-card-set-Suc*[*of c k* − *1*, *unfolded Suc-minus*[*OF* ‹$k \neq 0$›], *OF* ‹$c = [a]^{@}k$›].

lemma *takeWhile-eq-set-conv*[*simp*]: $takeWhile(\lambda\ x.\ x = a)\ w = w \longleftrightarrow set\ w \subseteq \{a\}$
using *takeWhile-eq-all-conv*[*of* $\lambda\ x.\ x = a\ w$] **by** *blast*

lemma *dropWhile-distinct*: **assumes** $\neg\ set\ w \subseteq \{a\}$
shows $takeWhile(\lambda\ x.\ x = a)\ w\ \cdot\ [hd\ (dropWhile\ (\lambda x.\ x = a)\ w)] \leq p\ w$
using *takeWhile-dropWhile-id*[*of* $\lambda\ x.\ x = a\ w$] **using** *assms*[*folded takeWhile-eq-set-conv*]
by *blast*

lemma *takeWhile-subset*: $set\ (takeWhile\ (\lambda\ x.\ x = a)\ w) \subseteq \{a\}$
using *takeWhile-eq-set-conv takeWhile-idem* **by** *metis*

lemma *takeWhile-sing-pow*: $takeWhile\ (\lambda\ x.\ x = a)\ w = w \longleftrightarrow w = [a]^{@}|w|$
by(*induct w*, *auto*)

lemma *letter-pref-exp-pref*: $[a]^{@}|takeWhile\ (\lambda\ x.\ x = a)\ u| \cdot dropWhile\ (\lambda x.\ x = a)\ u = u$
using *takeWhile-idem takeWhile-sing-pow*
takeWhile-dropWhile-id[*of* $\lambda\ x.\ x = a\ u$] **by** *metis*

lemma *letter-pref-exp-mismatch*: $u = [a]^{@}|takeWhile\ (\lambda\ x.\ x = a)\ u| \cdot v \Longrightarrow v \neq \varepsilon \Longrightarrow hd\ v \neq a$
by(*subst* (*asm*) *letter-pref-exp-pref*[*of a u*, *symmetric*])
(*use* *hd-dropWhile*[*of* $\lambda\ x.\ x = a\ u$] **in** *fast*)

lemma *dropWhile-sing-pow*: $dropWhile\ (\lambda\ x.\ x = a)\ w = \varepsilon \longleftrightarrow w = [a]^{@}|w|$
by(*induct w*, *auto*)

lemma *sing-exp-len*: $u = [a]^{@}m \Longrightarrow u = [a]^{@}|u|$
by *simp*

lemma *nemp-takeWhile-hd*: $us \neq \varepsilon \Longrightarrow hd\ (takeWhile\ (\lambda\ a.\ a = hd\ us)\ us) = hd\ us$
by (*simp add*: *pref-hd-eq takeWhile-eq-Nil-iff takeWhile-is-prefix*)

lemma $w = [a]^{@}|w| \longleftrightarrow set\ w \subseteq \{a\}$
using *sing-pow-set-sub*[*of w* $|w|$ *a*] *sing-set-word*[*of w a*] **by** *blast*

lemma *distinct-letter-in*: **assumes** $\neg\ set\ w \subseteq \{a\}$
obtains *m b q* **where** $[a]^{@}m \cdot [b] \cdot q = w$ **and** $b \neq a$
proof−
let *?u* = *takeWhile* $(\lambda\ x.\ x = a)\ w$
let *?v* = *dropWhile* $(\lambda\ x.\ x = a)\ w$
let *?b* = *hd ?v*

have $?v \neq \varepsilon$
using *assms* **by** *auto*
have *u*: $?u = [a]^{@}|?u|$
using *sing-set-word*[*OF takeWhile-subset*].
have $?b \neq a$
using *hd-dropWhile*[*OF* ‹*dropWhile* $(\lambda x.\ x = a)\ w \neq \varepsilon$›].

have *eq*: $?u \cdot [?b] \cdot tl\ ?v = w$
unfolding *hd-tl*[*OF* ‹$?v \neq \varepsilon$›] **by** *simp*
from *that*[*OF* - ‹$?b \neq a$›, *of* $|?u|$, *folded u*, *OF this*]
show *thesis*.
qed

lemma *distinct-letter-in-hd*: **assumes** $\neg\ set\ w \subseteq \{hd\ w\}$
obtains *m b q* **where** $[hd\ w]^{@}m \cdot [b] \cdot q = w$ **and** $b \neq hd\ w$ **and** $m \neq 0$
proof−
obtain *m b q* **where** *a1*: $[hd\ w]^{@}m \cdot [b] \cdot q = w$ **and** *a2*: $b \neq hd\ w$
using *distinct-letter-in*[*OF assms*].
have $m \neq 0$
proof (*rule notI*)
assume $m = 0$
note *a1*[*unfolded this pow-zero emp-simps*, *folded hd-word*]
thus *False* **using** *a2* **by** *force*
qed

from *that*[*OF a1 a2 this*]
show *thesis*.
qed

lemma *distinct-letter-in-hd'*: **assumes** ¬ *set w* ⊆ {*hd w*}
obtains *m b q* **where** [*hd w*]^@*Suc m* · [*b*] · *q* = *w* **and** *b* ≠ *hd w*
using *distinct-letter-in-hd*[*OF assms*] *Suc-minus* **by** *metis*

lemma *distinct-letter-in-suf*: **assumes** ¬ *set w* ⊆ {*a*}
obtains *m b* **where** [*b*] · [*a*]^@*m* ≤*s w* **and** *b* ≠ *a*
using *distinct-letter-in*[*reversed*, *unfolded rassoc*, *OF assms*]
unfolding *suffix-def* **by** *metis*

lemma *per-sing-one*: **assumes** *w* ≤*p* [*a*] · *w* **shows** *period w 1*
using *periodI-pref'*[*OF assms*, *unfolded sing-len*].

lemma *sing-pow-exp*: *set w* ⊆ {*a*} ⟷ *w* = [*a*]^@|*w*|
using *sing-pow-set-sub sing-set-word* **by** *metis*

lemma *sing-power'*: **assumes** *set w* ⊆ {*a*} **and** *i* < |*w*| **shows** *w* ! *i* = *a*
using *nth-pow-list-single*[*OF* ‹*i* < |*w*|›, *of a*, *folded* ‹*set w* ⊆ {*a*}›[*unfolded sing-pow-exp*]].

lemma *sing-set-palindrom*: *set w* ⊆ {*a*} ⟹ *rev w* = *w*
using *sing-pow-palindrom sing-set-word* **by** *metis*

lemma *lcp-letter-power*:
assumes *w* ≠ *ε* **and** *set w* ⊆ {*a*} **and** [*a*]^@*m* · [*b*] ≤*p z* **and** *a* ≠ *b*
shows *w* · *z* ∧*p z* · *w* = [*a*]^@*m*
proof–
obtain *k z'* **where** *w* = [*a*]^@*k z* = [*a*]^@*m* · [*b*] · *z' k* ≠ *0*
using ‹*set w* ⊆ {*a*}› ‹[*a*]^@*m* · [*b*] ≤*p z*› ‹*w* ≠ *ε*›
unfolding *prefix-def rassoc* **by** *blast*
hence *eq1*: *w* · *z* = [*a*]^@*m* · ([*a*]^@*k* · [*b*] · *z'*) **and** *eq2*: *z* · *w* = [*a*]^@*m* · ([*b*] · *z'*· [*a*]^@*k*)
by (*simp add*: ‹*w* = [*a*]^@*k*› ‹*z* = [*a*]^@*m* · [*b*] · *z'*› *pows-comm*)+
have *hd* ([*a*]^@*k* · [*b*] · *z'*) = *a*
using *hd-append2*[*OF* ‹*w* ≠ *ε*›, *of* [*b*]·*z'*,
unfolded ‹*w* = (*a* # *ε*)^@*k*› *hd-sing-pow*[*OF* ‹*k* ≠ *0*›, *of a*]].
moreover have *hd*([*b*] · *z'*· [*a*]^@*k*) = *b*
by *simp*
ultimately have [*a*]^@*k* · [*b*] · *z'* ∧*p* [*b*] · *z'*· [*a*]^@*k* = *ε*
by (*simp add*: ‹*a* ≠ *b*› *lcp-distinct-hd*)
thus *?thesis* **using** *eq1 eq2 lcp-ext-left*[*of* [*a*]^@*m* [*a*]^@*k* · [*b*] · *z'* [*b*] · *z'*· [*a*]^@*k*]
by *simp*
qed

2.17 Border

A non-empty word $x \neq w$ is a *border* of a word w if it is both its prefix and suffix. This elementary property captures how much the word w overlaps with itself, and it is in the obvious way related to a period of w. However, in many cases it is much easier to reason about borders than about periods.

definition *border* :: *'a list* ⇒ *'a list* ⇒ *bool* (- ≤*b* - [*51*,*51*] *60*)
where [*simp*]: *border x w* = (*x* ≤*p w* ∧ *x* ≤*s w* ∧ *x* ≠ *w* ∧ *x* ≠ *ε*)

definition *bordered* :: *'a list* ⇒ *bool*
where [*simp*]: *bordered w* = (∃ *b*. *b* ≤*b w*)

lemma *borderI*[*intro*]: *x* ≤*p w* ⟹ *x* ≤*s w* ⟹ *x* ≠ *w* ⟹ *x* ≠ *ε* ⟹ *x* ≤*b w*
unfolding *border-def* **by** *blast*

lemma *borderD-pref*: *x* ≤*b w* ⟹ *x* ≤*p w*
unfolding *border-def* **by** *blast*

lemma *borderD-spref*: *x* ≤*b w* ⟹ *x* <*p w*
unfolding *border-def* **by** *simp*

lemma *border-sprefE* [*elim*]: **assumes** *b* ≤*b w* **obtains** *r* **where** *b* · *r* = *w* **and** *r* ≠ *ε*
using *borderD-spref*[*OF assms*] **by** *blast*

lemma *borderD-suf*: *x* ≤*b w* ⟹ *x* ≤*s w*
unfolding *border-def* **by** *blast*

lemma *borderD-ssuf*: *x* ≤*b w* ⟹ *x* <*s w*
unfolding *border-def* **by** *blast*

lemma *borderD-nemp*: *x* ≤*b w* ⟹ *x* ≠ *ε*
using *border-def* **by** *blast*

lemma *border-ssufE* [*elim*]: **assumes** *b* ≤*b w* **obtains** *r* **where** *r* · *b* = *w* **and** *r* ≠ *ε*
using *borderD-ssuf*[*OF assms*] **by** *blast*

lemma *borderD-neq*: *x* ≤*b w* ⟹ *x* ≠ *w*
unfolding *border-def* **by** *blast*

lemma *borderedI*: *u* ≤*b w* ⟹ *bordered w*
unfolding *bordered-def* **by** *blast*

lemma *border-lq-nemp*: **assumes** *x* ≤*b w* **shows** $x^{-1>}w \neq \varepsilon$
using *assms borderD-spref lq-spref-nemp* **by** *blast*

lemma *border-rq-nemp*: **assumes** *x* ≤*b w* **shows** $w^{<-1}x \neq \varepsilon$

using *assms borderD-ssuf rq-ssuf-nemp* **by** *blast*

lemma *border-trans*[*trans*]: **assumes** $t \leq_b x$ $x \leq_b w$
shows $t \leq_b w$
using *assms* **unfolding** *border-def*
using *suffix-order.antisym pref-trans*[*of t x w*] *suf-trans*[*of t x w*] **by** *blast*

lemma *border-rev-conv*[*reversal-rule*]: $rev\ x \leq_b rev\ w \longleftrightarrow x \leq_b w$
unfolding *border-def*
using *rev-is-Nil-conv*[*of x*] *rev-swap*[*of w*] *rev-swap*[*of x*]
suf-rev-pref-iff[*of x w*] *pref-rev-suf-iff*[*of x w*]
by *fastforce*

lemma *border-lq-comp*: $x \leq_b w \Longrightarrow (w^{<-1}x) \bowtie x$
unfolding *border-def* **using** *rq-is-pref*[*of w x*] *ruler'* **by** *blast*

lemmas *border-lq-suf-comp* = *border-lq-comp*[*reversed*]

2.17.1 The shortest border

lemma *border-len*: **assumes** $x \leq_b w$
shows $1 < |w|$ **and** $0 < |x|$ **and** $|x| < |w|$
proof−
show $|x| < |w|$
using *assms prefix-length-less* **unfolding** *border-def strict-prefix-def*
by *blast*
show $0 < |x|$
using *assms* **unfolding** *border-def* **by** *blast*
thus $1 < |w|$
using *assms* ‹$|x| < |w|$› **unfolding** *border-def*
by *linarith*
qed

lemma *borders-compare*: **assumes** $x \leq_b w$ **and** $x' \leq_b w$ **and** $|x'| < |x|$
shows $x' \leq_b x$
using *ruler-le*[*OF borderD-pref*[*OF* ‹$x' \leq_b w$›] *borderD-pref*[*OF* ‹$x \leq_b w$›]
less-imp-le-nat[*OF* ‹$|x'| < |x|$›]]
suf-ruler-le[*OF borderD-suf*[*OF* ‹$x' \leq_b w$›] *borderD-suf*[*OF* ‹$x \leq_b w$›] *less-imp-le-nat*[*OF* ‹$|x'| < |x|$›]]
borderD-nemp[*OF* ‹$x' \leq_b w$›] ‹$|x'| < |x|$›
borderI **by** *blast*

lemma *unbordered-border*:
bordered w $\Longrightarrow$ $\exists\ x.\ x \leq_b w \wedge \neg$ *bordered x*
proof (*induction* $|w|$ *arbitrary*: *w rule*: *less-induct*)
case *less*
obtain x' **where** $x' \leq_b w$
using *bordered-def less.prems* **by** *blast*
show *?case*

```
  proof (cases bordered x')
    assume ¬ bordered x'
    thus ?case
      using ‹x' ≤b w› by blast
  next
    assume bordered x'
    from less.hyps[OF border-len(3)[OF ‹x' ≤b w›] this]
    show ?case
      using  border-trans[of - x' w] ‹x' ≤b w› by blast
  qed
qed

lemma unbordered-border-shortest: x ≤b w ⟹ ¬ bordered x ⟹  y ≤b w ⟹ |x|
≤ |y|
  using bordered-def[of x] borders-compare[of x w y] not-le-imp-less[of |x| |y|] by
blast

lemma long-border-bordered: assumes long: |w| < |x| + |x| and border: x ≤b w
  shows (w^<-1>x)^-1>x ≤b x
proof–
  define p s where p = w^<-1>x and s = x^-1>w
  hence eq: p·x = x·s
   using assms unfolding border-def using lq-pref[of x w] rq-suf[of x w]  by simp
  have |p| < |x|
    using lenarg[OF p-def] long unfolding rq-len by linarith
  have px: p · p^-1>x = x and sx: p^-1>x · s = x
    using eqd-pref[OF eq less-imp-le, OF ‹|p| < |x|›] by blast+
  have p^-1>x ≠ ε
    using ‹|p| < |x|› px by fastforce
  have p ≠ ε
    using border-rq-nemp[OF border] p-def
    by presburger
  have p^-1>x ≠ x
    using ‹p ≠ ε› px by force
  have (p^-1>x) ≤b x
    unfolding border-def
   using eqd-pref[OF eq less-imp-le, OF ‹|p| < |x|›] ‹p^-1>x ≠ ε› ‹p^-1>x ≠ x› by
blast
  thus ?thesis
    unfolding p-def.
qed

thm long-border-bordered[reversed]

lemma border-short-dec: assumes border: x ≤b w and short: |x| + |x| ≤ |w|
  shows x · x^-1>(w^<-1>x) · x = w
proof–
  have eq: x·x^-1>w = w^<-1>x·x
    using lq-pref[OF borderD-pref[OF border]] rq-suf[OF borderD-suf[OF border]]
```

by *simp*
have $|x| \leq |w^{<-1}x|$
using *short* **unfolding** *rq-len* **by** *linarith*
from *lq-pref*[*of x w*, *OF borderD-pref*[*OF border*], *folded conjunct2*[*OF eqd-pref*[*OF eq this*]]]
show *?thesis*.
qed

lemma *bordered-dec*: **assumes** *bordered w*
obtains *u v* **where** $u{\cdot}v{\cdot}u = w$ **and** $u \neq \varepsilon$
proof−
obtain *u* **where** $u \leq_b w$ **and** $\neg$ *bordered u*
using *unbordered-border*[*OF assms*] **by** *blast*
have $|u| + |u| \leq |w|$
using *long-border-bordered*[*OF* - ‹$u \leq_b w$›] ‹$\neg$ *bordered u*› *bordered-def leI* **by** *blast*
from *border-short-dec*[*OF* ‹$u \leq_b w$› *this*, *THEN that*, *OF borderD-nemp*[*OF* ‹$u \leq_b w$›]]
show *thesis*.
qed

lemma *emp-not-bordered*: $\neg$ *bordered* ε
by *simp*

lemma *bordered-nemp*: *bordered w* $\Longrightarrow$ $w \neq \varepsilon$
using *emp-not-bordered* **by** *blast*

lemma *sing-not-bordered*: $\neg$ *bordered* [*a*]
using *bordered-dec*[*of* [*a*] *False*] *append-eq-Cons-conv*[*of* - - *a* ε] *suf-nemp* **by** *fast*

2.17.2 Relation to period and conjugation

lemma *border-conjug-eq*: $x \leq_b w \Longrightarrow (w^{<-1}x) \cdot w = w \cdot (x^{-1>}w)$
using *lq-rq-reassoc-suf*[*OF borderD-pref borderD-suf*, *symmetric*] **by** *blast*

lemma *border-per-root*: $x \leq_b w \Longrightarrow w \leq_p (w^{<-1}x) \cdot w$
using *border-conjug-eq* **by** *blast*

lemma *per-root-border*: **assumes** $|r| < |w|$ **and** $r \neq \varepsilon$ **and** $w \leq_p r \cdot w$
shows $r^{-1>}w \leq_b w$
proof
have $|r| \leq |w|$ **and** $r \leq_p w$
using *less-imp-le*[*OF* ‹$|r| < |w|$›] *pref-prod-long*[*OF* ‹$w \leq_p r \cdot w$›] **by** *blast*+
show $r^{-1>}w \leq_p w$
using *pref-lq*[*OF* ‹$w \leq_p r \cdot w$›, *of r*] **unfolding** *lq-triv*.
show $r^{-1>}w \leq_s w$
using ‹$r \leq_p w$› **by** (*auto simp add*: *prefix-def*)
show $r^{-1>}w \neq w$
using ‹$r \leq_p w$› ‹$r \neq \varepsilon$› **unfolding** *prefix-def* **by** *fastforce*

show $r^{-1>}w \neq \varepsilon$
using *lq-pref*[*OF* ‹$r \leq p\ w$›] ‹$|r| < |w|$› **by** *force*
qed

lemma *pref-suf-neq-per*: **assumes** $x \leq p\ w$ **and** $x \leq s\ w$ **and** $x \neq w$ **shows** *period* $w\ (|w|-|x|)$
proof–
have $(w^{<-1}x)\cdot x = w$
using *rq-suf*[*OF* ‹$x \leq s\ w$›].
have $x\cdot(x^{-1>}w) = w$
using *lq-pref*[*OF* ‹$x \leq p\ w$›].
have *take*: $w^{<-1}x = take\ (|w|-|x|)\ w$
using *rq-take*.
have *nemp*: *take* $(|w|-|x|)\ w \neq \varepsilon$
using ‹$x \leq p\ w$› ‹$x \neq w$› **unfolding** *prefix-def* **by** *force*
have $w \leq p\ take\ (|w|-|x|)\ w \cdot w$
using *triv-pref*[*of* $w\ x^{-1>}w$]
unfolding *lassoc*[*of* $w^{<-1}x\ x\ x^{-1>}w$, *unfolded* ‹$x \cdot x^{-1>}w = w$› ‹$w^{<-1}x \cdot x = w$›, *symmetric*] *take*.
thus *period* $w\ (|w|-|x|)$
unfolding *period-def* **using** *nemp* **by** *blast*
qed

lemma *border-per*: $x \leq b\ w \Longrightarrow$ *period* $w\ (|w|-|x|)$
unfolding *border-def* **using** *pref-suf-neq-per* **by** *blast*

lemma *per-border*: **assumes** $n < |w|$ **and** $n \neq 0$ **and** *period* $w\ n$
shows *take* $(|w| - n)\ w \leq b\ w$
proof–
have *eq*: *take* $(|w| - n)\ w = drop\ n\ w$
using ‹*period* $w\ n$› *length-drop per-shift* *pref-take-conv* **by** *metis*
have *take* $(|w| - n)\ w \neq \varepsilon$
using ‹$n < |w|$› *take-eq-Nil* **by** *fastforce*
moreover have *take* $(|w| - n)\ w \neq w$
using ‹$n < |w|$› ‹$n \neq 0$› **unfolding** *take-all-iff*[*of* $|w|-n\ w$] **by** *fastforce*
ultimately show *?thesis*
unfolding *border-def* **using** *take-is-prefix*[*of* $|w|-n\ w$] *suffix-drop*[*of* $n\ w$, *folded* *eq*] **by** *blast*
qed

2.18 The longest border and the shortest period

2.18.1 The longest border

definition *max-borderP* :: *'a list* ⇒ *'a list* ⇒ *bool* **where**
max-borderP $u\ w = (u \leq p\ w \wedge u \leq s\ w \wedge (u = w \longrightarrow w = \varepsilon) \wedge (\forall\ v.\ v \leq b\ w \longrightarrow\ v \leq p\ u))$

lemma *max-borderP-emp-emp*: *max-borderP* $\varepsilon\ \varepsilon$

unfolding *max-borderP-def* **by** *simp*

lemma *max-borderP-exE*: **obtains** u **where** *max-borderP* u w
proof−
define P **where** $P = (\lambda\ x.\ x \leq p\ w \wedge x \leq s\ w \wedge (x = w \longrightarrow w = \varepsilon))$
have $P\ \varepsilon$
unfolding *P-def* **by** *blast*
obtain v **where** $v \leq p\ w$ **and** $P\ v$ **and** $(\bigwedge y.\ y \leq p\ w \Longrightarrow P\ y \Longrightarrow y \leq p\ v)$
using *max-pref*[*of* ε w P *thesis*, *OF prefix-bot.extremum* ‹$P\ \varepsilon$›] **by** *blast*
hence *max-borderP* v w
unfolding *max-borderP-def border-def P-def* **by** *presburger*
from *that*[*OF this*]
show *thesis*.
qed

lemma *max-borderP-of-nemp*: *max-borderP* $u\ \varepsilon \Longrightarrow u = \varepsilon$
by (*metis max-borderP-def suffix-bot.extremum-unique*)

lemma *max-borderP-D-neq*: $w \neq \varepsilon \Longrightarrow$ *max-borderP* $u\ w \Longrightarrow u \neq w$
by (*simp add*: *max-borderP-def*)

lemma *max-borderP-D-pref*: *max-borderP* $u\ w \Longrightarrow u \leq p\ w$
by (*simp add*: *max-borderP-def*)

lemma *max-borderP-D-suf*: *max-borderP* $u\ w \Longrightarrow u \leq s\ w$
by (*simp add*: *max-borderP-def*)

lemma *max-borderP-D-max*: *max-borderP* $u\ w \Longrightarrow v \leq b\ w \Longrightarrow\ v \leq p\ u$
by (*simp add*: *max-borderP-def*)

lemma *max-borderP-D-max′*: *max-borderP* $u\ w \Longrightarrow v \leq b\ w \Longrightarrow v \leq s\ u$
unfolding *max-borderP-def* **using** *borderD-suf suf-pref-eq suffix-same-cases* **by** *metis*

lemma *unbordered-max-border-emp*: ¬ *bordered* $w \Longrightarrow$ *max-borderP* $u\ w \Longrightarrow u = \varepsilon$
unfolding *max-borderP-def bordered-def border-def* **by** *blast*

lemma *bordered-max-border-nemp*: *bordered* $w \Longrightarrow$ *max-borderP* $u\ w \Longrightarrow u \neq \varepsilon$
unfolding *max-borderP-def bordered-def border-def* **using** *prefix-Nil* **by** *blast*

lemma *max-borderP-border*: *max-borderP* $u\ w \Longrightarrow u \neq \varepsilon \Longrightarrow u \leq b\ w$
unfolding *max-borderP-def border-def* **by** *blast*

lemma *max-borderP-rev*: *max-borderP* (*rev* u) (*rev* w) $\Longrightarrow$ *max-borderP* $u\ w$
proof−
assume *max-borderP* (*rev* u) (*rev* w)
from *this*[*unfolded max-borderP-def rev-is-rev-conv*, *folded pref-rev-suf-iff suf-rev-pref-iff*]
have $u = w \longrightarrow w = \varepsilon$ **and** $u \leq p\ w$ **and** $u \leq s\ w$ **and** *allv*: $v \leq b$ *rev* $w \Longrightarrow v$

$\leq p$ *rev u* **for** *v*
by *blast+*
show *max-borderP u w*
proof (*unfold max-borderP-def*, *intro conjI*, *simp-all only*: ‹$u \leq p$ w› ‹$u \leq s$ w›)
show $u = w \longrightarrow w = \varepsilon$ **by** *fact*
show $\forall v.\ v \leq b\ w \longrightarrow v \leq p\ u$
proof (*rule allI*, *rule impI*)
fix *v* **assume** $v \leq b\ w$
show $v \leq p\ u$
using ‹*max-borderP* (*rev u*) (*rev w*)› ‹$v \leq b\ w$› *border-rev-conv max-borderP-D-max′ pref-rev-suf-iff* **by** *metis*
qed
qed
qed

lemma *max-borderP-rev-conv*: *max-borderP* (*rev u*) (*rev w*) $\longleftrightarrow$ *max-borderP u w*
using *max-borderP-rev max-borderP-rev*[*of rev u rev w*, *unfolded rev-rev-ident*]
by *blast*

definition *max-border* :: *'a list* $\Rightarrow$ *'a list* **where**
max-border w = (*THE u*. (*max-borderP u w*))

lemma *max-border-unique*: **assumes** *max-borderP u w max-borderP v w*
shows *u* = *v*
using *max-borderP-D-max*[*OF* ‹*max-borderP u w*›, *OF max-borderP-border*[*OF* ‹*max-borderP v w*›]]
max-borderP-D-max[*OF* ‹*max-borderP v w*›, *OF max-borderP-border*[*OF* ‹*max-borderP u w*›]]
by *force*

lemma *max-border-ex*: *max-borderP* (*max-border w*) *w*
proof (*rule max-borderP-exE*[*of w*])
fix *u* **assume** *max-borderP u w*
with *max-border-unique*[*OF this*]
show *?thesis*
unfolding *max-border-def*
by (*elim theI*[*of* λ *x*. *max-borderP x w*]) *simp*
qed

lemma *max-borderP-max-border*: *max-borderP u w* $\Longrightarrow$ *max-border w* = *u*
using *max-border-unique*[*OF max-border-ex*].

lemma *max-border-len-rev*: |*max-border u*| = |*max-border* (*rev u*)|
by (*cases u* = ε, *simp*, *metis length-rev max-borderP-max-border max-borderP-rev-conv max-border-ex*)

lemma *max-border-border*: **assumes** *bordered w* **shows** *max-border w* $\leq b$ *w*
using *max-border-ex bordered-max-border-nemp*[*OF assms*, *of max-border w*]
unfolding *max-borderP-def border-def* **by** *metis*

theorem *max-border-border'*: *max-border w* $\neq \varepsilon \Longrightarrow$ *max-border w* $\leq_b$ *w*
using *max-borderP-border max-border-ex* **by** *blast*

lemma *max-border-sing-emp*: *max-border* [*a*] = ε
using *max-border-ex*[*THEN unbordered-max-border-emp*[*OF sing-not-bordered*]]
by *fast*

lemma *max-border-suf*: *max-border w* $\leq_s$ *w*
using *max-borderP-D-suf max-border-ex* **by** *auto*

lemma *max-border-nemp-neq*: *w* $\neq \varepsilon \Longrightarrow$ *max-border w* $\neq$ *w*
by (*simp add*: *max-borderP-D-neq max-border-ex*)

lemma *max-borderI*: **assumes** *u* $\neq$ *w* **and** *u* $\leq_p$ *w* **and** *u* $\leq_s$ *w* **and** $\forall$ *v*. *v* $\leq_b$ *w* $\longrightarrow$ *v* $\leq_p$ *u*
shows *max-border w* = *u*
using *assms max-border-ex*
by (*intro max-borderP-max-border*, *unfold max-borderP-def border-def*, *blast*)

lemma *max-border-less-len*: **assumes** *w* $\neq \varepsilon$ **shows** |*max-border w*| < |*w*|
using *assms border-len*(*3*) *leI list.size*(*3*) *max-border-border' npos-len* **by** *metis*

theorem *max-border-max-pref*: **assumes** *u* $\leq_b$ *w* **shows** *u* $\leq_p$ *max-border w*
using *max-borderP-D-max*[*OF max-border-ex* ‹*u* $\leq_b$ *w*›]**.**

theorem *max-border-max-suf*: **assumes** *u* $\leq_b$ *w* **shows** *u* $\leq_s$ *max-border w*
using *max-borderP-D-max'*[*OF max-border-ex* ‹*u* $\leq_b$ *w*›]**.**

lemma *bordered-max-bord-nemp-conv*[*code*]: *bordered w* $\longleftrightarrow$ *max-border w* $\neq \varepsilon$
using *bordered-max-border-nemp max-border-ex unbordered-max-border-emp* **by** *fast*

lemma *max-bord-take*: *max-border w* = *take* |*max-border w*| *w*
proof (*cases bordered w*)
assume *bordered w*
from *borderD-pref*[*OF max-border-border*[*OF this*]]
show *max-border w* = *take* |*max-border w*| *w*
by (*simp add*: *pref-take*)
next
assume ¬ *bordered w*
hence *max-border w* = ε
using *bordered-max-bord-nemp-conv* **by** *blast*
thus *max-border w* = *take* |*max-border w*| *w*
by *simp*
qed

2.18.2 The shortest period

definition *min-period-root* :: *'a list* ⇒ *'a list* (*π*) **where**
min-period-root w = *take* (*LEAST n*. *0* < *n* ∧ *period w n*) *w*

definition *min-period* :: *'a list* ⇒ *nat* **where**
min-period w = |*π w*|

lemma *min-per-emp*[*simp*]: *π ε* = *ε*
unfolding *min-period-root-def* **by** *simp*

lemma *min-per-zero*[*simp*]: *min-period ε* = *0*
by (*simp add*: *min-period-def*)

lemma *min-per-per*: *period w* (*min-period w*) **and**
min-per-pos: *w* ≠ *ε* ⟹ *0* < *min-period w*
proof (*cases w* = *ε*, *simp*)
assume *w* ≠ *ε*
let *?P* = *λ x*. *0* < *x* ∧ *period w x*
have *ex*: *?P* |*w*|
using *len-is-per*[*of w*] ‹*w* ≠ *ε*› **by** *auto*
have *le*: (*LEAST x*. *0* < *x* ∧ *period w x*) ≤ |*w*|
using *Least-le*[*of ?P*, *OF ex*]**.**
have *len*: |*take* (*LEAST x*. *0* < *x* ∧ *period w x*) *w*| = (*LEAST x*. *0* < *x* ∧ *period w x*)
using *take-len*[*OF le*]**.**
show *period w* (*min-period w*) *0* < *min-period w*
using *LeastI-ex*[*OF exI*, *of ?P*, *OF ex*]
unfolding *min-period-def min-period-root-def len* **by** *blast+*
qed

lemma *n* ≤ |*w*| ⟹ |*take n w*| = *n*
using *take-len***.**

lemma *u* ≤*p w* ⟹ *take* |*u*| *w* = *u*
using *pref-take***.**

lemma *min-per-root-take-eq* : *take* |*π w*| *w* = *π w*
by (*rule pref-take*, *unfold min-period-root-def*) (*rule take-is-prefix*)

lemma *min-per-len*: *min-period w* ≤ |*w*|
unfolding *min-period-def min-period-root-def* **by** *simp*

lemmas *min-per-root-len* = *min-per-len*[*unfolded min-period-def*]

lemma *min-per-sing*: *min-period* [*a*] = *1*
using *min-per-pos*[*of* [*a*]] *min-per-len*[*of* [*a*]] **by** *simp*

lemma *min-per-root-per*: **assumes** *w* ≠ *ε* **shows** *w* ≤*p* (*π w*) · *w*
using *min-per-per*[*of w*] **unfolding** *period-def min-period-def*

min-per-root-take-eq.

lemma *min-per-root-per-root*: **assumes** $w \neq \varepsilon$ **shows** $w <p (\pi\ w) \cdot w$
by (*rule sprefI*[*OF min-per-root-per*[*OF assms*]])
(*use min-per-pos*[*OF assms*, *unfolded min-period-def*] **in** *blast*)

lemma *min-per-pref*: $\pi\ w \leq p\ w$
unfolding *min-period-root-def* **using** *take-is-prefix* **by** *blast*

lemma *min-per-nemp*: $w \neq \varepsilon \Longrightarrow \pi\ w \neq \varepsilon$
using *min-per-root-per-root* **by** *blast*

lemma *min-per-min*: **assumes** $w <p r \cdot w$ **shows** $\pi\ w \leq p\ r$
proof (*cases* $w = \varepsilon$)
assume $w \neq \varepsilon$
have *period w* $|\pi\ w|$
using *sprefD1*[*OF min-per-root-per-root*[*OF* ‹$w \neq \varepsilon$›]] **unfolding** *period-def min-per-root-take-eq*.
have $0 < |r| \wedge$ *period w* $|r|$
using *periodI-pref'*[*OF sprefD1*[*OF assms*]] *assms* **by** *fast*
from *Least-le*[*of* $\lambda\ n.\ 0 < n \wedge$ *period w n*, *OF this*]
have $|\pi\ w| \leq |r|$
unfolding *min-period-root-def* **using** *dual-order.trans len-take1* **by** *metis*
with *pref-trans*[*OF min-per-pref sprefD1*[*OF* ‹$w <p r \cdot w$›]]
show $\pi\ w \leq p\ r$
using *pref-prod-le* **by** *blast*
qed *simp*

lemma *lq-min-per-pref*: $\pi\ w^{-1>} w \leq p\ w$
unfolding *same-prefix-prefix*[*of* $\pi\ w$ - *w*, *symmetric*] *lq-pref*[*OF min-per-pref*]
using *sprefD1*[*OF min-per-root-per-root*]
by (*cases* $w = \varepsilon$, *simp*)

lemma *max-bord-emp*: *max-border* $\varepsilon = \varepsilon$
by (*simp add*: *max-borderP-of-nemp max-border-ex*)

theorem *min-per-max-border*: $\pi\ w \cdot$ *max-border* $w = w$
proof (*cases* $w = \varepsilon$)
assume $w \neq \varepsilon$
have *max-border* $w = (\pi\ w)^{-1>} w$
proof (*intro max-borderI*)
show $\pi\ w^{-1>} w \neq w$
using *min-per-nemp*[*OF* ‹$w \neq \varepsilon$›] *lq-pref*[*OF min-per-pref*] *append-self-conv2*
by *metis*
show $\pi\ w^{-1>} w \leq s\ w$
by *force*
show $\pi\ w^{-1>} w \leq p\ w$
using *lq-min-per-pref* **by** *blast*
show $\forall\, v.\ v \leq b\ w \longrightarrow v \leq p\ \pi\ w^{-1>} w$

proof (*rule allI*, *rule impI*)
fix v **assume** $v \leq_b w$
have $w <_p (w^{<-1}v) \cdot w$
using *per-border* ‹$v \leq_b w$› *border-per-root*[*OF* ‹$v \leq_b w$›] *border-rq-nemp*[*OF* ‹$v \leq_b w$›] **by** *blast*
from *min-per-min*[*OF this*]
have $\pi\ w \leq_p w^{<-1}v$**.**
from *pref-rq-suf-lq*[*OF borderD-suf*[*OF* ‹$v \leq_b w$›] *this*]
have $v \leq_s \pi\ w^{-1>}w$**.**
from *suf-pref-eq*[*OF this*] *ruler*[*OF borderD-pref*[*OF* ‹$v \leq_b w$›] ‹$\pi\ w^{-1>}w \leq_p w$›]
show $v \leq_p \pi\ w^{-1>}w$
by *blast*
qed
qed
thus *?thesis*
using *lq-pref*[*OF min-per-pref*, *of w*] **by** *simp*
qed (*simp add*: *max-bord-emp*)

lemma *min-per-len-diff*: *min-period* $w = |w| - |$*max-border* $w|$
unfolding *min-period-def* **using** *lenarg*[*OF min-per-max-border*,*unfolded len-morph*,*of w*] **by** *linarith*

lemma *min-per-root-take* [*code*]: $\pi\ w =$ *take* $(|w| - |$*max-border* $w|)\ w$
using *cancel-right* *min-per-max-border suffix-take*[*OF max-border-suf*] **by** *metis*

2.19 Primitive words

If a word w is not a non-trivial power of some other word, we say it is primitive.

definition *primitive* :: *'a list* ⇒ *bool*
where *primitive* $u = (\forall\ r\ k.\ r^{@}k = u \longrightarrow k = 1)$

lemma *emp-not-prim*[*simp*]: ¬ *primitive* ε
unfolding *primitive-def*
by (*metis pow-list-eq-if zero-neq-one*)

lemma *primI*[*intro*]: $(\bigwedge\ r\ k.\ r^{@}k = u \Longrightarrow k = 1) \Longrightarrow$ *primitive* u
by (*simp add*: *primitive-def*)

lemma *prim-nemp*: *primitive* $u \Longrightarrow u \neq \varepsilon$
by *force*

lemma *prim-exp-one*: *primitive* $u \Longrightarrow r^{@}k = u \Longrightarrow k = 1$
using *primitive-def* **by** *blast*

lemma *pow-nemp-imprim*[*intro*]: $2 \leq k \implies$ ¬ *primitive* $(u^{@}k)$
using *prim-exp-one* **by** *fastforce*

lemma *pow-not-prim*: ¬ *primitive* ($u^{@}Suc(Suc\ k)$)
using *prim-exp-one* **by** *fastforce*

lemma *pow-non-prim*: $k \neq 1 \Longrightarrow$ ¬ *primitive* ($w^{@}k$)
using *prim-exp-one*
by *auto*

lemma *prim-exp-eq*: *primitive* $u \Longrightarrow r^{@}k = u \Longrightarrow u = r$
using *prim-exp-one pow-list-one* **by** *force*

lemma *prim-per-div*: **assumes** *primitive* v **and** $n \neq 0$ **and** $n \leq |v|$ **and** *period* v ($gcd\ |v|\ n$)
shows $n = |v|$
proof−
have $gcd\ |v|\ n\ dvd\ |v|$
by *simp*
from *prim-exp-eq*[*OF* ‹*primitive v*› *per-div*[*OF this* ‹*period v* (*gcd* |*v*| *n*)›]]
have $gcd\ |v|\ n = |v|$
using *take-len*[*OF le-trans*[*OF gcd-le2-nat*[*OF* ‹$n \neq 0$›] ‹$n \leq |v|$›], *of* |*v*|] **by** *presburger*
from *gcd-le2-nat*[*OF* ‹$n \neq 0$›, *of* |*v*|, *unfolded this*] ‹$n \leq |v|$›
show $n = |v|$ **by** *force*
qed

lemma *prim-comm-exp*[*elim*]: **assumes** *primitive* r **and** $u \cdot r = r \cdot u$
obtains k **where** $r^{@}k = u$
using *commE*[*OF* ‹$u \cdot r = r \cdot u$›] *prim-exp-eq*[*OF* ‹*primitive r*›] **by** *metis*

lemma *set-singleton-iff*: *card* (*set u*) = *1* ⟷ *set u* = {*hd u*}
unfolding *One-nat-def card-1-singleton-iff*
by (*cases u* = ε, *force*, *standard*)
(*metis hd-in-set singleton-iff*, *blast*)

lemma *set-empty-iff*: *card* (*set u*) = *0* ⟷ *set u* = {}
by *force*

lemma *set-large-iff*: *1* < *card* (*set u*) ⟷ ¬ *set u* ⊆ {*hd u*}
unfolding *subset-singleton-iff de-Morgan-disj*
set-empty-iff[*symmetric*] *set-singleton-iff*[*symmetric*] **by** *linarith*

lemma *prim-card-set*: **assumes** *primitive u* **and** $|u| \neq 1$ **shows** *1* < *card* (*set u*)
unfolding *set-large-iff* **using** ‹*primitive u*› *pow-non-prim*[*OF* ‹$|u| \neq 1$›, *of* [*hd u*]]
using *sing-set-word*[*of u hd u*] **by** *fastforce*

lemma *comm-not-prim*: **assumes** $u \neq \varepsilon$ $v \neq \varepsilon$ $u \cdot v = v \cdot u$ **shows** ¬ *primitive*

$(u \cdot v)$
proof–
obtain $t\ k\ m$ **where** $u = t^{@}k \quad v = t^{@}m$
using ‹$u{\cdot}v = v{\cdot}u$›[*unfolded comm*] **by** *blast*
show *?thesis* **using** *pow-non-prim*[*of k+m t*]
unfolding *pow-add*[*of k m t*]
using ‹$u = t\ ^{@}\ k$› ‹$v = t\ ^{@}\ m$› *assms*(*1,2*) **by** *fastforce*
qed

lemma *prim-rotate-conv*: *primitive* $w \longleftrightarrow$ *primitive* (*rotate n w*)
proof
assume *primitive w* **show** *primitive* (*rotate n w*)
proof (*rule primI*)
fix *r k* **assume** $r^{@}k =$ *rotate n w*
obtain *l* **where** (*rotate l r*)$^{@}k = w$
using *rotate-backE*[*of n w*, *folded* ‹$r^{@}k =$ *rotate n w*›, *unfolded rotate-pow-list-swap*]
by *blast*
from *prim-exp-one*[*OF* ‹*primitive w*› *this*]
show $k = 1$**.**
qed
next
assume *primitive* (*rotate n w*) **show** *primitive w*
proof (*rule primI*)
fix *r k* **assume** $r^{@}k = w$
from *prim-exp-one*[*OF* ‹*primitive* (*rotate n w*)›, *OF rotate-pow-list-swap*[*of n k r*, *unfolded this*, *symmetric*]]
show $k = 1$**.**
qed
qed

lemma *non-prim*: **assumes** ¬ *primitive w* **and** $w \neq \varepsilon$
obtains *r k* **where** $r \neq \varepsilon$ **and** $1 < k$ **and** $r^{@}k = w$ **and** $w \neq r$
proof–
from ‹¬ *primitive w*›[*unfolded primitive-def*]
obtain *r k* **where** $k \neq 1$ **and** $r^{@}k = w$ **by** *blast*
have $r \neq \varepsilon$
using ‹$w \neq \varepsilon$› ‹$r^{@}k = w$› **by** *blast*
have $k \neq 0$
using ‹$w \neq \varepsilon$› ‹$r^{@}k = w$› *pow-zero*[*of r*] **by** *meson*
have $w \neq r$
using ‹$k \neq 1$›[*folded eq-pow-exp*[*OF* ‹$r \neq \varepsilon$›, *of k 1*, *unfolded* ‹$r\ ^{@}\ k = w$›]] **by** *simp*
show *thesis*
using *that*[*OF* ‹$r \neq \varepsilon$› - ‹$r^{@}k = w$› ‹$w \neq r$›] ‹$k \neq 0$› ‹$k \neq 1$› *less-linear* **by** *blast*
qed

lemma *prim-no-rotate*: **assumes** *primitive w* **and** $0 < n$ **and** $n < |w|$
shows *rotate n w* $\neq w$

proof
assume *rotate n w = w*
have *take n w · drop n w = drop n w · take n w*
using *rotate-append*[*of take n w drop n w*]
unfolding *take-len*[*OF less-imp-le-nat*[*OF ‹n < |w|›*]] *append-take-drop-id ‹rotate n w = w›*.
have *take n w ≠ ε drop n w ≠ ε*
using *‹0 < n› ‹n < |w|›* **by** *auto+*
from *‹primitive w›* **show** *False*
using *comm-not-prim*[*OF ‹take n w ≠ ε› ‹drop n w ≠ ε› ‹take n w · drop n w = drop n w · take n w›, unfolded append-take-drop-id*]
by *simp*
qed

lemma *no-rotate-prim*: **assumes** *w ≠ ε* **and** *⋀ n. 0 < n ⟹ n < |w| ⟹ rotate n w ≠ w*
shows *primitive w*
proof (*rule ccontr*)
assume *¬ primitive w*
from *non-prim*[*OF this ‹w ≠ ε›*]
obtain *r l* **where** *r ≠ ε* **and** *1 < l* **and** *r*[@]*l = w* **and** *w ≠ r* **by** *blast*
have *rotate |r| w = w*
using *rotate-root-self*[*of r l, unfolded ‹r*[@]*l = w›*].
moreover have *0 < |r|*
by (*simp add*: *‹r ≠ ε›*)
moreover have *|r| < |w|*
unfolding *pow-len*[*of l r, unfolded ‹r*[@]*l = w›*] **using** *‹1 < l› ‹0 < |r|›* **by** *auto*
ultimately show *False*
using *assms*(*2*) **by** *blast*
qed

corollary *prim-iff-rotate*: **assumes** *w ≠ ε* **shows**
primitive w ⟷ (∀ n. 0 < n ∧ n < |w| ⟶ rotate n w ≠ w)
using *no-rotate-prim*[*OF ‹w ≠ ε›*] *prim-no-rotate* **by** *metis*

lemma *prim-sing*: *primitive* [*a*]
using *prim-iff-rotate*[*of* [*a*]] **by** *fastforce*

lemma *sing-pow-conv* [*simp*]: *[u] = t*[@]*k ⟷ t = [u] ∧ k = 1*
using *pow-non-prim pow-list-1 prim-sing* **by** *metis*

lemma *prim-rev-iff*[*reversal-rule*]: *primitive* (*rev u*) *⟷ primitive u*
unfolding *primitive-def*[*reversed*] **using** *primitive-def*..

lemma *prim-map-prim*: *primitive* (*map f ws*) *⟹ primitive ws*
unfolding *primitive-def* **using** *map-pow-list* **by** *metis*

lemma *inj-map-prim*: **assumes** *inj-on f A* **and** *u ∈ lists A* **and**

primitive u
shows *primitive* (*map f u*)
using *prim-map-prim*[*of the-inv-into A f map f u*, *folded inj-map-inv*[*OF assms*(*1–2*)], *OF assms*(*3*)]**.**

lemma *prim-map-iff* [*reversal-rule*]:
assumes *inj f* **shows** *primitive* (*map f ws*) = *primitive* (*ws*)
using *inj-map-prim*[*of - UNIV*, *unfolded lists-UNIV*, *OF* ‹*inj f*› *UNIV-I*]
prim-map-prim **by** (*intro iffI*)

lemma *prim-concat-prim*: *primitive* (*concat ws*) ⟹ *primitive ws*
unfolding *primitive-def* **using** *concat-pow-list* **by** *metis*

lemma *eq-append-not-prim*: *x* = *y* ⟹ ¬ *primitive* (*x* · *y*)
by (*metis append-Nil2 comm-not-prim prim-nemp*)

2.20 Primitive root

Given a non-empty word w which is not primitive, it is natural to look for the shortest u such that $w = u^k$. Such a word is primitive, and it is the primitive root of w.

definition *primitive-root* :: *'a list* ⇒ *'a list* (ϱ) **where**
primitive-root x = (*if x* ≠ ε *then* (*THE r*. *primitive r* ∧ (∃ *k*. *x* = *r*[@]*k*)) *else* ε)

definition *primitive-root-exp* :: *'a list* ⇒ *nat* (e_ϱ) **where**
primitive-root-exp x = (*if x* = ε *then 0 else* (*THE k*. *x* = (ϱ *x*)[@]*k*))

lemma *primroot-emp*[*simp*]: ϱ ε = ε
unfolding *primitive-root-def* **by** *simp*

lemma *comm-prim*: **assumes** *primitive r* **and** *primitive s* **and** *r*·*s* = *s*·*r*
shows *r* = *s*
using ‹*r*·*s* = *s*·*r*›[*unfolded comm*] *assms*[*unfolded primitive-def*, *rule-format*] **by** *metis*

lemma *primroot-ex*: **assumes** *x* ≠ ε **shows** ∃ *r k*. *primitive r* ∧ *k* ≠ *0* ∧ *x* = *r*[@]*k*
using ‹*x* ≠ ε›
proof(*induction* |*x*| *arbitrary*: *x rule*: *less-induct*)
case *less*
then show ∃ *r k*. *primitive r* ∧ *k* ≠ *0* ∧ *x* = *r*[@]*k*
proof (*cases primitive x*)
assume ¬ *primitive x*
from *non-prim*[*OF this* ‹*x* ≠ ε›]
obtain *r l* **where** *r* ≠ ε **and** *1* < *l* **and** *r*[@]*l* = *x* **and** *x* ≠ *r*
by *blast*
have |*r*| < |*x*|
unfolding *lenarg*[*OF* ‹*r*[@]*l* = *x*›, *unfolded pow-len*, *symmetric*]
using *nemp-len*[*OF* ‹*r* ≠ ε›] ‹*1* < *l*› **by** *force*

from *less.hyps*[*OF this* ‹$r \neq \varepsilon$›]
obtain k pr **where** *primitive pr* $k \neq 0$ $r = pr^{@}k$
by *blast*
have $k*l \neq 0$
using ‹$1 < l$› ‹$k \neq 0$› **by** *force*
have $x = pr^{@}(k*l)$
using *pow-mult*[*of k l pr*, *folded* ‹$r = pr^{@}k$›, *unfolded* ‹$r^{@}l = x$›, *symmetric*]**.**
thus $\exists\, r\ k.$ *primitive* $r \wedge k \neq 0 \wedge x = r\ ^{@}\ k$
using ‹*primitive pr*› ‹$k*l \neq 0$› **by** *fast*
next
assume *primitive x*
have $x = x^{@}Suc\ 0$
by *simp*
thus $\exists\ r\ k.$ *primitive* $r \wedge k \neq 0 \wedge x = r^{@}k$
using ‹*primitive x*› **by** *force*
qed
qed

lemma *primroot-exE*: **assumes**$x \neq \varepsilon$
obtains r k **where** *primitive* r **and** $0 < k$ **and** $x = r^{@}k$
using *assms* *primroot-ex*[*OF* ‹ $x \neq \varepsilon$›] **by** *blast*

Uniqueness of the primitive root follows from the following lemma

lemma *primroot-unique*: **assumes** $u \neq \varepsilon$ **and** *primitive* r **and** $u = r^{@}k$ **shows** $\varrho\ u = r$
proof–
have $0 < k$
using ‹$u \neq \varepsilon$› ‹$u = r^{@}k$› **by** *blast*
have $s = r$ **if** *primitive* s **and** $u = s^{@}l$ **for** s l
proof–
from *pow-list-comm-comm*[*OF* ‹$0 < k$› ‹$u = s^{@}l$›[*unfolded* ‹$u = r^{@}k$›]]
obtain t es er **where** $t^{@}es = s$ **and** $t^{@}er = r$
by *fast*
from *prim-exp-eq*[*OF* ‹*primitive r*› ‹$t^{@}er = r$›] *prim-exp-eq*[*OF* ‹*primitive s*› ‹$t^{@}es = s$›]
show $s = r$
by *simp*
qed
hence *primitive* $s \wedge (\exists\, k.\ u = s\ ^{@}\ k) \Longrightarrow s = r$ **for** s
by *presburger*
from *the-equality*[*of* $\lambda\ r.$ *primitive* $r \wedge (\exists\, k.\ u = r\ ^{@}\ k)$ r, *OF - this*]
show $\varrho\ u = r$
using ‹*primitive r*› ‹$u = r^{@}k$› **unfolding** *primitive-root-def if-P*[*OF* ‹$u \neq \varepsilon$›]
by *blast*
qed

lemma *primroot-unique'*: **assumes** $0 < k$ *primitive* r **and** $u = r^{@}k$ **shows** $\varrho\ u = r$
using *primroot-unique*[*OF - assms*(*2*,*3*)] **using** *prim-nemp*[*OF* ‹*primitive r*›] ‹*0*

$< k\rangle$ **unfolding** $\langle u = r^{@}k\rangle$
using *pow-list-Nil-iff-Nil* **by** *blast*

lemma *prim-primroot*[*intro*]: **assumes** *primitive x* **shows** $\varrho\ x = x$
using *assms emp-not-prim primroot-unique pow-list-1* **by** *metis*

lemma *primroot-exp-unique*: **assumes** $u \neq \varepsilon$ **and** $(\varrho\ u)^{@}k = u$ **shows** $e_\varrho\ u = k$
unfolding *primitive-root-exp-def if-not-P*[*OF* $\langle u \neq \varepsilon\rangle$]
proof (*rule the-equality*)
show $u = (\varrho\ u)^{@}k$ **using** $\langle(\varrho\ u)^{@}k = u\rangle$[*symmetric*].
have $\varrho\ u \neq \varepsilon$
using *assms* **by** *force*
show *ka* $= k$ **if** $u = \varrho\ u\ ^{@}\ ka$ **for** *ka*
using *eq-pow-exp*[*OF* $\langle\varrho\ u \neq \varepsilon\rangle$, *of k ka*, *folded* $\langle u = (\varrho\ u)^{@}k\rangle$ *that*] **by** *blast*
qed

lemma *primroot-prim*[*intro*]: $x \neq \varepsilon \Longrightarrow$ *primitive* $(\varrho\ x)$
using *primroot-unique primroot-ex* **by** *metis*

Existence and uniqueness of the primitive root justifies the function ϱ: it indeed yields the primitive root of a nonempty word.

lemma *primroot-expE*: **obtains** *k* **where** $(\varrho\ x)^{@}k = x$ **and** $0 < k$
proof (*cases* $x = \varepsilon$)
assume $x \neq \varepsilon$
with *that primroot-exE*[*OF this*]
show *thesis*
using *primroot-unique′* **by** *metis*
qed *auto*

lemma *primroot-exp-eq* [*simp*]: $(\varrho\ u)^{@}(e_\varrho\ u) = u$
using *primroot-expE*[*of u* $\varrho\ u\ ^{@}\ e_\varrho\ u = u$] *primroot-exp-unique pow-list-zero primitive-root-exp-def* **by** *metis*

lemma *prim-pow-exp-one*: *primitive* $(u^{@}i) \Longrightarrow i = 1$
unfolding *primitive-def* **by** *blast*

lemma *prim-exp-emp*[*simp*]: $e_\varrho\ \varepsilon = 0$
unfolding *primitive-root-exp-def* **by** *simp*

lemma *prim-exp-zero-iff*[*simp*]: $e_\varrho\ x = 0 \longleftrightarrow x = \varepsilon$
using *list-pow-emp*[*of* $\varrho\ x\ e_\varrho\ x$] *prim-exp-emp* **unfolding** *primroot-exp-eq*[*of x*]
by *blast*

lemma *prim-exp-one-iff*: *primitive x* $\longleftrightarrow e_\varrho\ x = 1$
proof
show *primitive x* $\Longrightarrow e_\varrho\ x = 1$
using *primitive-def primroot-exp-eq*[*of x*] **by** *blast*

show *primitive x* **if** $e_\varrho\ x = 1$
by (*cases* $x = \varepsilon$, *use that* **in** *force*)
(*use primroot-exp-eq*[*of x*, *unfolded that*]
pow-list-1[*of* $\varrho\ x$] *primroot-prim*[*of x*] **in** *presburger*)
qed

lemma *nemp-not-prim-exp*: $x \neq \varepsilon \Longrightarrow \neg$ *primitive* $x \longleftrightarrow 1 < e_\varrho\ x$
unfolding *prim-exp-zero-iff*[*symmetric*] *prim-exp-one-iff* **by** *fastforce*

lemma *not-prim-pref-primroots*: $\neg$ *primitive* $x \longleftrightarrow \varrho\ x \cdot \varrho\ x \leq p\ x$
proof (*cases* $x = \varepsilon$)
assume $x \neq \varepsilon$
show *?thesis*
unfolding *nemp-not-prim-exp*[*OF* ‹$x \neq \varepsilon$›]
using *le-exps-pref*[*of 2* $e_\varrho\ x\ \varrho\ x$, *unfolded primroot-exp-eq pow-list-2*]
‹$(\neg$ *primitive* $x) = (1 < e_\varrho\ x)$› *prim-primroot* **by** *fastforce*
qed *simp*

lemma *zero-prim-exp-eq*[*intro*]: $n = 0 \Longrightarrow e_\varrho(r^{@}n) = n$
using *primitive-root-exp-def pow-list-zero* **by** *metis*

lemma *primroot-exp-len*:
shows $e_\varrho\ w * |\varrho\ w| = |w|$
using *lenarg*[*OF primroot-exp-eq*] **unfolding** *pow-len*.

lemma *primroot-exp-nemp* [*intro*]: $u \neq \varepsilon \Longrightarrow 0 < e_\varrho\ u$
using *primroot-expE*[*of u*] *primroot-exp-unique*[*of u*] **by** *metis*

lemma *primroot-exp-zero* [*intro*]: $e_\varrho\ u = 0 \Longrightarrow u = \varepsilon$
using *primroot-exp-nemp* **by** *force*

lemma *primroot-exp-zero-conv* [*simp*]: $e_\varrho\ u = 0 \longleftrightarrow u = \varepsilon$
using *primroot-exp-nemp* **by** *force*

lemma *pop-primroot*: $u = \varrho\ u \cdot (\varrho\ u)^{@}(e_\varrho\ u - 1)$
by (*subst primroot-exp-eq*[*symmetric*], *subst pow-list-eq-if*) *force*

lemma *pop-primroot'*: $u = (\varrho\ u)^{@}(e_\varrho\ u - 1) \cdot \varrho\ u$
by (*subst pop-primroot*) *comparison*

lemma *primroot-exp-minus-Suc-eq* [*simp*]: $\varrho\ x\ ^{@}\ (e_\varrho\ x - Suc\ 0) \cdot \varrho\ x = x$
proof (*cases* $x = \varepsilon$)
assume $x \neq \varepsilon$
hence $0 < e_\varrho\ x$
using *primroot-exp-zero* **by** *blast*
show *?thesis*
using *pow-pos2*[*OF* ‹$0 < e_\varrho\ x$›, *of* $\varrho\ x$, *symmetric*, *unfolded primroot-exp-eq*]

by *force*
qed *simp*

lemma *primroot-nemp*[*intro*!]: $x \neq \varepsilon \Longrightarrow \varrho\ x \neq \varepsilon$
using *prim-nemp* **by** *blast*

lemma *primroot-idemp*[*simp*]: $\varrho\ (\varrho\ x) = \varrho\ x$
by (*cases* $x = \varepsilon$) (*simp only*: *primroot-emp*, *use* *prim-primroot* **in** *blast*)

lemma *prim-primroot-conv*: **assumes** $w \neq \varepsilon$ **shows** *primitive* $w \longleftrightarrow \varrho\ w = w$
using *assms prim-primroot primroot-prim*[*OF* ‹$w \neq \varepsilon$›] **by** *metis*

lemma *not-prim-primroot-expE*: **assumes** ¬ *primitive* w
obtains k **where** $\varrho\ w\ ^@k = w$ **and** $2 \leq k$
using *primroot-exp-eq primroot-prim assms*
proof (*cases* $w = \varepsilon$)
assume $w \neq \varepsilon$
with *primroot-exp-eq*[*of* w]
have $e_\varrho\ w \neq 1\ e_\varrho\ w \neq 0$
using *pow-zero primroot-prim*[*OF* ‹$w \neq \varepsilon$›] ‹¬ *primitive* w› **by** *force*+
with *that*[*OF* ‹$\varrho\ w\ ^@\ e_\varrho\ w = w$›]
show *thesis* **by** *force*
qed *force*

lemma *not-prim-expE*: **assumes** ¬ *primitive* x **and** $x \neq \varepsilon$
obtains $r\ k$ **where** *primitive* r **and** $2 \leq k$ **and** $r^@k = x$
using *not-prim-primroot-expE*[*OF* ‹¬ *primitive* x›] *primroot-prim*[*OF* ‹$x \neq \varepsilon$›]
by *metis*

lemma *primroot-len-le*: $u \neq \varepsilon \Longrightarrow |\varrho\ u| \leq |u|$
using *primroot-exp-len primroot-exp-zero-conv quotient-smaller* **by** *metis*

lemma *primroot-pref*: $\varrho\ u \leq_p u$
by (*cases primitive u*, *simp add*: *prim-primroot*)
(*unfold not-prim-pref-primroots*, *rule append-prefixD*)

lemma *primroot-take*: **assumes** $u \neq \varepsilon$ **shows** $\varrho\ u = (take\ (\ |\varrho\ u|\ )\ u)$
using *primroot-pref*[*folded pref-take-conv*, *symmetric*].

lemma *primroot-rotate-comm*: **assumes** $w \neq \varepsilon$ **shows** $\varrho\ (rotate\ n\ w) = rotate\ n\ (\varrho\ w)$
proof−
obtain l **where** $(\varrho\ w)^@l = w$
using *primroot-expE*.
have *rotate* $n\ w \neq \varepsilon$
using *assms* **by** *auto*
have *primitive* (*rotate* $n\ (\varrho\ w)$)
using *assms prim-rotate-conv* **by** *blast*
show *?thesis*

using *primroot-unique*[*OF* ‹*rotate n w* ≠ ε› ‹*primitive* (*rotate n* (ϱ *w*))›, *of l*]
unfolding ‹ϱ *w* ^@ *l* = *w*› *rotate-pow-list-swap*[*symmetric*] **by** *blast*
qed

lemma *primroot-rotate*: ϱ *w* = *r* ⟷ ϱ (*rotate* (*k*∗|*r*|) *w*) = *r* (**is** *?L* ⟷ *?R*)
proof(*cases w* = ε)
case *False*
show *?thesis*
unfolding *primroot-rotate-comm*[*OF* ‹*w* ≠ ε›, *of k*∗|*r*|]
using *length-rotate*[*of k*∗|*r*| ϱ *w*] *mod-mult-self2-is-0*[*of k* |*r*|]
rotate-id[*of k*∗|*r*| ϱ *w*]
by *metis*
qed (*simp add*: *rotate-is-Nil-conv*[*of k*∗|*r*| *w*])

lemma *primrootI*[*intro*]: **assumes** *pow*: *u* = *r*^@(*Suc k*) **and** *primitive r* **shows** ϱ *u* = *r*
proof−
have *u* ≠ ε
using *pow* ‹*primitive r*› *prim-nemp* **by** *auto*
show ϱ *u* = *r*
using *primroot-unique*[*OF* ‹*u* ≠ ε› ‹*primitive r*› ‹*u* = *r*^@(*Suc k*)›].
qed

lemma *sing-primroot*[*simp*]: ϱ [*a*] = [*a*]
using *pref-singE*[*OF primroot-pref*, *of a*] *primroot-nemp*[*of* [*a*], *OF list.distinct*(*2*)]
by *blast*

lemma *short-primroot*: **assumes** *u* ≠ ε ¬ *primitive u* **shows** |ϱ *u*| < |*u*|
using *primroot-prim*[*OF* ‹*u* ≠ ε›] *le-neq-implies-less pref-len primroot-pref long-pref assms* **by** *metis*

lemma *prim-primroot-cases*: **obtains** *u* = ε | *primitive u* | |ϱ *u*| < |*u*|
using *short-primroot* **by** *blast*

We also have the standard characterization of commutation for nonempty words.

theorem *comm-primroots*: **assumes** *u* ≠ ε **and** *v* ≠ ε **shows** *u* · *v* = *v* · *u* ⟷ ϱ *u* = ϱ *v*
proof
assume *u* · *v* = *v* · *u*
then obtain *t k m* **where** *u* = *t*^@*k* **and** *v* = *t*^@*m* **and** *t* ≠ ε
unfolding *commE* **by** *blast*
have *primitive* (ϱ *t*)
using ‹*t* ≠ ε› **by** *blast*
show ϱ *u* = ϱ *v*
using
primroot-unique[*OF* ‹*u* ≠ ε› ‹*primitive* (ϱ *t*)›, *of* e_{ϱ} *t*∗*k*]
primroot-unique[*OF* ‹*v* ≠ ε› ‹*primitive* (ϱ *t*)›, *of* e_{ϱ} *t*∗*m*]
unfolding *pow-mult primroot-exp-eq* ‹*u* = *t*^@*k*› ‹*v* = *t*^@*m*›

by *argo*
next
assume $\varrho\ u = \varrho\ v$
from *pows-comm*[*of* $e_\varrho\ u\ \varrho\ u\ \ e_\varrho\ v$]
show $u \cdot v = v \cdot u$
unfolding *primroot-exp-eq* **unfolding** ‹$\varrho\ u = \varrho\ v$› *primroot-exp-eq*.
qed

lemma *comm-primroots'*: $u \neq \varepsilon \Longrightarrow v \neq \varepsilon \Longrightarrow u \cdot v = v \cdot u \Longrightarrow \varrho\ u = \varrho\ v$
by (*simp add*: *comm-primroots*)

lemma *primroot-add-exp*: **assumes** $0 < l$ **shows** $\varrho\ (x^{@}l) = \varrho\ x$
using *comm-primroots'*[*OF - - pow-comm*, *of l x*] *pow-list-Nil-iff-Nil*[*OF* ‹$0 < l$›, *of x*]
by *fastforce*

lemma *same-primroots-comm*: $\varrho\ x = \varrho\ y \Longrightarrow x \cdot y = y \cdot x$
using *comm-primroots* **by** *blast*

lemma *pow-primroot*: **assumes** $x \neq \varepsilon$ **shows** $\varrho\ (x^{@}Suc\ k) = \varrho\ x$
using *comm-primroots'*[*OF nemp-Suc-pow-nemp*, *OF assms assms*, *of k*, *folded pow-Suc2 pow-Suc*] **by** *blast*

lemma *comm-primroot-exp*: **assumes** $v \neq \varepsilon$ **and** $u \cdot v = v \cdot u$
obtains n **where** $(\varrho\ v)^{@}n = u$
proof(*cases*)
assume $u = \varepsilon$ **thus** *thesis* **using** *that* **by** *blast*
next
assume $u \neq \varepsilon$ **thus** *thesis* **using** *that*[*OF primroot-expE*] ‹$u \cdot v = v \cdot u$›[*unfolded comm-primroots*[*OF* ‹$u \neq \varepsilon$› ‹$v \neq \varepsilon$›]] **by** *metis*
qed

lemma *primE*: **obtains** t **where** *primitive t*
using *prim-sing*[*of undefined*] **by** *fast*

lemma *comm-primrootE'*: **assumes** $x \cdot y = y \cdot x$
obtains $t\ m\ k$ **where** $x = t^{@}k$ **and** $y = t^{@}m$ **and** *primitive t*
by (*cases* $x = \varepsilon$)
(*use prim-sing primroot-ex pow-zero* **in** *metis*,
use assms comm-primroots' primroot-exp-eq primroot-prim pow-zero **in** *metis*)

lemma *comm-nemp-pows-posE*: **assumes** $x \cdot y = y \cdot x$ **and** $x \neq \varepsilon$ **and** $y \neq \varepsilon$
obtains $t\ k\ m$ **where** $x = t^{@}k$ **and** $y = t^{@}m$ **and** $0 < k$ **and** $0 < m$ **and** *primitive t*
using ‹$x \cdot y = y \cdot x$›[*unfolded comm-primroots*[*OF* ‹$x \neq \varepsilon$› ‹$y \neq \varepsilon$›]]
primroot-expE primroot-prim[*OF* ‹$x \neq \varepsilon$›] **by** *metis*

lemma *comm-primroot-conv*: $u \cdot v = v \cdot u \longleftrightarrow u \cdot \varrho\ v = \varrho\ v \cdot u$
proof (*cases* $u = \varepsilon \vee v = \varepsilon$)

assume $\neg\ (u = \varepsilon \vee v = \varepsilon)$
hence $u \neq \varepsilon\ v \neq \varepsilon$
by *blast+*
show *?thesis*
using *comm-primroots*[*OF* ‹$u \neq \varepsilon$› ‹$v \neq \varepsilon$›, *folded*
comm-primroots[*OF* ‹$u \neq \varepsilon$› *primroot-nemp*[*OF* ‹$v \neq \varepsilon$›], *unfolded primroot-idemp*]].
qed *force*

lemma *comm-primroot* [*simp*, *intro*]: $u \cdot \varrho\ u = \varrho\ u \cdot u$
using *comm-primroot-conv* **by** *blast*

lemma *comm-primroot-conv'*: **shows** $u \cdot v = v \cdot u \longleftrightarrow \varrho\ u \cdot \varrho\ v = \varrho\ v \cdot \varrho\ u$
using *comm-primroot-conv*[*of u v*] *comm-primroot-conv*[*of* $\varrho\ v\ u$]
unfolding *eq-sym-conv*[*of* $\varrho\ v \cdot u$] *eq-sym-conv*[*of* $\varrho\ v \cdot \varrho\ u$] **by** *blast*

lemma *comm-primrootI* [*intro*]: $\varrho\ u \cdot \varrho\ v = \varrho\ v \cdot \varrho\ u \Longrightarrow u \cdot v = v \cdot u$
unfolding *comm-primroot-conv'*[*of u*].

lemma *per-root-primroot*: $w <p\ r \cdot w \Longrightarrow\ w <p\ \varrho\ r \cdot w$
using *per-root-drop-exp*[*of w* e_ϱ r ϱ r, *unfolded primroot-exp-eq*].

lemma *per-root-primroot'*: $w <p\ \varrho\ r \cdot w \Longrightarrow\ w <p\ r \cdot w$
by (*cases* $e_\varrho\ r = 0$, *use primroot-exp-eq* **in** *force*)
(*use per-root-add-exp*[*of w* ϱ r e_ϱ r, *unfolded primroot-exp-eq*] **in** *blast*)

lemma *per-root-primroot-iff*: $w <p\ \varrho\ r \cdot w \longleftrightarrow\ w <p\ r \cdot w$
using *iffI*[*OF per-root-primroot' per-root-primroot*].

lemma *per-root-primroot-iff'*: $x \leq p\ \varrho\ u \cdot x \longleftrightarrow x \leq p\ u \cdot x$
using *per-root-primroot-iff nemp-per-root-conv primroot-emp primroot-nemp* **by** *metis*

lemma *primroot-per-root*: $r \neq \varepsilon \Longrightarrow\ r <p\ \varrho\ r \cdot r$
by *blast*

lemma *prim-comm-short-emp*: **assumes** *primitive p* **and** $u{\cdot}p{=}p{\cdot}u$ **and** $|u| < |p|$
shows $u = \varepsilon$
proof (*rule ccontr*)
assume $u \neq \varepsilon$
from ‹$u \cdot p = p \cdot u$›
have $\varrho\ u = \varrho\ p$
unfolding *comm-primroots*[*OF* ‹$u \neq \varepsilon$› *prim-nemp*, *OF* ‹*primitive p*›].
have $\varrho\ u = p$
using *prim-primroot*[*OF* ‹*primitive p*›, *folded* ‹$\varrho\ u = \varrho\ p$›].
from ‹$|u| < |p|$›[*folded this*]
show *False*
using *primroot-len-le*[*OF* ‹$u \neq \varepsilon$›] **by** *auto*
qed

lemma *primroot-rev*[*reversal-rule*]: **shows** $\varrho\ (rev\ u) = rev\ (\varrho\ u)$
proof (*cases* $u = \varepsilon$)
  assume $u \neq \varepsilon$
  hence $rev\ u \neq \varepsilon$
    by *simp*
  have *primitive* $(rev\ (\varrho\ u))$
    using *primroot-prim*[*OF* ‹$u \neq \varepsilon$›] **unfolding** *prim-rev-iff*.
  have $rev\ u = (rev\ (\varrho\ u))^{@} e_\varrho\ u$
    unfolding *rev-pow*[*symmetric*] *primroot-exp-eq*..
  from *primroot-unique*[*OF* ‹$rev\ u \neq \varepsilon$› ‹*primitive* $(rev\ (\varrho\ u))$› *this*]
  show *?thesis*.
qed *simp*

lemmas *primroot-suf* = *primroot-pref*[*reversed*]

lemma *per-le-prim-iff*:
  assumes $u \leq p\ p \cdot u$ **and** $p \neq \varepsilon$ **and** $2 * |p| \leq |u|$
  shows *primitive* $u \longleftrightarrow u \cdot p \neq p \cdot u$
proof
  have $|p| < |u|$ **using** ‹$2 * |p| \leq |u|$›
      nemp-len[*OF* ‹$p \neq \varepsilon$›] **by** *linarith*
  with ‹$p \neq \varepsilon$›
  show *primitive* $u \Longrightarrow u \cdot p \neq p \cdot u$
    by (*intro notI*, *elim notE*) (*rule prim-comm-short-emp*[*OF* - *sym*])
  show $u \cdot p \neq p \cdot u \Longrightarrow$ *primitive* u
  proof (*elim swap*[*of* - = -], *elim not-prim-primroot-expE*)
    fix $k\ z$ **assume** $2 \leq k$ **and** *eq*: $z\ ^{@}\ k = u$
    from *this*(*1*) *lenarg*[*OF this*(*2*)] ‹$2 * |p| \leq |u|$›
    have $|z| + |p| \leq |u|$
      by (*elim at-least2-Suc*) (*simp only*: *pow-Suc lenmorph*[*of z*])
    with ‹$u \leq p\ p \cdot u$› **have** $z \cdot p = p \cdot z$
      by (*rule two-pers*[*rotated 1*]) (*simp flip*: *eq pow-comm*)
    from *comm-pow-comm*[*OF this*, *of k*]
    show $u \cdot p = p \cdot u$ **unfolding** *eq*.
  qed
qed

lemma *per-root-mod-primE* [*elim*]: **assumes** $u <p\ r \cdot u$
  obtains $n\ p\ s$ **where** $p \cdot s = \varrho\ r$ **and** $(p \cdot s)^{@} n \cdot p = u$ **and** $s \neq \varepsilon$
 using *per-root-modE*[*OF sprefD1*[*OF per-root-primroot*[*OF assms*]]] *primroot-prim*[*OF per-root-nemp*[*OF assms*]]
   emp-not-prim **by** *metis*

lemmas[*shifts*] = *pows-comm*

lemma *per-root-shift*: **assumes** $x = (r \cdot s)^{@} k \cdot r\ \ y = (r \cdot s)^{@} m$
  shows $y \cdot x = x \cdot (s \cdot r)^{@} m$
  unfolding *assms* **by** *comparison*

lemma *root-comm-root*: **assumes** $x \leq p\ u \cdot x$ **and** $v \cdot u = u \cdot v$ **and** $u \neq \varepsilon$
shows $x \leq p\ v \cdot x$
proof (*cases* $v = \varepsilon$, *blast*)
assume $v \neq \varepsilon$
show $x \leq p\ v \cdot x$
using ‹$v \cdot u = u \cdot v$›[*unfolded comm-primroots*[*OF* ‹$v \neq \varepsilon$› ‹$u \neq \varepsilon$›]]
per-root-primroot-iff' ‹$x \leq p\ u \cdot x$› **by** *metis*
qed

2.20.1 Primitivity and the shortest period

lemma *min-per-primitive*: **assumes** $w \neq \varepsilon$ **shows** *primitive* $(\pi\ w)$
proof–
have $\varrho(\pi\ w) \neq \varepsilon$
using *assms min-per-nemp primroot-nemp* **by** *blast*
obtain k **where** $\pi\ w = (\varrho\ (\pi\ w))^{@}k$
using *primroot-expE* **by** *metis*
have $w <p\ (\varrho\ (\pi\ w)) \cdot w$
using *per-root-primroot*[*OF min-per-root-per-root*[*OF* ‹$w \neq \varepsilon$›]].
from *pow-pref-root-one*[*OF* - ‹$\varrho(\pi\ w) \neq \varepsilon$›, *of* k, *folded* ‹$\pi\ w = (\varrho\ (\pi\ w))^{@}k$›, *OF* - *min-per-min*[*OF this*]]
have $k = 1$
using ‹$\pi\ w = (\varrho\ (\pi\ w))^{@}k$› *min-per-nemp*[*OF* ‹$w \neq \varepsilon$›] *pow-zero*[*of* $\varrho\ (\pi\ w)$]
by *fastforce*
show *primitive* $(\pi\ w)$
using *primroot-prim*[*OF* ‹$\varrho\ (\pi\ w) \neq \varepsilon$›, *folded* ‹$\pi\ w = (\varrho\ (\pi\ w))^{@}k$›[*unfolded* ‹$k = 1$› *One-nat-def pow-list-one*]].
qed

lemma *min-per-short-primroot*: **assumes** $w \neq \varepsilon$ **and** $(\varrho\ w)^{@}k = w$ **and** $k \neq 1$
shows $\pi\ w = \varrho\ w$
proof–
have $k \neq 0$
using *assms*(*1–2*) **by** *force*
with ‹$k \neq 1$› **have** $2 \leq k$
by *fastforce*
have $w <p\ (\varrho\ w) \cdot w$
using *assms*(*1*) *assms*(*2*) *per-root-drop-exp root-self* **by** *metis*
have $w <p\ (\pi\ w) \cdot w$
using *assms*(*1*) *min-per-root-per-root* **by** *blast*
have $\pi\ w \leq p\ \varrho\ w$
using *min-per-min*[*OF* ‹$w <p\ (\varrho\ w) \cdot w$›].
from *prefix-length-le*[*OF this*]
have $|\pi\ w| + |\varrho\ w| \leq |w|$
unfolding *lenarg*[*OF* ‹$(\varrho\ w)^{@}k = w$›, *unfolded pow-len*, *symmetric*] **using**
mult-le-mono1[*OF* ‹$2 \leq k$›, *of* $|\varrho\ w|$] **unfolding** *one-add-one*[*symmetric*] *distrib-right mult-1*
by *simp*
from *two-pers*[*OF sprefD1*[*OF* ‹$w <p\ (\pi\ w) \cdot w$›] *sprefD1*[*OF* ‹$w <p\ (\varrho\ w) \cdot w$›]

this]
have $\pi\ w \cdot \varrho\ w = \varrho\ w \cdot \pi\ w$**.**
from *this*[*unfolded comm-primroots*[*OF per-root-nemp*[*OF* ‹$w <p (\pi\ w) \cdot w$›] *per-root-nemp*[*OF* ‹$w <p (\varrho\ w) \cdot w$›]]]
show $\pi\ w = \varrho\ w$
unfolding *prim-primroot*[*of* $\varrho\ w$, *OF primroot-prim*[*OF* ‹$w \neq \varepsilon$›]]
prim-primroot[*of* $\pi\ w$, *OF min-per-primitive*[*OF* ‹$w \neq \varepsilon$›]]**.**
qed

lemma *primitive-iff-per*: *primitive* $w \longleftrightarrow w \neq \varepsilon \wedge (\pi\ w = w \vee \pi\ w \cdot w \neq w \cdot \pi\ w)$
proof
assume *primitive w*
hence $w \neq \varepsilon$ **by** *fastforce*
show $w \neq \varepsilon \wedge (\pi\ w = w \vee \pi\ w \cdot w \neq w \cdot \pi\ w)$
proof (*rule conjI*)
show $\pi\ w = w \vee \pi\ w \cdot w \neq w \cdot \pi\ w$
using *comm-prim* [*OF min-per-primitive*[*OF* ‹$w \neq \varepsilon$›] ‹*primitive w*›]
by (*intro verit-or-neg*(*1*))
qed *fact*
next
assume *asm*: $w \neq \varepsilon \wedge (\pi\ w = w \vee \pi\ w \cdot w \neq w \cdot \pi\ w)$
have $w \neq \varepsilon$ **and** *imp*: $\pi\ w \cdot w = w \cdot \pi\ w \Longrightarrow \pi\ w = w$
using *asm* **by** *blast*+
obtain k **where** $(\varrho\ w)^{@}k = w\ 0 < k$
using *primroot-expE***.**
show *primitive w*
proof−
from *imp*[*unfolded min-per-short-primroot*[*OF* ‹$w \neq \varepsilon$› ‹$(\varrho\ w)^{@}k = w$›]]
have $\varrho\ w = w$
using *pow-comm*[*symmetric*, *of* $\varrho\ w\ k$, *unfolded* ‹$\varrho\ w\ ^{@}k = w$›]
‹$\varrho\ w\ ^{@}\ k = w$› *min-per-short-primroot*[*OF* ‹$w \neq \varepsilon$› ‹$\varrho\ w^{@}k = w$›] ‹$w \neq \varepsilon$›
by *force*
thus *primitive w*
using *prim-primroot-conv*[*OF* ‹$w \neq \varepsilon$›] **by** *simp*
qed
qed

2.21 Conjugation

Two words x and y are conjugated if one is a rotation of the other. Or, equivalently, there exists z such that

$$xz = zy.$$

definition *conjugate* (**infix** $\sim$ *51*)
where $u \sim v \equiv \exists r\ s.\ r \cdot s = u \wedge s \cdot r = v$

lemma *conjugE* [*elim*]:
assumes $u \sim v$
obtains $r\ s$ **where** $r \cdot s = u$ **and** $s \cdot r = v$
using *assms* **unfolding** *conjugate-def* **by** (*elim exE conjE*)

lemma *conjugE-nemp*[*elim*]:
assumes $u \sim v$ **and** $u \neq \varepsilon$
obtains $r\ s$ **where** $r \cdot s = u$ **and** $s \cdot r = v$ **and** $s \neq \varepsilon$
using *assms* **unfolding** *conjugate-def*
proof (*cases* $u = v$)
assume $u \neq v$
obtain $r\ s$ **where** $r \cdot s = u$ **and** $s \cdot r = v$ **using** *conjugE*[*OF* ‹$u \sim v$›]**.**
hence $s \neq \varepsilon$ **using** ‹$u \neq v$› **by** *force*
thus *thesis* **using** *that*[*OF* ‹$r \cdot s = u$› ‹$s \cdot r = v$›] **by** *blast*
qed (*simp add*: *that*[*OF* - - ‹$u \neq \varepsilon$›])

lemma *conjugE1* [*elim*]:
assumes $u \sim v$
obtains r **where** $u \cdot r = r \cdot v$
proof −
obtain $r\ s$ **where** *u*: $r \cdot s = u$ **and** *v*: $s \cdot r = v$ **using** *assms***..**
have $u \cdot r = r \cdot v$ **unfolding** *u*[*symmetric*] *v*[*symmetric*] **using** *rassoc***.**
then show *thesis* **by** *fact*
qed

lemma *conjug-rev-conv* [*reversal-rule*]: *rev* $u \sim$ *rev* $v \longleftrightarrow u \sim v$
unfolding *conjugate-def*[*reversed*] **using** *conjugate-def* **by** *blast*

lemma *conjug-rotate-iff*: $u \sim v \longleftrightarrow (\exists\ n.\ v =$ *rotate* $n\ u)$
unfolding *conjugate-def*
using *rotate-drop-take*[*of* - *u*] *takedrop*[*of* - *u*] *rotate-append*
by *metis*

lemma *rotate-conjug*: $w \sim$ *rotate* $n\ w$
using *conjug-rotate-iff* **by** *blast*

lemma *conjug-rotate-iff-le*:
shows $u \sim v \longleftrightarrow (\exists\ n \leq |u| - 1.\ v =$ *rotate* $n\ u)$
proof
show $\exists n \leq |u| - 1\ .\ v =$ *rotate* $n\ u \Longrightarrow u \sim v$
using *conjug-rotate-iff* **by** *blast*
next
assume $u \sim v$
thus $\exists\ n \leq |u| - 1.\ v =$ *rotate* $n\ u$
proof (*cases* $u = \varepsilon$)
assume $u \neq \varepsilon$
obtain $r\ s$ **where** $r \cdot s = u$ **and** $s \cdot r = v$ **and** $s \neq \varepsilon$
using *conjugE-nemp*[*OF* ‹$u \sim v$› ‹$u \neq \varepsilon$›]**.**
hence $v =$ *rotate* $|r|\ u$

using *rotate-append*[*of r s*] **by** *argo*
moreover have $|r| \leq |u| - 1$
using *lenarg*[*OF* ‹$r \cdot s = u$›, *unfolded lenmorph*] *nemp-len*[*OF* ‹$s \neq \varepsilon$›] **by** *linarith*
ultimately show $\exists\, n \leq |u| - 1.\ v = rotate\ n\ u$
by *blast*
qed *auto*
qed

lemma *conjugI* [*intro*]: $r \cdot s = u \Longrightarrow s \cdot r = v \Longrightarrow u \sim v$
unfolding *conjugate-def* **by** (*intro exI conjI*)

lemma *conjugI'* [*intro*!]: $r \cdot s \sim s \cdot r$
unfolding *conjugate-def* **by** (*intro exI conjI*) *standard+*

lemma *conjug-refl*: $u \sim u$
by *standard+*

lemma *conjug-sym*[*sym*]: $u \sim v \Longrightarrow v \sim u$
by (*elim conjugE*, *intro conjugI*) *assumption*

lemma *conjug-swap*: $u \sim v \longleftrightarrow v \sim u$
by *blast*

lemma *conjug-nemp-iff*: $u \sim v \Longrightarrow u = \varepsilon \longleftrightarrow v = \varepsilon$
by (*elim conjugE1*, *intro iffI*) *simp+*

lemma *conjug-len*: $u \sim v \ \Longrightarrow |u| = |v|$
by (*elim conjugE*, *hypsubst*, *rule swap-len*)

lemma *pow-conjug*:
assumes *eq*: $t^{@}i \cdot r \cdot u = t^{@}k$ **and** *t*: $r \cdot s = t$
shows $u \cdot t^{@}i \cdot r = (s \cdot r)^{@}k$
proof −
have $t^{@}i \cdot r \cdot u \cdot t^{@}i \cdot r = t^{@}i \cdot t^{@}k \cdot r$ **unfolding** *eq*[*unfolded lassoc*] *lassoc append-same-eq pows-comm*..
also have $\ldots = t^{@}i \cdot r \cdot (s \cdot r)^{@}k$ **unfolding** *conjug-pow*[*OF rassoc*, *symmetric*] *t*..
finally show $u \cdot t^{@}i \cdot r = (s \cdot r)^{@}k$ **unfolding** *same-append-eq*.
qed

lemma *conjug-set*: **assumes** $u \sim v$ **shows** *set u* = *set v*
using *conjugE*[*OF* ‹$u \sim v$›] *set-append Un-commute* **by** *metis*

lemma *conjug-concat-conjug*: $xs \sim ys \Longrightarrow concat\ xs \sim concat\ ys$
unfolding *conjugate-def* **using** *concat-morph* **by** *metis*

The solution of the equation

$$xz = zy$$

is given by the next lemma.

lemma *conjug-eqE* [*elim*, *consumes 2*]:
assumes *eq*: $x \cdot z = z \cdot y$ **and** $x \neq \varepsilon$
obtains $u\ v\ k$ **where** $u \cdot v = x$ **and** $v \cdot u = y$ **and** $(u \cdot v)^{@}k \cdot u = z$ **and** $v \neq \varepsilon$
proof −
have $z \leq p\ x \cdot z$ **using** *eq*[*symmetric*]**..**
from *this* **and** ‹$x \neq \varepsilon$›
obtain $k\ u\ v$ **where** $x^{@}k \cdot u = z$ **and** *x*: $u \cdot v = x$ **and** $v \neq \varepsilon$**..**
have *z*: $(u{\cdot}v)^{@}k \cdot u = z$ **unfolding** x ‹$x^{@}k \cdot u = z$›**..**
have $z \cdot y = (u{\cdot}v) \cdot ((u{\cdot}v)^{@}k \cdot u)$ **unfolding** z **unfolding** x *eq*.**.**
also have $\ldots = (u{\cdot}v)^{@}k \cdot u \cdot (v \cdot u)$ **unfolding** *lassoc pow-comm*[*symmetric*]**..**
finally have *y*: $v \cdot u = y$ **unfolding** *z*[*symmetric*] *rassoc same-append-eq*.**.**
from $x\ y\ z$ ‹$v \neq \varepsilon$› **show** *thesis*.**.**
qed

theorem *conjugation*: **assumes** $x{\cdot}z = z{\cdot}y$ **and** $x \neq \varepsilon$
shows $\exists\ u\ v\ k.\ u \cdot v = x \wedge v \cdot u\ = y \wedge (u \cdot v)^{@}k \cdot u = z$
using *assms* **by** *blast*

lemma *conjug-eqE′* [*elim*]:
assumes *eq*: $x \cdot z = z \cdot y$
obtains $u\ v\ k\ i$ **where** $(u \cdot v)^{@}i = x$ **and** $(v \cdot u)^{@}i = y$ **and** $(u \cdot v)^{@}k \cdot u = z$
proof (*cases* $x = \varepsilon$)
assume $x = \varepsilon$
hence $y = \varepsilon$
using *eq* **by** *simp*
from *that*[*of 0 z ε 0*, *unfolded* ‹$x = \varepsilon$› ‹$y = \varepsilon$› *cow-simps*]
show *thesis*
by *simp*
next
assume $x \neq \varepsilon$
from *conjug-eqE*[*OF eq this*, *of thesis*]
show *thesis*
using *that*[*of 1*, *unfolded cow-simps*] **by** *blast*
qed

lemma *conjug-eq-primrootE′* [*elim*, *consumes 2*]:
assumes *eq*: $x \cdot z = z \cdot y$ **and** $x \neq \varepsilon$
obtains $r\ s\ i\ n$ **where**
$(r \cdot s)^{@}i = x$ **and**
$(s \cdot r)^{@}i = y$ **and**
$(r \cdot s)^{@}n \cdot r = z$ **and**
$s \neq \varepsilon$ **and** $0 < i$ **and** *primitive* $(r \cdot s)$
proof −
obtain i **where** $(\varrho\ x)^{@}i = x\ 0 < i$
using *primroot-expE* **by** *blast*
have $z <p\ x \cdot z$ **using** *prefI*[*OF* ‹$x \cdot z = z \cdot y$›[*symmetric*]] ‹$x \neq \varepsilon$›**..**
from *per-root-primroot*[*OF this*]
have $z <p\ (\varrho\ x) \cdot z$.

from *per-root-modE*[*OF sprefD1*[*OF this*] *primroot-nemp*[*OF* ‹$x \neq \varepsilon$›]]
obtain $n\ r\ s$ **where** $r \cdot s = \varrho\ x\ \varrho\ x\ ^{@}\ n \cdot r = z\ s \neq \varepsilon$**.**
have x: $(r{\cdot}s)^{@}i = x$ **unfolding** ‹$r \cdot s = \varrho\ x$› ‹$(\varrho\ x)^{@}i = x$›**..**
have z: $(r{\cdot}s)^{@}n \cdot r = z$ **unfolding** ‹$r \cdot s = \varrho\ x$› **using** ‹$(\varrho\ x)^{@}n \cdot r = z$›**.**
have y [*symmetric*]: $y = (s{\cdot}r)^{@}i$
using *eq*[*symmetric*, *folded x z*, *unfolded lassoc pows-comm*[*of i*], *unfolded rassoc cancel*,
unfolded shift-pow cancel]**.**
from ‹$x \neq \varepsilon$› **have** *primitive* $(r \cdot s)$ **unfolding** ‹$r \cdot s = \varrho\ x$›**..**
from *that*[*OF x y z* ‹$s \neq \varepsilon$› ‹$0 < i$› *this*]
show *thesis***.**
qed

lemma *conjugI1* [*intro*]:
assumes *eq*: $u \cdot r = r \cdot v$
shows $u \sim v$
proof (*cases*)
assume $u = \varepsilon$
have $v = \varepsilon$ **using** *eq* **unfolding** ‹$u = \varepsilon$› **by** *simp*
show $u \sim v$ **unfolding** ‹$u = \varepsilon$› ‹$v = \varepsilon$› **using** *conjug-refl***.**
next
assume $u \neq \varepsilon$
show $u \sim v$ **using** *eq* ‹$u \neq \varepsilon$› **by** (*cases rule*: *conjug-eqE*, *intro conjugI*)
qed

lemma *pow-conjug-conjug-conv*: **assumes** $0 < k$ **shows** $u^{@}k \sim v^{@}k \longleftrightarrow u \sim v$
proof
assume $u\ ^{@}\ k \sim v\ ^{@}\ k$
obtain $r\ s$ **where** $r \cdot s = u^{@}k$ **and** $s \cdot r = v^{@}k$
using *conjugE*[*OF* ‹$u^{@}k \sim v^{@}k$›]**.**
hence $v^{@}k = (rotate\ |r|\ u)^{@}k$
using *rotate-append rotate-pow-list-swap* **by** *metis*
hence $v = rotate\ |r|\ u$
using *pow-eq-eq*[*OF* - ‹$0 < k$›] **by** *blast*
thus $u \sim v$
using *rotate-conjug* **by** *blast*
next
assume $u \sim v$
obtain $r\ s$ **where** $u = r \cdot s$ **and** $v = s \cdot r$
using *conjugE*[*OF* ‹$u \sim v$›] **by** *metis*
have $u^{@}k \cdot r = r \cdot v^{@}k$
unfolding ‹$u = r \cdot s$› ‹$v = s \cdot r$› *shift-pow*..
thus $u^{@}k \sim v^{@}k$
using *conjugI1* **by** *blast*
qed

lemma *conjug-trans* [*trans*]:
assumes *uv*: $u \sim v$ **and** *vw*: $v \sim w$
shows $u \sim w$

using *assms* **unfolding** *conjug-rotate-iff* **using** *rotate-rotate* **by** *blast*

lemma *conjug-trans'*: **assumes** *uv'*: $u \cdot r = r \cdot v$ **and** *vw'*: $v \cdot s = s \cdot w$ **shows** $u \cdot (r \cdot s) = (r \cdot s) \cdot w$
proof −
have $u \cdot (r \cdot s) = (r \cdot v) \cdot s$ **unfolding** *uv'*[*symmetric*] *rassoc*..
also have $\ldots = r \cdot (s \cdot w)$ **unfolding** *vw'*[*symmetric*] *rassoc*..
finally show $u \cdot (r \cdot s) = (r \cdot s) \cdot w$ **unfolding** *rassoc*.
qed

lemma *root-conjugE*:
assumes $x \leq p\ r \cdot x$
obtains $u\ v\ k\ i$ **where** $(u \cdot v)^{@} i = r$ **and** $(u \cdot v)^{@} k \cdot u = x$ **and** $(v \cdot u)\ ^{@}\ i = x^{-1>}(r \cdot x)$
using *conjug-eqE'*[*OF lq-pref*[*OF* ‹$x \leq p\ r \cdot x$›, *symmetric*]] **by** *metis*

lemmas *suf-root-conjugE* = *root-conjugE*[*reversed*]

Of course, conjugacy is an equivalence relation.

lemma *conjug-equiv*: *equivp* ($\sim$)
by (*simp add*: *conjug-refl conjug-sym conjug-trans equivpI reflpI sympI transpI*)

lemma *append-split-mod-primroot*:
obtains $r\ s\ i\ j$ **where** $(r{\cdot}s)^{@} i{\cdot}r = x\ (s{\cdot}r)^{@} j{\cdot}s = y\ \ r{\cdot}s = \varrho\ (x{\cdot}y)\ s{\cdot}r = x^{-1>}(\varrho\ (x{\cdot}y) \cdot x)$
proof−
have $x \leq p\ \varrho\ (x \cdot y) \cdot x$
by (*rule pref-pow-root*[*of* - $e_{\varrho}\ (x{\cdot}y)$], *unfold primroot-exp-eq*) (*rule triv-pref*)
from *root-conjugE*[*OF this*]
obtain $u\ v\ k\ i$ **where** $(u \cdot v)\ ^{@}\ i = \varrho\ (x \cdot y)\ (u \cdot v)\ ^{@}\ k \cdot u = x\ (v \cdot u)\ ^{@}\ i = x^{-1>}(\varrho\ (x{\cdot}y) \cdot x)$.
show *thesis*
proof (*cases* $y = \varepsilon$)
assume $y \neq \varepsilon$
hence $x \cdot y \neq \varepsilon$
by *blast*
have $i = 1$
using *prim-pow-exp-one*[*OF primroot-prim*[*OF* ‹$x{\cdot}y \neq \varepsilon$›, *folded* ‹$(u \cdot v)\ ^{@}\ i = \varrho\ (x{\cdot}y)$›]].
define e **where** $e = e_{\varrho}(x{\cdot}y)$
have $\neg\ e_{\varrho}(x{\cdot}y) \leq k$
using *mult-le-mono1*[*of* $e_{\varrho}(x{\cdot}y)\ k\ |u{\cdot}\ v|$]
unfolding *lenarg*[*OF primroot-exp-eq*[*of* $x \cdot y$], *unfolded pow-len* ‹$(u \cdot v)\ ^{@}\ i = \varrho\ (x \cdot y)$›[*symmetric*, *unfolded* ‹$i = 1$› *pow-list-1*]]
nemp-len[*OF* ‹$y \neq \varepsilon$›] *lenarg*[*OF* ‹$(u \cdot v)\ ^{@}\ k \cdot u = x$›, *unfolded pow-len lenmorph*[*of* - *u*], *symmetric*] *lenmorph*[*of x*] **using** *nemp-len*[*OF* ‹$y \neq \varepsilon$›] **by** *force*
hence $*$: $k + 1\ + (e - k - 1) = e$
unfolding *e-def* **by** *simp*
have *xy*: $x{\cdot}y = (u{\cdot}v)^{@}(k + 1\ + (e - k - 1))$

```
    unfolding ∗ unfolding e-def
    using primroot-exp-eq[of x · y, symmetric] unfolding ‹(u · v) @ i = ϱ (x · y)›[unfolded ‹i = 1› pow-list-1].
  have x · y = x · (v·u)@(e − k − 1) · v
    unfolding xy unfolding ‹(u · v) @ k · u = x›[symmetric] by comparison
  from this[unfolded cancel, symmetric]
  show thesis
    using that[OF ‹(u · v) @ k · u = x› -
        ‹(u · v) @ i = ϱ (x·y)›[unfolded ‹i=1› pow-list-1] ‹(v · u) @ i = x^-1>(ϱ (x·y) · x)›[unfolded ‹i=1› pow-list-1]]
    by blast
 next
  assume y = ε
  have (ϱ x · ε) @ (e_ϱ x − 1) · ϱ x = x
    by simp
  have ϱ x = x^-1>(ϱ x · x)
    unfolding comm-primroot[symmetric] lq-triv..
  then show thesis
    using that[unfolded ‹y = ε›, of e_ϱ x − 1 ϱ x ε 0] by simp
 qed
qed

lemma rotate-fac-pref: assumes u ≤f w
 obtains w' where w' ∼ w and u ≤p w'
proof−
 from facE[OF ‹u ≤f w›]
 obtain p s where w = p · u · s.
 from that[OF conjugI'[of u · s p, unfolded rassoc, folded this] triv-pref]
 show thesis.
qed

lemma rotate-into-pos-sq: assumes s·p ≤f w·w and |s| ≤ |w| and |p| ≤ |w|
obtains w' where w ∼ w' p ≤p w' s ≤s w'
proof−
 obtain pw where pw·s·p ≤p w·w
   by (meson assms(1) fac-pref)
 hence pw · s ≤p w· w
   unfolding lassoc prefix-def by force

 hence take |pw · s| (w · w) = pw · s
   using pref-take by blast

 have p ≤p drop |pw · s| (w · w)
   using pref-drop[OF ‹pw·s·p ≤p w·w›[unfolded lassoc]] drop-pref by metis

 let ?w = rotate |pw · s| w

 have |?w| = |w| by auto
```

have *rotate* $|pw \cdot s|\ (w \cdot w) = ?w \cdot ?w$
using *rotate-pow-comm-two*.

hence *eq*: $?w \cdot ?w = (drop\ |pw \cdot s|\ (w \cdot w)) \cdot take\ |pw \cdot s|\ (w \cdot w)$
by (*metis* ‹$pw \cdot s \le p\ w \cdot w$› *append-take-drop-id pref-take rotate-append*)

have $p \le p\ ?w$
using *pref-prod-le*[*OF* - ‹$|p| \le |w|$›[*folded* ‹$|?w| = |w|$›]]
prefix-prefix[*OF* ‹$p \le p\ drop\ |pw \cdot s|\ (w \cdot w)$›, *of take* $|pw \cdot s|\ (w \cdot w)$, *folded eq*].

have $s \le s\ ?w$
using *pref-prod-le*[*reversed*, *OF* - ‹$|s| \le |w|$›[*folded* ‹$|?w| = |w|$›], *of* ?w]
unfolding *eq* ‹*take* $|pw \cdot s|\ (w \cdot w) = pw \cdot s$› *lassoc* **by** *blast*

show *thesis*
using *that*[*OF rotate-conjug* ‹$p \le p\ ?w$› ‹$s \le s\ ?w$›].
qed

lemma *rotate-into-pref-sq*: **assumes** $p \le f\ w \cdot w$ **and** $|p| \le |w|$
obtains w' **where** $w \sim w'\ p \le p\ w'$
using *rotate-into-pos-sq*[*of* ε, *unfolded emp-simps*, *OF* ‹$p \le f\ w \cdot w$› - ‹$|p| \le |w|$›]
by *auto*

lemmas *rotate-into-suf-sq* = *rotate-into-pref-sq*[*reversed*]

lemma *rotate-into-pos*: **assumes** $s \cdot p \le f\ w$
obtains w' **where** $w \sim w'\ p \le p\ w'\ s \le s\ w'$
proof(*rule rotate-into-pos-sq*)
show $s \cdot p \le f\ w \cdot w$
using ‹$s \cdot p \le f\ w$› **by** *blast*
show $|s| \le |w|$
using *order.trans*[*OF pref-len' fac-len*[*OF* ‹$s \cdot p \le f\ w$›]].
show $|p| \le |w|$
using *order.trans*[*OF suf-len' fac-len*[*OF* ‹$s \cdot p \le f\ w$›]].
qed

lemma *rotate-into-pos-conjug*: **assumes** $w \sim v$ **and** $s \cdot p \le f\ v$
obtains w' **where** $w \sim w'\ p \le p\ w'\ s \le s\ w'$
using *assms conjug-trans rotate-into-pos* **by** *metis*

lemma *nconjug-neq*: $\neg\ u \sim v \Longrightarrow u \neq v$
by *blast*

lemma *prim-conjug*:
assumes *prim*: *primitive u* **and** *conjug*: $u \sim v$
shows *primitive v*
proof −
have $v \neq \varepsilon$ **using** *prim-nemp*[*OF prim*] **unfolding** *conjug-nemp-iff*[*OF conjug*].

from *conjug*[*symmetric*] **obtain** t **where** $v \cdot t = t \cdot u$**..**
from *this* ‹$v \neq \varepsilon$› **obtain** $r\ s\ i$ **where**
v: $(r \cdot s)^{@}i = v$ **and** u: $(s \cdot r)^{@}i = u$ **and** *prim'*: *primitive* $(r \cdot s)$ **and** $0 < i$**..**
have $r \cdot s = v$ **using** v **unfolding** *prim-exp-one*[*OF prim u*] *pow-list-1***.**
show *primitive* v **using** *prim'* **unfolding** ‹$r \cdot s = v$›**.**
qed

lemma *conjug-prim-iff*: **assumes** $u \sim v$ **shows** *primitive* u = *primitive* v
using *prim-conjug*[*OF* - ‹$u \sim v$›] *prim-conjug*[*OF* - *conjug-sym*[*OF* ‹$u \sim v$›]]**..**

lemmas *conjug-prim-iff'* = *conjug-prim-iff*[*OF conjugI'*]

lemmas *conjug-concat-prim-iff* = *conjug-concat-conjug*[*THEN conjug-prim-iff*]

lemma *conjug-eq-primrootE* [*elim*, *consumes 2*]:
assumes *eq*: $x \cdot z = z \cdot y$ **and** $x \neq \varepsilon$
obtains $r\ s\ i\ n$ **where**
$(r \cdot s)^{@}i = x$ **and**
$(s \cdot r)^{@}i = y$ **and**
$(r \cdot s)^{@}n \cdot r = z$ **and**
$s \neq \varepsilon$ **and** $0 < i$ **and** *primitive* $(r \cdot s)$
and *primitive* $(s \cdot r)$
using *conjug-eq-primrootE'*[*OF assms*] *conjug-prim-iff'* **by** *metis*

lemma *conjug-primrootsE*: **assumes** $\varrho\ p \sim \varrho\ q$
obtains $r\ s\ k\ l$ **where** $p = (r \cdot s)^{@}k$ **and** $q = (s \cdot r)^{@}l$ **and** *primitive* $(r{\cdot}s)$
proof(*cases*)
assume $p = \varepsilon \wedge q = \varepsilon$
obtain w::*'a list* **where** *primitive* w
by *blast*
from *that*[*of 0 w* ε *0*, *unfolded emp-simps*]
show *?thesis*
by (*simp add*: ‹$p = \varepsilon \wedge q = \varepsilon$› ‹*primitive* w›)
next
assume $\neg\ (p = \varepsilon \wedge q = \varepsilon)$
hence *primitive* $(\varrho\ p)$
using *assms conjug-prim-iff* **by** *auto*
from *conjugE*[*OF* ‹$\varrho\ p \sim \varrho\ q$›]
obtain $r\ s$ **where**
$r \cdot s = \varrho\ p$ **and**
$s \cdot r = \varrho\ q$**.**
from *that*[*of* $e_{\varrho}\ p\ r\ s$ $e_{\varrho}\ q$, *unfolded this*, *OF* - - ‹*primitive* $(\varrho\ p)$›]
show *?thesis*
using *primroot-exp-eq*[*symmetric*]
by *blast*
qed

lemma *root-conjug*: $u \leq p\ r \cdot u \Longrightarrow u^{-1>}(r{\cdot}u) \sim r$

using *conjugI1 conjug-sym lq-pref* **by** *metis*

lemmas *conjug-prim-iff-pref* = *conjug-prim-iff*[*OF root-conjug*]

lemma *conjug-primroot-word*:
assumes *conjug*: $u \cdot t = t \cdot v$
shows $(\varrho\ u) \cdot t = t \cdot (\varrho\ v)$
proof (*cases* $u = \varepsilon$)
assume $u \neq \varepsilon$
from ‹$u \cdot t = t \cdot v$› ‹$u \neq \varepsilon$› **obtain** $r\ s\ i\ n$ **where**
u: $(r \cdot s)^{@} i = u$ **and** v: $(s \cdot r)^{@} i = v$ **and** *prim*: *primitive* $(r \cdot s)$
and $(r \cdot s)^{@} n \cdot r = t$ **and** $0 < i$**..**
have *rs*: $\varrho\ u = r \cdot s$ **and** *sr*: $\varrho\ v = s \cdot r$
using *prim-conjug*[*OF prim conjugI'*] $u\ v$ ‹$0 < i$› *prim*
primroot-unique' **by** *meson+*
show *?thesis*
unfolding ‹$(r \cdot s)^{@} n \cdot r = t$›[*symmetric*] *rs sr*
by *comparison*
next
assume $u = \varepsilon$
hence $v = \varepsilon$
using *assms* **by** *force*
show *?thesis*
unfolding ‹$u = \varepsilon$› ‹$v = \varepsilon$› **by** *simp*
qed

lemma *conjug-primroot*:
assumes $u \sim v$
shows $\varrho\ u \sim \varrho\ v$
proof(*cases*)
assume $u = \varepsilon$ **with** ‹$u \sim v$› **show** $\varrho\ u \sim \varrho\ v$
using *conjug-nemp-iff* **by** *blast*
next
assume $u \neq \varepsilon$
from ‹$u \sim v$› **obtain** t **where** $u \cdot t = t \cdot v$**..**
from *conjug-primroot-word*[*OF this*]
show $\varrho\ u \sim \varrho\ v$
by (*simp add*: *conjugI1*)
qed

lemma *conjug-primroots-nemp*: **assumes** $x \cdot y \neq y \cdot x$ **and** $r \cdot s = \varrho\ (x \cdot y)$ **and** $s \cdot r = \varrho\ (y \cdot x)$
shows $r \neq \varepsilon$ **and** $s \neq \varepsilon$
proof−
have $x \cdot y \neq \varepsilon$ **and** $y \cdot x \neq \varepsilon$
using *assms*(*1*) **by** *force+*
have $r \neq \varepsilon \wedge s \neq \varepsilon$
proof (*rule contrapos-np*[*OF assms*(*1*)])
assume $\neg\ (r \neq \varepsilon \wedge s \neq \varepsilon)$

hence $\varrho\ (x \cdot y) = \varrho\ (y \cdot x)$
using *assms*(*2*−*3*) **by** *force*
with *comm-primroots*[*symmetric*, *OF* ‹$x \cdot y \neq \varepsilon$› ‹$y \cdot x \neq \varepsilon$›]
show $x \cdot y = y \cdot x$
using *eqd-eq*[*OF - swap-len*] **by** *meson*
qed
thus $r \neq \varepsilon$ **and** $s \neq \varepsilon$
by *blast*+
qed

lemma *conjugE-primrootsE*[*elim*]: **assumes** $x \cdot y \neq y \cdot x$
obtains $r\ s$ **where** $r \cdot s = \varrho\ (x \cdot y)$ **and** $s \cdot r = \varrho\ (y \cdot x)$ **and** $r \neq \varepsilon$ **and** $s \neq \varepsilon$
proof−
have $\varrho\ (x \cdot y) \neq \varepsilon$
using *assms* **by** *force*
from *conjugE-nemp*[*OF conjug-primroot*[*OF conjugI'*, *of x y*] *this*] *conjug-primroots-nemp*[*OF assms*] *that*
show *thesis*
by *auto*
qed

lemma *conjug-add-exp*: $u \sim v \Longrightarrow\ u^{@}k \sim v^{@}k$
by (*elim conjugE1*, *intro conjugI1*, *rule conjug-pow*)

lemma *conjug-primroot-iff*:
assumes *len*: $|u| = |v|$
shows $\varrho\ u\ \sim \varrho\ v \longleftrightarrow u \sim v$
proof
show $u \sim v \Longrightarrow \varrho\ u\ \sim \varrho\ v$ **using** *conjug-primroot*.
next
assume *conjug*: $\varrho\ u\ \sim \varrho\ v$
show $u \sim v$
proof (*cases* $u = \varepsilon$)
assume $u \neq \varepsilon$
hence $v \neq \varepsilon$
using *len* **by** *force*
have $|(\varrho\ u)^{@}e_{\varrho}\ u| = |(\varrho\ v)^{@}e_{\varrho}\ v|$
using *len* **unfolding** *primroot-exp-eq*.
then have $e_{\varrho}\ u = e_{\varrho}\ v$
using *primroot-nemp*[*OF* ‹$v \neq \varepsilon$›]
unfolding *pow-len conjug-len*[*OF conjug*] **by** *simp*
show $u \sim v$
using *conjug-add-exp*[*OF conjug*, *of* $e_{\varrho}\ u$]
unfolding *primroot-exp-eq* **unfolding** *primroot-exp-eq* ‹$e_{\varrho}\ u = e_{\varrho}\ v$›.
qed (*use len* **in** *fastforce*)
qed

lemma *two-conjugs-aux*: **assumes** $u{\cdot}v = x{\cdot}y$ **and** $v{\cdot}u = y{\cdot}x$ **and** $u \neq \varepsilon$ **and** $u \neq x$ **and** $|u| \leq |x|$

obtains $r\ s\ k\ l\ m\ n$ **where**
$u = (s \cdot r)^{@} k \cdot s$ **and** $v = (r \cdot s)^{@} l \cdot r$ **and**
$x = (s \cdot r)^{@} m \cdot s$ **and** $y = (r \cdot s)^{@} n \cdot r$ **and**
primitive $(r \cdot s)$ **and** *primitive* $(s \cdot r)$
proof–
have $|u| < |x|$
using ‹$u \neq x$› *eqd-eq(1)*[*OF* ‹$u{\cdot}v = x{\cdot}y$›] *le-neq-implies-less*[*OF* ‹$|u| \leq |x|$›] **by** *blast*
hence $x \neq \varepsilon$
by *force*
from *eqd-lessE*[*OF* ‹$u{\cdot}v = x{\cdot}y$› ‹$|u| < |x|$›]
obtain t **where** $u \cdot t = x\ t \cdot y = v\ t \neq \varepsilon$**.**
from ‹$v{\cdot}u = y{\cdot}x$›[*folded this(1–2)*]
obtain *exp* **where** $y \cdot u = (\varrho\ t)^{@} exp$
using *comm-primroot-exp*[*OF* ‹$t \neq \varepsilon$›, *of* $y \cdot u$] **unfolding** *rassoc* **by** *metis*
hence $0 < exp$
using ‹$u \neq \varepsilon$› **by** *force*
from *split-pow*[*OF* ‹$y \cdot u = (\varrho\ t)^{@} exp$› *this* ‹$u \neq \varepsilon$›]
obtain $r\ s\ n\ k$ **where** $u = (s \cdot r)^{@} k \cdot s\ y = (r \cdot s)^{@} n \cdot r\ r \cdot s = \varrho\ t$
by *metis*
have *primitive* $(r \cdot s)$
unfolding ‹$r \cdot s = \varrho\ t$› **using** ‹$t \neq \varepsilon$› **by** *blast*
hence *primitive* $(s \cdot r)$
using *conjug-prim-iff'* **by** *blast*
define e **where** $e = e_{\varrho}\ t$
have t: $t = (r{\cdot}s)^{@} e$
unfolding ‹$r \cdot s = \varrho\ t$› *e-def* **by** *simp*
have *eq1*: $t \cdot (r \cdot s)\ ^{@}\ n \cdot r = (r \cdot s)\ ^{@}\ (e_{\varrho}\ t + n) \cdot r$
unfolding *pow-add* ‹$r \cdot s = \varrho\ t$› *primroot-exp-eq rassoc***..**
have *eq2*: $((s \cdot r)\ ^{@}\ k \cdot s) \cdot t = (s \cdot r)\ ^{@}\ (k + e) \cdot s$
unfolding t **by** *comparison*
show *thesis*
using *that*[*OF* ‹$u = (s \cdot r)^{@} k \cdot s$› - - ‹$y = (r \cdot s)^{@} n \cdot r$› ‹*primitive* $(r \cdot s)$› ‹*primitive* $(s \cdot r)$›,
folded ‹$u \cdot t = x$› ‹$t \cdot y = v$›, *unfolded* ‹$u = (s \cdot r)^{@} k \cdot s$› ‹$y = (r \cdot s)^{@} n \cdot r$›, *OF eq1 eq2*]**.**
qed

lemma *two-conjugs*: **assumes** $u{\cdot}v = x{\cdot}y$ **and** $v{\cdot}u = y{\cdot}x$ **and** $u \neq \varepsilon$ **and** $x \neq \varepsilon$ **and** $u \neq x$
obtains $r\ s\ k\ l\ m\ n$ **where**
$u = (s \cdot r)^{@} k \cdot s$ **and** $v = (r \cdot s)^{@} l \cdot r$ **and**
$x = (s \cdot r)^{@} m \cdot s$ **and** $y = (r \cdot s)^{@} n \cdot r$ **and**
primitive $(r \cdot s)$ **and** *primitive* $(s \cdot r)$
by (*rule le-cases*[*of* $|u|$ $|x|$],
use two-conjugs-aux[*OF assms(1–3,5)*] **in** *metis*)
(*use two-conjugs-aux*[*OF assms(1–2)*[*symmetric*] *assms(4) assms(5)*[*symmetric*]]
in *metis*)

lemma *fac-pow-pref-conjug*:
assumes $u \leq f\ t^{@}k$
obtains t' **where** $t \sim t'$ **and** $u \leq p\ t'^{@}k$
proof (*cases* $t = \varepsilon$)
assume $t \neq \varepsilon$
obtain $p\ q$ **where** *eq*: $p \cdot u \cdot q = t^{@}k$ **using** *facE'*[*OF assms*].
obtain $i\ r$ **where** $i \leq k$ **and** $r <p\ t$ **and** p: $t^{@}i \cdot r = p$
using *pref-mod-pow*[*OF prefI*[*OF eq*] ‹$t \neq \varepsilon$›].
from ‹$r <p\ t$› **obtain** s **where** t: $r \cdot s = t$..
have *eq'*: $t^{@}i \cdot r \cdot (u \cdot q) = t^{@}k$ **using** *eq* **unfolding** *lassoc p*.
have $u \leq p\ (s \cdot r)^{@}k$ **using** *pow-conjug*[*OF eq' t*] **unfolding** *rassoc*..
with *conjugI'*[*of r s*] **show** *thesis* **unfolding** *t*..
next
assume $t = \varepsilon$
then show *thesis*
using *assms that*[*OF conjug-refl*] *emp-pref emp-pow-emp sublist-Nil-right* **by** *metis*
qed

lemmas *fac-pow-suf-conjug* = *fac-pow-pref-conjug*[*reversed*]

lemma *fac-pow-len-conjug*[*intro*]: **assumes** $|u| = |v|$ **and** $u \leq f\ v^{@}k$ **shows** $v \sim u$
proof–
obtain t **where** $v \sim t$ **and** $u \leq p\ t^{@}k$
using *fac-pow-pref-conjug*[*OF* ‹$u \leq f\ v\ ^{@}\ k$›].
have $u = t$
using *pref-prod-eq*[*OF pref-pow-root*[*OF* ‹$u \leq p\ t^{@}k$›] *conjug-len*[*OF* ‹$v \sim t$›,*folded* ‹$|u| = |v|$›]].
from ‹$v \sim t$›[*folded this*]
show $v \sim u$.
qed

lemma *conjug-fac-sq*:
$u \sim v \Longrightarrow u \leq f\ v \cdot v$
by (*elim conjugE*, *unfold eq-commute*[*of* - · -]) (*intro facI'*, *simp*)

lemma *conjug-fac-pow-conv*: **assumes** $|u| = |v|$ **and** $2 \leq k$
shows $u \sim v \longleftrightarrow u \leq f\ v^{@}k$
proof
assume $u \sim v$
have *f*: $v \cdot v \leq f\ v\ ^{@}k$
using ‹$2 \leq k$› **unfolding** *pow-list-2*[*symmetric*] **using** *le-exps-pref* **by** *blast*
from *fac-trans*[*OF conjug-fac-sq*[*OF* ‹$u \sim v$›] *this*]
show $u \leq f\ v\ ^{@}\ k$.
next
show $u \leq f\ v\ ^{@}\ k \Longrightarrow u \sim v$
using *fac-pow-len-conjug*[*OF* ‹$|u| = |v|$›, *THEN conjug-sym*].
qed

lemma *conjug-fac-Suc*: **assumes** $t \sim v$
shows $t^{@}k \leq_f v^{@}Suc\ k$
proof–
obtain $r\ s$ **where** $v = r \cdot s$ **and** $t = s \cdot r$
using ‹$t \sim v$› **by** *blast*
show *?thesis*
unfolding ‹$v = r \cdot s$› ‹$t = s \cdot r$›
unfolding *pow-list-slide*[*of r k s*, *symmetric*]
by *force*
qed

lemma *fac-pow-conjug*: **assumes** $u \leq_f v^{@}k$ **and** $t \sim v$
shows $u \leq_f t^{@}Suc\ k$
proof–
obtain $r\ s$ **where** $v = r \cdot s$ **and** $t = s \cdot r$
using ‹$t \sim v$› **by** *blast*
have $s \cdot v^{@}k \cdot r = t^{@}Suc\ k$
unfolding ‹$v = r \cdot s$› ‹$t = s \cdot r$› *shift-pow pow-Suc rassoc*..
from *facI*[*of* $v^{@}k\ s\ r$, *unfolded this*]
show $u \leq_f t^{@}Suc\ k$
using ‹$u \leq_f v^{@}k$› **by** *blast*
qed

lemma *border-conjug*: $x \leq_b w \Longrightarrow w^{<-1}x \sim x^{-1>}w$
using *border-conjug-eq conjugI1* **by** *blast*

lemma *count-list-conjug*: **assumes** $u \sim v$ **shows** *count-list u a* = *count-list v a*
proof–
from *conjugE*[*OF* ‹$u \sim v$›]
obtain $r\ s$ **where** $r \cdot s = u\ s \cdot r = v$.
show *count-list u a* = *count-list v a*
unfolding ‹$r \cdot s = u$›[*symmetric*] ‹$s \cdot r = v$›[*symmetric*] *count-list-append* **by** *presburger*
qed

lemma *conjug-in-lists*: $us \sim vs \Longrightarrow vs \in lists\ A \Longrightarrow us \in lists\ A$
unfolding *conjugate-def* **by** *auto*

lemma *conjug-in-lists'*: $us \sim vs \Longrightarrow us \in lists\ A \Longrightarrow vs \in lists\ A$
unfolding *conjugate-def* **by** *auto*

lemma *conjug-in-lists-iff*: $us \sim vs \Longrightarrow us \in lists\ A \longleftrightarrow vs \in lists\ A$
unfolding *conjugate-def* **by** *auto*

lemma *prim-conjug-unique*: **assumes** *primitive* $(u \cdot v)$ **and** $u \cdot v = r \cdot s$ **and** $v \cdot u = s \cdot r$ **and** $u \cdot v \neq v \cdot u$
shows $u = r$ **and** $v = s$
proof–

have $u = r$ **if** *primitive* $(u \cdot v)$ **and** $u \cdot v = r \cdot s$ **and** $v \cdot u = s \cdot r$ **and** $u \cdot v \neq v \cdot u$ **and** $|v| \leq |s|$ **for** $u\ v\ r\ s :: 'a\ list$
proof–
from *eqdE*[*OF* ‹$v \cdot u = s \cdot r$› ‹$|v| \leq |s|$›]
obtain t **where** $v \cdot t = s\ t \cdot r = u$**.**
have $t \cdot (r \cdot v) = (r \cdot v) \cdot t$
unfolding *lassoc* ‹$t \cdot r = u$› **unfolding** *rassoc* ‹$v \cdot t = s$› **by** *fact*
from *comm-not-prim*[*OF - - this*, *unfolded lassoc* ‹$t \cdot r = u$›]
have $t = \varepsilon$
using ‹*primitive* $(u \cdot v)$› ‹$u \cdot v \neq v \cdot u$› **by** *blast*
thus $u = r$
using ‹$t \cdot r = u$› **by** *force*
qed
from *this*[*OF assms*]
this[*OF* ‹*primitive* $(u \cdot v)$›[*unfolded* ‹$u \cdot v = r \cdot s$›] *assms*(*2*−*3*)[*symmetric*] *assms*(*4*)[*unfolded* ‹$u \cdot v = r \cdot s$› ‹$v \cdot u = s \cdot r$›]]
show $u = r$
by *fastforce*
thus $v = s$
using ‹$u \cdot v = r \cdot s$› **by** *fast*
qed

lemma *prim-conjugE*[*elim*, *consumes 3*]: **assumes** $(u \cdot v) \cdot z = z \cdot (v \cdot u)$ **and** *primitive* $(u \cdot v)$ **and** $v \neq \varepsilon$
obtains k **where** $(u \cdot v)^{@}k \cdot u = z$
proof–
from *conjug-eqE*[*OF assms*(*1*) *prim-nemp*[*OF assms*(*2*)]]
obtain $x\ y\ m$ **where** $x \cdot y = u \cdot v$ **and** $y \cdot x = v \cdot u$ **and** $(x \cdot y)^{@}m \cdot x = z$ **and** $y \neq \varepsilon$**.**
from *prim-conjug-unique*[*OF* ‹*primitive* $(u \cdot v)$› ‹$x \cdot y = u \cdot v$›[*symmetric*] ‹$y \cdot x = v \cdot u$›[*symmetric*]]
consider $u \cdot v = v \cdot u \mid u = x \wedge v = y$ **by** *blast*
thus *thesis*
proof (*cases*)
assume $u \cdot v = v \cdot u$
from *comm-not-prim*[*OF -* ‹$v \neq \varepsilon$› *this*] ‹*primitive* $(u \cdot v)$›
have $u = \varepsilon$ **by** *blast*
from ‹$(u \cdot v) \cdot z = z \cdot (v \cdot u)$›[*symmetric*] ‹*primitive* $(u \cdot v)$›
obtain k **where** $z = (u \cdot v)^{@}k \cdot u$
unfolding ‹$u = \varepsilon$› *emp-simps* **by** *blast*
from *that*[*OF this*[*symmetric*]]
show *thesis***.**
next
assume $u = x \wedge v = y$
with ‹$(x \cdot y)^{@}m \cdot x = z$› *that*
show *thesis* **by** *blast*
qed
qed

lemma *prim-conjugE'*[*elim*, *consumes 3*]: **assumes** $(r \cdot s) \cdot z = z \cdot (s \cdot r)$ **and** *primitive* $(r \cdot s)$ **and** $z \neq \varepsilon$
obtains k **where** $(r \cdot s)^{@} k \cdot r = z$
proof (*cases* ‹$s = \varepsilon$›)
assume $s = \varepsilon$
from *assms*(*1–2*)[*unfolded this emp-simps*]
have *primitive* r **and** $z \cdot r = r \cdot z$ **by** *force+*
from *prim-comm-exp*[*OF this*]
obtain k **where** $z = r^{@} k$ $0 < k$
using *nemp-exp-pos*[*OF* ‹$z \neq \varepsilon$›] **by** *metis*
have $r^{@}(k-1) \cdot r = z$
unfolding *pow-pos2*[*OF* ‹$0 < k$›, *of r*, *folded* ‹$z = r^{@} k$›]**..**
from *that*[*unfolded* ‹$s = \varepsilon$› *emp-simps*, *OF this*]
show *thesis***.**
qed (*use prim-conjugE*[*OF assms*(*1–2*)] **in** *blast*)

lemma *conjug-primroots-unique*: **assumes** $x \cdot y \neq y \cdot x$ **and**
$r \cdot s = \varrho\ (x \cdot y)$ **and** $s \cdot r = \varrho\ (y \cdot x)$ **and**
$r' \cdot s' = \varrho\ (x \cdot y)$ **and** $s' \cdot r' = \varrho\ (y \cdot x)$
shows $r = r'$ **and** $s = s'$
proof–
have $x \cdot y \neq \varepsilon$ **and** $y \cdot x \neq \varepsilon$ **and** $x \neq \varepsilon$ **and** $y \neq \varepsilon$ **and** $(x \cdot y) \cdot (y \cdot x) \neq (y \cdot x) \cdot (x \cdot y)$
using ‹$x \cdot y \neq y \cdot x$› *eqd-eq-suf*[*OF - swap-len*, *of* $x \cdot y$ y x $y \cdot x$] **by** *blast+*
show $r = r'$
proof (*rule prim-conjug-unique*(*1*))
from *primroot-prim*[*OF* ‹$x \cdot y \neq \varepsilon$›, *folded* ‹$r \cdot s = \varrho\ (x \cdot y)$›]
show *primitive* $(r \cdot s)$**.**
from ‹$r \cdot s = \varrho\ (x \cdot y)$›[*folded* ‹$r' \cdot s' = \varrho\ (x \cdot y)$›] ‹$s \cdot r = \varrho\ (y \cdot x)$›[*folded* ‹$s' \cdot r' = \varrho\ (y \cdot x)$›]
show $r \cdot s = r' \cdot s'$ **and** $s \cdot r = s' \cdot r'$**.**
show $r \cdot s \neq s \cdot r$
unfolding ‹$r \cdot s = \varrho\ (x \cdot y)$› ‹$s \cdot r = \varrho\ (y \cdot x)$›
using *same-primroots-comm* ‹$(x \cdot y) \cdot (y \cdot x) \neq (y \cdot x) \cdot (x \cdot y)$› **by** *blast*
qed
thus $s = s'$
using ‹$r \cdot s = \varrho\ (x \cdot y)$›[*folded* ‹$r' \cdot s' = \varrho\ (x \cdot y)$›] **by** *blast*
qed

lemma *prim-conjug-pref*: **assumes** *primitive* $(s \cdot r)$ **and** $u \cdot r \cdot s \leq p\ (s \cdot r)^{@} n$ **and** $r \neq \varepsilon$
obtains n **where** $(s \cdot r)^{@} n \cdot s = u$
proof–
have $u \cdot r \cdot s \leq p\ (s \cdot r \cdot u) \cdot r \cdot s$
using *pref-pow-root*[*OF* ‹$u \cdot r \cdot s \leq p\ (s \cdot r)^{@} n$›] **unfolding** *rassoc***.**
from *pref-prod-eq*[*OF this*, *unfolded lenmorph*]
have $(s \cdot r) \cdot u = u \cdot (r \cdot s)$
unfolding *rassoc* **by** *force*
from *prim-conjugE*[*OF this* ‹*primitive* $(s \cdot r)$› ‹$r \neq \varepsilon$›]

show *thesis*
using *that*.
qed

lemma *fac-per-conjug*: **assumes** *period w n* **and** $v \leq f\ w$ **and** $|v| = n$
shows $v \sim$ *take n w*
proof (*cases n = 0*)
assume $n = 0$
then show $v \sim$ *take n w*
using ‹$|v| = n$› **by** *auto*
next
assume $n \neq 0$
have $|take\ n\ w| = |v|$
using *fac-len*[*OF* ‹$v \leq f\ w$›] ‹$|v| = n$› *take-len* **by** *blast*
have $take\ n\ w \neq \varepsilon$
using ‹$|v| = n$› ‹$n \neq 0$› ‹$v \leq f\ w$› **by** *force*
with *per-root-powE*[*OF* ‹*period w n*›[*unfolded period-def*] *this*]
obtain k **where** $w <p\ take\ n\ w\ ^{@}\ k$
by *metis*
from *fac-pow-len-conjug*[*OF* ‹$|take\ n\ w| = |v|$›[*symmetric*], *THEN conjug-sym*]
fac-trans[*OF* ‹$v \leq f\ w$› *pref-fac*, *OF sprefD1*[*OF this*]]
show *?thesis*.
qed

lemma *fac-pers-conjug*: **assumes** *period w n* **and** $v \leq f\ w$ **and** $|v| = n$ **and** $u \leq f\ w$ **and** $|u| = n$
shows $v \sim u$
using *conjug-trans*[*OF fac-per-conjug*[*OF* ‹*period w n*› ‹$v \leq f\ w$› ‹$|v| = n$›]
conjug-sym[*OF fac-per-conjug*[*OF* ‹*period w n*› ‹$u \leq f\ w$› ‹$|u| = n$›]]].

lemma *conjug-pow-powE*: **assumes** $w \sim r^{@}k$ **obtains** s **where** $w = s^{@}k$
proof–
obtain $u\ v$ **where** $w = u \cdot v$ **and** $v \cdot u = r^{@}k$
using *assms* **by** *blast*
have $w = (v^{-1>}(r \cdot v))^{@}k$
unfolding ‹$w = u \cdot v$› *lq-conjug-pow*[*OF pref-pow-root*, *OF prefI*[*OF* ‹$v \cdot u = r\ ^{@}\ k$›], *symmetric*] ‹$v \cdot u = r\ ^{@}\ k$›[*symmetric*]
by *simp*
from *that*[*OF this*]
show *thesis*.
qed

lemma *find-second-letter*: **assumes** $a \neq b$ **and** *set ws* = {*a*,*b*}
shows *dropWhile* (λ *c*. $c = a$) *ws* $\neq \varepsilon$ **and** *hd* (*dropWhile* (λ *c*. $c = a$) *ws*) = *b*
proof–
let *?a* = (λ *c*. $c = a$)

define *wsb* **where** *wsb* = *dropWhile ?a ws* · *takeWhile ?a ws*
have *wsb* $\sim$ *ws*

unfolding *wsb-def* **using** *takeWhile-dropWhile-id*[*of ?a ws*] *conjugI'* **by** *blast*
hence *set wsb* = {*a*,*b*}
using ‹*set ws* = {*a*,*b*}› **by** (*simp add*: *conjug-set*)

have *takeWhile ?a ws* ≠ *ws*
unfolding *takeWhile-eq-all-conv* **using** ‹*set ws* = {*a*,*b*}› ‹*a* ≠ *b*› **by** *simp*
thus *dropWhile ?a ws* ≠ *ε* **by** *simp*
from *hd-dropWhile*[*OF this*] *set-dropWhileD*[*OF hd-in-set*[*OF this*], *unfolded* ‹*set ws* = {*a*,*b*}›]
show *hd* (*dropWhile ?a ws*) = *b*
by *blast*
qed

lemma *fac-conjuq-sq*:
assumes *u* ∼ *v* **and** |*w*| ≤ |*u*| **and** *w* ≤*f* *u* · *u*
shows *w* ≤*f* *v* · *v*
proof −
have *assm-le*: *w* ≤*f* *s* · *r* · *s* · *r*
if *p* · *w* · *q* = *r* · *s* · *r* · *s* **and** |*r*| ≤ |*p*| **for** *w s r p q* :: *'a list*
proof −
obtain *p'* **where** *r* · *p'* = *p*
using ‹*p* · *w* · *q* = *r* · *s* · *r* · *s*› ‹|*r*| ≤ |*p*|› **unfolding** *rassoc* **by** (*rule eqdE*[*OF sym*])
show *w* ≤*f* *s* · *r* · *s* · *r*
using ‹*p* · *w* · *q* = *r* · *s* · *r* · *s*›
by (*intro facI'*[*of p'* - *q* · *r*]) (*simp flip*: ‹*r* · *p'* = *p*›)
qed
obtain *r s* **where** *r* · *s* = *u* **and** *s* · *r* = *v* **using** ‹*u* ∼ *v*›**..**
obtain *p q* **where** *p* · *w* · *q* = *u* · *u* **using** ‹*w* ≤*f* *u* · *u*› **..**
from *lenarg*[*OF this*] ‹|*w*| ≤ |*u*|›
have |*r*| ≤ |*p*| ∨ |*s*| ≤ |*q*|
unfolding ‹*r* · *s* = *u*›[*symmetric*] *lenmorph* **by** *linarith*
then show *w* ≤*f* *v* · *v*
using ‹*p* · *w* · *q* = *u* · *u*› **unfolding** ‹*r* · *s* = *u*›[*symmetric*] ‹*s* · *r* = *v*›[*symmetric*]
by (*elim disjE*) (*simp only*: *assm-le rassoc*, *simp only*: *assm-le*[*reversed*] *lassoc*)
qed

lemma *fac-conjuq-sq-iff*:
assumes *u* ∼ *v* **shows** |*w*| ≤ |*u*| ⟹ *w* ≤*f* *u* · *u* ⟷ *w* ≤*f* *v* · *v*
using *fac-conjuq-sq*[*OF* ‹*u* ∼ *v*›] *fac-conjuq-sq*[*OF* ‹*u* ∼ *v*›[*symmetric*]]
unfolding *conjug-len*[*OF* ‹*u* ∼ *v*›[*symmetric*]]**..**

lemma *map-conjug*:
u ∼ *v* ⟹ *map f u* ∼ *map f v*
by (*elim conjugE*, *unfold eq-commute*[*of* - · -]) *auto*

lemma *concat-map-conjug*:
u ∼ *v* ⟹ *concat* (*map f u*) ∼ *concat* (*map f v*)
by (*intro conjug-concat-conjug map-conjug*)

lemma *map-conjug-iff* [*reversal-rule*]:
assumes *inj f* **shows** *map f u* $\sim$ *map f v* $\longleftrightarrow$ *u* $\sim$ *v*
using *map-conjug map-conjug*[*of map f u map f v inv f*]
unfolding *map-map inv-o-cancel*[*OF* ‹*inj f*›] *list.map-id* **by** (*intro iffI*)

lemma *set-conjug*: **assumes** $w \neq \varepsilon$
shows $\{w'.\ w \sim w'\} = \{rotate\ n\ w \mid n.\ n < |\varrho\ w|\}$
unfolding *set-eq-iff*
unfolding *mem-Collect-eq conjug-rotate-iff*
using *rotate-pow-mod*[*of - e*$_\varrho$ *w* ϱ *w*] *mod-less-divisor*[*OF nemp-len*[*OF prim-root-nemp*[*OF* ‹*w* $\neq \varepsilon$›]]]
unfolding *primroot-exp-eq* **by** *blast*

lemma *card-conjug*: **assumes** $w \neq \varepsilon$
shows *card* $\{w'.\ w \sim w'\} = |\varrho\ w|$
proof−
have *inj-on* ($\lambda n.$ *rotate n w*) $\{i.\ i < |\varrho\ w|\}$
proof (*rule inj-onI*, *unfold mem-Collect-eq*)
fix *x y*
assume $x < |\varrho\ w|$ $y < |\varrho\ w|$ *rotate x w* = *rotate y w*
have *roxy*: *rotate x* (ϱ *w*) = *rotate y* (ϱ *w*)
unfolding *primroot-rotate-comm*[*OF assms*, *symmetric*] ‹*rotate x w* = *rotate y w*›**..**
show *x* = *y*
using *prim-no-rotate*[*OF primroot-prim*[*OF* ‹*w* $\neq \varepsilon$›]]
rotate-back'[*OF roxy*] *rotate-back'*[*OF roxy*[*symmetric*]] ‹*x* $<|\varrho\ w|$› ‹*y* $<|\varrho\ w|$›
using *less-imp-diff-less linorder-neqE-nat zero-less-diff* **by** *meson*
qed
from *card-image*[*OF this*]
show *?thesis*
unfolding *set-conjug*[*OF assms*]
unfolding *card-Collect-less-nat*
unfolding *setcompr-eq-image***.**
qed

lemma *finite-Bex-conjug*: **assumes** *finite A*
shows *finite* $\{r.$ *Bex A* (*conjugate r*)$\}$
unfolding *finite-Collect-bex*[*OF* ‹*finite A*›, *of conjugate*]
proof
fix *y*
assume $y \in A$
show *finite* $\{r.\ r \sim y\}$
proof(*cases* $y = \varepsilon$)
case *True*
then show *?thesis*
unfolding *conjug-swap*[*of - y*]
by (*metis* (*mono-tags*, *opaque-lifting*) ‹$y \in A$› *assms conjug-nemp-iff finite-subset mem-Collect-eq subset-eq*)

next
case *False*
show *?thesis*
unfolding *conjug-swap*[*of - y*] **using** *card-conjug*[*OF* ‹$y \neq \varepsilon$›]
using *nemp-len*[*OF primroot-nemp*, *OF* ‹$y \neq \varepsilon$›]
card-ge-0-finite **by** *metis*
qed
qed

2.21.1 Enumerating conjugates

definition *bounded-conjug*
where *bounded-conjug* w' w $k \equiv (\exists\ n \leq k.\ w = rotate\ n\ w')$

named-theorems *bounded-conjug*

lemma[*bounded-conjug*]: *bounded-conjug* w' w $0 \longleftrightarrow w = w'$
unfolding *bounded-conjug-def* **by** *auto*

lemma[*bounded-conjug*]: *bounded-conjug* w' w (*Suc* k) $\longleftrightarrow$ *bounded-conjug* w' w k $\vee$ $w = rotate\ (Suc\ k)\ w'$
unfolding *bounded-conjug-def* **using** *le-SucE le-imp-less-Suc le-less* **by** *metis*

lemma[*bounded-conjug*]: $w' \sim w \longleftrightarrow$ *bounded-conjug* w w' ($|w|-1$)
unfolding *bounded-conjug-def conjug-swap*[*of* w'] **using** *conjug-rotate-iff-le*.

lemma $w \sim [a,b,c] \longleftrightarrow w = [a,b,c] \vee w = [b,c,a] \vee w = [c,a,b]$
by (*simp add*: *bounded-conjug*)

2.21.2 General lemmas using conjugation

lemma *switch-fac*: **assumes** $x \neq y$ **and** *set* $ws = \{x,y\}$ **shows** $[x,y] \simeq_f ws \cdot ws$
proof−
let *?y* = (λ *a*. $a = y$) **and** *?x* = (λ *a*. $a = x$)
have $ws \neq \varepsilon$
using ‹*set* $ws = \{x,y\}$› **by** *force*

define *wsx* **where** *wsx* = *dropWhile ?y ws* · *takeWhile ?y ws*
have *wsx* $\sim$ *ws*
unfolding *wsx-def* **using** *takeWhile-dropWhile-id*[*of ?y ws*] *conjugI'* **by** *blast*
have *set wsx* = $\{x,y\}$
unfolding *wsx-def* **using** ‹*set ws* = $\{x,y\}$› *conjugI' conjug-set takeWhile-dropWhile-id* **by** *metis*
from *find-second-letter*[*OF* ‹$x \neq y$›[*symmetric*] ‹*set ws* = $\{x,y\}$›[*unfolded insert-commute*[*of x*]]]
have *dropWhile* ($\lambda c.\ c = y$) *ws* $\neq \varepsilon$ **and** *hd wsx* = x
unfolding *wsx-def* **using** *hd-append* **by** *simp-all*
hence *takeWhile ?x wsx* $\neq \varepsilon$
unfolding *wsx-def takeWhile-eq-Nil-iff* **by** *blast*
obtain k **where** k [*symmetric*]: $[x]^{@}k$ = *takeWhile ?x wsx*

by (*metis takeWhile-idem takeWhile-sing-pow*)

have $0 < k$
using ‹*takeWhile ?x wsx* ≠ *ε*›[*unfolded k*] **by** *blast*
note *find-second-letter*[*OF* ‹$x \neq y$› ‹*set wsx* = {*x*,*y*}›]
have *wsx* = $[x]^{@}(k - 1) \cdot [x] \cdot [hd\ (dropWhile\ ?x\ wsx)] \cdot tl\ (dropWhile\ ?x\ wsx)$
unfolding *lassoc pow-pos2*[*OF* ‹*0* < *k*›,*symmetric*] ‹*takeWhile ?x wsx* = $[x]^{@}k$›[*symmetric*]
unfolding *rassoc hd-tl*[*OF* ‹*dropWhile ?x wsx* ≠ *ε*›] *takeWhile-dropWhile-id*..
from *this*[*unfolded* ‹*hd* (*dropWhile ?x wsx*) = *y*›]
have [*x*,*y*] ≤*f wsx* **by** (*auto simp add*: *fac-def*)
thus [*x*,*y*] ≤*f ws* · *ws*
using *fac-trans*[*OF* - *conjug-fac-sq*[*OF* ‹*wsx* ∼ *ws*›]] **by** *blast*
qed

lemma *imprim-ext-pref-comm*: **assumes** ¬ *primitive* (*u* · *v*) **and** ¬ *primitive* (*u* · *v* · *u*)
shows *u* · *v* = *v* · *u*
using ‹¬ *primitive* (*u* · *v*)› **proof** (*elim not-prim-primroot-expE*)
fix *z n* **assume** *z* $^{@}$ *n* = *u* · *v* **and** *2* ≤ *n*
have *2* ∗ |*z*| ≤ |*u* · *v* · *u*|
by (*simp add*: *pow-len* ‹*2* ≤ *n*› *trans-le-add1 flip*: ‹$z^{@}n$ = *u* · *v*› *rassoc*)
moreover have *u* · *v* · *u* ≤*p z* · *u* · *v* · *u*
by (*intro pref-pow-root*[*of* - *n* + *n*]) (*simp add*: ‹*z* $^{@}$ *n* = *u* · *v*› *pow-add*)
ultimately have (*u* · *v* · *u*) · *z* = *z* · *u* · *v* · *u*
using ‹¬ *primitive* (*u* · *v* · *u*)› *per-le-prim-iff*
by (*cases z* = *ε*) *blast*+
from *comm-pow-comm*[*OF this*[*symmetric*], *of n*]
show *u* · *v* = *v* · *u*
unfolding ‹*z* $^{@}$ *n* = *u* · *v*› **by** *simp*
qed

lemma *imprim-ext-suf-comm*:
¬ *primitive* (*u* · *v*) ⟹ ¬ *primitive* (*u* · *v* · *v*) ⟹ *u* · *v* = *v* · *u*
by (*intro imprim-ext-pref-comm*[*symmetric*])
(*unfold conjug-prim-iff*[*OF conjugI'*, *of v*] *rassoc*)

lemma *prim-xyky*: **assumes** *2* ≤ *k* **and** ¬ *primitive* ($(x \cdot y)^{@}k \cdot y$) **shows** *x* · *y* = *y* · *x*
proof−
have *k* ≠ *0* **using** ‹*2* ≤ *k*› **by** *simp*
have $(x \cdot y)^{@}k = (x \cdot y)^{@}(k - 1) \cdot x \cdot y$
unfolding *rassoc pow-Suc2*[*symmetric*] *Suc-minus*[*OF* ‹*k* ≠ *0*›]..
have $(x \cdot y)^{@}k \cdot y = ((x \cdot y)^{@}(k - 1) \cdot x) \cdot y \cdot y$
unfolding *lassoc cancel-right* **unfolding** *rassoc pow-Suc2*[*symmetric*] *Suc-minus*[*OF* ‹*k* ≠ *0*›]..
from *imprim-ext-suf-comm*[*OF* - ‹¬ *primitive* ($(x \cdot y)^{@}k \cdot y$)›[*unfolded this*],
unfolded rassoc pow-Suc2[*symmetric*] *Suc-minus*[*OF* ‹*k* ≠ *0*›], *OF pow-nemp-imprim*[*OF* ‹*2* ≤ *k*›]]

show $x \cdot y = y \cdot x$
unfolding ‹$(x \cdot y)^{@}k = (x \cdot y)^{@}(k - 1) \cdot x \cdot y$› *shift-pow*
pow-Suc2[*of* - $x \cdot y$, *unfolded rassoc*, *symmetric*] *pow-Suc*[*of* - $y \cdot x$, *unfolded rassoc*, *symmetric*]
using *pow-eq-eq* **by** *blast*
qed

lemma *fac-pow-div*: **assumes** $u \leq f\ w^{@}l$ *primitive* w
shows $w^{@}((|u|\ div\ |w|) - 1) \leq f\ u$
proof–
obtain w' **where**
$w \sim w'$ **and**
$u \leq p\ w'^{@}\ l$
using *fac-pow-pref-conjug*[*OF* ‹$u \leq f\ w^{@}l$›].

note *prim-nemp*[*OF* ‹*primitive* w›]
hence $w' \neq \varepsilon$
using *conjug-nemp-iff* ‹$w \sim w'$› **by** *blast*

obtain s **where** $s <p\ w'$ **and** $w'^{@}\ (|u|\ div\ |w'|) \cdot s = u$
using *per-root-modE'*[*OF pref-pow-root*, *OF* ‹$u \leq p\ w'^{@}\ l$› ‹$w' \neq \varepsilon$›].
have $w^{@}((|u|\ div\ |w|) - 1) \leq f\ w'^{@}\ (|u|\ div\ |w'|)$
unfolding *conjug-len*[*OF* ‹$w \sim w'$›]
using *conjug-fac-Suc*[*OF* ‹$w \sim w'$›]
by (*cases* $(|u|\ div\ |w'|) = 0$, *force*)
(*use Suc-minus* **in** *metis*)
thus *?thesis*
using *fac-ext-suf*[*of* - $w'^{@}\ (|u|\ div\ |w'|)\ s$, *unfolded* ‹$w'^{@}\ (|u|\ div\ |w'|) \cdot s = u$›]
by *presburger*
qed

2.22 Element of lists: a method for testing if a word is in lists A

lemma *append-in-lists*[*simp*, *intro*]: $u \in lists\ A \Longrightarrow v \in lists\ A \Longrightarrow u \cdot v \in lists\ A$
by *simp*

lemma *pref-in-lists*: $u \leq p\ v \Longrightarrow v \in lists\ A \Longrightarrow u \in lists\ A$
by (*auto simp add*: *prefix-def*)

lemmas *suf-in-lists* = *pref-in-lists*[*reversed*]

lemma *fac-in-lists*: $ws \in lists\ S \Longrightarrow vs \leq f\ ws \Longrightarrow vs \in lists\ S$
by *force*

lemma *lq-in-lists*: $v \in lists\ A \Longrightarrow u^{-1>}v \in lists\ A$
unfolding *left-quotient-def* **using** *fac-in-lists*[*OF* - *sublist-drop*].

lemmas *rq-in-lists* = *lq-in-lists*[*reversed*]

lemma *take-in-lists*: $w \in \mathit{lists}\ A \Longrightarrow \mathit{take}\ j\ w \in \mathit{lists}\ A$
using *pref-in-lists*[*OF take-is-prefix*].

lemma *drop-in-lists*: $w \in \mathit{lists}\ A \Longrightarrow \mathit{drop}\ j\ w \in \mathit{lists}\ A$
using *suf-in-lists*[*OF suffix-drop*].

lemma *lcp-in-lists*: $u \in \mathit{lists}\ A \Longrightarrow\ u \wedge_p v \in \mathit{lists}\ A$
using *pref-in-lists*[*OF lcp-pref*].

lemma *lcp-in-lists'*: $v \in \mathit{lists}\ A \Longrightarrow\ u \wedge_p v \in \mathit{lists}\ A$
using *pref-in-lists*[*OF lcp-pref'*].

lemma *append-in-lists-dest*[*elim*]: $u \cdot v \in \mathit{lists}\ A \Longrightarrow u \in \mathit{lists}\ A$
by *simp*

lemma *append-in-lists-dest'*: $u \cdot v \in \mathit{lists}\ A \Longrightarrow v \in \mathit{lists}\ A$
by *simp*

lemma *pow-in-lists*: $u \in \mathit{lists}\ A \Longrightarrow u^{@}k \in \mathit{lists}\ A$
by (*induct k*) *auto*

lemma *takeWhile-in-list*: $u \in \mathit{lists}\ A \Longrightarrow \mathit{takeWhile}\ P\ u \in \mathit{lists}\ A$
using *take-in-lists*[*of u - |takeWhile P u|, folded takeWhile-eq-take*].

lemma *rev-in-lists*: $u \in \mathit{lists}\ A \Longrightarrow \mathit{rev}\ u \in \mathit{lists}\ A$
by *auto*

lemma *append-in-lists-dest1* [*elim*]: $u \cdot v = w \Longrightarrow w \in \mathit{lists}\ A \Longrightarrow u \in \mathit{lists}\ A$
by *auto*

lemma *append-in-lists-dest2*: $u \cdot v = w \Longrightarrow w \in \mathit{lists}\ A \Longrightarrow v \in \mathit{lists}\ A$
by *auto*

lemma *pow-in-lists-dest1*: $u \cdot v = w^{@}n \Longrightarrow w \in \mathit{lists}\ A \Longrightarrow u \in \mathit{lists}\ A$
using *append-in-lists-dest pow-in-lists* **by** *metis*

lemma *pow-in-lists-dest1-sym*: $w^{@}n = u \cdot v \Longrightarrow w \in \mathit{lists}\ A \Longrightarrow u \in \mathit{lists}\ A$
using *append-in-lists-dest pow-in-lists* **by** *metis*

lemma *pow-in-lists-dest2*: $u \cdot v = w^{@}n \Longrightarrow w \in \mathit{lists}\ A \Longrightarrow v \in \mathit{lists}\ A$
using *append-in-lists-dest' pow-in-lists* **by** *metis*

lemma *pow-in-lists-dest2-sym*: $w^{@}n = u \cdot v \Longrightarrow w \in \mathit{lists}\ A \Longrightarrow v \in \mathit{lists}\ A$
using *append-in-lists-dest' pow-in-lists* **by** *metis*

lemma *per-in-lists*: $w <_p r \cdot w \Longrightarrow r \in \mathit{lists}\ A \Longrightarrow w \in \mathit{lists}\ A$

using *pow-in-lists*[*of r A*] *pref-in-lists per-root-pow-conv* **by** *metis*

lemma *nth-in-lists*: $j < |w| \Longrightarrow w \in lists\ A \Longrightarrow w\ !\ j \in A$
using *in-lists-conv-set nth-mem* **by** *force*

method *inlists* =
(*insert method-facts*, *use nothing* **in** ‹
((*elim suf-in-lists* | *elim pref-in-lists*[*elim-format*] | *rule lcp-in-lists* | *rule drop-in-lists*
|
rule lq-in-lists | *rule rq-in-lists* | *rule lists-butlast* | *rule tl-in-lists* |
rule take-in-lists | *intro lq-in-lists* | *rule nth-in-lists* |
rule append-in-lists | *elim conjug-in-lists* | *rule pow-in-lists* | *rule takeWhile-in-list*
| *elim append-in-lists-dest1* | *elim append-in-lists-dest2*
| *elim pow-in-lists-dest2* | *elim pow-in-lists-dest2-sym*
| *elim pow-in-lists-dest1* | *elim pow-in-lists-dest1-sym*)
| (*simp* | *fact*))+›)

2.23 Reversed mappings

definition *rev-map* :: ('*a list* ⇒ '*b list*) ⇒ ('*a list* ⇒ '*b list*) **where**
rev-map f = *rev* ∘ *f* ∘ *rev*

lemma *rev-map-idemp*[*simp*]: *rev-map* (*rev-map f*) = *f*
unfolding *rev-map-def* **by** *auto*

lemma *rev-map-arg*: *rev-map f u* = *rev* (*f* (*rev u*))
by (*simp add*: *rev-map-def*)

lemma *rev-map-arg'*: *rev* ((*rev-map f*) *w*) = *f* (*rev w*)
by (*simp add*: *rev-map-def*)

lemmas *rev-map-arg-rev*[*reversal-rule*] = *rev-map-arg*[*reversed add*: *rev-rev-ident*]

lemma *rev-map-sing*: *rev-map f* [*a*] = *rev* (*f* [*a*])
unfolding *rev-map-def* **by** *simp*

lemma *rev-maps-eq-iff*[*simp*]: *rev-map g* = *rev-map h* ⟷ *g* = *h*
using *arg-cong*[*of rev-map g rev-map h rev-map*, *unfolded rev-map-idemp*] **by** *fast*

lemma *rev-map-funpow*[*reversal-rule*]: (*rev-map* (*f*::'*a list* ⇒'*a list*)) ^^ *k* = *rev-map* (*f* ^^ *k*)
unfolding *funpow.simps rev-map-def*
by(*induct k*, *simp*+)

2.24 Overlapping powers, periods, prefixes and suffixes

lemma *pref-suf-overlapE*: **assumes** $p \leq p\ w$ **and** $s \leq s\ w$ **and** $|w| \leq |p| + |s|$
obtains *p1 u s1* **where** $p1 \cdot u \cdot s1 = w$ **and** $p1 \cdot u = p$ **and** $u \cdot s1 = s$
proof–
define *u* **where** $u = (w^{<-1}s)^{-1>}p$
have $u \leq s\ p$
unfolding *u-def lq-def* **using** *suffix-drop*.
obtain *p1 s1* **where** $p1 \cdot u = p$ **and** $p \cdot s1 = w$
using *suffixE*[*OF* ‹$u \leq s\ p$›] *prefixE*[*OF* ‹$p \leq p\ w$›] **by** *metis*
note ‹$p \cdot s1 = w$›[*folded* ‹$p1 \cdot u = p$›, *unfolded rassoc*]

have $|s1| \leq |s|$
using ‹$|w| \leq |p| + |s|$›[*folded* ‹$p \cdot s1 = w$›, *unfolded lenmorph*] **by** *force*
hence $s1 \leq s\ s$
using ‹$p \cdot s1 = w$› ‹$s \leq s\ w$› *suf-prod-long* **by** *blast*

from *rq-lq-assoc*[*OF rq-is-pref*, *of s1*] *u-def*[*folded rqI*[*OF* ‹$p \cdot s1 = w$›]]
have $u = s^{<-1}s1$
using *suf-rq-lq-id*[*OF* ‹$s \leq s\ w$›] ‹$s1 \leq s\ s$› **by** *presburger*
hence $u \cdot s1 = s$
using *rq-suf*[*OF* ‹$s1 \leq s\ s$›] **by** *blast*

from *that*[*OF* ‹$p1 \cdot u \cdot s1 = w$› ‹$p1 \cdot u = p$› *this*]
show *thesis*.
qed

lemma *mid-sq*: **assumes** $p{\cdot}x{\cdot}q{=}x{\cdot}x$ **shows** $x{\cdot}p{=}p{\cdot}x$ **and** $x{\cdot}q{=}q{\cdot}x$
proof–
have $(x{\cdot}p){\cdot}x{\cdot}q = (p{\cdot}x){\cdot}q{\cdot}x$
using *assms* **by** *auto*
from *eqd-eq*[*OF this*]
show $x{\cdot}p{=}p{\cdot}x$ **and** $x{\cdot}q{=}q{\cdot}x$
by *simp*+
qed

lemma *mid-sq′*: **assumes** $p{\cdot}x{\cdot}q{=}x{\cdot}x$ **shows** $q \cdot p = x$ **and** $p \cdot q = x$
proof–
have $p{\cdot}q{\cdot}x = x{\cdot}x$
using *assms*[*unfolded mid-sq*(*2*)[*OF assms*]].
thus $p{\cdot}q = x$ **by** *auto*
from *assms*[*folded this*] *this*
show $q{\cdot}p = x$ **by** *fastforce*
qed

lemma *mid-sq-pref*: $p \cdot u \leq p\ u \cdot u \Longrightarrow p \cdot u = u \cdot p$
using *mid-sq*(*1*)[*symmetric*] **unfolding** *prefix-def rassoc* **by** *metis*

lemmas *mid-sq-suf* = *mid-sq-pref*[*reversed*]

lemma *mid-sq-pref-suf*: **assumes** $p{\cdot}x{\cdot}q{=}x{\cdot}x$ **shows** $p \leq p\ x$ **and** $p \leq s\ x$ **and** $q \leq p\ x$ **and** $q \leq s\ x$
using *assms mid-sq'*[*OF assms*] **by** *blast+*

lemma *mid-sq-primroot*: **assumes** $p \cdot x <_p x \cdot x$
shows $x \cdot x = p \cdot x \cdot (\varrho\ x)^{@}(e_\varrho\ ((p \cdot x)^{-1>}(x \cdot x)))$ (**is** $x \cdot x = p \cdot x \cdot (\varrho\ x)^{@}e_\varrho$ *?z*)
$0 < e_\varrho\ ((p \cdot x)^{-1>}(x \cdot x))$ (**is** $0 < e_\varrho$ *?z*)
proof–
from *assms lq-spref-nemp*[*OF assms*]
have $x \neq \varepsilon$
by *fastforce*
from *lq-spref*[*OF assms*, *unfolded rassoc*] *lq-spref-nemp*[*OF assms*]
have $\varrho\ x = \varrho$ *?z*
using *mid-sq*(*2*) *comm-primroots*[*OF* ‹$x \neq \varepsilon$›] **by** *meson*
from ‹$p \cdot x \cdot$ *?z* $= x \cdot x$›[*symmetric*, *folded this*]
show $x \cdot x = p \cdot x \cdot (\varrho\ x)^{@}e_\varrho$ *?z*
unfolding *primroot-exp-eq*[*of ?z*, *folded* ‹$\varrho\ x = \varrho$ *?z*›].
show $0 < e_\varrho$ *?z*
using ‹*?z* $\neq \varepsilon$› **by** *blast*
qed

lemma *mid-pow*: **assumes** $p{\cdot}x^{@}(Suc\ l){\cdot}q = x^{@}k$
shows $x{\cdot}p{=}p{\cdot}x$ **and** $x{\cdot}q{=}q{\cdot}x$
proof–
have $x{\cdot}p{\cdot}x^{@}l{\cdot}x{\cdot}q = x{\cdot}(p{\cdot}x^{@}Suc\ l \cdot q)$
by *comparison*
also have ... $= (p{\cdot}x^{@}Suc\ l \cdot q) \cdot x$
unfolding *rassoc assms* **by** *comparison*
also have ... = $p{\cdot}x{\cdot}x^{@}l{\cdot}q{\cdot}x$ **by** *simp*
finally have *eq*: $x{\cdot}p{\cdot}x^{@}l{\cdot}x{\cdot}q = p{\cdot}x{\cdot}x^{@}l{\cdot}q{\cdot}x$**.**

have $(x{\cdot}p){\cdot}x^{@}l{\cdot}x{\cdot}q = (p{\cdot}x){\cdot}x^{@}l{\cdot}q{\cdot}x$
using *eq* **unfolding** *rassoc*.
from *eqd-comp*[*OF this*]
show $x{\cdot}p = p{\cdot}x$
using *comm-ruler* **by** *blast*

have $(x{\cdot}p{\cdot}x^{@}l){\cdot}(x{\cdot}q) = (x{\cdot}p{\cdot}x^{@}l){\cdot}(q{\cdot}x)$
using *eq* **unfolding** *lassoc* ‹$x{\cdot}p = p{\cdot}x$›.
from *this*[*unfolded cancel*]
show $x{\cdot}q = q{\cdot}x$**.**
qed

lemma *root-suf-comm*: **assumes** $x \leq_p r \cdot x$ **and** $r \leq_s r \cdot x$ **shows** $r \cdot x = x \cdot r$
proof–
have $r \cdot x = x \cdot x^{-1>}(r \cdot x)$

using *lq-pref*[*OF* ‹$x \leq_p r \cdot x$›, *symmetric*].
from *this* **and** *eq-conjug-len*[*OF this*]
have $r = x^{-1>}(r \cdot x)$
using *lq-pref*[*OF* ‹$x \leq_p r \cdot x$›] *suf-ruler-eq-len*[*OF* ‹$r \leq_s r \cdot x$›, *of* $x^{-1>}(r \cdot x)$]
by *blast*
from ‹$r \cdot x = x \cdot x^{-1>}(r \cdot x)$›[*folded this*]
show $r \cdot x = x \cdot r$.
qed

lemma *pref-marker*: **assumes** $w \leq_p v \cdot w$ **and** $u \cdot v \leq_p w$
shows $u \cdot v = v \cdot u$
using *append-prefixD*[*OF* ‹$u \cdot v \leq_p w$›] *comm-ruler*[*OF* ‹$u \cdot v \leq_p w$›, *of* $v \cdot w$, *unfolded same-prefix-prefix*]
‹$w \leq_p v \cdot w$› **by** *blast*

lemma *pref-marker-ext*: **assumes** $|x| \leq |y|$ **and** $v \neq \varepsilon$ **and** $y \cdot v \leq_p x \cdot v^{@}k$
obtains n **where** $y = x \cdot (\varrho\ v)^{@}n$
proof–
note *pref-prod-long-ext*[*OF* ‹$y \cdot v \leq_p x \cdot v^{@}k$› ‹$|x| \leq |y|$›]
have $x^{-1>}y \cdot v \leq_p v^{@}k$
using *pref-cancel-lq-ext*[*OF* ‹$y \cdot v \leq_p x \cdot v^{@}k$› ‹$|x| \leq |y|$›].
from *pref-marker*[*OF* - *this*]
have $x^{-1>}y \cdot v = v \cdot x^{-1>}y$
unfolding *pow-comm*[*symmetric*] **by** *blast*
then obtain n **where** $x^{-1>}y = (\varrho\ v)^{@}n$
using ‹$v \neq \varepsilon$›
using *comm-primroots pow-zero primroot-expE* **by** *metis*
hence $y = x \cdot (\varrho\ v)^{@}n$
using ‹$x \leq_p y$› **by** (*auto simp add*: *prefix-def*)
from *that*[*OF this*] **show** *thesis*.
qed

lemma *pref-marker-sq*: $p \cdot x \leq_p x \cdot x \Longrightarrow p \cdot x = x \cdot p$
using *pref-marker same-prefix-prefix triv-pref* **by** *metis*

lemmas *suf-marker-sq* = *pref-marker-sq*[*reversed*]

lemma *pref-marker-conjug*: **assumes** $w \neq \varepsilon$ **and** $w \cdot r \cdot s \leq_p s \cdot (r \cdot s)^{@}m$ **and** *primitive* $(r \cdot s)$
obtains n **where** $w = s \cdot (r \cdot s)^{@}n$
proof–
have $(r \cdot w) \cdot r \cdot s \leq_p (r \cdot s)^{@}Suc\ m$
using ‹$w \cdot r \cdot s \leq_p s \cdot (r \cdot s)^{@}m$› **by** *auto*
from *pref-marker*[*OF* - *this*, *folded pow-comm*, *OF triv-pref*]
have $(r \cdot w) \cdot r \cdot s = (r \cdot s) \cdot r \cdot w$.
from *comm-primroots'*[*OF* - *prim-nemp*[*OF* ‹*primitive* $(r \cdot s)$›] *this*, *unfolded prim-primroot*[*OF* ‹*primitive* $(r \cdot s)$›]]
have $\varrho\ (r \cdot w) = r \cdot s$
using ‹$w \neq \varepsilon$› **by** *blast*

then obtain n **where** $r \cdot w = (r \cdot s)^{@} n\ 0 < n$
using ‹$w \neq \varepsilon$› *primroot-expE* **by** *metis*
thus *thesis*
using *pow-pos*[*OF* ‹$0 < n$›, *of* $r \cdot s$, *folded* ‹$r \cdot w = (r \cdot s)^{@} n$›, *unfolded rassoc cancel*] *that* **by** *force*
qed

lemmas *pref-marker-reversed* = *pref-marker*[*reversed*]

lemma *suf-marker-per-root*: **assumes** $w \leq_p v \cdot w$ **and** $p \cdot v \cdot u \leq_p w$
shows $u \leq_p v \cdot u$
proof–
have $p \cdot v = v \cdot p$
using *pref-marker*[*OF* ‹$w \leq_p v \cdot w$›, *of* p] ‹$p \cdot v \cdot u \leq_p w$› **by** (*auto simp add*: *prefix-def*)
from *pref-trans*[*OF* ‹$p \cdot v \cdot u \leq_p w$›[*unfolded lassoc this*, *unfolded rassoc*] ‹$w \leq_p v \cdot w$›]
have $p \cdot u \leq_p w$
using *pref-cancel* **by** *auto*
from *ruler-le*[*OF this* ‹$p \cdot v \cdot u \leq_p w$›]
have $p \cdot u \leq_p p \cdot v \cdot u$
by *force*
thus *?thesis*
unfolding *pref-cancel-conv*.
qed

lemma *suf-marker-per-root'*: **assumes** $w \leq_p v \cdot w$ **and** $p \cdot v \cdot u \leq_p w$ **and** $v \neq \varepsilon$
shows $u \leq_p p \cdot u$
proof–
have $p \cdot v = v \cdot p$
using *pref-marker*[*OF* ‹$w \leq_p v \cdot w$›, *of* p] ‹$p \cdot v \cdot u \leq_p w$› **by** (*fastforce simp add*: *prefix-def*)
from *root-comm-root*[*OF suf-marker-per-root*[*OF* ‹$w \leq_p v \cdot w$› ‹$p \cdot v \cdot u \leq_p w$›] *this* ‹$v \neq \varepsilon$›]
show $u \leq_p p \cdot u$
by *blast*
qed

lemma *marker-fac-pref*: **assumes** $u \leq_f r^{@} k$ **and** $r \leq_p u$ **shows** $u \leq_p r^{@} k$
using *assms*
proof (*cases* $r = \varepsilon$)
assume $r \neq \varepsilon$
have $|u| \leq |r^{@} k|$
using ‹$u \leq_f r^{@} k$› **by** *force*
obtain u' **where** $r \cdot u' = u$
using ‹$r \leq_p u$› **by** (*auto simp add*: *prefix-def*)
obtain $p\ s$ **where** $p \cdot u \cdot s = r^{@} k$
using ‹$u \leq_f r^{@} k$› **by** *blast*
from *suf-marker-per-root*[*of* $r^{@} k\ r\ p\ u' \cdot s$, *folded pow-comm*, *OF triv-pref*]

have $u' \cdot s \leq_p r \cdot (u' \cdot s)$
using ‹$p \cdot u \cdot s = r^{@}k$›[*folded* ‹$r \cdot u' = u$›, *unfolded rassoc*] **by** *fastforce*
hence $u' \cdot s \leq_p r^{@}k \cdot (u' \cdot s)$
using *per-exp-pref* **by** *blast*
hence $u \leq_p (r^{@}k \cdot r) \cdot (u' \cdot s)$
unfolding ‹$r \cdot u' = u$›[*symmetric*] *pow-Suc2*[*symmetric*] *pow-Suc rassoc*
by (*auto simp add*: *prefix-def*)
thus $u \leq_p r^{@}k$
unfolding *rassoc* **using** ‹$|u| \leq |r^{@}k|$› **by** *blast*
next
assume $r = \varepsilon$
hence $u = \varepsilon$
using *assms*(*1*) *emp-pow-emp sublist-Nil-right* **by** *metis*
then show *?thesis*
by *blast*
qed

lemma *marker-fac-pref-len*: **assumes** $u \leq_f r^{@}k$ **and** $t \leq_p u$ **and** $|t| = |r|$
shows $u \leq_p t^{@}k$
proof–
have $|u| \leq |r^{@}k|$
using ‹$u \leq_f r^{@}k$› **by** *force*
hence $|u| \leq |t^{@}k|$
unfolding *pow-len* ‹$|t| = |r|$›**.**
have $t \leq_f r^{@}k$
using *assms* **by** *blast*
hence $t \sim r$
using ‹$|t| = |r|$› **by** (*simp add*: *conjug-sym fac-pow-len-conjug*)
from *fac-pow-conjug*[*OF* ‹$u \leq_f r^{@}k$› *this*]
have $u \leq_p t^{@}Suc\ k$
using *marker-fac-pref*[*OF* - ‹$t \leq_p u$›] **by** *blast*
thus $u \leq_p t^{@}k$
using ‹$|u| \leq |t^{@}k|$› **unfolding** *pow-Suc2* **by** *blast*
qed

lemma *pref-comm-cancel*: $u \cdot z \leq_p v \Longrightarrow u \cdot v = v \cdot u \Longrightarrow z \leq_p v$
unfolding *pref-cancel-conv*[*of u z v*, *symmetric*] **using** *prefix-prefix*[*of* $u \cdot z\ v\ u$]
by *argo*

lemma *marker-fac-pref-ext*: **assumes** $w \leq_f v^{@}k\ v = p \cdot s\ s \cdot v \bowtie w\ |v| \leq |w|$
shows $p \cdot w \leq_p v \cdot p \cdot w$
proof–
have $s \cdot p \leq_p w$
using ‹$s \cdot v \bowtie w$› ‹$|v| \leq |w|$›
unfolding ‹$v = p \cdot s$› *swap-len*[*of p*] *lassoc*
using *comp-monotone comp-shorter prefI* **by** *meson*
moreover have $w \leq_f (s \cdot p)^{@}Suc\ k$
using ‹$w \leq_f v^{@}k$› **unfolding** ‹$v = p \cdot s$›
using *conjugI' fac-pow-conjug* **by** *meson*

ultimately have $w \leq_p (s \cdot p) \cdot w$
using *marker-fac-pref pref-pow-root* **by** *meson*
thus *?thesis*
unfolding ‹$v = p \cdot s$› *rassoc pref-cancel-conv*.
qed

lemma *root-suf-comm'*: $x \leq_p r \cdot x \Longrightarrow r \leq_s x \Longrightarrow r \cdot x = x \cdot r$
using *root-suf-comm suffix-appendI*[*of r x r*] **by** *blast*

lemmas *suf-root-pref-comm* = *root-suf-comm'*[*reversed*]

lemma *mid-marker-root*: **assumes** $u \cdot v \leq_p r^{@}k$ $r \leq_s u$ **shows** $v \leq_p r \cdot v$
using *pref-pow-root*[*OF pref-comm-cancel*[*OF - comm-pow-comm*[*OF root-suf-comm'*[*OF pref-pow-root*[*OF append-prefixD*[*OF* ‹$u \cdot v \leq_p r^{@}k$›]] ‹$r \leq_s u$›], *symmetric*]], *OF* ‹$u \cdot v \leq_p r^{@}k$›].

lemmas *mid-marker-root'* = *mid-marker-root*[*reversed*]

lemma *marker-pref-suf-fac*: **assumes** $u \leq_p v$ **and** $u \leq_s v$ **and** $v \leq_f u^{@}k$
shows $u \cdot v = v \cdot u$
using *root-suf-comm'*[*OF pref-pow-root*[*OF marker-fac-pref*[*OF* ‹$v \leq_f u^{@}k$› ‹$u \leq_p v$›]] ‹$u \leq_s v$›].

lemma *pref-suf-per-fac-comm*:
assumes $v \leq_p u \cdot v$ **and** $v \leq_s v \cdot u$ **and** $u \leq_f v^{@}k$ **shows** $u \cdot v = v \cdot u$
using *marker-pref-suf-fac*[*OF - -* ‹$u \leq_f v^{@}k$›] *root-suf-comm*[*OF* ‹$v \leq_p u \cdot v$› *suf-ext*] *root-suf-comm*[*reversed*, *OF* ‹$v \leq_s v \cdot u$› *pref-ext*]
ruler-pref'[*OF* ‹$v \leq_p u \cdot v$›] *ruler-suf'*[*OF* ‹$v \leq_s v \cdot u$›] **by** *argo*

lemma *mid-long-pow*: **assumes** *eq*: $y^{@}m = u \cdot x^{@}(Suc\ k) \cdot v$ **and** $|y| \leq |x^{@}k|$
shows $(u \cdot v) \cdot y = y \cdot (u \cdot v)$ **and** $(u \cdot x^{@}l \cdot v) \cdot y = y \cdot (u \cdot x^{@}l \cdot v)$ **and** $(u^{-1>}(y \cdot u)) \cdot x = x \cdot (u^{-1>}(y \cdot u))$
proof–
have *eq'*: $x \cdot x \cdot v \cdot u = u^{-1>}(u \cdot x \cdot x \cdot v) \cdot u$ **by** *simp*
let *?y* = $u^{-1>}(y \cdot u)$
have $u \leq_p y \cdot u$
using *eq prefI pref-pow-root*[*of u m y*,*unfolded eq*] **by** *simp*
hence *?y* $\sim y$
using *root-conjug* **by** *blast*
from *conjug-len*[*OF this*]
have $|?y| \leq |x^{@}k|$
using ‹$|y| \leq |x^{@}k|$› **by** *simp*
from *lq-conjug-pow*[*OF* ‹$u \leq_p y \cdot u$›, *of m*]
have *?y*$^{@}m = x^{@}Suc\ k \cdot v \cdot u$
unfolding *eq eq'* **by** *simp*
hence $x^{@}Suc\ k \leq_p$ *?y* $\cdot\ x^{@}Suc\ k$
using *rassoc prefI pref-pow-root*[*of* $x^{@}Suc\ k\ m$ *?y*] **by** *blast*
have $x\ ^{@}\ Suc\ k \leq_p x \cdot x\ ^{@}\ Suc\ k$
using *pref-pow-ext'* **by** *blast*

have *com*: $?y \cdot x = x \cdot ?y$
using $\langle |?y| \leq |x^{@}k| \rangle$ *two-pers*[*OF* $\langle x^{@}Suc\ k \leq_p ?y \cdot x^{@}Suc\ k \rangle$ $\langle x\ ^{@}\ Suc\ k \leq_p x \cdot x\ ^{@}\ Suc\ k \rangle$]
unfolding *pow-Suc2 lenmorph* **by** *linarith*
thus $?y \cdot x = x \cdot ?y$
by *blast*
have $?y \cdot x^{@}Suc\ k = x^{@}Suc\ k \cdot ?y$
using *com comm-pow-comm* **by** *metis*
from *pow-comm*[*of m ?y*, *unfolded* $\langle ?y\ ^{@}\ m = x^{@}(Suc\ k) \cdot v \cdot u \rangle$, *unfolded lassoc this*, *unfolded rassoc*]
have $x^{@}Suc\ k \cdot v \cdot u \cdot ?y = x^{@}Suc\ k \cdot ?y \cdot v \cdot u$**.**
hence $u \cdot ?y \cdot v \cdot u = u \cdot v \cdot u \cdot ?y$ **by** *simp*
thus $(u \cdot v) \cdot\ \ y = y \cdot (u \cdot v)$
unfolding *lassoc lq-pref*[*OF* $\langle u \leq_p y \cdot u \rangle$] **by** *fastforce*
have $u \cdot x^{@}l \cdot v \cdot\ \ u \cdot ?y = \ \ u \cdot (?y \cdot x^{@}l) \cdot v \cdot u$
unfolding *comm-pow-comm*[*OF com*[*symmetric*], *of l*, *symmetric*] *rassoc cancel*
using $\langle u \cdot ?y \cdot v \cdot u = u \cdot v \cdot u \cdot ?y \rangle$[*unfolded cancel*, *symmetric*]**.**
thus $(u \cdot x^{@}l \cdot v) \cdot y = y \cdot (u \cdot x^{@}l \cdot v)$
unfolding *lq-pref*[*OF* $\langle u \leq_p y \cdot u \rangle$] *lassoc* **by** *blast*
qed

lemma *mid-pow-pref-suf'*: **assumes** $s \cdot w^{@}(Suc\ l) \cdot p \leq_f w^{@}k$ **shows** $p \leq_p w^{@}k$ **and** $s \leq_s w^{@}k$
proof−
obtain *v u* **where** *dec*: $v \cdot s \cdot\ \ w^{@}(Suc\ l) \cdot p \cdot u = w^{@}k$
using *facE'*[*OF assms*, *unfolded rassoc*]**.**
hence $(v \cdot s) \cdot w = w \cdot (v \cdot s)$ **and** $w \cdot (p \cdot u) = (p \cdot u) \cdot w$
using *mid-pow*[*of v · s l w p · u k*] **unfolding** *rassoc* **by** *presburger*+
have $|p| \leq |w^{@}k|$ **and** $|s| \leq |w^{@}k|$
using *fac-len*[*OF assms*] **unfolding** *lenmorph* **by** *linarith*+

from *per-exp-pref*[*of p · u w k*, *unfolded* $\langle w \cdot (p \cdot u) = (p \cdot u) \cdot w \rangle$, *OF triv-pref*]
have $p \leq_p w^{@}k \cdot (p \cdot u)$
using *prefix-order.trans*[*OF triv-pref*[*of p u*]] **by** *blast*
thus $p \leq_p w^{@}k$
using $\langle |p| \leq |w\ ^{@}\ k| \rangle$ *pref-prod-le* **by** *blast*

from *per-exp-suf*[*of v · s w k*, *unfolded* $\langle (v \cdot s) \cdot w = w \cdot (v \cdot s) \rangle$, *OF triv-suf*]
have $s \leq_s (v \cdot s) \cdot w^{@}k$
using *suffix-order.trans*[*OF triv-suf*[*of s v*], *of* $(v \cdot s) \cdot w^{@}k$] **by** *blast*
thus $s \leq_s w^{@}k$
using $\langle |s| \leq |w\ ^{@}\ k| \rangle$ *suf-prod-le* **by** *blast*
qed

lemma *mid-pow-pref-suf*: **assumes** $s \cdot w \cdot p \leq_f w^{@}k$ **shows** $p \leq_p w^{@}k$ **and** $s \leq_s w^{@}k$
using *mid-pow-pref-suf'*[*of s 0 w p k*, *unfolded pow-list-one*, *OF assms*]**.**

lemma *fac-marker-pref*: $y \cdot x \leq_f y^{@}k \Longrightarrow x \leq_p y \cdot x$

using *mid-pow-pref-suf(1)*[*of* ε, *unfolded emp-simps*, *THEN pref-pow-root*].

lemmas *fac-marker-suf* = *fac-marker-pref*[*reversed*]

lemma *fac-two-markers*: **assumes** $u \cdot v \cdot u \leq f\ u^{@}k$ **shows** $u \cdot v = v \cdot u$
using *fac-marker-pref*[*OF assms*, *THEN pref-comm-eq'*, *symmetric*].

lemma *prim-overlap-sqE* [*consumes 2*]:
assumes *prim*: *primitive r* **and** *eq*: $p \cdot r \cdot q = r \cdot r$
obtains (*pref-emp*) $p = \varepsilon$ | (*suff-emp*) $q = \varepsilon$
proof (*cases* $|p| = 0$, *blast*)
assume $|p| \neq 0$ **and** *qemp*: $q = \varepsilon \Longrightarrow$ *thesis*
hence $|q| < |r|$
using *lenarg*[*OF eq*] **unfolding** *lenmorph* **by** *linarith*
have $q = \varepsilon$
using *prim-comm-short-emp*[*OF prim mid-sq(2)*[*OF eq*, *symmetric*] ‹$|q| < |r|$›].
from *qemp*[*OF this*]
show *thesis*.
qed

lemma *prim-overlap-sqE'* [*consumes 2*]:
assumes *prim*: *primitive r* **and** *eq*: $p \cdot r \cdot q = r \cdot r$
obtains (*pref-emp*) $p = \varepsilon$ | (*suff-emp*) $p = r$
using *append-Nil2 eq mid-sq'(2) prim prim-overlap-sqE* **by** *metis*

lemma *prim-overlap-sq-prefE*:
assumes *primitive r* **and** $p \cdot r \leq p\ r \cdot r$
obtains (*pref-emp*) $p = \varepsilon$ | (*suff-emp*) $p = r$
using *prim-overlap-sqE'*[*OF* ‹*primitive r*› *lq-pref*[*OF* ‹$p \cdot r \leq p\ r \cdot r$›, *unfolded rassoc*]].

lemma *prim-overlap-sq*:
assumes *prim*: *primitive r* **and** *eq*: $p \cdot r \cdot q = r \cdot r$
shows $p = \varepsilon \vee q = \varepsilon$
using *prim-overlap-sqE*[*OF prim eq disjI1 disjI2*].

lemma *prim-overlap-sq'*:
assumes *prim*: *primitive r* **and** *pref*: $p \cdot r \leq p\ r \cdot r$ **and** *len*: $|p| < |r|$
shows $p = \varepsilon$
using *mid-sq(1)*[*symmetric*, *THEN prim-comm-short-emp*[*OF prim - len*]] *prefD*[*OF pref*] **by** *force*

lemma *zxy-fac-xyzx*: **assumes** *nemp*: $y \neq \varepsilon\ z \neq \varepsilon$ **and** *fac*: $z \cdot x \cdot y \leq f\ x \cdot y \cdot z \cdot x$
shows ¬ *primitive* ($z{\cdot}x{\cdot}y$)
proof
assume *primitive* ($z{\cdot}x{\cdot}y$)
from *facE*[*OF fac*]
obtain *p s* **where** $x \cdot y \cdot z \cdot x = p \cdot (z \cdot x \cdot y) \cdot s$

by *blast*
hence *sq*: $(z \cdot p) \cdot (z \cdot x \cdot y) \cdot (s \cdot y) = (z \cdot x \cdot y) \cdot (z \cdot x \cdot y)$
by *simp*
from *prim-overlap-sqE*[*OF* ‹*primitive* $(z{\cdot}x{\cdot}y)$› *sq*]
show *False*
using *nemp append-is-Nil-conv* **by** *blast*
qed

lemma *prim-overlap-pow*[*elim*]:
assumes *prim*: *primitive* r **and** *pref*: $u \cdot r \leq p\ r^{@}k$
obtains i **where** $u = r^{@}i$ **and** $i < k$
proof−
obtain q **where** *eq*: $u \cdot r\ ^{@}\ Suc\ 0 \cdot q = r\ ^{@}\ k$
using *pref* **by** (*auto simp add*: *prefix-def*)
from *mid-pow*(*1*)[*OF this*, *symmetric*]
have $u \cdot r = r \cdot u$**.**
from *prim-comm-exp*[*OF* ‹*primitive* r› *this*]
obtain i **where** $r^{@}i = u$**.**
hence $|r\ ^{@}\ Suc\ i| \leq |r\ ^{@}\ k|$
using *pref* **by** (*auto simp add*: *prefix-def*)
from *mult-right-le-imp-le*[*OF this*[*unfolded pow-len*] *nemp-len*[*OF prim-nemp*[*OF prim*]]]
have $i < k$ **by** *auto*
from *that*[*OF* ‹$r^{@}i = u$›[*symmetric*] *this*]
show *thesis***.**
qed

lemma *prim-overlap-pow′*:
assumes *prim*: *primitive* r **and** *pref*: $u \cdot r \leq p\ r^{@}k$ **and** *less*: $|u| < |r|$
shows $u = \varepsilon$
proof−
obtain i **where** $u = r^{@}i$
using *prim-overlap-pow*[*OF prim pref*] **by** *force*
from *less*[*unfolded pow-len*[*of i r*, *folded this*]]
have $i = 0$ **by** *force*
from ‹$u = r^{@}i$›[*unfolded this pow-zero*]
show $u = \varepsilon$**.**
qed

lemma *prim-sqs-overlap*:
assumes *prim*: *primitive* r **and** *comp*: $u \cdot r \cdot r \bowtie v \cdot r \cdot r$
and *len-u*: $|u| < |v| + |r|$ **and** *len-v*: $|v| < |u| + |r|$
shows $u = v$
proof (*cases rule*: *le-cases*)
have *wlog-le*: $u = v$ **if** *comp*: $u \cdot (r \cdot r) \bowtie v \cdot (r \cdot r)$ **and** *len-v*: $|v| < |u| + |r|$
and $|u| \leq |v|$ **for** $u\ v$
proof −
obtain w **where** v: $u \cdot w = v$
using *comp-shorter*[*OF comp-prefs-comp*[*OF comp*] ‹$|u| \leq |v|$›] **by** (*auto simp*

add: *prefix-def*)
  have $|w| < |r|$ **using** *len-v* **unfolding** *v*[*symmetric*] **by** *simp*
  have *comp'*: $r \cdot r \bowtie (w \cdot r) \cdot r$ **using** *comp* **unfolding** *v*[*symmetric*] *rassoc comp-cancel*.
  moreover have $|w \cdot r| \leq |r \cdot r|$ **using** *less-imp-le-nat*[*OF* ‹$|w| < |r|$›] **by** *simp*
  ultimately have *pref*: $w \cdot r \leq_p r \cdot r$
    by (*rule pref-comp-len-trans*[*OF triv-pref*])
  from *this* ‹$|w| < |r|$› **have** $w = \varepsilon$ **by** (*rule prim-overlap-sq'*[*OF prim*])
  show $u = v$ **using** *v* **unfolding** ‹$w = \varepsilon$› *append-Nil2*.
 qed
 show $|u| \leq |v| \Longrightarrow u = v$ **using** *wlog-le*[*OF comp len-v*].
 show $|v| \leq |u| \Longrightarrow u = v$ **using** *wlog-le*[*OF comp*[*symmetric*] *len-u*, *symmetric*].
qed

lemma *drop-pref-prim*: **assumes** *Suc n* $< |w|$ **and** $w \leq_p$ *drop* (*Suc n*) $(w \cdot w)$ **and** *primitive w*
 shows *False*
 using *assms*
proof (*cases* $w = \varepsilon$)
 assume $w \neq \varepsilon$
 obtain *s* **where** *drop* (*Suc n*) $(w \cdot w) = w \cdot s$
  using *prefD*[*OF* ‹$w \leq_p$ *drop* (*Suc n*) $(w \cdot w)$›] **by** *blast*
 note *takedrop*[*of Suc n* $w \cdot w$, *unfolded this*]
 from ‹*Suc n* $< |w|$› ‹$w \neq \varepsilon$› *prim-overlap-sqE'*[*OF* ‹*primitive w*› *this*]
 show *False* **by** *auto*
qed *simp*

lemma *root-suf-conjug*: **assumes** *primitive* $(s \cdot r)$ **and** $y \leq_p (s \cdot r) \cdot y$ **and** $y \leq_s y \cdot (r \cdot s)$ **and** $|s \cdot r| \leq |y|$
  obtains *l* **where** $y = (s \cdot r)^{@} l \cdot s$
proof−
 have $y \neq \varepsilon$
  using *assms*(*1*) *assms*(*4*) **by** *force*
 have $r \cdot s \leq_s y$
  using *suf-prod-long*[*OF* ‹$y \leq_s y \cdot (r \cdot s)$› ‹$|s \cdot r| \leq |y|$›[*unfolded swap-len*]].
 have *primitive* $(r \cdot s)$
  using *prim-conjug*[*OF* ‹*primitive* $(s \cdot r)$› *conjugI'*].
 have $r \cdot y \leq_p (r \cdot s) \cdot (r \cdot y)$
  using ‹$y \leq_p (s \cdot r) \cdot y$› **by** *auto*
 from *prim-comm-exp*[*OF* ‹*primitive* $(r \cdot s)$› *root-suf-comm'*[*OF this suf-ext*[*OF* ‹$r \cdot s \leq_s y$›], *symmetric*]]
 obtain *k* **where** [*symmetric*]: $(r \cdot s)^{@} k = r \cdot y$ **and** $0 < k$
  using ‹$y \neq \varepsilon$› **using** *nemp-exp-pos sufI suf-emp* **by** *metis*
 hence $y = (s \cdot r)^{@}(k-1) \cdot s$
  unfolding *pow-pos*[*of* - $r \cdot s$, *OF* ‹$0 < k$›] *rassoc cancel shift-pow* **by** *blast*
 from *that*[*OF this*]
 show *thesis*.
qed

lemma *root-suf-pow-comm*: **assumes** $x \leq_p r \cdot x$ **and** $r \leq_s x^{@}(Suc\ k)$ **shows** $r \cdot x = x \cdot r$
using *root-suf-comm*[*OF* ‹$x \leq_p r \cdot x$› *suf-prod-root*[*OF* ‹$r \leq_s x^{@}(Suc\ k)$›]].

lemma *suf-pow-short-suf*: $r \leq_s x^{@}k \Longrightarrow |x| \leq |r| \Longrightarrow x \leq_s r$
using *suf-prod-root*[*THEN suf-prod-long*].

thm *suf-pow-short-suf*[*reversed*]

lemma *sq-short-per*: **assumes** $|u| \leq |v|$ **and** $v{\cdot}v \leq_p u{\cdot}(v{\cdot}v)$
shows $u{\cdot}v = v{\cdot}u$
using
pref-marker[*of* $v{\cdot}v$, *OF* ‹$v{\cdot}v \leq_p u{\cdot}(v{\cdot}v)$›
pref-prod-long[*OF append-prefixD*[*OF* ‹$v{\cdot}v \leq_p u{\cdot}(v{\cdot}v)$›] ‹$|u| \leq |v|$›,
THEN pref-cancel'], *symmetric*].

lemma *fac-marker*: **assumes** $w \leq_p u{\cdot}w$ **and** $u{\cdot}v{\cdot}u \leq_f w$
shows $u \cdot v = v \cdot u$
proof–
obtain $p\ s$ **where** $w = p{\cdot}u{\cdot}v{\cdot}u{\cdot}s$
using ‹$u{\cdot}v{\cdot}u \leq_f w$›[*unfolded fac-def*]
by *auto*

hence $p{\cdot}u{\cdot}v{\cdot}u = u{\cdot}p{\cdot}u{\cdot}v$
using *pref-marker*[*OF* ‹$w \leq_p u{\cdot}w$›, *unfolded* ‹$w = p{\cdot}u{\cdot}v{\cdot}u{\cdot}s$›, *of* $p \cdot u \cdot v$]
by *force*

thus $u{\cdot}v = v{\cdot}u$
using *eqd-eq*[*of* $p \cdot u\ v \cdot u\ u \cdot p\ u \cdot v$, *unfolded rassoc*, *OF - swap-len*]
by *presburger*
qed

lemma $4 = Suc(Suc(Suc(Suc\ 0)))$
using [[*simp-trace*]] **by** *simp*

lemma *xyxy-conj-yxxy*: **assumes** $x \cdot y \cdot x \cdot y \sim y \cdot x \cdot x \cdot y$
shows $x \cdot y = y \cdot x$
proof–
have *four*: $x^{@}4 = x{\cdot}x{\cdot}x{\cdot}x$ **for** $x :: 'a\ list$
unfolding *numeral-Bit0* **by** *simp*
from *conjug-fac-sq*[*OF assms*[*symmetric*]]
have $y \cdot x \cdot x \cdot y \leq_f (x \cdot y)^{@}4$
unfolding *four rassoc*.
from *marker-fac-pref*[*reversed*,
OF this triv-suf[*of* $x{\cdot}y$ $y{\cdot}x$, *unfolded rassoc*]]
have $y \cdot x \cdot x \cdot y \leq_s (x \cdot y)\ ^{@}\ 4$.
hence $y \cdot x \cdot x \cdot y \leq_s (x{\cdot}y{\cdot}x{\cdot}y){\cdot}x{\cdot}y{\cdot}x{\cdot}y$
unfolding *four rassoc*.

from *suf-prod-eq*[*OF this*]
show $x \cdot y = y \cdot x$
by *simp*
qed

lemma *per-glue*: **assumes** *period* u n **and** *period* v n **and** $u \leq p$ w **and** $v \leq s$ w **and**
$|w| + n \leq |u| + |v|$
shows *period* w n
proof (*rule indeces-period*)
fix i **assume** $i + n < |w|$
show $w ! i = w ! (i + n)$
proof (*cases*)
assume $i + n < |u|$
hence $w ! i = u ! i$ **and** $w ! (i+n) = u ! (i+n)$
using *add-lessD1* ‹$u \leq p$ w› *pref-index* **by** *metis+*
thus $w ! i = w ! (i + n)$
unfolding ‹$w ! i = u ! i$› ‹$w ! (i+n) = u ! (i+n)$›
using *period-indeces*[*OF* ‹*period* u n› ‹$i + n < |u|$›] **by** *blast*
next
assume $\neg\ i + n < |u|$
obtain p **where** $w = p \cdot v$
using ‹$v \leq s$ w› **by** (*auto simp add*: *suffix-def*)
have $\neg\ i < |p|$
using ‹$\neg\ i + n < |u|$› ‹$|w| + n \leq |u| + |v|$› **unfolding** *lenarg*[*OF* ‹$w = p \cdot v$›, *unfolded lenmorph*]
by *auto*
hence $w!i = v!(i - |p|)$ **and** $w!(i+n) = v!((i - |p|) + n)$
unfolding ‹$w = p \cdot v$› *nth-append* **by** *simp-all*
have $i - |p| + n < |v|$
using ‹$\neg\ i < |p|$› ‹$i + n < |w|$› ‹$w = p \cdot v$› **by** *auto*
from *period-indeces*[*OF* ‹*period* v n› *this*]
show $w ! i = w ! (i + n)$
unfolding ‹$w!i = v!(i - |p|)$› ‹$w!(i+n) = v!(i - |p| + n)$›.
qed
qed

lemma *per-glue-facs*: **assumes** $u \cdot z \leq f\ w^{@} k$ **and** $z \cdot v \leq f\ w^{@} m$ **and** $|w| \leq |z|$
obtains l **where** $u \cdot z \cdot v \leq f\ w^{@} l$
using *assms*
proof (*cases* $k = 0$)
assume $k \neq 0$
have $z \leq f\ w^{@} k$
using ‹$u \cdot z \leq f\ w^{@} k$› **by** *blast*
have $z \leq f\ w^{@} m$
using ‹$z \cdot v \leq f\ w^{@} m$› **by** *blast*
define t **where** $t = take\ |w|\ z$
have $|t| = |w|$ **and** $t \leq p\ z$

unfolding *t-def* **using** ‹$|w| \leq |z|$› *take-is-prefix* **by** (*force*,*blast*)
hence $w \sim t$
using ‹$z \leq_f w^{@}m$› **by** *blast*
from *marker-fac-pref-len*[*OF* ‹$z \cdot v \leq_f (w)^{@} m$› - ‹$|t| = |w|$›]
have $z \cdot v \leq_p t^{@}m$
using ‹$t \leq_p z$› **by** *force*
have $u \cdot z \leq_f t^{@}Suc\ k$
using *fac-pow-conjug*[*OF* ‹$u \cdot z \leq_f w^{@}k$› ‹$w \sim t$›[*symmetric*]].
with ‹$t \leq_p z$›
have $u \leq_s t^{@}Suc\ k$
using *mid-pow-pref-suf*(*2*)[*of u t* $t^{-1>}z$ *Suc k*] *lq-pref* **by** *metis*
have $(t^{@}Suc\ k^{<-1}u) \cdot (u \cdot z \cdot v) \cdot (z \cdot v)^{-1>}(t^{@}m) = t^{@}Suc\ k \cdot t^{@}m$
unfolding *lassoc rq-suf*[*OF* ‹$u \leq_s t^{@}Suc\ k$›] **unfolding** *rassoc cancel* **using** *lq-pref*[*OF* ‹$z \cdot v \leq_p t^{@}m$›] **unfolding** *rassoc*.
from *facI*[*of* $u \cdot z \cdot v$ $t^{@}Suc\ k^{<-1}u$ $(z \cdot v)^{-1>}(t^{@}m)$, *unfolded this*, *folded pow-add*]
obtain *l* **where** $u \cdot z \cdot v \leq_f t^{@}l$
by *metis*
from *that*[*OF fac-pow-conjug*[*OF this* ‹$w \sim t$›]]
show *thesis*.
qed *simp*

lemma *per-fac-pow-fac*: **assumes** *period w n* **and** $v \leq_f w$ **and** $|v| = n$ **and** $n \neq 0$
obtains *k* **where** $w \leq_f v^{@}k$
proof–
have *take n w* $\neq \varepsilon$
using *assms*(*2*–*4*) **by** *force*
obtain *m* **where** $w <_p (take\ n\ w)^{@}m$
using *per-root-powE*[*OF* ‹*period w n*›[*unfolded period-def*], *OF* ‹*take n w* $\neq \varepsilon$›].
hence $w \leq_f (take\ n\ w)^{@}m$
by *simp*
obtain *r s* **where** $r \cdot s = v$ **and** $s \cdot r = take\ n\ w$
using *fac-per-conjug*[*OF assms*(*1*,*2*,*3*), *THEN conjugE*].
hence $r \cdot (take\ n\ w)^{@}m \cdot s = v^{@}Suc\ m$
by (*metis pow-list-slide*)
from *that*[*OF fac-trans*, *OF* ‹$w \leq_f (take\ n\ w)^{@}m$›] *sublist-appendI*[*of* (*take n w*)$^{@}$*m r s*, *unfolded this*]
show *thesis*
by *blast*
qed

lemma *refine-per*: **assumes** *period w n* **and** $v \leq_f w$ **and** $n \leq |v|$ **and** *period v k* **and** *k dvd n* **and** $n \neq 0$
shows *period w k*
proof–
have |*take n w*| = *n*
using *take-len*[*OF le-trans*[*OF* ‹$n \leq |v|$› *fac-len*[*OF* ‹$v \leq_f w$›]]].
have |*take n v*| = *n*
using *take-len*[*OF* ‹$n \leq |v|$›].

have *period v n*
using *period-fac-period'*[*OF* ‹*period w n*› ‹*v* ≤*f w*›] **by** *blast*
have *take n v* ≤*f w*
using ‹*v* ≤*f w*› ‹*n* ≤ |*v*|› *sublist-order.dual-order.trans sublist-take* **by** *metis*
have *period* (*take n v*) *k*
using ‹*period w n*› ‹*period v k*› *period-pref-period take-is-prefix* **by** *metis*
have *k* ≤ *n*
using ‹*k dvd n*› ‹*n* ≠ *0*› **by** *force*
hence *take k* (*take n v*) = *take k v*
using *take-le-take* **by** *blast*
hence (*take k v*)@(*n div k*) = *take n v*
using *per-div*[*OF* - ‹*period* (*take n v*) *k*›, *unfolded* ‹|*take n v*| = *n*›, *OF* ‹*k dvd n*›] **by** *presburger*
have |*take k v*| = *k*
using *order.trans*[*OF* ‹*k* ≤ *n*› ‹*n* ≤ |*v*|›, *THEN take-len*].
obtain *e* **where** *w* ≤*f* (*take n v*)@*e*
using *per-fac-pow-fac*[*OF* ‹*period w n*› ‹*take n v* ≤*f w*› ‹|*take n v*| = *n*›] ‹*n* ≠ *0*› **by** *blast*
from *per-fac*[*OF this*[*folded* ‹(*take k v*)@(*n div k*) = *take n v*›, *folded pow-mult*]]
show *?thesis*
unfolding ‹|*take k v*| = *k*› **by** *blast*
qed

lemma *xy-per-comp*: **assumes** *x*·*y* ≤*p q*·*x*·*y*
and *q* ≠ *ε* **and** *q* ⋈ *y*
shows *x* ⋈ *y*
proof(*cases rule*: *pref-compE*[*OF* ‹*q* ⋈ *y*›])
assume *q* ≤*p y*
have *x*·*q* = *q*·*x*
using
pref-cancel'[*OF* ‹*q* ≤*p y*›, *of x*, *THEN pref-trans*, *OF* ‹*x* · *y* ≤*p q* · *x* · *y*›]
unfolding *lassoc*
using *ruler-eq-len*[*OF* - *triv-pref swap-len*]
by *blast*
thus *?thesis*
using *assms*(*1*) *assms*(*2*) *pref-comp-sym root-comm-root*
ruler-pref'' *same-prefix-prefix* **by** *metis*
next
assume *y* ≤*p q*
then show *?thesis*
by (*meson append-prefixD prefix-append ruler' assms*)
qed

lemma *prim-xyxyy*: *x* · *y* ≠ *y* · *x* ⟹ *primitive* (*x* · *y* · *x* · *y* · *y*)
proof (*rule prim-conjug*)
show *y* · *x* · *y* · *x* · *y* ∼ *x* · *y* · *x* · *y* · *y*
by (*intro conjugI1*) *simp*
show *x* · *y* ≠ *y* · *x* ⟹ *primitive* (*y* · *x* · *y* · *x* · *y*)
by (*intro iffD2*[*OF per-le-prim-iff*[*of* - *y* · *x*]]) *auto*

qed

lemma *prim-min-per-suf-eq*: **assumes** *primitive x* **and** $\pi\ x \leq s\ x$ **shows** $\pi\ x = x$
using *assms(1) min-per-root-per-root*[*OF prim-nemp*[*OF ‹primitive x›*], *unfolded*] *root-suf-comm'*[*OF - ‹*$\pi\ x \leq s\ x$*›*]
unfolding *primitive-iff-per* **by** *blast*

lemma *primroot-code*[*code*]: $\varrho\ x = (if\ x \neq \varepsilon\ then\ (if\ \pi\ x \leq s\ x\ then\ \pi\ x\ else\ x)\ else\ Code.abort\ (STR\ ''Empty\ word\ has\ no\ primitive\ root.'')\ (\lambda\text{-}.\ (\varrho\ x)))$
proof(*cases* $x = \varepsilon$)
assume $x \neq \varepsilon$
thus *?thesis*
unfolding *if-P*[*OF ‹*$x \neq \varepsilon$*›*]
proof(*cases*)
assume $e_\varrho\ x = 1$
have *primitive x*
using *primroot-exp-eq*[*of x, unfolded ‹*$e_\varrho\ x = 1$*› exp-simps*]
unfolding *prim-primroot-conv*[*OF ‹*$x \neq \varepsilon$*›*].
from *prim-min-per-suf-eq*[*OF this*] *prim-primroot*[*OF this*]
show $\varrho\ x = (if\ \pi\ x \leq s\ x\ then\ \pi\ x\ else\ x)$
by *argo*
next
assume $e_\varrho\ x \neq 1$
show $\varrho\ x = (if\ \pi\ x \leq s\ x\ then\ \pi\ x\ else\ x)$
using *primroot-suf*
unfolding *min-per-short-primroot*[*OF ‹*$x \neq \varepsilon$*› primroot-exp-eq ‹*$e_\varrho\ x \neq 1$*›*]
by *auto*
qed
qed (*simp add*: *primitive-root-def*)

lemma *per-lemma-pref-suf*: **assumes** $w <p\ p \cdot w$ **and** $w <s\ w \cdot q$ **and**
fw: $|p| + |q| \leq |w|$
obtains $r\ s\ k\ l\ m$ **where** $p = (r \cdot s)^{@} k$ **and** $q = (s \cdot r)^{@} l$ **and** $w = (r \cdot s)^{@} m \cdot r$ **and** *primitive* $(r \cdot s)$
proof−
let *?q* $= (w \cdot q)^{<-1} w$
have $w <p$ *?q* $\cdot\ w$
using *ssufD1*[*OF ‹*$w <s\ w \cdot q$*›*] *rq-suf*[*symmetric, THEN per-rootI*[*OF prefI rq-ssuf-nemp*[*OF ‹*$w <s\ w \cdot q$*›*]]]
by *argo*
have $q \sim$ *?q*
by (*meson assms(2) conjugI1 conjug-sym rq-suf suffix-order.less-imp-le*)

have *nemps'*: $p \neq \varepsilon$ *?q*$\neq \varepsilon$
using *assms(1) ‹*$w <p$ *?q*$\cdot w$*›* **by** *fastforce*+
from *two-pers*[*OF sprefD1*[*OF ‹*$w <p\ p \cdot w$*›*] *sprefD1*[*OF ‹*$w <p$ *?q*$\cdot w$*›*]] *fw*
have $p \cdot$ *?q* $=$ *?q* $\cdot\ p$
unfolding *conjug-len*[*OF ‹*$q \sim (w \cdot q)^{<-1} w$*›*]
by *blast*

then have $\varrho\ p = \varrho\ ?q$ **using** *comm-primroots*[*OF nemps'*] **by** *force*
hence [*symmetric*]: $\varrho\ q \sim \varrho\ p$
using *conjug-primroot*[*OF* ‹$q \sim (w \cdot q)^{<-1} w$›]
by *argo*
from *conjug-primrootsE*[*OF this*]
obtain $r\ s\ k\ l$ **where**
$p = (r \cdot s)\ ^{@}\ k$ **and**
$q = (s \cdot r)\ ^{@}\ l$ **and**
primitive $(r \cdot s)$**.**
have $w \leq p\ (r{\cdot}s){\cdot}w$
using *assms per-root-drop-exp sprefD1* ‹$p = (r \cdot s)\ ^{@}\ k$›
by *meson*
have $w \leq s\ w{\cdot}(s{\cdot}r)$
using *assms*(*2*) *per-root-drop-exp*[*reversed*] *ssufD1* ‹$q = (s \cdot r)\ ^{@}\ l$›
by *meson*
have $|r \cdot s| \leq |w|$
using *conjug-nemp-iff*[*OF* ‹$q \sim ?q$›] *dual-order.trans length-0-conv nemps' primroot-len-le*[*OF nemps'*(*1*)] *fw*
unfolding *primroot-unique*[*OF nemps'*(*1*) ‹*primitive* $(r \cdot s)$› ‹$p = (r \cdot s)\ ^{@}\ k$›]
by *linarith*
then
obtain m **where** $w = (r \cdot s)\ ^{@}\ m \cdot r$
by (*meson* ‹*primitive* $(r \cdot s)$› ‹$w \leq p\ (r \cdot s) \cdot w$› ‹$w \leq s\ w \cdot s \cdot r$› *root-suf-conjug*)
from *that*[*OF* ‹$p = (r \cdot s)\ ^{@}\ k$› ‹$q = (s \cdot r)\ ^{@}\ l$› *this* ‹*primitive* $(r \cdot s)$›]
show *?thesis***.**
qed

lemma *fac-two-conjug-primrootE*:
assumes *facs*: $w \leq f\ p^{@}k\ w \leq f\ q^{@}l$ **and** *nemps*: $p \neq \varepsilon\ q \neq \varepsilon$ **and** *len*: $|p| + |q| \leq |w|$
obtains $r\ s\ m$ **where** $\varrho\ p \sim r \cdot s$ **and** $\varrho\ q \sim r \cdot s$ **and** $w = (r \cdot s)^{@}m \cdot r$ **and** *primitive* $(r{\cdot}s)$
proof −
obtain p' **where** $w <p\ p'{\cdot}w\ p \sim p'\ p' \neq \varepsilon$
using *conjug-nemp-iff fac-pow-pref-conjug*[*OF facs*(*1*)] *nemps*(*1*) *per-rootI'* **by** *metis*
obtain q' **where** $w <s\ w{\cdot}q'\ q \sim q'\ q' \neq \varepsilon$
using *fac-pow-pref-conjug*[*reversed*, *OF* ‹$w \leq f\ q^{@}l$›] *conjug-nemp-iff nemps*(*2*) *per-rootI'*[*reversed*] **by** *metis*
from *per-lemma-pref-suf*[*OF* ‹$w <p\ p'{\cdot}w$› ‹$w <s\ w{\cdot}q'$›]
obtain $r\ s\ k\ l\ m$ **where**
$p' = (r \cdot s)\ ^{@}\ k$ **and**
$q' = (s \cdot r)\ ^{@}\ l$ **and**
$w = (r \cdot s)\ ^{@}\ m \cdot r$ **and**
primitive $(r \cdot s)$
using *len*[*unfolded conjug-len*[*OF* ‹$p \sim p'$›] *conjug-len*[*OF* ‹$q \sim q'$›]]
by *metis*
moreover have $\varrho\ p' = r{\cdot}s$
using ‹$p' = (r \cdot s)\ ^{@}\ k$› ‹*primitive* $(r \cdot s)$› ‹$p' \neq \varepsilon$› *primroot-unique* **by** *metis*

hence $\varrho\ p \sim r{\cdot}s$
using *conjug-primroot*[*OF* ‹$p \sim p'$›]
by *simp*
moreover have $\varrho\ q' = s{\cdot}r$
using ‹$q' = (s \cdot r)\ ^{@}\ l$› ‹*primitive* $(r \cdot s)$›[*unfolded conjug-prim-iff'*[*of* r]] ‹$q' \neq \varepsilon$› *primroot-unique* **by** *metis*
hence $\varrho\ q \sim s{\cdot}r$
using *conjug-primroot*[*OF* ‹$q \sim q'$›] **by** *simp*
hence $\varrho\ q \sim r{\cdot}s$
using *conjug-trans*[*OF* - *conjugI'*]
by *meson*
ultimately show *?thesis*
using *that* **by** *blast*
qed

corollary *fac-two-conjug-primroot*:
assumes *facs*: $u \leq_f r^{@}k\ u \leq_f s^{@}l$ **and** *len*: $|r| + |s| \leq |u|$
shows $\varrho\ r \sim \varrho\ s$
proof (*cases* $u = \varepsilon$)
assume $u = \varepsilon$
thus *?thesis*
using *len* **by** *auto*
next
assume $u \neq \varepsilon$
hence *nemps*: $r \neq \varepsilon\ s \neq \varepsilon$
using *facs emp-pow-emp sublist-Nil-right* **by** *metis+*
thus *?thesis*
using *fac-two-conjug-primrootE*[*OF facs nemps len*] *conjug-trans*[*OF* - *conjug-sym*[*of* ϱ s]]
by *metis*
qed

lemma *fac-two-prim-conjug*:
assumes $w \leq_f u^{@}n\ w \leq_f v^{@}m$ *primitive u primitive v* $|u| + |v| \leq |w|$
shows $u \sim v$
using *fac-two-conjug-primroot*[*OF assms*(*1*−*2*) *assms*(*5*)]
unfolding *prim-primroot*[*OF* ‹*primitive u*›] *prim-primroot*[*OF* ‹*primitive v*›].

lemma *fac-pow-conjug-primroot*: **assumes** $u^{@}k \leq_f v^{@}l$ **and** $|u^{@}k| \geq 2{*}|v|$ **and** $2 \leq k$ **and** $u \neq \varepsilon$
shows $\varrho\ u \sim \varrho\ v$
proof(*rule fac-two-conjug-primroot*[*OF* - *assms*(*1*)], *force*)
have $0 < k$
using ‹$2 \leq k$› **by** *linarith*
show $|u| + |v| \leq |u\ ^{@}\ k|$
proof(*cases* $|u|$ $|v|$ *rule*: *le-cases*)
assume $|u| \leq |v|$
thus *?thesis*
using *assms*(*2*) **by** *linarith*

```
  next
    assume |v| ≤ |u|
    hence  |u| + |v| ≤ 2*|u|
      by simp
    thus ?thesis
      unfolding pow-len
      using mult-le-mono1[OF ‹2 ≤ k›] le-trans
      by blast
  qed
qed
```

2.25 Testing primitivity

This section defines a proof method used to prove that a word is primitive.

```
lemma primitive-iff [code]: primitive w ⟷ ¬ w ≤f tl w · butlast w
proof–
  have ¬ primitive w ⟷ w ≤f tl w · butlast w
  proof
    assume ¬ primitive w
    then obtain r k where k ≠ 1 and w = r^@k
      unfolding primitive-def by blast
    show w ≤f tl w · butlast w
    proof (cases w = ε)
      assume w ≠ ε
      from this[unfolded ‹w = r^@k›]
      have  0 < k
        by blast
      have r ≠ ε
        using pow-zero ‹r ^@ k ≠ ε› by force
      have r^@(k−1) ≠ ε
        unfolding nemp-emp-pow[OF ‹r ≠ ε›, of k−1]
        using ‹0 < k› ‹k ≠ 1› by force
      have r · w · r^@(k−1) = w · w
        unfolding ‹w = r^@k› pows-comm[of k r k − 1]
        unfolding lassoc cancel-right pow-pos[OF ‹0 < k›]..
      hence [hd r] · tl r · w · butlast (r^@(k−1)) · [last (r^@(k−1))] = [hd w] · tl w ·
butlast w · [last w]
        unfolding hd-tl[reversed, OF ‹r^@(k−1) ≠ ε›] hd-tl[reversed, OF ‹w ≠ ε›]
        unfolding lassoc hd-tl[OF ‹r ≠ ε›] hd-tl[OF ‹w ≠ ε›].
      hence tl r · w · butlast (r^@(k−1)) = tl w · butlast w
        by force
      thus ?thesis
        unfolding fac-def by metis
    qed simp
  next
    assume w ≤f tl w · butlast w
    show ¬ primitive w
    proof (cases w = ε)
```

assume $w \neq \varepsilon$
from *facE*[*OF* ‹$w \leq_f tl\ w \cdot butlast\ w$›]
obtain $p\ s$ **where** $tl\ w \cdot butlast\ w = p \cdot w \cdot s$.
have $[hd\ w] \cdot (p \cdot w \cdot s) \cdot [last\ w] = w \cdot w$
unfolding ‹$tl\ w \cdot butlast\ w = p \cdot w \cdot s$›[*symmetric*]
unfolding *lassoc hd-tl*[*OF* ‹$w \neq \varepsilon$›]
unfolding *rassoc hd-tl*[*reversed*, *OF* ‹$w \neq \varepsilon$›]..
from *prim-overlap-sqE*[*of* w $[hd\ w] \cdot p\ s \cdot [last\ w]$ *False*, *unfolded rassoc*, *OF* - *this*[*unfolded rassoc*]]
show ¬ *primitive w*
by *blast*
qed *simp*
qed
thus *?thesis* **by** *blast*
qed

method *primitivity-inspection* = (*insert method-facts*, *use nothing* **in** ‹*simp add*: *primitive-iff pow-pos*›)

— This is out of scope of the method, and has to be proved separately
lemma *alternate-prim*: **assumes** $x \neq y$ **shows** *primitive* $([x,y]^{@}n{\cdot}[x])$
proof−
consider $n = 0 \mid n = 1 \mid 2 \leq n$ **by** *linarith*
then show *?thesis*
proof(*cases*)
assume $2 \leq n$
have *pref*: $[x,\ y]\ ^{@}\ n \cdot [x] \leq_p [x,\ y] \cdot [x,\ y]\ ^{@}\ n \cdot [x]$
by *comparison*
have *neq*: $([x,\ y]\ ^{@}\ n \cdot [x]) \cdot [x,\ y] \neq [x,\ y] \cdot [x,\ y]\ ^{@}\ n \cdot [x]$
using ‹$x \neq y$› **by** *force*
then show *?thesis*
using *per-le-prim-iff*[*of* $[x,y]^{@}n{\cdot}[x]$ $[x,y]$, *OF pref*] ‹$2 \leq n$›
unfolding *lenmorph pow-len*
by *fastforce*
qed (*simp-all add*: ‹$x \neq y$› *primitive-iff*)
qed

2.26 Factor interpretation

definition *factor-interpretation* :: *'a list* ⇒ *'a list* ⇒ *'a list* ⇒ *'a list list* ⇒ *bool* (- - - $\sim_{\mathcal{I}}$ - [*52*,*52*,*52*,*52*] *60*)
where *factor-interpretation p u s ws* = (*p* $<_p$ *hd ws* ∧ *s* $<_s$ *last ws* ∧ $p \cdot u \cdot s$ = *concat ws*)

lemma *fac-interp-nemp*: $u \neq \varepsilon \Longrightarrow p\ u\ s \sim_{\mathcal{I}} ws \Longrightarrow ws \neq \varepsilon$
unfolding *factor-interpretation-def* **by** *force*

lemma *fac-interpD*: **assumes** $p\ u\ s \sim_{\mathcal{I}} ws$
shows $p <_p hd\ ws$ **and** $s <_s last\ ws$ **and** $p \cdot u \cdot s = concat\ ws$

using *assms* **unfolding** *factor-interpretation-def* **by** *blast+*

lemma *fac-interpI*[*intro*]:
p <*p hd ws* ⟹ *s* <*s last ws* ⟹ *p* · *u* · *s* = *concat ws* ⟹ *p u s* ∼$_{\mathcal{I}}$ *ws*
unfolding *factor-interpretation-def* **by** *blast*

— Strict limitation on the existence of an interpretation: empty word on edge has no interpretation
lemma *fac-fac-interpE*: **assumes** *pu* · *u* · *su* = *concat ws* **and** *u* ≠ *ε*
obtains *ps ss p s vs* **where** *p u s* ∼$_{\mathcal{I}}$ *vs* **and** *ps* · *vs* · *ss* = *ws* **and** *concat ps* · *p* = *pu* **and**
s · *concat ss* = *su*
using *assms*
proof (*induction* |*ws*| *arbitrary*: *ws pu su thesis rule*: *less-induct*)
case *less*
then show *?case*
proof−
have *ws* ≠ *ε*
using ‹*u* ≠ *ε*› ‹*pu* · *u* · *su* = *concat ws*› **by** *force*
have |*tl ws*| < |*ws*| **and** |*butlast ws*| < |*ws*|
using ‹*ws* ≠ *ε*› **by** *force+*
show *thesis*
proof (*cases*)
assume *hd ws* ≤*p pu* ∨ *last ws* ≤*s su*
then show *thesis*
proof
assume *hd ws* ≤*p pu*
from *prefixE*[*OF this*]
obtain *pu*′ **where** *pu* = *hd ws* · *pu*′**.**
from ‹*pu* · *u* · *su* = *concat ws*›[*unfolded this*, *folded arg-cong*[*OF hd-tl*[*OF* ‹*ws* ≠ *ε*›], *of concat*]]
have *pu*′ · *u* · *su* = *concat* (*tl ws*)
by *force*
from *less.hyps*[*OF* ‹|*tl ws*| < |*ws*|› - ‹*pu*′ · *u* · *su* = *concat* (*tl ws*)› ‹*u* ≠ *ε*›]
obtain *p s vs ps*′ *ss* **where** *p u s* ∼$_{\mathcal{I}}$ *vs* **and** *ps*′ · *vs* · *ss* = *tl ws* **and** *concat ps*′ · *p* = *pu*′
and *s* · *concat ss* = *su***.**
have (*hd ws* # *ps*′) · *vs* · *ss* = *ws*
using ‹*ws* ≠ *ε*› ‹*ps*′ · *vs* · *ss* = *tl ws*› **by** *auto*
have *concat* (*hd ws* # *ps*′) · *p* = *pu*
using ‹*concat ps*′ · *p* = *pu*′› **unfolding** ‹*pu* = *hd ws* · *pu*′› **by** *fastforce*
from *less.prems*(*1*)[*OF* ‹*p u s* ∼$_{\mathcal{I}}$ *vs*› ‹(*hd ws* # *ps*′) · *vs* · *ss* = *ws*› ‹*concat* (*hd ws* # *ps*′) · *p* = *pu*› ‹*s* · *concat ss* = *su*›]
show *thesis***.**
next
assume *last ws* ≤*s su*
from *suffixE*[*OF this*]
obtain *su*′ **where** *su* = *su*′ · *last ws***.**
from ‹*pu* · *u* · *su* = *concat ws*›[*unfolded this*, *folded arg-cong*[*OF hd-tl*[*reversed*,

OF ‹*ws* ≠ *ε*›], *of concat*]]
have *pu* · *u* · *su′* = *concat* (*butlast ws*)
by *force*
from *less.hyps*[*OF* ‹|*butlast ws*| < |*ws*|› - ‹*pu* · *u* · *su′* = *concat* (*butlast ws*)› ‹*u* ≠ *ε*›]
obtain *p s vs ps ss′* **where** *p u s* ∼*I vs* **and** *ps* · *vs* · *ss′* = *butlast ws* **and** *concat ps* · *p* = *pu* **and** *s* · *concat ss′* = *su′*.
have *ps* · *vs* · (*ss′* · [*last ws*]) = *ws*
using *append-butlast-last-id*[*OF* ‹*ws* ≠ *ε*›, *folded* ‹*ps* · *vs* · *ss′* = *butlast ws*›] **unfolding** *rassoc*.
have *s* · *concat* (*ss′* · [*last ws*]) = *su*
using ‹*s* · *concat ss′* = *su′*› ‹*su* = *su′* · *last ws*› **by** *fastforce*
from *less.prems*(*1*)[*OF* ‹*p u s* ∼*I vs*› ‹*ps* · *vs* · (*ss′* · [*last ws*]) = *ws*› ‹*concat ps* · *p* = *pu*› ‹*s* · *concat* (*ss′* · [*last ws*]) = *su*›]
show *thesis*.
qed
next
assume *not-or*: ¬ (*hd ws* ≤*p pu* ∨ *last ws* ≤*s su*)
hence *pu* <*p hd ws* **and** *su* <*s last ws*
using *ruler*[*OF concat-hd-pref*[*OF* ‹*ws* ≠ *ε*›] *prefI′*[*OF* ‹*pu* · *u* · *su* = *concat ws*›[*symmetric*]]]
ruler[*reversed*, *OF concat-hd-pref*[*reversed*, *OF* ‹*ws* ≠ *ε*›] *prefI′*[*reversed*, *OF* ‹*pu* · *u* · *su* = *concat ws*›[*symmetric*, *unfolded lassoc*]]] **by** *auto*
from *fac-interpI*[*OF this* ‹*pu* · *u* · *su* = *concat ws*›]
have *pu u su* ∼*I ws*.
from *less.prems*(*1*)[*OF this*, *of ε ε*]
show *thesis* **by** *simp*
qed
qed
qed

lemma *pref-interpE*: **assumes** *p u s* ∼*I ws v* ≤*p u v* ≠ *ε*
obtains *vs s′* **where** *p v s′* ∼*I vs vs* ≤*p ws*
proof−
have *p* · (*v* · *v*$^{-1>}$*u*) · *s* = *concat ws*
using *fac-interpD*(*3*)[*OF* ‹*p u s* ∼*I ws*›] **unfolding** *lq-pref*[*OF* ‹*v* ≤*p u*›].
from *fac-fac-interpE*[*of p v v*$^{-1>}$*u* · *s ws*, *OF this*[*unfolded rassoc*] ‹*v* ≠ *ε*›] *fac-interpD*(*3*)[*OF* ‹*p u s* ∼*I ws*›] *assms*
obtain *ps ss pa sa vs* **where** *pa v sa* ∼*I vs ps* · *vs* · *ss* = *ws concat ps* · *pa* = *p sa* · *concat ss* = *v*$^{-1>}$*u* · *s*
by *metis*
from *fac-interpD*(*1*)[*OF* ‹*p u s* ∼*I ws*›, *folded* ‹*concat ps* · *pa* = *p*› ‹*ps* · *vs* · *ss* = *ws*›]
have *ps* = *ε*
using *concat-hd-pref hd-append2 prefix-order.leD prefix-prefix* **by** *metis*
hence *pa* = *p* **and** *vs* ≤*p ws*
using ‹*concat ps* · *pa* = *p*› ‹*ps* · *vs* · *ss* = *ws*› **by** *force*+
from *that*[*OF* ‹*pa v sa* ∼*I vs*›[*unfolded this*] *this*(*2*)]
show *thesis*.

qed

lemma *fac-fac-interpE'*: **assumes** $u \leq_f concat\ ws$ **and** $u \neq \varepsilon$
obtains $p\ s\ vs$ **where** $p\ u\ s \sim_{\mathcal{I}} vs$ **and** $vs \leq_f ws$
proof−
from *facE*[*OF* ‹$u \leq_f concat\ ws$›]
obtain $pu\ su$ **where** $concat\ ws = pu \cdot u \cdot su$**.**
from *fac-fac-interpE*[*OF this*[*symmetric*] ‹$u \neq \varepsilon$›] *that*
show *thesis*
using *facI'* **by** *metis*
qed

lemma *fac-pow-longE*: **assumes** $w \leq_f v^{@}k$ **and** $|v| \leq |w|$
obtains $m\ v1\ v2$ **where** $v1 \leq_s v\ v2 \leq_p v\ w = v1 \cdot v^{@}m \cdot v2$
using *assms*
proof (*cases* $w = \varepsilon \vee w = v$)
assume $w = \varepsilon \vee w = v$
then show *thesis*
by (*rule disjE*) (*use that*[*of* $\varepsilon\ \varepsilon\ 0$] **in** *fastforce*, *use that*[*of* $\varepsilon\ \varepsilon\ 1$] **in** *auto*)
next
assume $\neg\ (w = \varepsilon \vee w = v)$
hence $w \neq \varepsilon$ **and** $w \neq v$ **by** *blast*+
have $v^{@}k = concat\ ([v]^{@}k)$
by *auto*
from *fac-fac-interpE'*[*OF* ‹$w \leq_f v^{@}k$›[*unfolded this*] ‹$w \neq \varepsilon$›]
obtain $p\ vs\ s$ **where** $p\ w\ s \sim_{\mathcal{I}} vs\ vs \leq_f [v]^{@}k$**.**
note *fac-interpD*[*OF this*(*1*)]
obtain m **where** $vs = [v]^{@}m$
using ‹$vs \leq_f [v]^{@}k$› *unique-letter-fac-expE* **by** *meson*
hence $concat\ vs = v^{@}m$
by *simp*
from *lenarg*[*OF* ‹$p \cdot w \cdot s = concat\ vs$›, *unfolded this lenmorph pow-len*]
have $0 \neq m$
using ‹$|v| \leq |w|$› ‹$\neg\ (w = \varepsilon \vee w = v)$› **by** *force*
hence $hd\ vs = v$ **and** $last\ vs = v$
unfolding ‹$vs = [v]^{@}m$›
by (*simp-all add*: *hd-pow-list-single hd-pow-list-single*[*reversed*])

obtain *v1* **where** $v = p \cdot v1$
using ‹$p <_p hd\ vs$› **unfolding** ‹$hd\ vs = v$› *strict-prefix-def prefix-def* **by** *force*
obtain *v2* **where** $v = v2 \cdot s$
using ‹$s <_s last\ vs$› **unfolding** ‹$last\ vs = v$› *strict-suffix-def suffix-def* **by** *force*
have $m \neq 1$
using ‹$p \cdot w \cdot s = concat\ vs$› **unfolding** ‹$concat\ vs = v^{@}m$›
using ‹$w \neq v$› ‹$|v| \leq |w|$› **by** *force*
hence $2 \leq m$
using ‹$0 \neq m$› **by** *linarith*
from *Suc-minus2*[*OF this*]
have $concat\ vs = v \cdot v^{@}(m-2) \cdot v$

unfolding *pow-Suc2*[*symmetric*] *pow-Suc*[*symmetric*] ‹*concat vs* = $v^{@}m$› **by** *argo*
hence *w* = *v1* · $v^{@}$(*m*−*2*) · *v2*
by (*subst*(*asm*) ‹*v* = *p* · *v1*›, *subst*(*asm*) (*2*) ‹*v* = *v2* · *s*›)
(*simp add*: ‹*p* · *w* · *s* = *concat vs*›[*symmetric*])
from *that*[*OF - - this*]
show *thesis*
using ‹*v* = *p* · *v1*› ‹*v* = *v2* · *s*› **by** *blast*
qed

lemma *fac-interp-inner*: **assumes** *u* ≠ *ε* **and** *p u s* ∼$_{\mathcal{I}}$ *ws* **and** *1* < |*ws*|
shows $p^{-1>}$(*hd ws*)·*concat*(*butlast* (*tl ws*))·(*last ws*)$^{<-1}$*s* = *u*
proof−
have *p* <*p hd ws* **and** *s* <*s last ws* **and** *p* · *u* · *s* = *concat ws*
using *assms*[*unfolded factor-interpretation-def*] **by** *blast*+
have *last* (*tl ws*) = *last ws*
using *last-tl long-list-tl*[*OF* ‹*1* < |*ws*|›] **by** *blast*
have *ws-eq*: [*hd ws*] · *butlast* (*tl ws*) · [*last ws*] = *ws*
using *hd-tl*[*OF fac-interp-nemp*[*OF* ‹*u* ≠ *ε*› ‹*p u s* ∼$_{\mathcal{I}}$ *ws*›]] *append-butlast-last-id*[*OF long-list-tl*[*OF* ‹*1* < |*ws*|›], *unfolded* ‹*last* (*tl ws*) = *last ws*›] **by** *simp*
from *arg-cong*[*OF this*, *of concat*, *unfolded concat-morph*, *unfolded concat-sing′*, *folded* ‹*p* · *u* · *s* = *concat ws*›]
have (*hd ws*)·*concat*(*butlast* (*tl ws*))·(*last ws*) = *p* · *u* · *s*.
thus $p^{-1>}$(*hd ws*)·*concat*(*butlast* (*tl ws*))·(*last ws*)$^{<-1}$*s* = *u*
unfolding *cancel-right*[*of* $p^{-1>}$(*hd ws*)·*concat* (*butlast* (*tl ws*)) · *last ws*$^{<-1}$*s s u*, *symmetric*]
unfolding *rassoc rq-suf*[*OF ssufD1*[*OF* ‹*s* <*s last ws*›]]
unfolding *cancel*[*of p* $p^{-1>}$*hd ws* · *concat* (*butlast* (*tl ws*)) · *last ws u*·*s*, *symmetric*]
unfolding *lassoc lq-spref*[*OF* ‹*p* <*p hd ws*›].
qed

lemma *fac-interp-inner-len*: **assumes** *u* ≠ *ε* **and** *p u s* ∼$_{\mathcal{I}}$ *ws*
shows |*concat*(*butlast* (*tl ws*))| < |*u*|
proof (*cases* |*ws*| ≤ *1*)
assume |*ws*| ≤ *1*
hence *tl ws* = *ε*
by *blast*
thus *?thesis*
using ‹*u* ≠ *ε*› **by** *simp*
next
assume *neg*: ¬ |*ws*| ≤ *1* **hence** *1* < |*ws*| **by** *auto*
from *lenarg*[*OF fac-interp-inner*[*OF* ‹*u* ≠ *ε*› ‹*p u s* ∼$_{\mathcal{I}}$ *ws*› *this*]] ‹*p u s* ∼$_{\mathcal{I}}$ *ws*›
show *?thesis*
unfolding *factor-interpretation-def lenmorph*
using *rq-ssuf-nemp*[*of s last ws*, *folded length-greater-0-conv*]
by *linarith*
qed

lemma *rev-in-set-map-rev-conv*: *rev u* ∈ *set* (*map rev ws*) ⟷ *u* ∈ *set ws*
by *auto*

lemma *rev-fac-interp*: **assumes** *p u s* ∼$_{\mathcal{I}}$ *ws* **shows** (*rev s*) (*rev u*) (*rev p*) ∼$_{\mathcal{I}}$ *rev* (*map rev ws*)
proof (*rule fac-interpI*)
note *fac-interpD*[*OF assms*]
show ‹*rev s* <*p hd* (*rev* (*map rev ws*))›
using ‹*s* <*s last ws*›
by (*metis* ‹*p* <*p hd ws*› ‹*p* · *u* · *s* = *concat ws*› *append-is-Nil-conv concat.simps*(*1*) *hd-rev last-map list.simps*(*8*) *rev-is-Nil-conv strict-suffix-to-prefix*)
show *rev p* <*s last* (*rev* (*map rev ws*))
using ‹ *p* <*p hd ws*›
by (*metis* ‹*p* · *u* · *s* = *concat ws*› ‹*s* <*s last ws*› *append-is-Nil-conv concat.simps*(*1*) *last-rev list.map-sel*(*1*) *list.simps*(*8*) *rev-is-Nil-conv spref-rev-suf-iff*)
show *rev s* · *rev u* · *rev p* = *concat* (*rev* (*map rev ws*))
using ‹*p* · *u* · *s* = *concat ws*›
by (*metis append-assoc rev-append rev-concat rev-map*)
qed

lemma *rev-fac-interp-iff* [*reversal-rule*]: (*rev s*) (*rev u*) (*rev p*) ∼$_{\mathcal{I}}$ *rev* (*map rev ws*) ⟷ *p u s* ∼$_{\mathcal{I}}$ *ws*
using *rev-fac-interp*
by (*metis* (*no-types*, *lifting*) *map-rev-involution rev-map rev-rev-ident*)

lemma *fac-interp-mid-fac*: **assumes** *p u s* ∼$_{\mathcal{I}}$ *ws*
shows *concat* (*butlast* (*tl ws*)) ≤*f u*
proof(*rule le-less-cases*)
assume *1* < |*ws*|
note *fac-interpD*[*OF* ‹*p u s* ∼$_{\mathcal{I}}$ *ws*›]
hd-middle-last[*OF* ‹*1* <|*ws*|›]
note *ex* = *sprefD1*[*OF this*(*1*)] *sprefE*[*reversed*, *OF this*(*2*)]
obtain *p*′ **where** *hd ws* = *p* · *p*′
using *ex*(*1*) *prefixE*
by *blast*
obtain *s*′ **where** *last ws* = *s*′ · *s*
using ‹*s* <*s last ws*› **by** (*blast elim*: *ssufE sufE*)
show *?thesis*
using ‹*p* · *u* · *s* = *concat ws*›
unfolding *arg-cong*[*OF* ‹[*hd ws*] · *butlast* (*tl ws*) · [*last ws*] = *ws*›[*symmetric*], *of concat*]
unfolding *concat-morph concat-sing*′
unfolding ‹*hd ws* = *p* · *p*′› ‹*last ws* = *s*′ · *s*›
by *simp*
qed (*simp add*: *tl-emp*)

definition *disjoint-interpretation* :: *'a list* ⇒ *'a list list* ⇒ *'a list* ⇒ *'a list list* ⇒ *bool* (- - - ∼$_{\mathcal{D}}$ - [*51*,*51*,*51*,*51*] *60*)
where *p us s* ∼$_{\mathcal{D}}$ *ws* ≡ *p* (*concat us*) *s* ∼$_{\mathcal{I}}$ *ws* ∧

$(\forall\ u\ v.\ u \leq_p us \wedge v \leq_p ws \longrightarrow p \cdot concat\ u \neq concat\ v)$

lemma *disj-interpI*: $p\ (concat\ us)\ s \sim_{\mathcal{I}} ws \Longrightarrow$
$(\forall\ u\ v.\ u \leq_p us \wedge v \leq_p ws \longrightarrow p \cdot concat\ u \neq concat\ v) \Longrightarrow p\ us\ s \sim_{\mathcal{D}} ws$
unfolding *disjoint-interpretation-def* **by** *blast*

lemma *disj-interpI'*[*intro*]: $p\ (concat\ us)\ s \sim_{\mathcal{I}} ws \Longrightarrow$
$(\bigwedge u\ v.\ u \leq_p us \Longrightarrow v \leq_p ws \Longrightarrow\ p \cdot concat\ u \neq concat\ v) \Longrightarrow p\ us\ s \sim_{\mathcal{D}} ws$
unfolding *disjoint-interpretation-def* **by** *blast*

lemma *disj-interpD0*: $p\ us\ s \sim_{\mathcal{D}} ws \Longrightarrow p\ (concat\ us)\ s \sim_{\mathcal{I}} ws$
unfolding *disjoint-interpretation-def* **by** *blast*

lemmas *disj-interpD* = *fac-interpD*[*OF disj-interpD0*]

lemma *disj-interpD1*: **assumes** $p\ us\ s \sim_{\mathcal{D}} ws$ **and** $us' \leq_p us$ **and** $ws' \leq_p ws$
shows $p \cdot concat\ us' \neq concat\ ws'$
using *assms* **unfolding** *disjoint-interpretation-def* **by** *blast*

lemma *disj-interp-nemp*: **assumes** $p\ us\ s \sim_{\mathcal{D}} ws$
shows $p \neq \varepsilon$ **and** $s \neq \varepsilon$ **and** $ws \neq \varepsilon$
proof−
show $p \neq \varepsilon$ **and** $s \neq \varepsilon$
using *disj-interpD1*[*OF assms emp-pref emp-pref*]
disj-interpD1[*OF assms self-pref self-pref*, *folded*
fac-interpD(*3*)[*OF disj-interpD0*, *OF assms*], *unfolded cancel*]
by *blast*+
thus $ws \neq \varepsilon$
using *fac-interpD*(*3*)[*OF disj-interpD0*[*OF assms*]] **by** *force*
qed

lemma *non-disjoint-interpE*: **assumes** $p\ (concat\ us)\ s \sim_{\mathcal{I}} ws$ **and**
$\neg\ p\ us\ s \sim_{\mathcal{D}} ws$
obtains *ws1 ws2 us1 us2* **where** $us1 \cdot us2 = us\ ws1 \cdot ws2 = ws$
$p \cdot concat\ us1 = concat\ ws1\ concat\ us2 \cdot s = concat\ ws2$
proof−
obtain *ws1 us1* **where** $ws1 \leq_p ws\ us1 \leq_p us\ p \cdot concat\ us1 = concat\ ws1$
using ‹$\neg\ p\ us\ s \sim_{\mathcal{D}} ws$› ‹$p\ (concat\ us)\ s \sim_{\mathcal{I}} ws$›
unfolding *disjoint-interpretation-def*
by *blast*
have $p \cdot (concat\ us1 \cdot concat\ (us1^{-1>}us)) \cdot s = concat\ ws1 \cdot concat\ (ws1^{-1>}ws)$
unfolding *concat-morph*[*symmetric*] *lq-pref*[*OF* ‹*ws1* $\leq_p$ *ws*›] *lq-pref*[*OF* ‹*us1* $\leq_p$ *us*›]
using *fac-interpD*(*3*)[*OF* ‹$p\ (concat\ us)\ s \sim_{\mathcal{I}} ws$›].
hence $concat\ (us1^{-1>}us) \cdot s = concat\ (ws1^{-1>}ws)$
unfolding *lassoc* ‹$p \cdot concat\ us1 = concat\ ws1$› **unfolding** *rassoc cancel*.
from *that*[*OF lq-pref*[*OF* ‹*us1* $\leq_p$ *us*›] *lq-pref*[*OF* ‹*ws1* $\leq_p$ *ws*›] ‹$p \cdot concat\ us1 = concat\ ws1$› *this*]
show *thesis*.

qed

lemma *emp-disj-interp*: **assumes** $p \neq \varepsilon$ $p <p u$
shows $p\ \varepsilon\ (p^{-1>}u) \sim_{\mathcal{D}} [u]$
proof
show $p\ (concat\ \varepsilon)\ (p^{-1>}u) \sim_{\mathcal{I}} [u]$
proof
show $p^{-1>}u <s last\ [u]\ p <p hd\ [u]\ p \cdot concat\ \varepsilon \cdot p^{-1>}u = concat\ [u]$
by (*metis* ‹$p <p u$› ‹$p \neq \varepsilon$› *last-sing lq-pref ssufI1 strict-prefixE*)
(*use* ‹$p <p u$› **in** *force*)+
qed
show $\bigwedge y\ v.\ y \leq p\ \varepsilon \Longrightarrow v \leq p\ [u] \Longrightarrow p \cdot concat\ y \neq concat\ v$
using ‹$p \neq \varepsilon$› ‹$p <p u$›
using *concat.simps(1) concat-sing nemp-pref-nemp pref-singE append-Nil2 sprefD* **by** *metis*
qed

lemma *emp-fac-interp*: $p \neq \varepsilon \Longrightarrow s \neq \varepsilon \Longrightarrow p\ \varepsilon\ s \sim_{\mathcal{I}} [p \cdot s]$
by (*rule fac-interpI*, *auto*)

lemma *emp-disj-interp′*:
assumes $1 < |w|$ **shows** $[hd\ w]\ \varepsilon\ (tl\ w) \sim_{\mathcal{D}} [w]$
proof (*rule disj-interpI*)
have $w \neq \varepsilon\ [hd\ w] \neq \varepsilon\ tl\ w \neq \varepsilon$
using *assms long-list-tl* **by** *auto*
show $[hd\ w]\ (concat\ \varepsilon)\ (tl\ w) \sim_{\mathcal{I}} [w]$
using *emp-fac-interp*[*OF* ‹$[hd\ w] \neq \varepsilon$› ‹$tl\ w \neq \varepsilon$›, *unfolded hd-tl*[*OF* ‹$w \neq \varepsilon$›]]
by *simp*
have $[hd\ w] \cdot concat\ u \neq concat\ v$ **if** $u \leq p\ \varepsilon$ **and** $v \leq p\ [w]$ **for** $u\ v$
unfolding *pref-emp*[*OF* ‹$u \leq p\ \varepsilon$›] *concat.simps cow-simps*
proof (*rule*, *rule pref-singE*[*OF* ‹$v \leq p\ [w]$›])
assume $v = [w]\ [hd\ (w)] = concat\ v$
from *lenarg*[*OF this*(*2*), *unfolded this*(*1*)]
show *False*
using ‹$1 < |w|$› **by** *force*
qed *auto*
then show $\forall u\ v.\ u \leq p\ \varepsilon \wedge v \leq p\ [w] \longrightarrow [hd\ w] \cdot concat\ u \neq concat\ v$
by *blast*
qed

lemma *pref-disj-interpE*: **assumes** $p\ us\ s \sim_{\mathcal{D}} ws\ vs \leq p\ us$
obtains $ws'\ s'$ **where** $p\ vs\ s' \sim_{\mathcal{D}} ws'\ ws' \leq p\ ws$
proof (*cases concat vs* $= \varepsilon$)
have $p \neq \varepsilon\ p <p hd\ ws$
using *disj-interp-nemp*(*1*)[*OF* ‹$p\ us\ s \sim_{\mathcal{D}} ws$›] *fac-interpD*(*1*)[*OF disj-interpD0*[*OF* ‹$p\ us\ s \sim_{\mathcal{D}} ws$›]].
from *emp-disj-interp*[*OF this*]
have *emp*: $p\ \varepsilon\ (p^{-1>}hd\ ws) \sim_{\mathcal{D}} [hd\ ws]$.
from *disj-interpD0*[*OF emp*, *simplified*]

have $p\ \varepsilon\ (p^{-1>}hd\ ws) \sim_{\mathcal{I}} [hd\ ws]$**.**
assume *concat vs* $= \varepsilon$
have $p\ vs\ (p^{-1>}hd\ ws) \sim_{\mathcal{D}} [hd\ ws]$
by (*rule disj-interpI*, *unfold* ‹*concat vs* $= \varepsilon$›, *fact*)
(*use* ‹$p <p\ hd\ ws$› *disj-interpD1*[*OF emp*] ‹*concat vs* $= \varepsilon$›
emp-pref nemp-pref-nemp pref-concat-pref **in** *metis*)
from *that*[*OF this*]
show *thesis*
using *disj-interp-nemp*(*3*)[*OF* ‹$p\ us\ s \sim_{\mathcal{D}} ws$›] **by** *blast*
next
assume *concat vs* $\neq \varepsilon$
have *disj*: $\forall\ u\ v.\ u \leq p\ vs \wedge v \leq p\ w \longrightarrow p \cdot concat\ u \neq concat\ v$ **if** $w \leq p\ ws$ **for** *w*
using *disj-interpD1*[*OF* ‹$p\ us\ s \sim_{\mathcal{D}} ws$›] *pref-trans*[*OF* - ‹$vs \leq p\ us$›] *pref-trans*[*OF* - ‹$w \leq p\ ws$›]
by *presburger*
have *concat vs* $\leq p$ *concat us*
using ‹$vs \leq p\ us$› **by** *blast*
have $p\ (concat\ us)\ s \sim_{\mathcal{I}} ws$
using *disj-interpD0*[*OF* ‹$p\ us\ s \sim_{\mathcal{D}} ws$›]**.**
from *pref-interpE*[*OF this* ‹*concat vs* $\leq p$ *concat us*› ‹*concat vs* $\neq \varepsilon$›, *of thesis*] *that*[*OF disj-interpI*, *OF* - *disj*]
show *thesis*
by *blast*
qed

lemma *rev-disj-interp*: **assumes** $p\ us\ s \sim_{\mathcal{D}} ws$ **shows** $(rev\ s)\ (rev\ (map\ rev\ us))\ (rev\ p) \sim_{\mathcal{D}} rev\ (map\ rev\ ws)$
proof
show $(rev\ s)\ (concat\ (rev\ (map\ rev\ us)))\ (rev\ p) \sim_{\mathcal{I}} rev\ (map\ rev\ ws)$
using *disj-interpD0*[*OF assms*] **unfolding** *concat-rev-map-rev rev-fac-interp-iff*.
have *aux*: *concat u* $\cdot$ *s* $\neq$ *concat v* **if** $u \leq s\ us\ v \leq s\ ws$ **for** *u v*
proof
assume *concat u* $\cdot$ *s* $=$ *concat v*
note *fac-interpD*(*3*)[*OF disj-interpD0*[*OF assms*]] **and**
arg-cong[*OF rq-suf*[*OF* ‹$u \leq s\ us$›], *of concat*, *unfolded concat-morph*] **and**
arg-cong[*OF rq-suf*[*OF* ‹$v \leq s\ ws$›], *of concat*, *unfolded concat-morph*]
from *this*(*1*)[*folded this*(*2*−*3*)]
have $p \cdot (concat\ (us^{<-1}u)) = concat\ (ws^{<-1}v)$
unfolding *rassoc* ‹*concat u* $\cdot$ *s* $=$ *concat v*› **by** *simp*
thus *False*
using *disj-interpD1*[*OF assms*, *of* $us^{<-1}u\ ws^{<-1}v$] **by** *simp*
qed
show *rev s* $\cdot$ *concat u* $\neq$ *concat v* **if** $u \leq p\ rev\ (map\ rev\ us)\ v \leq p\ rev\ (map\ rev\ ws)$ **for** *u v*
proof–
have $rev\ (map\ rev\ u) \leq s\ us\ rev\ (map\ rev\ v) \leq s\ ws$
using *map-mono-prefix*[*OF* ‹$u \leq p\ rev\ (map\ rev\ us)$›, *of rev*] *map-mono-prefix*[*OF* ‹$v \leq p\ rev\ (map\ rev\ ws)$›, *of rev*]
unfolding *suf-rev-pref-iff rev-map rev-rev-ident map-rev-involution*.

from *aux*[*reversed*, *OF this*]
show *rev s · concat u ≠ concat v*
unfolding *rev-append*[*of rev* (*concat u*) *s*, *unfolded rev-rev-ident*, *symmetric*] *rev-rev-ident rev-swap*.
qed
qed

lemma *rev-disj-interp-iff* [*reversal-rule*]: (*rev s*) (*rev* (*map rev us*)) (*rev p*) $\sim_{\mathcal{D}}$ *rev* (*map rev ws*) $\longleftrightarrow$ *p us s* $\sim_{\mathcal{D}}$ *ws*
using *rev-disj-interp*[*of rev s rev* (*map rev us*) *rev p rev* (*map rev ws*)] *rev-disj-interp*
unfolding *map-rev-involution rev-disj-interp rev-map rev-rev-ident* **by** *blast+*

lemma *disj-interp-split*:
assumes *disj*: *p us s* $\sim_{\mathcal{D}}$ *ws* **and** *us* = $us_1 \cdot us_2$
obtains ws_1 ws_2 p' s' **where** *p* us_1 s' $\sim_{\mathcal{D}}$ $ws_1 \cdot [p' \cdot s']$ p' us_2 *s* $\sim_{\mathcal{D}}$ $[p' \cdot s'] \cdot ws_2$ *ws* = $ws_1 \cdot [p' \cdot s'] \cdot ws_2$
proof–
from *pref-disj-interpE*[*OF disj triv-pref*[*of* us_1 us_2, *folded* ‹*us* = $us_1 \cdot us_2$›]]
obtain $ws_1{}'$ s' **where** *p* us_1 s' $\sim_{\mathcal{D}}$ $ws_1{}'$ $ws_1{}' \leq_p ws$.
note *disj-interpD*[*OF* ‹*p us s* $\sim_{\mathcal{D}}$ *ws*›]
note *disj-interpD*[*OF* ‹*p* us_1 s' $\sim_{\mathcal{D}}$ $ws_1{}'$›]
note *disj-interp-nemp*[*OF* ‹*p* us_1 s' $\sim_{\mathcal{D}}$ $ws_1{}'$›]
obtain p' **where** $p' \cdot s' = last\ ws_1{}'\ p' \neq \varepsilon$
using *ssuf-exE*[*OF* ‹$s' <_s last\ ws_1{}'$›] **by** *metis*
hence $p' <_p last\ ws_1{}'$
using ‹$p' \cdot s' = last\ ws_1{}'$› ‹$s' \neq \varepsilon$› **by** *blast*
obtain $ws_2{}'$ **where** $butlast\ ws_1{}' \cdot ws_2{}' = ws$ **and** $hd\ ws_2{}' = p' \cdot s'$
using ‹$p' \cdot s' = last\ ws_1{}'$›
append-butlast-last-id[*OF* ‹$ws_1{}' \neq \varepsilon$›] ‹$ws_1{}' \leq_p ws$›
append-assoc hd-Cons-append prefD **by** *metis*
have $ws_2{}' \neq \varepsilon$
using ‹$butlast\ ws_1{}' \cdot ws_2{}' = ws$› ‹$ws_1{}' \leq_p ws$› ‹$ws_1{}' \neq \varepsilon$› **by** *fastforce*
from ‹$p \cdot concat\ us_1 \cdot s' = concat\ ws_1{}'$›
have $p \cdot concat\ us_1 \cdot s' = concat\ (butlast\ ws_1{}') \cdot last\ ws_1{}'$
unfolding *concat-butlast-last*[*OF* ‹$ws_1{}' \neq \varepsilon$›].
have p' us_2 *s* $\sim_{\mathcal{D}}$ $ws_2{}'$
proof (*rule disj-interpI*)
have *eq*: $p \cdot concat\ us_1 = concat\ (butlast\ ws_1{}') \cdot p'$
using ‹$p \cdot concat\ us_1 \cdot s' = concat\ (butlast\ ws_1{}') \cdot last\ ws_1{}'$›
unfolding ‹$p' \cdot s' = last\ ws_1{}'$›[*symmetric*] *lassoc cancel-right*.
show p' (*concat* us_2) *s* $\sim_{\mathcal{I}}$ $ws_2{}'$
proof (*rule fac-interpI*)
show $p' <_p hd\ ws_2{}'$
using ‹$hd\ ws_2{}' = p' \cdot s'$› ‹$p' <_p last\ ws_1{}'$› ‹$p' \cdot s' = last\ ws_1{}'$› **by** *auto*
show *s* $<_s$ *last* $ws_2{}'$
using ‹$butlast\ ws_1{}' \cdot ws_2{}' = ws$› ‹*s* $<_s$ *last ws*› ‹$ws_2{}' \neq \varepsilon$› **by** *force*
show $p' \cdot concat\ us_2 \cdot s = concat\ ws_2{}'$
proof–

show *?thesis*
using ‹$p \cdot concat\ us \cdot s = concat\ ws$›
unfolding ‹$us = us_1 \cdot us_2$› ‹$butlast\ ws_1{}' \cdot ws_2{}' = ws$›[*symmetric*]
unfolding *lassoc cancel-right concat-morph lassoc eq*
unfolding *rassoc cancel*.
qed
qed
have $p' \cdot concat\ u \neq concat\ v$ **if** $u \leq_p us_2$ $v \leq_p ws_2{}'$ **for** $u\ v$
proof
note *disj-interpD1*[*OF disj*] *eq*
have *p1*: $us_1 \cdot u \leq_p us$
using ‹$us = us_1 \cdot us_2$› ‹$u \leq_p us_2$› **by** *simp*
have *p2*: $butlast\ ws_1{}' \cdot v \leq_p ws$
using ‹$v \leq_p ws_2{}'$› ‹$butlast\ ws_1{}' \cdot ws_2{}' = ws$› **by** *force*
assume $p' \cdot concat\ u = concat\ v$
hence $p \cdot concat\ (us_1 \cdot u) = concat\ (butlast\ ws_1{}' \cdot v)$
unfolding *concat-morph lassoc eq* **by** *simp*
thus *False*
using *disj-interpD1*[*OF disj p1 p2*] **by** *blast*
qed
then show $\forall\, u\ v.\ u \leq_p us_2 \wedge v \leq_p ws_2{}' \longrightarrow p' \cdot concat\ u \neq concat\ v$
by *blast*
qed
have $ws_2{}' \neq \varepsilon$
using *disj-interp-nemp(3)*[*OF* ‹$p'\ us_2\ s \sim_{\mathcal{D}} ws_2{}'$›] **by** *blast*
have $p\ us_1\ s' \sim_{\mathcal{D}} butlast\ ws_1{}' \cdot [p' \cdot s']\ p'\ us_2\ s \sim_{\mathcal{D}} [p' \cdot s'] \cdot tl\ ws_2{}'$
unfolding *hd-tl*[*reversed*, *OF* ‹$ws_1{}' \neq \varepsilon$›, *folded* ‹$p' \cdot s' = last\ ws_1{}'$›]
hd-tl[*OF* ‹$ws_2{}' \neq \varepsilon$›, *unfolded* ‹$hd\ ws_2{}' = p' \cdot s'$›] **by** *fact+*
from *that*[*OF this*]
show *thesis*
using ‹$butlast\ ws_1{}' \cdot ws_2{}' = ws$› *hd-tl*[*OF* ‹$ws_2{}' \neq \varepsilon$›, *unfolded* ‹$hd\ ws_2{}' = p' \cdot s'$›] **by** *force*
qed

corollary *disj-interp-fac*:
assumes *disj*: $p\ us\ s \sim_{\mathcal{D}} ws$ **and** $us = us_1 \cdot us_2 \cdot us_3$
obtains $ws_1\ ws_2\ ws_3\ p'\ s'$ **where** $p'\ us_2\ s' \sim_{\mathcal{D}} ws_2$
and $p \cdot concat\ us_1 = concat\ ws_1 \cdot p'\ concat\ us_3 \cdot s = s' \cdot concat\ ws_3$
and $ws = ws_1 \cdot ws_2 \cdot ws_3$
proof–
from *disj-interp-split*[*OF disj* ‹$us = us_1 \cdot us_2 \cdot us_3$›]
obtain $ws_1\ ws_2{}'\ p'\ s''$ **where** $p\ us_1\ s'' \sim_{\mathcal{D}} ws_1 \cdot [p' \cdot s'']\ p'\ us_2 \cdot us_3\ s \sim_{\mathcal{D}} [p' \cdot s''] \cdot ws_2{}'\ ws = ws_1 \cdot [p' \cdot s''] \cdot ws_2{}'$.

from *disj-interp-split*[*OF* ‹$p'\ us_2 \cdot us_3\ s \sim_{\mathcal{D}} [p' \cdot s''] \cdot ws_2{}'$› *refl*]
obtain $ws_2{}''\ ws_3\ p''\ s'$ **where** $p'\ us_2\ s' \sim_{\mathcal{D}} ws_2{}'' \cdot [p'' \cdot s']\ p''\ us_3\ s \sim_{\mathcal{D}} [p'' \cdot s'] \cdot ws_3$
$[p' \cdot s''] \cdot ws_2{}' = ws_2{}'' \cdot [p'' \cdot s'] \cdot ws_3$.

```
  have ws = ws₁ · (ws₂'' · [p'' · s'']) · ws₃
    using ‹[p' · s''] · ws₂' = ws₂'' · [p'' · s''] · ws₃› ‹ws = ws₁ · [p' · s''] · ws₂'› by
auto
  from fac-interpD(3)[OF disj-interpD0, OF ‹p'' us₃ s ∼_D [p'' · s''] · ws₃›]
  have concat us₃ · s = s' · concat ws₃
    by force
  from fac-interpD(3)[OF disj-interpD0, OF ‹p us₁ s'' ∼_D ws₁ · [p' · s'']›]
  have p · concat us₁ = concat ws₁ · p'
    by force
  show thesis
    by (rule that) fact+
qed

lemma disj-interp-alternate:
  assumes disj: p us s ∼_D ws and 1 < |ws|
  obtains us₁ us₂ us' p' s' where p' (butlast (tl ws)) s' ∼_D us'
        hd ws = p · concat us₁ · p' last ws = s' · concat us₂ · s us₁ · us' · us₂ = us
proof−
  note disj-interpD[OF disj]
  note disj-interpD1[OF disj]
  let ?ws = butlast (tl ws)
  from hd-middle-last[OF ‹1 < |ws|›]
  have dec: [hd ws] · ?ws · [last ws] = ws.
  from arg-cong[OF this, of concat]
  have eq: hd ws · (concat ?ws) · last ws = concat ([p] · us · [s])
    unfolding concat-morph concat-sing' ‹p · concat us · s = concat ws›.
  with lq-spref[OF ‹p <p hd ws›] rq-suf[OF ssufD1, OF ‹s <s last ws›]
  have (p · p^-1>hd ws) · concat ?ws · (last ws^<-1 s · s) = p · concat us · s
    by simp
  hence p^-1>hd ws · concat ?ws · last ws^<-1 s = concat us
    by force
  show thesis
  proof (cases concat ?ws = ε)
    assume concat ?ws ≠ ε
    from fac-fac-interpE[OF ‹p^-1>hd ws · concat ?ws · last ws^<-1 s = concat us›
this]
    obtain ps ss pa sa vs where pa (concat ?ws) sa ∼_I vs ps · vs · ss = us
      concat ps · pa = p^-1>hd ws sa · concat ss = last ws^<-1 s.
    show thesis
    proof (rule that)
      show hd ws = p · concat ps · pa
        using ‹concat ps · pa = p^-1>hd ws› ‹p <p hd ws› by auto
      show last ws = sa · concat ss · s
        unfolding lassoc ‹sa · concat ss = last ws^<-1 s› ‹last ws^<-1 s · s = last
ws›..
      show pa ?ws sa ∼_D vs
      proof (rule disj-interpI[OF ‹pa (concat ?ws) sa ∼_I vs›])
        have pa · concat u ≠ concat v if u ≤p butlast (tl ws) and v ≤p vs for u v
        proof
```

```
        assume pa · concat u = concat v
        have ps · v ≤p us
          using ‹ps · vs · ss = us› ‹v ≤p vs› by force
        moreover have [hd ws] · u ≤p  [hd ws] · butlast (tl ws) · [last ws]
          using  ‹u ≤p butlast (tl ws)› by force
        moreover have  p · concat (ps · v) = concat ([hd ws] · u)
          by (simp add: ‹hd ws = p · concat ps · pa› ‹pa · concat u = concat v›)
        ultimately show False
          unfolding dec
           using ‹ps · v ≤p us ⟹ [hd ws] · u ≤p ws ⟹ p · concat (ps · v) ≠
concat ([hd ws] · u)›
          by blast
      qed
      thus ∀ u v. u ≤p butlast (tl ws) ∧ v ≤p vs ⟶ pa · concat u ≠ concat v
        by blast
    qed
  qed fact
 next
  assume concat ?ws = ε
  have p⁻¹>hd ws ≠ ε
    using ‹p <p hd ws› lq-spref-nemp by auto
  note ‹p⁻¹>hd ws · concat ?ws · last ws<⁻¹s = concat us›[unfolded ‹concat ?ws
= ε› cow-simps]
  from pref-mod-list[OF - ‹p⁻¹>hd ws ≠ ε›, of us, folded this, OF triv-pref]
  obtain ps pa sa where ps ≤p us pa ≠ ε pa · sa = last ps  concat ps = p⁻¹>hd
ws · sa.
  have concat us = concat ps · concat (ps⁻¹>us)
    unfolding concat-morph[symmetric] lq-pref[OF ‹ps ≤p us›]..
  have ps ≠ ε
    using ‹concat ps = p⁻¹>hd ws · sa› ‹p⁻¹>hd ws ≠ ε› by fastforce
  have concat ps = concat (butlast ps) · pa · sa
    unfolding ‹pa · sa = last ps›
    using  concat-butlast-last[OF ‹ps ≠ ε›, symmetric].
  show thesis
  proof (rule that)
    show hd ws = p · concat (butlast ps) · pa
      using ‹concat ps = concat (butlast ps) · pa · sa› ‹concat ps = p⁻¹>hd ws ·
sa› ‹p · p⁻¹>hd ws = hd ws› by fastforce
    show last ws = sa · concat (ps⁻¹>us) · s
      using ‹p⁻¹>hd ws · last ws<⁻¹s  = concat us› ‹last ws<⁻¹s · s = last ws›
      unfolding ‹concat us = concat ps · concat (ps⁻¹>us)› ‹concat ps = concat
(butlast ps) · pa · sa› rassoc
        ‹hd ws = p · concat (butlast ps) · pa›  by auto
    show pa ?ws sa ∼𝒟 [last ps]
    proof (rule disj-interpI, unfold ‹concat ?ws = ε› cow-simps)
      have pa · ε · sa = concat [last ps]
        using ‹pa · sa = last ps› by auto
      have [hd ws] ≤p ws
        using dec by blast
```

from ‹$ps \leq_p us \Longrightarrow [hd\ ws] \leq_p ws \Longrightarrow p \cdot concat\ ps \neq concat\ [hd\ ws]$›[*OF* ‹$ps \leq_p us$› *this*]
have $sa \neq \varepsilon$
using ‹$concat\ ps = p^{-1>} hd\ ws \cdot sa$› ‹$hd\ ws = p \cdot concat\ (butlast\ ps) \cdot pa$›
by *force*
show $pa\ \varepsilon\ sa \sim_{\mathcal{I}} [last\ ps]$
proof (*rule fac-interpI*)
show $pa <_p hd\ [last\ ps]$
using ‹$pa \cdot sa = last\ ps$› ‹$sa \neq \varepsilon$› **by** *auto*
show $sa <_s last\ [last\ ps]$
using ‹$pa \cdot sa = last\ ps$› ‹$pa \neq \varepsilon$› **by** *auto*
qed *fact*
have $pa \neq concat\ v$ **if** $v \leq_p [last\ ps]$ **for** v
by (*rule pref-singE*[*OF* ‹$v \leq_p [last\ ps]$›]) (*use* ‹$pa \neq \varepsilon$› ‹$pa \cdot sa = last\ ps$› ‹$sa \neq \varepsilon$› **in** *auto*)
thus $\forall u\ v.\ u \leq_p butlast\ (tl\ ws) \wedge v \leq_p [last\ ps] \longrightarrow pa \cdot concat\ u \neq concat\ v$
using ‹$concat\ (butlast\ (tl\ ws)) = \varepsilon$› *pref-concat-emp self-append-conv* **by** *metis*
qed
show $butlast\ ps \cdot [last\ ps] \cdot ps^{-1>} us = us$
using ‹$ps \leq_p us$› ‹$ps \neq \varepsilon$› *append-butlast-last-id lassoc lq-pref* **by** *metis*
qed
qed
qed

lemma *disj-interp-long*: **assumes** $p\ us\ s \sim_{\mathcal{D}} ws$
shows $1 < |hd\ ws|\ 1 < |last\ ws|$
using *nemp-spref-len*[*OF disj-interp-nemp*(*1*)[*OF assms*] *fac-interpD*(*1*)[*OF disj-interpD0*[*OF assms*]]]
nemp-ssuf-len[*OF disj-interp-nemp*(*2*)[*OF assms*] *fac-interpD*(*2*)[*OF disj-interpD0*[*OF assms*]]].

end

theory *Border-Array*

imports
CoWBasic

begin

2.26.1 Auxiliary lemmas on suffix and border extension

lemma *border-ConsD*: **assumes** $b\#x \leq_b a\#w$
shows $a = b$ **and**
$x \neq \varepsilon \Longrightarrow x \leq_b w$ **and**
border-ConsD-neq: $x \neq w$ **and**
border-ConsD-pref: $x \leq_p w$ **and**

border-ConsD-suf: $x \leq_s w$
proof−
show $a = b$
using *borderD-pref*[*OF assms*] **by** *force*
show $x \neq w$ **and** $x \leq_p w$ **and** $x \leq_s w$
using *borderD-neq*[*OF assms*, *unfolded* ‹$a = b$›]
borderD-pref[*OF assms*, *unfolded Cons-prefix-Cons*]
suffix-ConsD2[*OF borderD-suf*[*OF assms*]] **by** *force*+
thus $x \neq \varepsilon \Longrightarrow x \leq_b w$
unfolding *border-def* **by** *blast*
qed

lemma *ext-suf-Cons*:
$Suc\ i + |u| = |w| \Longrightarrow u \leq_s w \Longrightarrow (w!i)\#u \leq_s (w!i)\#w$
proof−
assume $Suc\ i + |u| = |w|$ **and** $u \leq_s w$
hence $u = drop\ (Suc\ i)\ w$
unfolding *suffix-def* **using** ‹$Suc\ i + |u| = |w|$› **by** *auto*
have $i < |w|$
using ‹$Suc\ i + |u| = |w|$› **by** *auto*
from *id-take-nth-drop*[*OF this*, *folded* ‹$u = drop\ (Suc\ i)\ w$›]
show $w\ !\ i\ \#\ u \leq_s w\ !\ i\ \#\ w$
using *suffix-ConsI triv-suf* **by** *metis*
qed

lemma *ext-suf-Cons-take-drop*: **assumes** $take\ k\ (drop\ (Suc\ i)\ w) \leq_s drop\ (Suc\ i)\ w$ **and** $w\ !\ i = w\ !\ (|w| - Suc\ k)$
shows $take\ (Suc\ k)\ (drop\ i\ w) \leq_s drop\ i\ w$
proof (*cases* $(Suc\ k) + i < |w|$, *simp-all*)
assume $Suc\ (k + i) < |w|$

hence $i < |w|$
by *simp*

have $Suc\ (|w| - Suc\ i - Suc\ k) = |w| - Suc(i+k)$
using *Suc-diff-Suc* ‹$Suc\ (k + i) < |w|$›
by (*simp add*: *Suc-diff-Suc*)

have $|take\ k\ (drop\ (Suc\ i)\ w)| = k$
using ‹$Suc\ (k + i) < |w|$› **by** *fastforce*

have $Suc\ (|w| - Suc\ i - Suc\ k) + |take\ k\ (drop\ (Suc\ i)\ w)| = |drop\ (Suc\ i)\ w|$
unfolding ‹$|take\ k\ (drop\ (Suc\ i)\ w)| = k$› ‹$Suc\ (|w| - Suc\ i - Suc\ k) = |w| - Suc(i+k)$›
using ‹$Suc\ (k + i) < |w|$› **by** *simp*

hence $|drop\ (Suc\ (|w| - Suc\ i - k))\ (drop\ i\ w)| = k$
using ‹$i < |w|$› **by** *fastforce*

```
  have |w| − Suc i − k < |drop i w|
    by (metis Suc-diff-Suc ‹i < |w|› diff-less-Suc length-drop)

  have (drop i w)!(|w| − Suc i − k) = w ! i
    using ‹Suc (k + i) < |w|› ‹w ! i = w ! (|w| − Suc k)› by auto

  have take (Suc k) (drop i w) = w!i#take k (drop (Suc i) w)
    using Cons-nth-drop-Suc[OF ‹i < |w|›] take-Suc-Cons[of k w!i drop (Suc i) w]
by argo

  have drop (Suc (|w| − Suc i − k)) (drop i w) = drop (|w| − Suc i − k) (drop
(Suc i) w)
    by auto
  hence drop (Suc (|w| − Suc i − k)) (drop i w) = take k (drop (Suc i) w)
    using ‹|take k (drop (Suc i) w)| = k›
     ‹take k (drop (Suc i) w) ≤s drop (Suc i) w› suf-drop-conv length-drop by metis

  with
    id-take-nth-drop[OF ‹|w| − Suc i − k < |drop i w|›]
  show ?thesis
    unfolding ‹(drop i w)!(|w| − Suc i − k) = w ! i›
      ‹take (Suc k) (drop i w) = w!i#take k (drop (Suc i) w)›
    unfolding suffix-def by auto
qed

lemma ext-border-Cons:
  Suc i + |u| = |w| ⟹ u ≤b w ⟹ (w!i)#u ≤b (w!i)#w
  unfolding border-def using ext-suf-Cons Cons-prefix-Cons list.discI list.inject
by metis

lemma border-add-Cons-len: assumes max-borderP u w and v ≤b (a#w) shows
|v| ≤ Suc |u|
proof−
  have v ≠ ε
    using ‹v ≤b (a#w)› by simp
  then obtain v' where v = a#v'
    using  borderD-pref[OF ‹v ≤b (a#w)›, unfolded prefix-Cons] by blast
  show |v| ≤ Suc |u|
  proof (cases v' = ε)
    assume v' ≠ ε
    have w ≠ ε
      using  borderedI[OF ‹v ≤b (a#w)›] sing-not-bordered[of a] by blast
    have v' ≤b w
      using border-ConsD(2)[OF ‹v ≤b (a#w)›[unfolded ‹v = a # v'›] ‹v' ≠ ε›].
    thus |v| ≤ Suc |u|
      unfolding ‹v = a # v'› length-Cons Suc-le-mono
      using ‹max-borderP u w›[unfolded max-borderP-def]
        prefix-length-le by blast
  qed (simp add: ‹v = a#v'›)
```

qed

2.27 Computing the Border Array

The computation is a special case of the Knuth-Morris-Pratt algorithm.

- KMP w arr bord pos
- w: processed word does not change; it is processed starting from the last letter
- pos: actually examined pos-th letter; that is, it is w!(pos-1)
- arr: already calculated suffix-border-array of w; that is, the length of array is (|w| - pos) and arr!(|w| - pos - bord) is the max border length of the suffix of w of length bord
- bord: length of the current max border length candidate to see whether it can be extended we compare: w!(pos-1) ?= w!(|w| - (Suc bord)); (Suc bord) is the length of the max border if the comparison is succesful
- if the comparison fails we move to the max border of the suffix of length bord; its max border length is stored in arr!(|w| - pos - bord)
- if bord was 0 and the comparison failed, the word is unbordered

```
fun   KMP-arr :: 'a list ⇒ nat list ⇒ nat ⇒ nat ⇒ nat list
 where   KMP-arr - arr - 0 = arr |
   KMP-arr w arr bord (Suc i) =
        (if w!i = w!(|w| − (Suc bord))
        then  (Suc bord) # arr
        else (if bord = 0
             then  0#arr
             else (if (arr!(|w| − (Suc i) − bord)) < bord — always True, for sake
of termination
                  then arr
                  else  undefined#arr — else: dummy termination condition
                  )
             )
        )

fun KMP-bord :: 'a list ⇒ nat list ⇒ nat ⇒ nat ⇒ nat
 where    KMP-bord - - bord 0 = bord |
   KMP-bord w arr bord (Suc i) =
        (if w!i = w!(|w| − (Suc bord))
        then Suc bord
        else (if bord = 0
```

```
            then  0
            else (if (arr!(|w| − (Suc i) − bord)) < bord — always True, for sake
of termination
                  then arr!(|w| − (Suc i) − bord)
                  else  0 — dummy termination condition
                 )
            )
      )

fun KMP-pos :: 'a list ⇒ nat list ⇒ nat ⇒ nat ⇒ nat
  where
    KMP-pos - - - 0 = 0 |
    KMP-pos w arr bord (Suc i) =
      (if w!i = w!(|w| − (Suc bord))
      then i
      else (if bord = 0
            then  i
            else (if (arr!(|w| − (Suc i) − bord)) < bord — always True, for sake
of termination
                  then Suc i
                  else  i — else: dummy termination condition
                 )
            )
      )

thm prod-cases
    nat.exhaust
    prod.exhaust
    prod-cases3

function KMP :: 'a list ⇒ nat list ⇒ nat ⇒ nat ⇒ nat list where
  KMP w arr bord 0 = arr  |
  KMP w arr bord (Suc i) = KMP w (KMP-arr w arr bord (Suc i)) (KMP-bord w
arr bord (Suc i)) (KMP-pos w arr bord (Suc i))
  using not0-implies-Suc by (force+)
termination
  by (relation measures [λ(-, - , compar, pos). pos,λ(-, - , compar, pos). compar],
simp-all)

lemma KMP-len: |KMP w arr bord pos| = |arr| + pos
  by (induct w arr bord pos rule: KMP.induct, auto)

value[nbe] KMP [a] [0] 0 0

value KMP [ 0::nat] [0] 0 0
value KMP [5,4,5,3,5,5::nat] [0] 0 5
value KMP [5,4::nat,5,3,5,5] [1,0] 1 4
value KMP [0,1,1,0::nat,0,0,1,1,1] [0] 0 8
value KMP [0::nat,1] [0] 0 1
```

2.27.1 Verification of the computation

definition *KMP-valid* :: *'a list* ⇒ *nat list* ⇒ *nat* ⇒ *nat* ⇒ *bool*
where *KMP-valid w arr bord pos* = (|*arr*| + *pos* = |*w*| ∧
— bord is the length of a border of (drop pos w), or 0
pos + *bord* < |*w*| ∧
take bord (*drop pos w*) ≤*p* (*drop pos w*) ∧
take bord (*drop pos w*) ≤*s* (*drop pos w*) ∧
— ... and no longer border can be extended
(∀ *v*. *v* ≤*b* *w*!(*pos* − *1*)#(*drop pos w*) ⟶ |*v*| ≤ *Suc bord*) ∧
— the array gives maximal border lengths of corresponding suffixes
(∀ *k* < |*arr*|. *max-borderP* (*take* (*arr*!*k*) (*drop* (*pos* + *k*) *w*)) (*drop* (*pos* + *k*) *w*))
)

lemma *KMP-valid w arr bord pos* ⟹ *w* ≠ *ε*
unfolding *KMP-valid-def*
using *le-antisym less-imp-le-nat less-not-refl2 take-Nil take-all-iff* **by** *metis*

lemma *KMP-valid-base*: **assumes** *w* ≠ *ε* **shows** *KMP-valid w* [*0*] *0* (|*w*|−*1*)
proof (*unfold KMP-valid-def*, *intro conjI*)
show |[*0*]| + (|*w*| − *1*) = |*w*|
by (*simp add*: *assms*)
show |*w*| − *1* + *0* < |*w*|
using ‹*w* ≠ *ε*› **by** *simp*
show *take 0* (*drop* (|*w*| − *1*) *w*) ≤*p* *drop* (|*w*| − *1*) *w*
by *simp*
show *take 0* (*drop* (|*w*| − *1*) *w*) ≤*s* *drop* (|*w*| − *1*) *w*
by *simp*
show ∀ *v*. *v* ≤*b* *w* ! (|*w*| − *1* − *1*) # *drop* (|*w*| − *1*) *w* ⟶ |*v*| ≤ *Suc 0*
proof (*rule allI*, *rule impI*)
fix *v* **assume** *b*: *v* ≤*b* *w* ! (|*w*| − *1* − *1*) # *drop* (|*w*| − *1*) *w*
have |*w* ! (|*w*| − *1* − *1*) # *drop* (|*w*| − *1*) *w*| = *Suc* (*Suc 0*)
using ‹|[*0*]| + (|*w*| − *1*) = |*w*|› **by** *auto*
from *border-len*(*3*)[*OF b*, *unfolded this*]
show |*v*| ≤ *Suc 0*
using *border-len*(*3*)[*OF b*] **by** *simp*
qed
have |*w*| − *Suc 0* = |*butlast w*|
by *simp*
have *all*: ∀ *v*. *v* ≤*b* [*last w*] ⟶ *v* ≤*p* *ε*
by (*meson borderedI sing-not-bordered*)
have *butlast w* · [*last w*] = *w*
by (*simp add*: *assms*)
hence *last*: *drop* (|*w*| − *Suc 0*) *w* = [*last w*]
unfolding ‹|*w*| − *Suc 0* = |*butlast w*|› **using** *drop-pref* **by** *metis*
hence *max-borderP ε* (*drop* (|*w*| − *Suc 0*) *w*)
unfolding *max-borderP-def* **using** *all* **by** *simp*

thus $\forall k<|[0]|$. *max-borderP* (*take* ([*0*] ! *k*) (*drop* (|*w*| − *1* + *k*) *w*)) (*drop* (|*w*| − *1* + *k*) *w*)
by *simp*
qed

lemma *KMP-valid-step*: **assumes** *KMP-valid w arr bord* (*Suc i*)
shows *KMP-valid w* (*KMP-arr w arr bord* (*Suc i*)) (*KMP-bord w arr bord* (*Suc i*)) (*KMP-pos w arr bord* (*Suc i*))
proof−
— Consequences of the assumption
have *all-k*: $\forall k<|arr|$. *max-borderP* (*take* (*arr* ! *k*) (*drop* (*Suc i* + *k*) *w*)) (*drop* (*Suc i* + *k*) *w*)
using *assms*[*unfolded KMP-valid-def*] **by** *blast*
have |*arr*| + *Suc i* = |*w*| **and**
Suc i + *bord* < |*w*| **and**
bord-pref: *take bord* (*drop* (*Suc i*) *w*) ≤*p drop* (*Suc i*) *w* **and**
bord-suf: *take bord* (*drop* (*Suc i*) *w*) ≤*s drop* (*Suc i*) *w* **and**
up-bord: $\bigwedge$ *v*. *v* ≤*b w*!*i*#(*drop* (*Suc i*) *w*) ⟹ |*v*| ≤ *Suc bord* **and**
all-k-neq0: $\bigwedge$ *k*. *k* < |*arr*| ⟹ *take* (*arr* ! *k*) (*drop* (*Suc i* + *k*) *w*) = *drop* (*Suc i* + *k*) *w* ⟶ *drop* (*Suc i* + *k*) *w* = *ε* **and**
all-k-pref: $\bigwedge$ *k*. *k* < |*arr*| ⟹ *take* (*arr* ! *k*) (*drop* (*Suc i* + *k*) *w*) ≤*p drop* (*Suc i* + *k*) *w* **and**
all-k-suf: $\bigwedge$ *k*. *k* < |*arr*| ⟹ *take* (*arr* ! *k*) (*drop* (*Suc i* + *k*) *w*) ≤*s drop* (*Suc i* + *k*) *w* **and**
all-k-v: $\bigwedge$ *k v*. *k* < |*arr*| ⟹ *v* ≤*b drop* (*Suc i* + *k*) *w* ⟹ *v* ≤*p take* (*arr* ! *k*) (*drop* (*Suc i* + *k*) *w*)
using *assms*[*unfolded KMP-valid-def max-borderP-def diff-Suc-1*] **by** *blast*+
have *all-k-neq*: $\bigwedge$ *k*. *k* < |*arr*| ⟹ *take* (*arr* ! *k*) (*drop* (*Suc i* + *k*) *w*) ≠ *drop* (*Suc i* + *k*) *w*
using ‹*Suc i* + *bord* < |*w*|› ‹|*arr*| + *Suc i* = |*w*|› *all-k-neq0*
add.commute add-le-imp-le-left drop-eq-Nil le-antisym less-imp-le-nat less-not-refl2
by *metis*

have *w* ≠ *ε*
using ‹|*arr*| + *Suc i* = |*w*|› **by** *auto*
have *Suc i* < |*w*|
using ‹*Suc i* + *bord* < |*w*|› **by** *simp*
have *pop-i*: *drop i w* = (*w*!*i*)# (*drop* (*Suc i*) *w*)
by (*simp add*: *Cons-nth-drop-Suc Suc-lessD* ‹*Suc i* < |*w*|›)
have *drop* (*Suc i*) *w* ≠ *ε*
using ‹*Suc i* < |*w*|› **by** *fastforce*
have *Suc i* + (|*w*| − *Suc i* − *bord*) = |*w*| − *bord*
unfolding *diff-right-commute*[*of* - - *bord*] **using** ‹*Suc i* + *bord* < |*w*|› **by** *linarith*

show *KMP-valid w* (*KMP-arr w arr bord* (*Suc i*)) (*KMP-bord w arr bord* (*Suc i*)) (*KMP-pos w arr bord* (*Suc i*))
proof (*cases w* ! *i* = *w* ! (|*w*| − *Suc bord*))
assume *match*: *w* ! *i* = *w* ! (|*w*| − *Suc bord*) — The current candidate is

extendable
show *?thesis*
proof (*unfold KMP-valid-def KMP-arr.simps KMP-bord.simps KMP-pos.simps if-P*[*OF match*], *intro conjI*)
show |*Suc bord # arr*| + *i* = |*w*|
using ‹|*arr*| + *Suc i* = |*w*|› **by** *auto*
show *i* + *Suc bord* < |*w*|
using ‹*Suc i* + *bord* < |*w*|› **by** *auto*
show *take* (*Suc bord*) (*drop i w*) ≤*p drop i w*
using *take-is-prefix* **by** *auto*
show *take* (*Suc bord*) (*drop i w*) ≤*s drop i w*
using ‹*take bord* (*drop* (*Suc i*) *w*) ≤*s drop* (*Suc i*) *w*› *ext-suf-Cons-take-drop match* **by** *blast*
— The new border array is correct
show *all-k-new*: ∀ *k*<|*Suc bord # arr*|. *max-borderP* (*take* ((*Suc bord # arr*) ! *k*) (*drop* (*i* + *k*) *w*)) (*drop* (*i* + *k*) *w*)
proof (*rule allI*, *rule impI*)
fix *k* **assume** *k* < |*Suc bord # arr*|
show *max-borderP* (*take* ((*Suc bord # arr*) ! *k*) (*drop* (*i* + *k*) *w*)) (*drop* (*i* + *k*) *w*)
proof (*cases* 0 < *k*)
assume 0 < *k* — old entries are valid:
thus *?thesis* **using** *all-k*
by (*metis Suc-less-eq* ‹*k* < |*Suc bord # arr*|› *add.right-neutral add-Suc-shift gr0-implies-Suc list.size*(4) *nth-Cons-Suc*)
next
assume ¬ 0 < *k* **hence** *k* = 0 **by** *simp*
show *?thesis* — the extended border is maximal:
unfolding *max-borderP-def* ‹*k* = 0›
proof (*intro conjI*)
show *take* ((*Suc bord # arr*) ! 0) (*drop* (*i* + 0) *w*) = *drop* (*i* + 0) *w* ⟶ *drop* (*i* + 0) *w* = *ε*
using ‹*i* + *Suc bord* < |*w*|› **by** *fastforce*
show *take* ((*Suc bord # arr*) ! 0) (*drop* (*i* + 0) *w*) ≤*p drop* (*i* + 0) *w*
using ‹*take* (*Suc bord*) (*drop i w*) ≤*p drop i w*› **by** *auto*
show *take* ((*Suc bord # arr*) ! 0) (*drop* (*i* + 0) *w*) ≤*s drop* (*i* + 0) *w*
by *simp fact*
show ∀ *v*. *v* ≤*b drop* (*i* + 0) *w* ⟶ *v* ≤*p take* ((*Suc bord # arr*) ! 0) (*drop* (*i* + 0) *w*)
proof (*rule allI*, *rule impI*)
fix *v* **assume** *v* ≤*b drop* (*i* + 0) *w* **hence** *v* ≤*b drop i w* **by** *simp*
from *borderD-pref*[*OF this*] *up-bord*[*OF this*[*unfolded pop-i*]]
show *v* ≤*p take* ((*Suc bord # arr*) ! 0) (*drop* (*i* + 0) *w*)
unfolding *prefix-def* **by** *force*
qed
qed
qed
qed
— the extended border is the longest candidate:

have *max-borderP* (*take* (*Suc bord*) (*drop i w*)) (*drop i w*)
using *all-k-new*[*rule-format*, *of 0*, *unfolded length-Cons nth-Cons-0 add-0-right*, *OF zero-less-Suc*].
from *border-add-Cons-len*[*OF this*] *max-borderP-D-max*[*OF this*] *max-borderP-D-neq*[*OF - this*]
show ∀ *v*. *v* ≤*b* *w* ! (*i* − *1*) # *drop i w* ⟶ |*v*| ≤ *Suc* (*Suc bord*)
using *nat-le-linear take-all take-len list.discI pop-i* **by** *metis*
qed
next
assume *mismatch*: *w* ! *i* ≠ *w* ! (|*w*| − *Suc bord*) — The current candidate is not extendable
show *?thesis*
proof (*cases bord* = *0*)
assume *bord* ≠ *0* — Recursion: try the maximal border of the current candidate...
let *?k* = |*w*| − *Suc i* − *bord* **and**
?w′ = *drop* (*Suc i*) *w*
have *?k* < |*arr*|
using ‹*Suc i* + *bord* < |*w*|› ‹|*arr*| + *Suc i* = |*w*|› ‹*bord* ≠ *0*› **by** *linarith*
from *all-k-neq*[*OF this*]
have *arr* ! *?k* < *bord* — ... which is stored in the array, and is shorter
by (*simp add*: ‹*take* (*arr* ! *?k*) (*drop* (*Suc i* + *?k*) *w*) ≠ *drop* (*Suc i* + *?k*) *w*› ‹*Suc i* + *?k* = |*w*| − *bord*› ‹*Suc i* + *bord* < |*w*|› *add-diff-inverse-nat diff-add-inverse2 gr0I less-diff-conv nat-diff-split-asm*)
let *?old-pref* = *take bord ?w′* **and**
?old-suf = *drop* (|*w*| − *bord*) *w* **and**
?new-pref = *take* (*arr* ! *?k*) *?w′*
show *?thesis*
proof (*unfold KMP-valid-def KMP-arr.simps KMP-bord.simps KMP-pos.simps if-not-P*[*OF mismatch*] *if-not-P*[*OF* ‹*bord* ≠ *0*›] *if-P*[*OF* ‹*arr* ! *?k* < *bord*›] *diff-Suc-1*, *intro conjI*)
show |*arr*| + *Suc i* = |*w*|
using ‹|*arr*| + *Suc i* = |*w*|› **by** *auto*
show *Suc i* + *arr* ! *?k* < |*w*|
using ‹*Suc i* + *bord* < |*w*|› ‹*arr* ! *?k* < *bord*› **by** *linarith*
show *take* (*arr* ! *?k*) (*drop* (*Suc i*) *w*) ≤*p* *drop* (*Suc i*) *w*
using *take-is-prefix* **by** *blast*

— Next goal: the new border is a suffix

have *?old-suf* ≤*s* *?w′*
by (*meson* ‹*Suc i* + *bord* < |*w*|› *le-suf-drop less-diff-conv nat-less-le*)
have |*?old-pref*| = *bord*
using ‹*Suc i* + *bord* < |*w*|› *take-len len-after-drop nat-less-le* **by** *blast*
also have ... = |*?old-suf*|
using ‹*Suc i* + *bord* < |*w*|› **by** *simp*
ultimately have *eq1*: *?old-pref* = *?old-suf* — bord defines a border
using ‹*take bord* (*drop* (*Suc i*) *w*) ≤*s* *drop* (*Suc i*) *w*› ‹*drop* (|*w*| − *bord*) *w* ≤*s* *drop* (*Suc i*) *w*› *suf-ruler-eq-len* **by** *metis*

have $|?new\text{-}pref| = arr!?k$
using *take-len* ‹*Suc i + bord < |w|*› ‹*arr ! ?k < bord*› *diff-diff-left* **by** *force*
have *take* (*arr ! ?k*) *?old-suf* $\leq_p$ *?old-pref*
using *take-is-prefix* ‹*?old-pref = ?old-suf*› **by** *metis*
from *pref-take*[*OF pref-trans*[*OF this take-is-prefix*], *unfolded* ‹$|?new\text{-}pref| = arr!?k$›, *symmetric*]
have *take* (*arr ! ?k*) *?old-suf = take* (*arr ! ?k*) *?w′*
using *take-len* ‹*arr ! ?k < bord*› ‹*bord* = $|drop\ (|w| - bord)\ w|$› *less-imp-le-nat* **by** *metis*
from *all-k-suf*[*OF* ‹*?k* < $|arr|$›, *unfolded* ‹*Suc i + ?k* = $|w|$ − *bord*›] *this*
have *take* (*arr ! ?k*) *?w′* $\leq_s$ *?old-suf* **by** *simp* — The new prefix is a suffix of the old suffix

with ‹*?old-pref* $\leq_s$ *?w′*›[*unfolded* ‹*?old-pref = ?old-suf*›]
show *take* (*arr ! ?k*) *?w′* $\leq_s$ *?w′*
using *suf-trans* **by** *blast*

— Key facts about borders of the w'
have *?old-pref* $\neq \varepsilon$
using ‹*bord* $\neq$ *0*› ‹$|?old\text{-}pref| = bord$› **by** *force*
moreover have *?old-pref* $\neq$ *?w′*
using ‹*Suc i + bord < |w|*›
by (*intro lenarg-not*, *unfold length-drop* ‹$|take\ bord\ ?w'| = bord$›, *linarith*)
ultimately have *?old-pref* $\leq_b$ *?w′* — bord is the length of a border
by (*intro borderI*[*OF bord-pref bord-suf*])

— We want to prove that the new border is the longest candidate
show $\forall v.\ v \leq_b w\ !i\ \#\ ?w' \longrightarrow |v| \leq Suc\ (arr\ !\ ?k)$
proof (*rule allI*,*rule impI*)
have *extendable*: $w\ !\ i\ \#\ v' \leq_b w\ !\ i\ \#\ ?w' \Longrightarrow v' \neq \varepsilon \Longrightarrow |v'| \leq arr\ !\ ?k$ **for** *v′* — First consider a border of w', which is extendable
proof–
assume $w!i\ \#\ v' \leq_b w!i\ \#\ ?w'$ **and** $v' \neq \varepsilon$
from *suf-trans*[*OF borderD-suf*[*OF* ‹$w!i\ \#\ v' \leq_b w\ !\ i\ \#\ ?w'$›, *folded pop-i*] *suffix-drop*]
have $w!i\ \#\ v' \leq_s w$.
from *this*[*unfolded suf-drop-conv*, *THEN nth-via-drop*] *mismatch*
have $|w!i\ \#\ v'| \neq Suc\ bord$
by *force*
with *up-bord*[*OF* ‹$w!i\ \#\ v' \leq_b w\ !\ i\ \#\ ?w'$›]
have $|v'| < bord$ — It is shorter than the old candidate border
by *simp*
from *border-ConsD*(*2*)[*OF* ‹$w!i\ \#\ v' \leq_b w\ !\ i\ \#\ ?w'$› ‹$v' \neq \varepsilon$›]
have $v' \leq_b$ *?w′*.
from *borders-compare*[*OF* ‹*?old-pref* $\leq_b$ *?w′*› *this*, *unfolded* ‹$|?old\text{-}pref| = bord$›, *unfolded* ‹*?old-pref = ?old-suf*›, *OF* ‹$|v'| < bord$›]
have $v' \leq_b$ *?old-suf*. — ... and therefore its border

```
            from prefix-length-le[OF max-borderP-D-max[OF all-k[rule-format,
OF ‹?k < |arr|›], unfolded ‹Suc i + ?k = |w| − bord›, OF this]]
              show |v'| ≤ arr!?k — ... and hence short
                using len-take1[of arr!?k, of w] by simp
            qed
          fix v assume v ≤b w!i # ?w' — Now consider a border of the extended
word
           show |v| ≤ Suc (arr ! ?k)
           proof (cases |v| ≤ Suc 0)
             assume ¬ |v| ≤ Suc 0
             hence Suc 0 < |v| 0 < |v| by auto
             from hd-tl-lenE[OF ‹0 < |v|›]
             obtain a v' where v = a#v'
               by blast
             have v' ≠ ε
                using ‹Suc 0 < |v|›[unfolded ‹v = a#v'›] by simp
             with borderD-pref[OF ‹v ≤b w!i # ?w'›, unfolded prefix-Cons]
             have v = w!i#v'
               using ‹v = a#v'› by simp
            from extendable[OF ‹v ≤b w!i # ?w'›[unfolded ‹v = w!i#v'›] ‹v' ≠ ε›]
             show ?thesis
               by (simp add: ‹v = a # v'›)
         qed simp
       qed
        show ∀k<|arr|. max-borderP (take (arr ! k) (drop (Suc i + k) w)) (drop
(Suc i + k) w)
          using all-k by blast
      qed
    next
      assume bord = 0 — End of recursion.
      show ?thesis
     proof (unfold KMP-valid-def KMP-arr.simps KMP-bord.simps KMP-pos.simps
if-not-P[OF mismatch] if-P[OF ‹bord = 0›], intro conjI)
       show |0 # arr| + i = |w|
         using ‹|arr| + Suc i = |w|› by auto
       show i + 0 < |w|
         by (simp add: Suc-lessD ‹Suc i < |w|›)
       show take 0 (drop i w) ≤p drop i w
         by simp
       show take 0 (drop i w) ≤s drop i w
         using ext-suf-Cons-take-drop by simp
           — The extension is unbordered
       have max-borderP ε (drop i w)
       proof(rule ccontr)
         assume ¬ max-borderP ε (drop i w)
         then obtain a t where max-borderP (a#t) (drop i w)
              unfolding pop-i  using max-border-ex[of w ! i # drop (Suc i) w]
neq-Nil-conv by metis
       from up-bord[OF max-borderP-border[OF this list.simps(3), unfolded pop-i],
```

```
unfolded ‹bord = 0›]
        have t = ε by simp
          from max-borderP-border[OF ‹max-borderP (a#t) (drop i w)›[unfolded
this] list.simps(3)]
        have [a] ≤b drop i w.
        from borderD-pref[OF this]
        have w!i = a
          by (simp add: pop-i)
        moreover have w!(|w| − 1) = a
       using borderD-suf[OF ‹[a] ≤b drop i w›] nth-via-drop sing-len suf-drop-conv
suf-share-take suffix-drop suffix-length-le by metis
        ultimately show False
          using mismatch[unfolded ‹bord = 0›] by simp
      qed
      thus ∀ v. v ≤b w ! (i − 1) # drop i w ⟶ |v| ≤ Suc 0
        by (metis border-add-Cons-len list.size(3))
          — The array is valid: old values from assumption, the first 0 since the
extension is unbordered
      show ∀ k<|0 # arr|. max-borderP (take ((0 # arr) ! k) (drop (i + k) w))
(drop (i + k) w)
      proof (rule allI, rule impI)
        fix k assume k < |0 # arr|
        show max-borderP (take ((0 # arr) ! k) (drop (i + k) w)) (drop (i + k)
w)
        proof (cases 0 < k)
          assume 0 < k
          thus ?thesis using all-k
            by (metis Suc-less-eq ‹k < |0 # arr|› add.right-neutral add-Suc-shift
gr0-implies-Suc list.size(4) nth-Cons-Suc)
        next
          assume ¬ 0 < k hence k = 0 by simp
          thus ?thesis
            using ‹max-borderP ε (drop i w)› by auto
        qed
      qed
    qed
  qed
qed

lemma KMP-valid-max: assumes KMP-valid w arr bord pos k < |w|
  shows max-borderP (take ((KMP w arr bord pos)!k) (drop k w)) (drop k w)
  using assms
proof (induct w arr bord pos arbitrary: k rule: KMP.induct)
  case (2 w arr bord i)
  then show ?case
    unfolding KMP.simps using KMP-valid-step by blast
qed (simp add: KMP-valid-def)
```

2.28 Border array

fun *border-array* :: *'a list* ⇒ *nat list* **where**
  border-array ε = ε
| *border-array (a#w) = rev (KMP (rev (a#w)) [0] 0 (|a#w|−1))*

lemma *border-array-len*: *|border-array w| = |w|*
  by (*induct w*, *simp-all add*: *KMP-len*)

theorem *bord-array*: **assumes** *Suc k ≤ |w|* **shows** *(border-array w)!k = |max-border (take (Suc k) w)|*
proof−
  define *m* **where** *m = |w| − Suc k*
  hence *m < |rev w|*
    by (*simp add*: *Suc-diff-Suc assms less-eq-Suc-le*)
  have *rev w ≠ ε* **and** *k < |rev w|*
    using ‹*Suc k ≤ |w|*› **by** *auto*
  hence *w = hd w#tl w*
    by *simp*
 from *arg-cong*[*OF border-array.simps*(*2*)[*of hd w tl w*, *folded this*], *of rev*, *unfolded rev-rev-ident*]
  have *rev (border-array w) = (KMP (rev w) [0] 0 (|w|−1))*.
  hence *max-borderP (take (rev (border-array w)!m) (drop m (rev w))) (drop m (rev w))*
    using *KMP-valid-max*[*rule-format*, *OF KMP-valid-base*[*OF* ‹*rev w ≠ ε*›] ‹*m < |rev w|*›] **by** *simp*
  hence *max-border (drop m (rev w)) = take (rev (border-array w)!m) (drop m (rev w))*
    using *max-borderP-max-border* **by** *blast*
  hence *|max-border (drop m (rev w))| = rev (border-array w)!m*
    by (*metis* ‹*m < |rev w|*› *drop-eq-Nil leD max-border-nemp-neq nat-le-linear take-all take-len*)
  thus *?thesis*
    using *m-def*
  by (*metis Suc-diff-Suc* ‹*k < |rev w|*› ‹*m < |rev w|*› *border-array-len diff-diff-cancel drop-rev length-rev less-imp-le-nat max-border-len-rev rev-nth*)
qed

lemma *max-border-comp* [*code*]: *max-border w = take ((border-array w)!(|w|−1)) w*
proof (*cases w = ε*)
  assume *w = ε*
  thus *max-border w = take ((border-array w)!(|w|−1)) w*
    using *max-bord-take take-Nil* **by** *metis*
next
  assume *w ≠ ε*
  hence *Suc (|w| − 1) ≤ |w|* **by** *simp*
  from *bord-array*[*OF this*]
  have *(border-array w)!(|w|−1) = |max-border w|*

by (*simp add*: ‹$w \neq \varepsilon$›)
thus *max-border* w = *take* ((*border-array* w)!(|w|−*1*)) w
using *max-bord-take* **by** *auto*
qed

value[*nbe*] *primitive* [*a*,*b*,*a*]

value *primitive* [*0*,*1*,*0*::*nat*]

value *border-array* [*5*,*4*,*5*,*3*,*5*,*5*,*5*,*4*,*5*::*nat*]

value *primitive* [*5*,*4*,*5*,*3*,*5*,*5*,*5*,*4*,*5*::*nat*]

value *primitive* [*5*,*4*,*5*,*3*,*5*,*5*,*5*,*4*,*5*::*nat*]

value[*nbe*] *bordered* []

value *border-array* [*0*,*1*,*1*,*0*,*0*,*0*,*1*,*1*,*1*,*1*,*1*,*0*,*0*,*0*,*1*,*1*,*1*,*0*,*1*,*1*,*0*,*0*,*0*,*1*,*1*,*1*,*1*,*1*,*1*,*0*,*0*,*0*,*1*,*1*,*1*,*0*,*1*,*1*,*0*,*0*,*0*,*1*,*1*,*1*,*0*,*1*,*1*,*0*,

value[*nbe*] *border-array* ε

value *border-array* [*1*,*0*,*1*,*0*,*1*,*1*,*0*,*0*::*nat*]

value *max-border* [*1*,*0*,*1*,*0*,*1*,*1*,*0*,*0*,*1*,*0*,*1*,*1*,*0*,*1*,*0*,*1*,*1*,*0*,*0*,*1*,*0*,*1*,*1*,*0*,*1*,*0*,*1*,*1*,*0*,*1*,*0*,*1*,*1*,*0*,*0*,*1*,*0*,*1*,*0*,*0*,*1*::*nat*]

thm *max-border-comp* — code for *max-border*, based on *border-array*

value *bordered* [*1*,*0*::*nat*,*1*,*0*,*1*,*1*,*0*,*1*]

value π [*1*::*nat*,*0*,*1*,*0*,*1*,*1*,*0*,*1*]

thm *min-per-root-take* — code for π, based on *max-border*

value |π [*1*::*nat*,*0*,*1*,*0*,*1*,*1*,*0*,*1*]|

value ϱ [*1*::*nat*,*0*,*1*,*1*,*0*,*1*,*1*,*0*,*1*]

thm *primroot-code* — code for ϱ, based on π

value ϱ [*1*::*nat*,*0*,*1*,*1*,*0*,*1*,*1*,*0*]

value[*nbe*] π ε

end

theory *Submonoids*
  imports *CoWBasic HOL.Hull*
begin

Chapter 3

Submonoids of a free monoid

This chapter deals with properties of submonoids of a free monoid, that is, with monoids of words. See more in Chapter 1 of [4].

3.1 Generated monoid (also called hull)

First, we define the hull of a set of words, that is, the monoid generated by them.

definition *word-monoid* **where**
word-monoid M ≡ (∀ *w1 w2*. *w1* ∈ *M* ⟶ *w2* ∈ *M* ⟶ *w1* · *w2* ∈ *M*) ∧ *ε* ∈ *M*

lemma *word-monoidI*[*intro*]: **assumes** ⋀ *w1 w2*. *w1* ∈ *M* ⟹ *w2* ∈ *M* ⟹ *w1* · *w2* ∈ *M ε* ∈ *M*
shows *word-monoid M*
by (*simp add*: *assms word-monoid-def*)

inductive-set *hull* :: *'a list set* ⇒ *'a list set* (⟨-⟩)
for *G* **where**
emp-in[*simp*]: *ε* ∈ ⟨*G*⟩ |
prod-cl: *w1* ∈ *G* ⟹ *w2* ∈ ⟨*G*⟩ ⟹ *w1* · *w2* ∈ ⟨*G*⟩

lemmas [*intro*] = *hull.intros*

lemma *hull-closed*[*intro*]: *w1* ∈ ⟨*G*⟩ ⟹ *w2* ∈ ⟨*G*⟩ ⟹ *w1* · *w2* ∈ ⟨*G*⟩
by (*rule hull.induct*[*of w1 G* λ *x*. (*x*·*w2*)∈⟨*G*⟩]) *auto*+

lemma *gen-in* [*intro*]: *w* ∈ *G* ⟹ *w* ∈ ⟨*G*⟩
using *hull.prod-cl* **by** *force*

lemma *gen-monoid-monoid*: *word-monoid* ⟨*G*⟩
unfolding *word-monoid-def* **using** *hull-closed emp-in* **by** *blast*

lemma *gen-monoid-hull*: *word-monoid hull G* = ⟨*G*⟩

proof (*rule hull-unique*, *blast*, *use gen-monoid-monoid* **in** *blast*,*rule subsetI*)
fix M x
assume $G \subseteq M$ **and** *word-monoid* M $x \in \langle G \rangle$
show $x \in M$
by (*induction rule*: *hull.induct*[*OF* ‹$x \in \langle G \rangle$›])
(*use* ‹*word-monoid* M›[*unfolded word-monoid-def*] ‹$G \subseteq M$› **in** *blast*)+
qed

lemma *gen-monoid-induct*: **assumes** $x \in \langle G \rangle$ $P\ \varepsilon$ $\bigwedge w.\ w \in G \Longrightarrow P\ w$
$\bigwedge w1\ w2.\ w1 \in \langle G \rangle \Longrightarrow P\ w1 \Longrightarrow w2 \in \langle G \rangle \Longrightarrow P\ w2 \Longrightarrow P\ (w1 \cdot w2)$ **shows** $P\ x$
using *hull.induct*[*of* - - P, *OF* ‹$x \in \langle G \rangle$› ‹$P\ \varepsilon$›]
assms **by** (*simp add*: *gen-in*)

lemma *genset-sub*[*simp*]: $G \subseteq \langle G \rangle$
using *gen-in* **..**

lemma *lists-sub*: $ws \in lists\ G \Longrightarrow ws \in lists\ \langle G \rangle$
using *sub-lists-mono*[*OF genset-sub*]**.**

lemma *in-lists-conv-set-subset*: $set\ ws \subseteq G \longleftrightarrow ws \in lists\ G$
unfolding *in-lists-conv-set subset-code*(*1*)**..**

lemma *concat-in-hull* [*intro*]:
assumes $set\ ws \subseteq G$
shows $concat\ ws \in \langle G \rangle$
using *assms* **by** (*induction ws*) *auto*

lemma *concat-in-hull′* [*intro*]:
assumes $ws \in lists\ G$
shows $concat\ ws \in \langle G \rangle$
using *assms* **by** (*induction ws*) *auto*

lemma *hull-concat-lists0*:
$w \in \langle G \rangle \Longrightarrow \exists\ ws \in lists\ G.\ concat\ ws = w$ (**is** $w \in \langle G \rangle \Longrightarrow ?P\ w$)
proof(*rule hull.induct*[*of* - G])
show $\exists\, ws{\in}lists\ G.\ concat\ ws = w1 \cdot w2$ **if** $w1 \in G$ $w2 \in \langle G \rangle$
$\exists\, ws{\in}lists\ G.\ concat\ ws = w2$ **for** $w1\ w2$
using *that Cons-in-lists-iff concat.simps*(*2*) **by** *metis*
qed (*use concat-nemp* **in** *blast*)+

lemma *hull-concat-listsE*[*elim*]: **assumes** $w \in \langle G \rangle$
obtains ws **where** $ws \in lists\ G$ **and** $concat\ ws = w$
using *assms hull-concat-lists0* **by** *blast*

lemma *hull-concat-lists*: $\langle G \rangle = concat\ `\ lists\ G$
using *hull-concat-lists0* **by** *blast*

lemma *hull-concat-lists′*: $\langle G \rangle = \{concat\ xs \mid xs.\ xs \in lists\ G\}$

using *Setcompr-eq-image hull-concat-lists* **by** *blast*

lemma *hull-mono*: $A \subseteq B \Longrightarrow \langle A\rangle \subseteq \langle B\rangle$
using *gen-monoid-hull hull-mono* **by** *blast*

lemma *hull-concat-lists-nempE*[*elim*]: **assumes** $w \in \langle G\rangle$
obtains *ws* **where** $ws \in lists\ (G - \{\varepsilon\})$ **and** $concat\ ws = w$
proof−
obtain *ws* **where** $ws \in lists\ G\ concat\ ws = w$
using *assms* **by** *blast*
with *that*[*of filter* $(\lambda\ x.\ x \neq \varepsilon)$ *ws*]
show *thesis*
by *force*
qed

lemma *hull-nemp-eq-hull*[*simp*]: $\langle G - \{\varepsilon\}\rangle = \langle G\rangle$
by (*rule equalityI*, *use hull-mono* **in** *fast*, *rule subsetI*, *blast*)

lemma *emp-gen-set*: $\langle\{\}\rangle = \{\varepsilon\}$
unfolding *hull-concat-lists* **by** *auto*

lemma *concat-lists-minus*[*simp*]: $concat\ `\ lists\ (G - \{\varepsilon\}) = concat\ `\ lists\ G$
proof
show $concat\ `\ lists\ G \subseteq concat\ `\ lists\ (G - \{\varepsilon\})$
proof
fix *x* **assume** $x \in concat\ `\ lists\ G$
from *imageE*[*OF this*]
obtain *y* **where** $x = concat\ y\ y \in lists\ G$**.**
from *lists-minus'*[*OF* ‹$y \in lists\ G$›] *del-emp-concat*[*of y*, *folded* ‹$x = concat\ y$›]
show $x \in concat\ `\ lists\ (G - \{\varepsilon\})$
by *blast*
qed
qed (*simp add*: *image-mono lists-mono*)

lemma *hull-drop-one*: $\langle G - \{\varepsilon\}\rangle = \langle G\rangle$
proof (*intro equalityI subsetI*)
fix *x* **assume** $x \in \langle G\rangle$ **thus** $x \in \langle G - \{\varepsilon\}\rangle$
unfolding *hull-concat-lists* **by** *simp*
next
fix *x* **assume** $x \in \langle G - \{\varepsilon\}\rangle$ **thus** $x \in \langle G\rangle$
unfolding *hull-concat-lists image-iff* **by** *auto*
qed

lemma *sing-gen-power*: $u \in \langle\{x\}\rangle \Longrightarrow \exists\, k.\ u = x^{@}k$
unfolding *hull-concat-lists* **using** *one-generated-list-power* **by** *auto*

lemma *pow-sing-gen*[*simp*]: $x^{@}k \in \langle\{x\}\rangle$
using *concat-in-hull*[*OF sing-pow-set-sub*, *of* - k x] *concat-pow-list-single* **by** *simp*

lemma *sing-gen-pow-ex-conv*: $u \in \langle\{x\}\rangle \longleftrightarrow (\exists\, k.\ u = x^{@}k)$
using *sing-gen-power pow-sing-gen* **by** *blast*

lemma *sing-gen-primroot-gen*: **assumes** $w \in \langle\{x\}\rangle$
shows $w \in \langle\{\varrho\ x\}\rangle$
using *sing-gen-power*[*OF assms*] *pow-sing-gen*
by (*subst* (*asm*) (*1*) *primroot-exp-eq*[*of x*, *symmetric*],
unfold pow-mult[*symmetric*]) *blast*

lemma *sing-gen-primroot-eq*:
assumes $w \in \langle\{x\}\rangle$ $w \neq \varepsilon$
shows $\varrho\ w = \varrho\ x$
proof−
have $\varrho\ x \neq \varepsilon$
using *assms concat.simps*(*1*) **by** *fastforce*
from *sing-gen-primroot-gen*[*OF* ‹$w \in \langle\{x\}\rangle$›]
show $\varrho\ w = \varrho\ x$
using *primroot-unique*[*OF* ‹$w \neq \varepsilon$› *primroot-prim*[*OF* ‹$\varrho\ x \neq \varepsilon$›, *unfolded primroot-idemp*]] *sing-gen-power*[*OF* ‹$w \in \langle\{\varrho\ x\}\rangle$›] **by** *blast*
qed

lemma *sing-gen-pow-conv*: $w \in \langle\{\varrho\ x\}\rangle \longleftrightarrow w = \varrho\ x^{@}e_{\varrho}\ w$
proof (*rule iffI*, *cases* $w = \varepsilon$, *force*)
assume $w \in \langle\{\varrho\ x\}\rangle$ $w \neq \varepsilon$
have $\varrho\ w = \varrho\ x$
using *sing-gen-primroot-eq*[*OF* ‹$w \in \langle\{\varrho\ x\}\rangle$› ‹$w \neq \varepsilon$›, *unfolded primroot-idemp*].
show $w = \varrho\ x\ ^{@}\ e_{\varrho}\ w$
using *primroot-exp-eq*[*of w*, *unfolded* ‹$\varrho\ w = \varrho\ x$›, *symmetric*].
qed (*metis pow-sing-gen*)

lemma *two-primroots-comm*: **assumes** $w \neq \varepsilon$ **and** $w \in \langle\{\varrho\ x\}\rangle$ **and** $w \in \langle\{\varrho\ y\}\rangle$
shows $x \cdot y = y \cdot x$
proof (*rule comm-primrootI*)
from *assms*[*unfolded sing-gen-pow-conv*]
show $\varrho\ x \cdot \varrho\ y = \varrho\ y \cdot \varrho\ x$
using *comm-drop-exps primroot-exp-nemp* **by** *metis*
qed

lemma *comm-rootI*: $x \in \langle\{r\}\rangle \Longrightarrow y \in \langle\{r\}\rangle \Longrightarrow x \cdot y = y \cdot x$
unfolding *comm sing-gen-pow-ex-conv* **by** *blast*

lemma *sing-genE*[*elim*]:
assumes $u \in \langle\{x\}\rangle$
obtains k **where** $x^{@}k = u$
using *assms* **using** *sing-gen-power* **by** *blast*

lemma *lists-gen-to-hull*: $us \in lists\ (G - \{\varepsilon\}) \Longrightarrow us \in lists\ (\langle G\rangle - \{\varepsilon\})$
using *lists-mono genset-sub* **by** *blast*

lemma *rev-hull*: $rev`\langle G\rangle = \langle rev`G\rangle$
proof
show $rev\ `\ \langle G\rangle \subseteq \langle rev\ `\ G\rangle$
proof
fix x **assume** $x \in rev\ `\ \langle G\rangle$
then obtain xs **where** $x = rev\ (concat\ xs)$ **and** $xs \in lists\ G$
unfolding *hull-concat-lists* **by** *auto*
from *rev-in-lists*[*OF* ‹$xs \in lists\ G$›]
have $(map\ rev\ (rev\ xs)) \in lists\ (rev\ `\ G)$
by *fastforce*
thus $x \in \langle rev\ `\ G\rangle$
unfolding *image-iff hull-concat-lists*
using ‹$x = rev\ (concat\ xs)$›[*unfolded rev-concat*] **by** *blast*
qed
show $\langle rev\ `\ G\rangle \subseteq rev\ `\ \langle G\rangle$
proof
fix x **assume** $x \in \langle rev\ `\ G\rangle$
then obtain xs **where** $x = concat\ xs$ **and** $xs \in lists\ (rev\ `\ G)$
unfolding *hull-concat-lists* **by** *blast*
from *rev-in-lists*[*OF* ‹$xs \in lists\ ((rev\ `\ G))$›]
have $map\ rev\ (rev\ xs) \in lists\ G$
by *fastforce*
hence $rev\ x \in \langle G\rangle$
unfolding ‹$x = concat\ xs$› *rev-concat*
by *fast*
thus $x \in rev\ `\ \langle G\rangle$
unfolding *rev-in-conv*.
qed
qed

lemma *power-in*[*intro*]: $x \in \langle G\rangle \Longrightarrow x^{@}k \in \langle G\rangle$
by (*induction k*, *auto*)

lemma *hull-closed-lists*: $us \in lists\ \langle G\rangle \Longrightarrow concat\ us \in \langle G\rangle$
by (*induct us*, *auto*)

lemma *hull-I* [*intro*]:
$\varepsilon \in H \Longrightarrow (\bigwedge x\ y.\ x \in H \Longrightarrow y \in H \Longrightarrow x \cdot y \in H) \Longrightarrow \langle H\rangle = H$
by (*standard*, *use hull.induct*[*of - H* $\lambda\ x.\ x \in H$] **in** *force*) (*simp only*: *genset-sub*)

lemma *gen-idemp*: $\langle\langle G\rangle\rangle = \langle G\rangle$
using *image-subsetI*[*of lists* $\langle G\rangle$ *concat* $\langle G\rangle$, *unfolded hull-concat-lists*[*of* $\langle G\rangle$, *symmetric*],
THEN subset-antisym[*OF - genset-sub*[*of* $\langle G\rangle$]]] *hull-closed-lists*[*of - G*] **by**

blast

lemma *hull-mono'[intro]*: $A \subseteq \langle B\rangle \Longrightarrow \langle A\rangle \subseteq \langle B\rangle$
using *hull-mono[of A ⟨B⟩]* **unfolding** *gen-idemp*.

lemma *hull-conjug [elim]*: $w \in \langle\{r{\cdot}s,s{\cdot}r\}\rangle \Longrightarrow w \in \langle\{r,s\}\rangle$
using *hull-mono[of {r·s,s·r} ⟨{r,s}⟩, unfolded gen-idemp]* **by** *blast*

lemma *root-divisor*: **assumes** *period w k* **and** *k dvd |w|* **shows** $w \in \langle\{take\ k\ w\}\rangle$
using *pow-sing-gen[of (|w| div k) take k w]* **unfolding**
take-several-pers[OF ‹period w k›, of |w| div k, unfolded dvd-div-mult-self[OF ‹k dvd |w|›] take-self, OF , OF order-refl].

Intersection of hulls is a hull.

lemma *hulls-inter*: $\langle\bigcap\ \{\langle G\rangle \mid G.\ G \in S\}\rangle = \bigcap\ \{\langle G\rangle \mid G.\ G \in S\}$
proof
{fix *G* **assume** $G \in S$
hence $\langle\bigcap\ \{\langle G\rangle \mid G.\ G \in S\}\rangle \subseteq \langle G\rangle$
using *Inter-lower[of ⟨G⟩ {⟨G⟩ |G. G ∈ S}] mem-Collect-eq[of ⟨G⟩ λ A. ∃ G. G ∈ S ∧ A = ⟨G⟩]*
hull-mono[of ⋂ {⟨G⟩ |G. G ∈ S} ⟨G⟩] **unfolding** *gen-idemp* **by** *auto***}**
thus $\langle\bigcap\ \{\langle G\rangle \mid G.\ G \in S\}\rangle \subseteq \bigcap\ \{\langle G\rangle \mid G.\ G \in S\}$ **by** *blast*
next
show $\bigcap\ \{\langle G\rangle \mid G.\ G \in S\} \subseteq \langle\bigcap\ \{\langle G\rangle \mid G.\ G \in S\}\rangle$
by *simp*
qed

theorem *hull-minimal*: $\langle G\rangle = \bigcap\ \{S\ .\ G \subseteq S \wedge \langle S\rangle = S\}$
proof
show $\langle G\rangle \subseteq \bigcap\ \{S.\ G \subseteq S \wedge \langle S\rangle = S\}$
by (*rule Inter-greatest*, *unfold mem-Collect-eq*) (*use hull-mono'[of G]* **in** *blast*)
show $\bigcap\ \{S.\ G \subseteq S \wedge \langle S\rangle = S\} \subseteq \langle G\rangle$
by (*rule Inter-lower*, *unfold mem-Collect-eq gen-idemp*) (*use genset-sub* **in** *blast*)
qed

lemma *all-gen-comm-hull-comm*: **assumes** $\bigwedge x\ y.\ x \in G \Longrightarrow y \in G \Longrightarrow x \cdot y = y \cdot x$
$u \in \langle G\rangle\ v \in \langle G\rangle$
shows $u \cdot v = v \cdot u$
proof (*rule hull.induct[OF ‹u ∈ ⟨G⟩›]*, *blast*)
fix *w1 w2*
assume $w1 \in G\ w2 \in \langle G\rangle\ w2 \cdot v = v \cdot w2$
show $(w1 \cdot w2) \cdot v = v \cdot w1 \cdot w2$
unfolding *rassoc ‹w2 · v = v · w2›* **unfolding** *lassoc cancel-right*
proof (*rule hull.induct[OF ‹v ∈ ⟨G⟩›]*, *blast*)
fix *w1' w2'*
assume $w1' \in G\ w2' \in \langle G\rangle\ w1 \cdot w2' = w2' \cdot w1$

show *w1* · *w1′* · *w2′* = (*w1′* · *w2′*) · *w1*
unfolding *rassoc* ‹*w1* · *w2′* = *w2′* · *w1*›[*symmetric*] **unfolding** *lassoc cancel-right*
using ‹*w1′* ∈ *G*› ‹*w1* ∈ *G*› *assms*(*1*) **by** *force*
qed
qed

lemma *bin-comm-hull-comm*: **assumes** *x* · *y* = *y* · *x* *u* ∈ ⟨{*x*,*y*}⟩ *v* ∈ ⟨{*x*,*y*}⟩
shows *u* · *v* = *v* · *u*
by (*rule all-gen-comm-hull-comm*[*OF - assms*(*2*−*3*)]) (*use assms*(*1*) **in** *fastforce*)

lemma[*reversal-rule*]: *rev* ‘ ⟨{*rev u*, *rev v*}⟩ = ⟨{*u*,*v*}⟩
by (*simp add*: *rev-hull*)

lemma[*reversal-rule*]: *rev w* ∈ ⟨*rev* ‘ *G*⟩ ≡ *w* ∈ ⟨*G*⟩
unfolding *rev-in-conv rev-hull rev-rev-image-eq*.

3.2 Factorization into generators

We define a decomposition (or a factorization) of a into elements of a given generating set. Such a decomposition is well defined only if the decomposed word is an element of the hull. Even int that case, however, the decomposition need not be unique.

definition *decompose* :: *'a list set* ⇒ *'a list* ⇒ *'a list list* (*Dec - -* [*55*,*55*] *56*)
where
decompose G u = (*SOME us*. *us* ∈ *lists* (*G* − {*ε*}) ∧ *concat us* = *u*)

lemma *dec-ex*: **assumes** *u* ∈ ⟨*G*⟩ **shows** ∃ *us*. (*us* ∈ *lists* (*G* − {*ε*}) ∧ *concat us* = *u*)
using *assms* **unfolding** *image-def hull-concat-lists*[*of G*] *mem-Collect-eq*
using *del-emp-concat lists-minus′* **by** *metis*

lemma *dec-in-lists′*: *u* ∈ ⟨*G*⟩ ⟹ (*Dec G u*) ∈ *lists* (*G* − {*ε*})
unfolding *decompose-def* **using** *someI-ex*[*OF dec-ex*] **by** *blast*

lemma *concat-dec*[*simp*, *intro*] : *u* ∈ ⟨*G*⟩ ⟹ *concat* (*Dec G u*) = *u*
unfolding *decompose-def* **using** *someI-ex*[*OF dec-ex*] **by** *blast*

lemma *dec-emp* [*simp*]: *Dec G ε* = *ε*
proof−
have *ex*: *ε* ∈ *lists* (*G* − {*ε*}) ∧ *concat ε* = *ε*
by *simp*
have *all*: (*us* ∈ *lists* (*G* − {*ε*}) ∧ *concat us* = *ε*) ⟹ *us* = *ε* **for** *us*
using *emp-concat-emp* **by** *auto*
show *?thesis*
unfolding *decompose-def*
using *all*[*OF someI*[*of* λ *x*. *x* ∈ *lists* (*G* − {*ε*}) ∧ *concat x* = *ε*, *OF ex*]].
qed

lemma *dec-nemp*: $u \in \langle G \rangle - \{\varepsilon\} \Longrightarrow \ Dec\ G\ u \neq \varepsilon$
using *concat-dec*[*of u G*] **by** *force*

lemma *dec-nemp'*[*simp*, *intro*]: $u \neq \varepsilon \Longrightarrow u \in \langle G \rangle \Longrightarrow Dec\ G\ u \neq \varepsilon$
using *dec-nemp* **by** *blast*

lemma *dec-eq-emp-iff* [*simp*]: **assumes** $u \in \langle G \rangle$ **shows** $Dec\ G\ u = \varepsilon \longleftrightarrow u = \varepsilon$
using *dec-nemp'*[*OF* - ‹$u \in \langle G \rangle$›] **by** *auto*

lemma *dec-in-lists*[*simp*]: $u \in \langle G \rangle \Longrightarrow Dec\ G\ u \in lists\ G$
using *dec-in-lists'* **by** *auto*

lemma *set-dec-sub*: $x \in \langle G \rangle \Longrightarrow set\ (Dec\ G\ x) \subseteq G$
using *dec-in-lists* **by** *blast*

lemma *dec-hd*: $u \neq \varepsilon \Longrightarrow u \in \langle G \rangle \Longrightarrow hd\ (Dec\ G\ u) \in G$
by *simp*

lemma *non-gen-dec*: **assumes** $u \in \langle G \rangle\ u \notin G$ **shows** $Dec\ G\ u\ \neq [u]$
using *dec-in-lists*[*OF* ‹$u \in \langle G \rangle$›] *Cons-in-lists-iff*[*of u ε G*] ‹$u \notin G$› **by** *argo*

lemma *fac-fac-interpE-dec*: **assumes** $w \in \langle G \rangle\ \ u \leq_f w\ u \neq \varepsilon$
obtains $p\ s\ ws$ **where** $ws \in lists\ (G - \{\varepsilon\})\ p\ u\ s \sim_{\mathcal{I}} ws\ ws \leq_f Dec\ G\ w$
proof–
from *fac-fac-interpE'*[*OF* - ‹$u \neq \varepsilon$›, *of Dec G w*, *unfolded concat-dec*[*OF* ‹$w \in \langle G \rangle$›], *OF* ‹$u \leq_f w$›]
obtain $p\ s\ ws$ **where** $*$: $p\ u\ s \sim_{\mathcal{I}} ws\ ws \leq_f Dec\ G\ w$**.**
have $ws \in lists\ (G - \{\varepsilon\})$
using *fac-in-lists*[*OF dec-in-lists'*[*OF* ‹$w \in \langle G \rangle$›] ‹$ws \leq_f Dec\ G\ w$›]**.**
from *that*[*OF this* $*$]
show *thesis***.**
qed

lemma *set-len-mod-concat*: **assumes** $\forall\ x \in set\ us.\ |x|\ mod\ n = 0\ 0 < n$
shows $|concat\ us|\ mod\ n = 0$
using ‹$\forall\ x \in set\ us.\ |x|\ mod\ n = 0$›
by (*induct us*, *force*, *unfold concat-simps lenmorph*, *force*)

lemma *hull-len-mod-concat*: **assumes** $\forall\ x \in G.\ |x|\ mod\ n = 0\ 0 < n\ w \in \langle G \rangle$
shows $|w|\ mod\ n = 0$
by (*rule set-len-mod-concat*[*of Dec G w*, *OF* - ‹$0 < n$›, *unfolded concat-dec*[*OF* ‹$w \in \langle G \rangle$›]] *concat-dec*[*OF* ‹$w \in \langle G \rangle$›])
(*use* ‹$\forall\ x \in G.\ |x|\ mod\ n = 0$› ‹$w \in \langle G \rangle$› *set-dec-sub* **in** *blast*)

3.2.1 Refinement into a specific decomposition

We extend the decomposition to lists of words. This can be seen as a refinement of a previous decomposition of some word.

definition *refine* :: *'a list set ⇒ 'a list list ⇒ 'a list list* (*Ref - - [55,56] 56*) **where**
refine G us = concat(*map* (*decompose G*) *us*)

lemma *ref-morph*: *us ∈ lists ⟨G⟩ ⟹ vs ∈ lists ⟨G⟩ ⟹ refine G* (*us · vs*) = *refine G us · refine G vs*
unfolding *refine-def* **by** *simp*

lemma *ref-morph-plus*: *us ∈ lists* (⟨*G*⟩ − {*ε*}) ⟹ *vs ∈ lists* (⟨*G*⟩ − {*ε*}) ⟹ *refine G* (*us · vs*) = *refine G us · refine G vs*
unfolding *refine-def* **by** *simp*

lemma *ref-pref-mono*: *ws ∈ lists ⟨G⟩ ⟹ us ≤p ws ⟹ Ref G us ≤p Ref G ws*
unfolding *prefix-def* **using** *ref-morph append-in-lists-dest' append-in-lists-dest*
by *metis*

lemma *ref-suf-mono*: *ws ∈ lists ⟨G⟩ ⟹ us ≤s ws ⟹* (*Ref G us*) ≤*s Ref G ws*
unfolding *suffix-def* **using** *ref-morph append-in-lists-dest' append-in-lists-dest*
by *metis*

lemma *ref-fac-mono*: *ws ∈ lists ⟨G⟩ ⟹ us ≤f ws ⟹* (*Ref G us*) ≤*f Ref G ws*
unfolding *sublist-altdef'* **using** *ref-pref-mono ref-suf-mono suf-in-lists* **by** *metis*

lemma *ref-pop-hd*: *us ≠ ε ⟹ us ∈ lists ⟨G⟩ ⟹ refine G us = decompose G* (*hd us*) *· refine G* (*tl us*)
unfolding *refine-def* **using** *list.simps*(*9*)[*of decompose G hd us tl us*] **by** *simp*

lemma *ref-in*: *us ∈ lists ⟨G⟩ ⟹* (*Ref G us*) ∈ *lists* (*G* − {*ε*})
proof (*induction us*)
case (*Cons a us*)
then show *?case*
using *Cons.IH Cons.prems dec-in-lists'* **by** (*auto simp add*: *refine-def*)
qed (*simp add*: *refine-def*)

lemma *ref-in'*[*intro*]: *us ∈ lists ⟨G⟩ ⟹* (*Ref G us*) ∈ *lists G*
using *ref-in* **by** *auto*

lemma *concat-ref*: *us ∈ lists ⟨G⟩ ⟹ concat* (*Ref G us*) = *concat us*
proof (*induction us*)
case (*Cons a us*)
then show *?case*
using *Cons.IH Cons.prems concat-dec refine-def* **by** (*auto simp add*: *refine-def*)
qed (*simp add*: *refine-def*)

lemma *ref-gen*: *us ∈ lists B ⟹ B ⊆ ⟨G⟩ ⟹ Ref G us ∈ ⟨decompose G ‘ B⟩*
by (*induct us*, *auto simp add*: *refine-def*)

lemma *ref-set*: *ws ∈ lists ⟨G⟩ ⟹ set* (*Ref G ws*) = ⋃ (*set‘*(*decompose G*)*‘set ws*)
by (*simp add*: *refine-def*)

lemma *emp-ref*: **assumes** *us* ∈ *lists* (⟨*G*⟩ − {*ε*}) **and** *Ref G us* = *ε* **shows** *us* = *ε*
using *emp-concat-emp*[*OF* ‹*us* ∈ *lists* (⟨*G*⟩ − {*ε*})›]
concat-ref [*OF lists-minus*[*OF assms*(*1*)], *unfolded* ‹*Ref G us* = *ε*› *concat.simps*(*1*),*symmetric*]
by *blast*

lemma *sing-ref-sing*:
assumes *us* ∈ *lists* (⟨*G*⟩ − {*ε*}) **and** *refine G us* = [*b*]
shows *us* = [*b*]
proof−
have *us* ≠ *ε*
using ‹*refine G us* = [*b*]› **by** (*auto simp add*: *refine-def*)
have *tl us* ∈ *lists* (⟨*G*⟩ − {*ε*}) **and** *hd us* ∈ ⟨*G*⟩ − {*ε*}
using *list.collapse*[*OF* ‹*us* ≠ *ε*›] ‹*us* ∈ *lists* (⟨*G*⟩ − {*ε*})› *Cons-in-lists-iff*[*of hd us tl us* ⟨*G*⟩ − {*ε*}]
by *auto*
have *Dec G* (*hd us*) ≠ *ε*
using *dec-nemp*[*OF* ‹*hd us* ∈ ⟨*G*⟩ − {*ε*}›].
have *us* ∈ *lists* ⟨*G*⟩
using ‹*us* ∈ *lists* (⟨*G*⟩ − {*ε*})› *lists-minus* **by** *auto*
have *concat us* = *b*
using ‹*us* ∈ *lists* ⟨*G*⟩› *assms*(*2*) *concat-ref* **by** *force*
have *refine G* (*tl us*) = *ε*
using *ref-pop-hd*[*OF* ‹*us* ≠ *ε*› ‹*us* ∈ *lists* ⟨*G*⟩›] **unfolding** ‹*refine G us* = [*b*]›
using ‹*Dec G* (*hd us*) ≠ *ε*› *Cons-eq-append-conv*[*of b ε* (*Dec G* (*hd us*)) (*Ref G* (*tl us*))]
Cons-eq-append-conv[*of b ε* (*Dec G* (*hd us*)) (*Ref G* (*tl us*))] *append-is-Nil-conv*[*of* - (*Ref G* (*tl us*))]
by *blast*
from *emp-ref*[*OF* ‹*tl us* ∈ *lists* (⟨*G*⟩ − {*ε*})› *this*, *symmetric*]
have *ε* = *tl us*.
from *this*[*unfolded Nil-tl*]
show *?thesis*
using ‹*us* ≠ *ε*› ‹*concat us* = *b*› **by** *auto*
qed

lemma *ref-ex*: **assumes** *Q* ⊆ ⟨*G*⟩ **and** *us* ∈ *lists Q*
shows *Ref G us* ∈ *lists* (*G* − {*ε*}) **and** *concat* (*Ref G us*) = *concat us*
using *ref-in*[*OF sub-lists-mono*[*OF assms*]] *concat-ref*[*OF sub-lists-mono*[*OF assms*]].

lemma *ref-emp* [*simp*]: *Ref G ε* = *ε*
unfolding *refine-def* **by** *force*

3.3 Basis

An important property of monoids of words is that they have a unique minimal generating set. Which is the set consisting of indecomposable elements.

Indecomposable element is an element that is not generated by other ones.

definition *ungenerated* :: *'a list* ⇒ *'a list set* ⇒ *bool* (- ∈*U* - [*51*,*51*] *50*) **where**
ungenerated b G ≡ *b* ∈ *G* ∧ *b* ∉ ⟨*G* − {*b*}⟩

lemma *ungen-nemp*[*simp*]: *b* ∈*U G* ⟹ *b* ≠ *ε*
unfolding *ungenerated-def* **by** *blast*

lemma *ungen-in*[*intro*]: *ungenerated b G* ⟹ *b* ∈ *G* **and**
ungenD[*intro*]: *ungenerated b G* ⟹ *b* ∉ ⟨*G* − {*b*}⟩
unfolding *ungenerated-def* **by** *blast*+

An ungenerated element has no nontrivial decomposition

lemma *ungen-dec-triv*: **assumes** *u* ∈ ⟨*G*⟩ *v* ∈ ⟨*G*⟩ *u* · *v* ∈*U* ⟨*G*⟩ **shows** *u* = *ε* ∨ *v* = *ε*
proof (*rule ccontr*)
assume ¬ (*u* = *ε* ∨ *v* = *ε*)
hence *u* · *v* ≠ *u u* · *v* ≠ *v*
by *blast*+
hence *u* ∈ ⟨⟨*G*⟩ − {*u* · *v*}⟩ *v* ∈ ⟨⟨*G*⟩ − {*u* · *v*}⟩
using *assms*(*1*−*2*) **by** *blast*+
from *hull-closed*[*OF this*]
show *False*
using ‹*u* · *v* ∈*U* ⟨*G*⟩› **unfolding** *ungenerated-def* **by** *blast*
qed

lemma *ungen-dec-triv'*: **assumes** *us* ∈ *lists* (⟨*G*⟩ − {*ε*}) *concat us* ∈*U* ⟨*G*⟩ **shows** |*us*| = *1*
proof−
have *us* ≠ *ε*
using ‹*concat us* ∈*U* ⟨*G*⟩› *ungen-nemp* **by** *force*
hence *hd us* ≠ *ε hd us* ∈ ⟨*G*⟩
using ‹*us* ∈ *lists* (⟨*G*⟩ − {*ε*})› **by** *force*+
have *concat* (*tl us*) ∈ ⟨*G*⟩
using *concat-in-hull'*[*OF tl-in-lists*, *OF* ‹*us* ∈ *lists* (⟨*G*⟩ − {*ε*})›, *unfolded gen-idemp*]
by (*simp add*: *gen-idemp*)
have *hd us* · *concat* (*tl us*) ∈*U* ⟨*G*⟩
using ‹*concat us* ∈*U* ⟨*G*⟩› **by** (*subst* (*asm*) (*1*) *hd-tl*[*OF* ‹*us* ≠ *ε*›, *symmetric*], *simp*)
from *ungen-dec-triv*[*of hd us* - *concat* (*tl us*), *OF* ‹*hd us* ∈ ⟨*G*⟩› ‹*concat* (*tl us*) ∈ ⟨*G*⟩› *this*]
have *tl us* = *ε*
using *tl-in-lists*[*OF* ‹*us* ∈ *lists* (⟨*G*⟩ − {*ε*})›] ‹*hd us* ≠ *ε*› **by** *force*
then show |*us*| = *1*
using *nemp-le-len*[*OF* ‹*us* ≠ *ε*›] *long-list-tl* **by** *force*
qed

Conversely, any nonempty element that is not ungenerated is a product of at least two shorter elements

lemma *gen-elem-list*: **assumes** *u* ∈ ⟨*G* − {*u*}⟩ *u* ≠ *ε*

obtains *us* **where** $us \in lists\ (G - \{u\} - \{\varepsilon\})\ concat\ us = u\ 1 < |us|$
$\bigwedge c.\ c \in set\ us \Longrightarrow |c| < |u|$
proof–
from *hull-concat-lists-nempE*[*OF* ‹$u \in \langle G - \{u\}\rangle$›]
obtain *ws* **where** *cond*: $ws \in lists\ (G - \{u\} - \{\varepsilon\})\ concat\ ws = u$.
have $1 < |ws|$
proof (*rule ccontr*)
assume $\neg\ 1 < |ws|$
hence $|ws| = 1$
using *nemp-len*[*of ws*] ‹$u \neq \varepsilon$›[*folded* ‹$concat\ ws = u$›]
by (*metis concat-nemp less-one linorder-neqE-nat nemp-len-not0*)
hence $ws = [u]$
using ‹$concat\ ws = u$› *sing-word-concat* **by** *fastforce*
thus *False*
using ‹$ws \in lists\ (G - \{u\} - \{\varepsilon\})$› **by** *force*
qed
have $|c| < |u|$ **if** $c \in set\ ws$ **for** *c*
proof–
obtain *ws1 ws2* **where** $ws = ws1 \cdot [c] \cdot ws2$
using ‹$c \in set\ ws$› *split-listE* **by** *meson*
hence *ws-lists*: $ws1 \in lists\ (G - \{u\} - \{\varepsilon\})\ ws2 \in lists\ (G - \{u\} - \{\varepsilon\})$
using ‹$ws \in lists\ (G - \{u\} - \{\varepsilon\})$› **by** *simp-all*
have $ws1 \cdot\ ws2 \neq \varepsilon$
using ‹$1 < |ws|$›[*unfolded* ‹$ws = ws1 \cdot [c] \cdot ws2$›] **by** *force*
hence $|concat\ ws1| + |concat\ ws2| \neq 0$
using *ws-lists* **by** *force*
thus $|c| < |u|$
using *lenarg*[*OF arg-cong*[*OF* ‹$ws = ws1 \cdot [c] \cdot ws2$›, *of concat*]]
unfolding *concat-morph lenmorph concat-sing*′ ‹$concat\ ws = u$› **by** *linarith*
qed
from *that*[*OF cond* ‹$1 < |ws|$› *this*]
show *thesis*.
qed

lemma *gen-elem-dec*: **assumes** $b \in \langle G - \{b\}\rangle\ b \neq \varepsilon$
obtains *u v* **where** $u \in \langle G\rangle\ u \neq \varepsilon\ v \in \langle G\rangle\ v \neq \varepsilon\ u \cdot v = b$
proof–
from *gen-elem-list*[*OF assms*]
obtain *us* **where** $us \in lists\ (G - \{b\} - \{\varepsilon\})\ concat\ us = b\ 1 < |us|$.
have $us \neq \varepsilon\ tl\ us \neq \varepsilon\ [hd\ us] \neq \varepsilon$
using *long-list-tl*[*OF* ‹$1 < |us|$›] **by** *fastforce+*
note $b =$ *arg-cong*[*OF hd-tl*[*OF* ‹$us \neq \varepsilon$›], *of concat*, *unfolded* ‹$concat\ us = b$› *concat-morph*]
have $concat\ (tl\ us) \neq \varepsilon$
using ‹$tl\ us \neq \varepsilon$› ‹$us \in lists\ (G - \{b\} - \{\varepsilon\})$› *emp-concat-emp tl-in-lists* **by** *meson*
have $concat\ (tl\ us) \in \langle G\rangle$
using ‹$us \in lists\ (G - \{b\} - \{\varepsilon\})$› *concat-in-hull*′ *lists-minus tl-in-lists* **by** *meson*

have *concat* [*hd us*] ≠ *ε*
using ‹*us* ∈ *lists* (*G* − {*b*} − {*ε*})› ‹*us* ≠ *ε*› **by** *fastforce*
have *concat* [*hd us*] ∈ ⟨*G*⟩
using ‹*us* ∈ *lists* (*G* − {*b*} − {*ε*})› *Diff-iff* ‹*us* ≠ *ε*› *concat-sing′ gen-in lists-hd-in-set* **by** *metis*
show *thesis*
by (*rule that*[*OF - - - - b*]) *fact+*
qed

This yields several criteria for being ungenerated

lemma *ungeneratedI*:
assumes *b* ∈ *G* **and** *b* ≠ *ε*
and *all*: ⋀ *u v*. *u* ≠ *ε* ⟹ *u* ∈ ⟨*G*⟩ ⟹ *v* ≠ *ε* ⟹ *v* ∈ ⟨*G*⟩ ⟹ *u* · *v* ≠ *b*
shows *b* ∈*U G*
unfolding *ungenerated-def*
proof
show *b* ∉ ⟨*G* − {*b*}⟩
proof
assume *b* ∈ ⟨*G* − {*b*}⟩
from *gen-elem-dec*[*OF this* ‹*b* ≠ *ε*›]
show *False*
using *all* **by** *metis*
qed
qed *fact*

lemma *ungeneratedI′*:
assumes *b* ∈ *G* **and** *b* ≠ *ε*
and *all*: ⋀ *us*. *us* ∈ *lists* (*G* − {*b*} − {*ε*}) ⟹ *concat us* = *b* ⟹ |*us*| ≤ *1*
shows *b* ∈*U G*
unfolding *ungenerated-def*
proof
show *b* ∉ ⟨*G* − {*b*}⟩
proof
assume *b* ∈ ⟨*G* − {*b*}⟩
from *gen-elem-list*[*OF this* ‹*b* ≠ *ε*›]
show *False*
using *all le-antisym nless-le* **by** *metis*
qed
qed *fact*

lemma *ungenerated-shortest*:
assumes *b* ∈ *G* **and** *b* ≠ *ε*
and *all*: ⋀ *c*. *c* ∈ *G* − {*ε*} ⟹ |*b*| ≤ |*c*|
shows *b* ∈*U G*
unfolding *ungenerated-def*
proof
show *b* ∉ ⟨*G* − {*b*}⟩
proof
assume *b* ∈ ⟨*G* − {*b*}⟩

from *gen-elem-list*[*OF this* ‹$b \neq \varepsilon$›]
obtain *us* **where** *us* ∈ *lists* ($G - \{b\} - \{\varepsilon\}$) *concat us* = *b* **and** *all'*: ($\bigwedge c.\ c \in set\ us \Longrightarrow |c| < |b|$)
by *metis*
have $c \in set\ us \Longrightarrow |b| \leq |c|$ **for** *c*
using *all* ‹*us* ∈ *lists* ($G - \{b\} - \{\varepsilon\}$)› **by** *blast*
then show *False*
using *all'* ‹$b \neq \varepsilon$› ‹*concat us* = *b*› **unfolding** *linorder-not-less*[*symmetric*]
by *fastforce*
qed
qed *fact*

lemma *ungenerated-sing*:
assumes [*a*] ∈ *G*
shows [*a*] ∈*U G*
using *ungenerated-shortest*[*OF assms*] *nemp-le-len* **unfolding** *sing-len*
by *blast*

— Ungeneretad element is ungenerated by the whole monoid

lemma *ungen-hull-ungen*: *b* ∈*U* ⟨*G*⟩ ⟷ *b* ∈*U G*
proof
assume *b* ∈*U G*
show *b* ∈*U* ⟨*G*⟩
unfolding *ungenerated-def*
proof
show *b* ∉ ⟨⟨*G*⟩ − {*b*}⟩
proof
assume *b* ∈ ⟨⟨*G*⟩ − {*b*}⟩
from *gen-elem-list*[*OF this ungen-nemp*[*OF* ‹*b* ∈*U G*›]]
obtain *us* **where** *us* ∈ *lists* (⟨*G*⟩ − {*b*} − {ε}) **and** *concat us* = *b*
and *1* < |*us*| **and** *short*: $\bigwedge c.\ c \in set\ us \Longrightarrow |c| < |b|$
by *blast*
have *no-b*: *b* ∉ *set* (*Dec G u*) **if** *u* ∈ *set us* **for** *u*
proof
assume *b* ∈ *set* (*Dec G u*)
have *concat* (*Dec G u*) = *u*
using ‹*us* ∈ *lists* (⟨*G*⟩ − {*b*} − {ε})› ‹*u* ∈ *set us*› **by** *blast*
hence |*b*| ≤ |*u*|
using *split-list-first*[*OF* ‹*b* ∈ *set* (*Dec G u*)›] **by** *force*
thus *False*
using *short*[*OF* ‹*u* ∈ *set us*›] **by** *force*
qed
define *vs* **where** *vs* = *refine G us*
have *concat vs* = *b vs* ∈ *lists* ($G - \{\varepsilon\}$)
using *ref-ex*[*OF* - ‹*us* ∈ *lists* (⟨*G*⟩ − {*b*} − {ε})›, *of G*]
unfolding *vs-def* ‹*concat us* = *b*› **by** *blast+*
have *b* ∉ *set vs*
using *no-b* **unfolding** *vs-def refine-def* **by** *simp*

```
    hence vs ∈ lists (G − {b})
      using ‹vs ∈ lists (G − {ε})› by blast
    with ungenD[OF ‹b ∈U G›]
    show False
      unfolding ungenerated-def using ‹concat vs = b› by blast
  qed
  show b ∈ ⟨G⟩
    using ungen-in[OF ‹b ∈U G›] by blast
qed
next
  assume b ∈U ⟨G⟩
  show b ∈U G
    unfolding ungenerated-def
  proof
    have G − {b} ⊆ ⟨G⟩ − {b}
      by blast
    from hull-mono[OF this]
    show b ∉ ⟨G − {b}⟩
      using ungenD[OF ‹b ∈U ⟨G⟩›] by blast
    then show b ∈ G
      using ungen-in[OF ‹b ∈U ⟨G⟩›] by auto
  qed
qed
```

The *basis* is the set of all ungenerated elements.

```
definition basis :: 'a list set ⇒ 'a list set (𝔅 - [51] ) where
  basis G = {x. x ∈U G}

lemma basisD: x ∈ 𝔅 G ⟹ x ∈U G
  unfolding basis-def by simp

lemma basis-in: x ∈ 𝔅 G ⟹ x ∈ G
  unfolding basis-def ungenerated-def by simp

lemma emp-not-basis: x ∈ 𝔅 G ⟹ x ≠ ε
  unfolding basis-def ungenerated-def by blast

lemma basis-sub-gen: 𝔅 G ⊆ G
  unfolding  basis-def ungenerated-def by simp
```

The basis generates the generating set and therefore also the whole monoid

```
lemma gen-sub-basis: G ⊆ ⟨𝔅 G⟩
proof
  fix w show w ∈ G ⟹ w ∈ ⟨𝔅 G⟩
  proof (induct length w arbitrary: w rule: less-induct)
    case less
    show ?case
    proof (cases w ∈ 𝔅 G ∨ w = ε, blast)
      assume ¬ (w ∈ 𝔅 G ∨ w = ε)
```

hence $w \in \langle G - \{w\}\rangle$ $w \neq \varepsilon$
using ‹$w \in G$› **unfolding** *basis-def ungenerated-def* **by** *blast+*
from *gen-elem-list*[*OF this*[*unfolded ungenerated-def*]]
obtain *us* **where** $us \in lists\ (G - \{w\} - \{\varepsilon\})$ $concat\ us = w$ $1 < |us|$ **and**
small: $(\bigwedge c.\ c \in set\ us \Longrightarrow |c| < |w|)$
by *blast*
have $c \in \langle \mathfrak{B}\ G\rangle$ **if** $c \in set\ us$ **for** c
using *less.hyps*[*of c*, *OF small*, *OF that*] ‹$c \in set\ us$› ‹$us \in lists\ (G - \{w\} - \{\varepsilon\})$› **by** *blast*
thus $w \in \langle \mathfrak{B}\ G\rangle$
unfolding ‹$concat\ us = w$›[*symmetric*]
using *hull-closed-lists*[*OF in-listsI*] **by** *blast*
qed
qed
qed

lemma *basis-concat-listsE*:
assumes $w \in G$
obtains *ws* **where** $ws \in lists\ \mathfrak{B}\ G$ **and** $concat\ ws = w$
using *assms* **by** (*rule hull-concat-listsE*[*OF subsetD*, *OF gen-sub-basis*])

theorem *basis-gen-hull*: $\langle \mathfrak{B}\ G\rangle = \langle G\rangle$
by (*rule equalityI*; *simp only*: *hull-mono*[*OF basis-sub-gen*] *hull-mono*[*OF gen-sub-basis*, *unfolded gen-idemp*])

theorem *basis-of-hull*: $\mathfrak{B}\ \langle G\rangle = \mathfrak{B}\ G$
unfolding *basis-def ungen-hull-ungen*..

lemma *basis-gen-hull'*: $\langle \mathfrak{B}\ \langle G\rangle\rangle = \langle G\rangle$
unfolding *basis-of-hull* **using** *basis-gen-hull*.

lemma *basis-hull-sub*: $\mathfrak{B}\ \langle G\rangle \subseteq G$
unfolding *basis-of-hull* **using** *basis-sub-gen*.

The basis is the smallest generating set.

theorem *gen-basis-sub*: $\langle S\rangle = \langle G\rangle \Longrightarrow \mathfrak{B}\ G \subseteq S$
using *basis-hull-sub*[*of S*] *basis-of-hull*[*of G*] **by** *simp*

lemma *basis-min-gen*: $S \subseteq \mathfrak{B}\ G \Longrightarrow \langle S\rangle = G \Longrightarrow S = \mathfrak{B}\ G$
using *basis-of-hull basis-sub-gen* **by** *blast*

lemma *basisI*: $(\bigwedge B.\ \langle B\rangle = \langle C\rangle \Longrightarrow C \subseteq B) \Longrightarrow \mathfrak{B}\ \langle C\rangle = C$
using *basis-gen-hull basis-min-gen basis-of-hull* **by** *metis*

An arbitrary set between basis and the hull is generating...

lemma *gen-sets*: **assumes** $\mathfrak{B}\ G \subseteq S$ **and** $S \subseteq \langle G\rangle$ **shows** $\langle S\rangle = \langle G\rangle$
using *hull-mono*[*OF* ‹$S \subseteq \langle G\rangle$›, *unfolded gen-idemp*] *hull-mono*[*OF* ‹$\mathfrak{B}\ G \subseteq S$›, *unfolded basis-gen-hull*] **by** *blast*

... and has the same basis

lemma *basis-sets*: $\mathfrak{B}\ G \subseteq S \Longrightarrow S \subseteq \langle G \rangle \Longrightarrow \mathfrak{B}\ G = \mathfrak{B}\ S$
by (*metis basis-of-hull gen-sets*)

3.4 Code

locale *nemp-words* =
fixes G
assumes *emp-not-in*: $\varepsilon \notin G$

begin
lemma *drop-empD*: $G - \{\varepsilon\} = G$
using *emp-not-in* **by** *simp*

lemmas *emp-concat-emp'* = *emp-concat-emp*[*of* - G, *unfolded drop-empD*]

thm *disjE*[*OF ruler*[*OF take-is-prefix take-is-prefix*]]

lemma *nemp*: $x \in G \Longrightarrow x \neq \varepsilon$
using *emp-not-in* **by** *blast*

lemma *concat-eq-emp-conv* [*simp*]: $us \in lists\ G \Longrightarrow concat\ us = \varepsilon \longleftrightarrow us = \varepsilon$
unfolding *in-lists-conv-set concat-eq-Nil-conv*
by (*simp add*: *nemp*)

lemma *root-dec-inj-on*: *inj-on* $(\lambda\ x.\ [\varrho\ x]^{@}(e_{\varrho}\ x))\ G$
unfolding *inj-on-def* **using** *primroot-exp-eq*
unfolding *concat-pow-list-single*[*of* - ϱ -, *symmetric*] **by** *metis*

lemma *concat-len-ruler*: **assumes** $ws \in lists\ G\ us \leq p\ ws\ vs \leq p\ ws\ |concat\ us| \leq |concat\ vs|$
shows $us \leq p\ vs$
proof (*rule ccontr*)
assume $\neg\ us \leq p\ vs$
with *ruler*[*OF assms*(*2*−*3*)]
have $vs < p\ us$
by *blast*
from *arg-cong*[*OF lq-spref*[*OF this*], *of* $\lambda\ x.\ |concat\ x|$, *unfolded concat-morph lenmorph*]
have $|concat\ (vs^{-1>}us)| = 0$
using *assms*(*4*) **by** *linarith*
hence $vs^{-1>}us = \varepsilon$
using *emp-concat-emp'*[*OF lq-in-lists*[*OF pref-in-lists*[*OF* ‹$us \leq p\ ws$› ‹$ws \in lists\ G$›]]] **by** *blast*
thus *False*
using *lq-spref-nemp*[*OF* ‹$vs < p\ us$›] **by** *contradiction*
qed

end

lemma *concat-emp*:
  $\varepsilon \notin G \Longrightarrow us \in lists\ G \Longrightarrow concat\ us = \varepsilon \Longrightarrow us = \varepsilon$
using *nemp-words.concat-eq-emp-conv*[*OF nemp-words.intro*] **by** *blast*

A basis freely generating its hull is called a *code*. By definition, this means that generated elements have unique factorizations into the elements of the code.

locale *code* =
  fixes $\mathcal{C}$
  assumes *is-code*: $xs \in lists\ \mathcal{C} \Longrightarrow ys \in lists\ \mathcal{C} \Longrightarrow concat\ xs = concat\ ys \Longrightarrow xs = ys$
begin

lemma *code-comm-eq*: $x \in \mathcal{C} \Longrightarrow y \in \mathcal{C} \Longrightarrow x \cdot y = y \cdot x \Longrightarrow x = y$
  using *is-code*[*of* [*x*,*y*] [*y*,*x*], *THEN arg-cong*[*of* - - *hd*]] **by** *simp*

lemma *emp-not-in*: $\varepsilon \notin \mathcal{C}$
proof
  assume $\varepsilon \in \mathcal{C}$
  hence $[] \in lists\ \mathcal{C}$ **and** $[\varepsilon] \in lists\ \mathcal{C}$ **and** $concat\ [] = concat\ [\varepsilon]$ **and** $[] \neq [\varepsilon]$
    by *simp+*
  thus *False*
    using *is-code* **by** *blast*
qed

lemma *nemp*: $u \in \mathcal{C} \Longrightarrow u \neq \varepsilon$
  using *emp-not-in* **by** *force*

sublocale *nemp-words* $\mathcal{C}$
  using *emp-not-in* **by** *unfold-locales*

lemma *code-elem-dec*: $us \in lists\ \mathcal{C} \Longrightarrow concat\ us = c \Longrightarrow c \in \mathcal{C} \Longrightarrow us = [c]$
  using *is-code*[*of us* [*c*]] **by** *simp*

lemma *code-ungen*: **assumes** $c \in \mathcal{C}$ **shows** $c \in U\ \mathcal{C}$
  unfolding *ungenerated-def*
proof
  show $c \notin \langle \mathcal{C} - \{c\} \rangle$
  proof
    assume ‹$c \in \langle \mathcal{C} - \{c\} \rangle$›
    from *gen-elem-list*[*OF this nemp*[*OF* ‹$c \in \mathcal{C}$›]]
    obtain *us* **where** $us \in lists\ (\mathcal{C} - \{c\} - \{\varepsilon\})\ concat\ us = c\ 1 < |us|$**.**
    show *False*
      using *code-elem-dec*[*OF* - ‹*concat us* = *c*› ‹$c \in \mathcal{C}$›]
        ‹*1* < |*us*|› *sing-len*[*of c*] ‹$us \in lists\ (\mathcal{C} - \{c\} - \{\varepsilon\})$› **by** *fastforce*
  qed
qed *fact*

lemma *code-is-basis*: $\mathfrak{B}\ \mathcal{C} = \mathcal{C}$
using *code-ungen basis-def*[*of* $\mathcal{C}$] *basis-sub-gen* **by** *blast*

lemma *code-unique-dec'*: $us \in lists\ \mathcal{C} \Longrightarrow Dec\ \mathcal{C}\ (concat\ us) = us$
using *dec-in-lists*[*of concat us* $\mathcal{C}$, *THEN is-code*, *of us*]
concat-dec[*of concat us* $\mathcal{C}$] *hull-concat-lists*[*of* $\mathcal{C}$] *image-eqI*[*of concat us concat us lists* $\mathcal{C}$]
by *argo*

lemma *code-unique-dec* [*intro*!]: $us \in lists\ \mathcal{C} \Longrightarrow concat\ us = u \Longrightarrow\ Dec\ \mathcal{C}\ u = us$
using *code-unique-dec'* **by** *blast*

lemma *triv-refine*[*intro*!] : $us \in lists\ \mathcal{C} \Longrightarrow concat\ us = u \Longrightarrow\ Ref\ \mathcal{C}\ [u] = us$
using *code-unique-dec'* **by** (*auto simp add*: *refine-def*)

lemma *code-unique-ref*: $us \in lists\ \langle\mathcal{C}\rangle \Longrightarrow refine\ \mathcal{C}\ us = decompose\ \mathcal{C}\ (concat\ us)$
proof–
assume $us \in lists\ \langle\mathcal{C}\rangle$
hence $concat\ (refine\ \mathcal{C}\ us) = concat\ us$
using *concat-ref* **by** *blast*
hence *eq*: $concat\ (refine\ \mathcal{C}\ us) = concat\ (decompose\ \mathcal{C}\ (concat\ us))$
using *concat-dec*[*OF hull-closed-lists*[*OF* ‹$us \in lists\ \langle\mathcal{C}\rangle$›]] **by** *auto*
have *dec*: $Dec\ \mathcal{C}\ (concat\ us) \in lists\ \mathcal{C}$
using ‹$us \in lists\ \langle\mathcal{C}\rangle$› *dec-in-lists hull-closed-lists*
by *metis*
have $Ref\ \mathcal{C}\ us \in lists\ \mathcal{C}$
using *lists-minus*[*OF ref-in*[*OF* ‹$us \in lists\ \langle\mathcal{C}\rangle$›]].
from *is-code*[*OF this dec eq*]
show *?thesis*.
qed

lemma *refI* [*intro*]: $us \in lists\ \langle\mathcal{C}\rangle \Longrightarrow vs \in lists\ \mathcal{C} \Longrightarrow concat\ vs = concat\ us \Longrightarrow Ref\ \mathcal{C}\ us = vs$
unfolding *code-unique-ref code-unique-dec*..

lemma *code-dec-morph*: **assumes** $x \in \langle\mathcal{C}\rangle\ y \in \langle\mathcal{C}\rangle$
shows $(Dec\ \mathcal{C}\ x) \cdot (Dec\ \mathcal{C}\ y) = Dec\ \mathcal{C}\ (x{\cdot}y)$
proof–
have *eq*: $(Dec\ \mathcal{C}\ x) \cdot (Dec\ \mathcal{C}\ y) = Dec\ \mathcal{C}\ (concat\ ((Dec\ \mathcal{C}\ x) \cdot (Dec\ \mathcal{C}\ y)))$
using *dec-in-lists*[*OF* ‹$x \in \langle\mathcal{C}\rangle$›] *dec-in-lists*[*OF* ‹$y \in \langle\mathcal{C}\rangle$›]
code.code-unique-dec[*OF code-axioms*, *of* $(Dec\ \mathcal{C}\ x) \cdot (Dec\ \mathcal{C}\ y)$, *unfolded append-in-lists-conv*, *symmetric*]
by *presburger*
moreover have $concat\ ((Dec\ \mathcal{C}\ x) \cdot (Dec\ \mathcal{C}\ y)) = (x \cdot y)$
using *concat-morph*[*of Dec* $\mathcal{C}$ *x Dec* $\mathcal{C}$ *y*]
unfolding *concat-dec*[*OF* ‹$x \in \langle\mathcal{C}\rangle$›] *concat-dec*[*OF* ‹$y \in \langle\mathcal{C}\rangle$›].
ultimately show $(Dec\ \mathcal{C}\ x) \cdot (Dec\ \mathcal{C}\ y) = Dec\ \mathcal{C}\ (x{\cdot}y)$
by *argo*

qed

lemma *dec-pow*: $rs \in \langle \mathcal{C} \rangle \Longrightarrow Dec\ \mathcal{C}\ (rs^{@}k) = (Dec\ \mathcal{C}\ rs)^{@}k$
proof(*induction k arbitrary*: *rs*, *fastforce*)
case (*Suc k*)
then show *?case*
using *code-dec-morph pow-Suc power-in* **by** *metis*
qed

lemma *code-el-dec*: $c \in \mathcal{C} \Longrightarrow decompose\ \mathcal{C}\ c = [c]$
by *fastforce*

lemma *code-ref-list*: $us \in lists\ \mathcal{C} \Longrightarrow refine\ \mathcal{C}\ us = us$
proof (*induct us*)
case (*Cons a us*)
then show *?case* **using** *code-el-dec*
unfolding *refine-def* **by** *simp*
qed (*simp add*: *refine-def*)

lemma *code-ref-gen*: **assumes** $G \subseteq \langle \mathcal{C} \rangle$ $u \in \langle G \rangle$
shows $Dec\ \mathcal{C}\ u \in \langle decompose\ \mathcal{C}\ `\ G \rangle$
proof−
have *refine* $\mathcal{C}$ (*Dec G u*) = *Dec* $\mathcal{C}$ *u*
using *dec-in-lists*[*OF* ‹$u \in \langle G \rangle$›] ‹$G \subseteq \langle \mathcal{C} \rangle$› *code-unique-ref*[*of Dec G u*, *unfolded concat-dec*[*OF* ‹$u \in \langle G \rangle$›]] **by** *blast*
from *ref-gen*[*of Dec G u G*, *OF dec-in-lists*[*OF* ‹$u \in \langle G \rangle$›], *of* $\mathcal{C}$, *unfolded this*, *OF* ‹$G \subseteq \langle \mathcal{C} \rangle$›]
show *?thesis***.**
qed

find-theorems ϱ *?x* $^{@}$ *?k* = *?x 0* < *?k*

lemma *code-rev-code*: *code* (*rev* ‘ $\mathcal{C}$)
proof
fix *xs ys* **assume** *xs* ∈ *lists* (*rev* ‘ $\mathcal{C}$) *ys* ∈ *lists* (*rev* ‘ $\mathcal{C}$) *concat xs* = *concat ys*
have *map rev* (*rev xs*) ∈ *lists* $\mathcal{C}$ **and** *map rev* (*rev ys*) ∈ *lists* $\mathcal{C}$
using *rev-in-lists*[*OF* ‹*xs* ∈ *lists* (*rev* ‘ $\mathcal{C}$)›] *rev-in-lists*[*OF* ‹*ys* ∈ *lists* (*rev* ‘ $\mathcal{C}$)›] *map-rev-lists-rev*
by (*metis imageI*)+
moreover have *concat* (*map rev* (*rev xs*)) = *concat* (*map rev* (*rev ys*))
unfolding *rev-concat*[*symmetric*] **using** ‹*concat xs* = *concat ys*› **by** *blast*
ultimately have *map rev* (*rev xs*) = *map rev* (*rev ys*)
using *is-code* **by** *blast*
thus *xs* = *ys*
using ‹*concat xs* = *concat ys*› **by** *simp*
qed

lemma *dec-rev* [*simp*, *reversal-rule*]:
u ∈ ⟨$\mathcal{C}$⟩ ⟹ *Dec rev* ‘ $\mathcal{C}$ (*rev u*) = *rev* (*map rev* (*Dec* $\mathcal{C}$ *u*))

by (*auto simp only*: *rev-map lists-image rev-in-lists rev-concat*[*symmetric*] *dec-in-lists intro*!: *code-rev-code code.code-unique-dec imageI del*: *in-listsI*)

lemma *elem-comm-sing-set*: **assumes** $ws \in lists\ \mathcal{C}$ **and** $ws \neq \varepsilon$ **and** $u \in \mathcal{C}$ **and** $concat\ ws \cdot u = u \cdot concat\ ws$
shows $set\ ws = \{u\}$
using *assms*
proof−
have $concat\ (ws \cdot [u]) = concat\ ([u] \cdot ws)$
using *assms* **by** *simp*
have $ws \cdot [u] = [u] \cdot ws$
using ‹$u \in \mathcal{C}$› ‹$ws \in lists\ \mathcal{C}$› *is-code*[*OF - -* ‹$concat\ (ws \cdot [u]) = concat\ ([u] \cdot ws)$›]
by *simp*
from *comm-nemp-pows-posE*[*OF this* ‹$ws \neq \varepsilon$› *not-Cons-self2*[*of u* ε]]
obtain $t\ k\ m$ **where** $ws = t^{@}k\ [u] = t^{@}m\ 0 < k\ 0 < m$ *primitive t*.
hence $t = [u]$
by *force*
show $set\ ws = \{u\}$
using ‹$ws = t^{@}k$›[*unfolded* ‹$t = [u]$›] *set-sing-nemp-eq*[*OF sing-pow-set-sub* ‹$ws \neq \varepsilon$›] **by** *blast*
qed

lemma *pure-code-pres-prim*: **assumes** *pure*: $\forall u \in \langle\mathcal{C}\rangle.\ \varrho\ u \in \langle\mathcal{C}\rangle$ **and** $w \in \langle\mathcal{C}\rangle$ **and** $primitive\ (Dec\ \mathcal{C}\ w)$
shows $primitive\ w$
proof−
obtain k **where** $(\varrho\ w)^{@}k = w$
using *primroot-expE* **by** *blast*

have $\varrho\ w \in \langle\mathcal{C}\rangle$
using *assms*(*2*) *pure* **by** *auto*

have $(Dec\ \mathcal{C}\ (\varrho\ w))^{@}k \in lists\ \mathcal{C}$
by (*metis* ‹$\varrho\ w \in \langle\mathcal{C}\rangle$› *concat-pow-list-single dec-in-lists flatten-lists order-refl sing-pow-lists*)

have $(Dec\ \mathcal{C}\ (\varrho\ w))^{@}k = Dec\ \mathcal{C}\ w$
using ‹$(Dec\ \mathcal{C}\ (\varrho\ w))^{@}k \in lists\ \mathcal{C}$› *code.code-unique-dec code-axioms concat-morph-power* ‹$(\varrho\ w)^{@}k = w$› *concat-dec*[*OF* ‹$\varrho\ w \in \langle\mathcal{C}\rangle$›] **by** *metis*
hence $k = 1$
using ‹$primitive\ (Dec\ \mathcal{C}\ w)$› **unfolding** *primitive-def* **by** *blast*
thus $primitive\ w$
by (*metis pow-list-1* ‹$\varrho\ w^{@}k = w$› *assms*(*3*) *dec-emp prim-nemp primroot-prim*)
qed

lemma *inj-on-dec*: *inj-on* (*decompose* $\mathcal{C}$) $\langle\mathcal{C}\rangle$
by (*rule inj-on-inverseI*[*of - concat*]) *simp*

lemma *ref-disj-interp*: **assumes** *vs* ∈ *lists* ⟨*C*⟩ *p Ref C vs s* ∼$_D$ *ws*
shows *p vs s* ∼$_D$ *ws*
proof(*rule disj-interpI*)
show *p* (*concat vs*) *s* ∼$_I$ *ws*
using *disj-interpD0*[*OF* ‹*p Ref C vs s* ∼$_D$ *ws*›]
unfolding *concat-ref*[*OF* ‹*vs* ∈ *lists* ⟨*C*⟩›].
show ∀ *u v*. *u* ≤*p vs* ∧ *v* ≤*p ws* ⟶ *p* · *concat u* ≠ *concat v*
proof (*rule allI*, *rule allI*, *rule impI*, *elim conjE*)
fix *u v* **assume** *u* ≤*p vs v* ≤*p ws*
have *Ref C u* ≤*p Ref C vs*
using *ref-pref-mono*[*OF* ‹*vs* ∈ *lists* ⟨*C*⟩› ‹*u* ≤*p vs*›].
have *concat* (*Ref C u*) = *concat u*
using *concat-ref*[*OF pref-in-lists*[*OF* ‹*u* ≤*p vs*› ‹*vs* ∈ *lists* ⟨*C*⟩›]].
then show *p* · *concat u* ≠ *concat v*
using *disj-interpD1*[*OF* ‹*p Ref C vs s* ∼$_D$ *ws*› ‹*Ref C u* ≤*p Ref C vs*› ‹*v* ≤*p ws*›]
by *simp*
qed
qed

end — end context code

lemma *emp-is-code*: *code* {}
using *code.intro empty-iff insert-iff lists-empty* **by** *metis*

lemma *code-rev-code-iff* [*reversal-rule*]: *code* (*rev* ‘ *C*) ⟷ *code C*
by (*rule iffI*[*OF code.code-rev-code*[*of rev* ‘ *C*, *unfolded rev-rev-image-eq*] *code.code-rev-code*])

lemma *code-induct-hd*: **assumes** *ε* ∉ *C* **and**
⋀ *xs ys*. *xs* ∈ *lists C* ⟹ *ys* ∈ *lists C* ⟹ *concat xs* = *concat ys* ⟹ *hd xs* = *hd ys*
shows *code C*
proof
show *xs* ∈ *lists C* ⟹ *ys* ∈ *lists C* ⟹ *concat xs* = *concat ys* ⟹ *xs* = *ys* **for** *xs ys*
proof (*induct xs ys rule*: *list-induct2′*)
case (*4 x xs y ys*)
from *assms*(*2*)[*OF 4.prems*]
have *x* = *y* **by** *force*
from *4.prems*[*unfolded this*]
have *xs* ∈ *lists C* **and** *ys* ∈ *lists C* **and** *concat xs* = *concat ys*
by *simp-all*
from *4.hyps*[*OF this*] ‹*x* = *y*›
show *?case*
by *simp*
qed (*auto simp add*: ‹*ε* ∉ *C*›)
qed

lemma *ref-set-primroot*: **assumes** *ws* ∈ *lists* (*G* − {*ε*}) **and** *code* (*ϱ*‘*G*)

```
  shows set (Ref ϱ‘G ws) = ϱ‘(set ws)
proof–
  have G ⊆ ⟨ϱ‘G⟩
  proof
    fix x
    assume x ∈ G
    show x ∈ ⟨ϱ ‘ G⟩
      by (metis ‹x ∈ G› genset-sub image-subset-iff power-in primroot-expE)
  qed
  hence ws ∈ lists ⟨ϱ‘G⟩
    using assms by blast

  have set (decompose (ϱ‘G) a) = {ϱ a} if a ∈ set ws for a
  proof–
    have ϱ a ∈ ϱ‘G a ≠ ε
      using ‹a ∈ set ws› ‹ws ∈ lists (G − {ε})› by force+
    obtain k where (Dec (ϱ‘G) a) = [ϱ a]^@k 0 < k
    using code.code-unique-dec[OF ‹code (ϱ ‘ G)› sing-pow-lists concat-pow-list-single,
OF ‹ϱ a ∈ ϱ ‘ G›]
       primroot-expE by metis
    hence Dec (ϱ‘G) a ≠ ε
      by simp
    from set-sing-nemp-eq[OF - this]
    show set (decompose (ϱ‘G) a) = {ϱ a}
      unfolding ‹Dec ϱ ‘ G a = [ϱ a] ^@ k› using sing-pow-set' by metis
  qed

  have (set‘(decompose (ϱ‘G))‘set ws) = {{ϱ a} |a. a ∈ set ws} (is ?L = ?R)
  proof
    show ?L ⊆ ?R
      using ‹⋀a. a ∈ set ws ⟹ set (Dec ϱ ‘ G a) = {ϱ a}› by blast
    show ?R ⊆ ?L
      using ‹⋀a. a ∈ set ws ⟹ set (Dec ϱ ‘ G a) = {ϱ a}› by blast
  qed

  show ?thesis
    using ref-set[OF ‹ws ∈ lists ⟨ϱ‘G⟩›]
     Setcompr-eq-image ‹set ‘ decompose (ϱ ‘ G) ‘ set ws = {{ϱ a} |a. a ∈ set ws}›
by (auto simp add: refine-def)
qed
```

3.5 Prefix code

```
locale prefix-code =
  fixes C
  assumes
    emp-not-in: ε ∉ C and
    pref-free: u ∈ C ⟹ v ∈ C ⟹ u ≤p v ⟹ u = v
```

begin

lemma *nemp*: $u \in \mathcal{C} \Longrightarrow u \neq \varepsilon$
using *emp-not-in* **by** *force*

lemma *concat-pref-concat*:
assumes $us \in lists\ \mathcal{C}$ $vs \in lists\ \mathcal{C}$ $concat\ us \leq p\ concat\ vs$
shows $us \leq p\ vs$
using *assms* **proof** (*induction us vs rule*: *list-induct2'*)
case (*4 x xs y ys*)
from *4.prems*
have $x = y$
by (*auto elim*!: *ruler-prefE intro*: *pref-free sym del*: *in-listsD*)
with *4* **show** $x \# xs \leq p\ y \# ys$
by *simp*
qed (*simp-all add*: *nemp*)

lemma *concat-pref-concat-conv*:
assumes $us \in lists\ \mathcal{C}$ $vs \in lists\ \mathcal{C}$
shows $concat\ us \leq p\ concat\ vs \longleftrightarrow us \leq p\ vs$
using *concat-pref-concat*[*OF assms*] *pref-concat-pref*..

sublocale *code*
by *standard* (*simp-all add*: *pref-antisym concat-pref-concat*)

lemmas *is-code* = *is-code* **and**
code = *code-axioms*

lemma *dec-pref-unique*:
$w \in \langle \mathcal{C} \rangle \Longrightarrow p \in \langle \mathcal{C} \rangle \Longrightarrow p \leq p\ w \Longrightarrow Dec\ \mathcal{C}\ p \leq p\ Dec\ \mathcal{C}\ w$
using *concat-pref-concat-conv*[*of Dec* $\mathcal{C}$ *p Dec* $\mathcal{C}$ *w*, *THEN iffD1*]
by *simp*

lemma *concat-suf-eq*: **assumes**
$us \in lists\ \mathcal{C}$ $ws \in lists\ \mathcal{C}$ **and**
$concat\ us \cdot s = concat\ ws$ **and** $s <s last\ ws$
shows $us = ws$ **and** $s = \varepsilon$
proof−
from *concat-pref-concat-conv*[*OF assms*(*1,2*), *folded* ‹$concat\ us \cdot s = concat\ ws$›]
have $us \leq p\ ws$
by *blast*
hence $concat\ (us^{-1>}ws) = s$
using *assms*(*3*) *concat-morph-lq lq-triv* **by** *metis*
from *concat-butlast-last*[*of* $us^{-1>}ws$, *unfolded this*]
show $us = ws$
using *empty-lq-eq*[*OF* ‹$us \leq p\ ws$›] *assms*(*4*) *last-appendR lq-pref*[*OF* ‹$us \leq p$ *ws*›]
ssuf-extD suffix-order.strict-iff-not **by** *metis*
show $s = \varepsilon$

using ‹*concat us* · *s* = *concat ws*›[*unfolded* ‹*us* = *ws*›] **by** *blast*
qed

end

thm *prefix-code.concat-suf-eq*[*reversed*]

3.5.1 Suffix code

locale *suffix-code* = *prefix-code* (*rev* ‘ $\mathcal{C}$) **for** $\mathcal{C}$
begin

thm *dec-rev*
code

sublocale *code*
using *code-rev-code* **unfolding** *rev-rev-image-eq*.

lemmas *concat-suf-concat* = *concat-pref-concat*[*reversed*] **and**
concat-suf-concat-conv = *concat-pref-concat-conv*[*reversed*] **and**
nemp = *nemp*[*reversed*] **and**
suf-free = *pref-free*[*reversed*] **and**
dec-suf-unique = *dec-pref-unique*[*reversed*]

lemma *concat-pref-eq*: **assumes**
us ∈ *lists* $\mathcal{C}$ *ws* ∈ *lists* $\mathcal{C}$ **and**
p · *concat us* = *concat ws* **and** *p* <*p hd ws*
shows *us* = *ws* **and** *p* = ε
proof (*atomize*(*full*))
show *us* = *ws* ∧ *p* = ε
proof (*cases ws* = ε)
assume *ws* = ε
hence *p* · *concat us* = ε
unfolding ‹*p* · *concat us* = *concat ws*› **by** *simp*
thus *us* = *ws* ∧ *p* = ε
unfolding ‹*ws* = ε› **using** ‹*us* ∈ *lists* $\mathcal{C}$›
using *concat-emp*[*OF emp-not-in* ‹*us* ∈ *lists* $\mathcal{C}$›] **by** *blast*
next
assume *ws* ≠ ε
from *concat-suf-eq*[*reversed*, *OF assms*(*1*−*3*)]
show *us* = *ws* ∧ *p* = ε
unfolding *hd-map*[*OF* ‹*ws* ≠ ε›] *spref-rev-suf-iff*[*symmetric*]
using ‹*p* <*p hd ws*› **by** *blast*
qed
qed

thm *is-code*
code-axioms
code

end

3.5.2 Bifix code

locale *bifix-code* = *prefix-code* + *suf*: *suffix-code*
begin

lemma *joint-interp-triv*: **assumes**
us ∈ *lists* $\mathcal{C}$ *ws* ∈ *lists* $\mathcal{C}$ **and**
interp: *p* (*concat us*) *s* $\sim_{\mathcal{I}}$ *ws* **and**
joint: ¬ *p us s* $\sim_{\mathcal{D}}$ *ws*
shows *p* = ε **and** *s* = ε **and** *us* = *ws*
proof−
from *non-disjoint-interpE*[*OF interp joint*]
obtain *ws1 ws2 us1 us2* **where** *us1* · *us2* = *us ws1* · *ws2* = *ws*
p · *concat us1* = *concat ws1 concat us2* · *s* = *concat ws2*.
hence *ws1* ∈ *lists* $\mathcal{C}$ *ws2* ∈ *lists* $\mathcal{C}$ *us1* ∈ *lists* $\mathcal{C}$ *us2* ∈ *lists* $\mathcal{C}$
using ‹*us* ∈ *lists* $\mathcal{C}$› ‹*ws* ∈ *lists* $\mathcal{C}$› **by** *inlists*
have *eq1*: *us2* = *ws2* ∧ *s* = ε
proof (*cases ws2* = ε)
assume *ws2* = ε
show *?thesis*
thm *emp-concat-emp′*
using ‹*concat us2* · *s* = *concat ws2*› *emp-concat-emp′*[*OF* ‹*us2* ∈ *lists* $\mathcal{C}$›]
unfolding ‹*ws2* = ε›
concat.simps **by** *blast*
next
assume *ws2* ≠ ε
with *concat-suf-eq*[*OF* ‹*us2* ∈ *lists* $\mathcal{C}$› ‹*ws2* ∈ *lists* $\mathcal{C}$› ‹*concat us2* · *s* = *concat ws2*›]
show *?thesis*
using *fac-interpD*(*2*)[*OF interp*, *folded* ‹*ws1* · *ws2* = *ws*›] **by** *force*
qed
have *eq2*: *us1* = *ws1* ∧ *p* = ε
proof (*cases ws1* = ε)
assume *ws1* = ε
show *?thesis*
using ‹*p* · *concat us1* = *concat ws1*› *emp-concat-emp′*[*OF* ‹*us1* ∈ *lists* $\mathcal{C}$›]
unfolding ‹*ws1* = ε›
concat.simps **by** *blast*
next
assume *ws1* ≠ ε
with *suf.concat-pref-eq*[*OF* ‹*us1* ∈ *lists* $\mathcal{C}$› ‹*ws1* ∈ *lists* $\mathcal{C}$› ‹*p* · *concat us1* = *concat ws1*›]
show *?thesis*
using *fac-interpD*(*1*)[*OF interp*, *folded* ‹*ws1* · *ws2* = *ws*›] **by** *force*
qed
show *p* = ε **and** *s* = ε **and** *us* = *ws*

using *eq1 eq2* ‹*us1* · *us2* = *us*› ‹*ws1* · *ws2* = *ws*› **by** *blast+*
qed

end

3.6 Marked code

locale *marked-code* =
fixes $\mathcal{C}$
assumes
emp-not-in: $\varepsilon \notin \mathcal{C}$ **and**
marked: $u \in \mathcal{C} \Longrightarrow v \in \mathcal{C} \Longrightarrow hd\ u = hd\ v \Longrightarrow u = v$

begin

lemma *nemp*: $u \in \mathcal{C} \Longrightarrow u \neq \varepsilon$
using *emp-not-in* **by** *blast*

sublocale *prefix-code*
by (*unfold-locales*) (*simp-all add*: *emp-not-in marked nemp pref-hd-eq*)

lemma *marked-concat-lcp*: $us \in lists\ \mathcal{C} \Longrightarrow vs \in lists\ \mathcal{C} \Longrightarrow concat\ (us \wedge_p vs) = (concat\ us) \wedge_p (concat\ vs)$
proof (*induct us vs rule*: *list-induct2'*)
case (*4 x xs y ys*)
hence $x \in \mathcal{C}$ **and** $y \in \mathcal{C}$ **and** $xs \in lists\ \mathcal{C}$ **and** $ys \in lists\ \mathcal{C}$
by *simp-all*
show *?case*
proof (*cases*)
assume $x = y$
thus $concat\ (x \# xs \wedge_p y \# ys) = concat\ (x \# xs) \wedge_p concat\ (y \# ys)$
using *4.hyps*[*OF* ‹$xs \in lists\ \mathcal{C}$› ‹$ys \in lists\ \mathcal{C}$›] **by** (*simp add*: *lcp-ext-left*)
next
assume $x \neq y$
with *marked*[*OF* ‹$x \in \mathcal{C}$› ‹$y \in \mathcal{C}$›] **have** $hd\ x \neq hd\ y$ **by** *blast*
hence $concat\ (x \# xs) \wedge_p concat\ (y \# ys) = \varepsilon$
by (*simp add*: ‹$x \in \mathcal{C}$› ‹$y \in \mathcal{C}$› *nemp lcp-distinct-hd*)
moreover have $concat\ (x \# xs \wedge_p y \# ys) = \varepsilon$
using ‹$x \neq y$› **by** *simp*
ultimately show *?thesis* **by** *presburger*
qed
qed *simp-all*

lemma *hd-concat-hd*: **assumes** $xs \in lists\ \mathcal{C}$ **and** $ys \in lists\ \mathcal{C}$ **and** $xs \neq \varepsilon$ **and** $ys \neq \varepsilon$ **and**
$hd\ (concat\ xs) = hd\ (concat\ ys)$
shows $hd\ xs = hd\ ys$
proof–

have *hd* (*hd xs*) = *hd* (*hd ys*)
using *assms hd-concat*[*OF* ‹*xs* ≠ *ε*› *lists-hd-in-set*[*THEN nemp*]] *hd-concat*[*OF* ‹*ys* ≠ *ε*› *lists-hd-in-set*[*THEN nemp*]]
by *presburger*

from *marked*[*OF lists-hd-in-set lists-hd-in-set this*] *assms*(*1−4*)
show *hd xs* = *hd ys*
by *simp*
qed

end

3.7 Non-overlapping code

locale *non-overlapping* =
fixes $\mathcal{C}$
assumes
emp-not-in: $\varepsilon \notin \mathcal{C}$ **and**
no-overlap: $u \in \mathcal{C} \Longrightarrow v \in \mathcal{C} \Longrightarrow z \leq p\ u \Longrightarrow z \leq s\ v \Longrightarrow z \neq \varepsilon \Longrightarrow u = v$ **and**
no-fac: $u \in \mathcal{C} \Longrightarrow v \in \mathcal{C} \Longrightarrow u \leq f\ v \Longrightarrow\ u = v$
begin

lemma *nemp*: $u \in \mathcal{C}\ \Longrightarrow u \neq \varepsilon$
using *emp-not-in* **by** *force*

sublocale *prefix-code*
using *nemp non-overlapping.no-fac non-overlapping-axioms prefix-code.intro prefix-imp-sublist* **by** *metis*

lemma *rev-non-overlapping*: *non-overlapping* (*rev* ‘ $\mathcal{C}$)
proof
show $\varepsilon \notin$ *rev* ‘ $\mathcal{C}$
using *nemp* **by** *force*
show $u \in$ *rev* ‘ $\mathcal{C} \Longrightarrow v \in$ *rev* ‘ $\mathcal{C} \Longrightarrow z \leq p\ u \Longrightarrow z \leq s\ v \Longrightarrow z \neq \varepsilon \Longrightarrow u = v$
for *u v z*
using *no-overlap*[*reversed*] **unfolding** *rev-in-conv*..
show $u \in$ *rev* ‘ $\mathcal{C} \Longrightarrow v \in$ *rev* ‘ $\mathcal{C} \Longrightarrow u \leq f\ v \Longrightarrow u = v$ **for** *u v*
using *no-fac*[*reversed*] **unfolding** *rev-in-conv* **by** *presburger*
qed

sublocale *suf*: *suffix-code* $\mathcal{C}$
proof−
interpret *i*: *non-overlapping rev* ‘ $\mathcal{C}$
using *rev-non-overlapping*.
from *i.prefix-code-axioms*
show *suffix-code* $\mathcal{C}$
by *unfold-locales*
qed

lemma *overlap-concat-last*: **assumes** $u \in \mathcal{C}$ **and** $vs \in lists\ \mathcal{C}$ **and** $vs \neq \varepsilon$ **and** $r \neq \varepsilon$ **and** $r \leq p\ u$ **and** $r \leq s\ p \cdot concat\ vs$
shows $u = last\ vs$
proof–
from *suffix-same-cases*[*OF suf-ext*[*OF concat-last-suf*[*OF* ‹$vs \neq \varepsilon$›]] ‹$r \leq s\ p \cdot concat\ vs$›]
show $u = last\ vs$
proof (*rule disjE*)
assume $r \leq s\ last\ vs$
from *no-overlap*[*OF* ‹$u \in \mathcal{C}$› - ‹$r \leq p\ u$› *this* ‹$r \neq \varepsilon$›]
show $u = last\ vs$
using ‹$vs \in lists\ \mathcal{C}$› ‹$vs \neq \varepsilon$› **by** *force*
next
assume $last\ vs \leq s\ r$
from *no-fac*[*OF* - ‹$u \in \mathcal{C}$› *pref-suf-fac*, *OF* - ‹$r \leq p\ u$› *this*]
show $u = last\ vs$
using ‹$vs \in lists\ \mathcal{C}$› ‹$vs \neq \varepsilon$› **by** *force*
qed
qed

lemma *overlap-concat-hd*: **assumes** $u \in \mathcal{C}$ **and** $vs \in lists\ \mathcal{C}$ **and** $vs \neq \varepsilon$ **and** $r \neq \varepsilon$ **and** $r \leq s\ u$ **and** $r \leq p\ concat\ vs \cdot s$
shows $u = hd\ vs$
proof–
interpret *c*: *non-overlapping rev* ‘ $\mathcal{C}$ **by** (*simp add*: *rev-non-overlapping*)
from *c.overlap-concat-last*[*reversed*, *OF assms*]
show *?thesis*.
qed

lemma *fac-concat-fac*:
assumes $us \in lists\ \mathcal{C}$ $vs \in lists\ \mathcal{C}$
and $1 < card\ (set\ us)$
and $concat\ vs = p \cdot concat\ us \cdot s$
obtains *ps ss* **where** $concat\ ps = p$ **and** $concat\ ss = s$ **and** $ps \cdot us \cdot ss = vs$
proof–
have $us \neq \varepsilon$
using ‹$1 < card\ (set\ us)$› **by** *fastforce*
let *?a* = $hd\ us$
define *us1* **where** $us1 = takeWhile\ (\lambda\ b.\ b = ?a)\ us$
define *us2* **where** $us2 = dropWhile\ (\lambda\ b.\ b = ?a)\ us$
define *k* **where** $k = |us1|$
have $us1 \neq \varepsilon$
unfolding *us1-def takeWhile-eq-Nil-iff* **using** ‹$us \neq \varepsilon$› **by** *blast*
note *nemp-len*[*OF this*, *folded k-def*]
have $us = us1 \cdot us2$
unfolding *us1-def us2-def* **by** *simp*
have $set\ us1 = \{?a\}$
using *set-sing-nemp-eq takeWhile-subset* ‹$us1 \neq \varepsilon$› **unfolding** *us1-def* **by** *metis*
hence $us1 = [?a]^{@}k$

using *sing-pow-exp* **unfolding** *k-def* **by** *fastforce*
have *last us1 = ?a*
unfolding ‹*us1 = [?a]*$^{@}$*k*›[*unfolded pow-pos2*[*OF* ‹*0 < k*›]] **using** *last-snoc*.
have *us2 ≠ ε*
using ‹*1 < card* (*set us*)›[*unfolded* ‹*us = us1 · us2*›] ‹*set us1 = {?a}*› **by** *force*
have *hd us2 ≠ ?a*
using *hd-dropWhile*[*OF* ‹*us2 ≠ ε*›[*unfolded us2-def*]] **unfolding** *us2-def*.
note *this*[*symmetric*, *folded* ‹*last us1 = ?a*›]

have *us2 ∈ lists C us1 ∈ lists C*
using ‹*us = us1 · us2*› ‹*us ∈ lists C*› **by** *simp-all*
hence *concat us2 ≠ ε*
using ‹*us2 ≠ ε*› *nemp* **by** *force*
hence *p · concat us1 <p concat vs*
using ‹*us = us1 · us2*› **unfolding** ‹*concat vs = p · concat us · s*› **by** *simp*
from *pref-mod-list'*[*OF this*]
obtain *j r* **where** *j < |vs| r <p vs ! j concat* (*take j vs*) *· r = p · concat us1*.
have *r = ε*
proof (*rule ccontr*)
assume *r ≠ ε*
from *spref-exE*[*OF* ‹*r <p vs ! j*›]
obtain *z* **where** *r · z = vs ! j z ≠ ε*.
from *overlap-concat-last*[*OF* - ‹*us1 ∈ lists C*› ‹*us1 ≠ ε*› ‹*r ≠ ε*› *sprefD1*[*OF* ‹*r <p vs ! j*›] *sufI*[*OF* ‹*concat* (*take j vs*) *· r = p · concat us1*›]]
have *vs ! j = last us1*
using *nth-in-lists*[*OF* ‹*j < |vs|*› ‹*vs ∈ lists C*›].

have *concat-vs*: *concat vs = concat* (*take j vs*) *· vs*!*j · concat* (*drop* (*Suc j*) *vs*)
unfolding *lassoc concat-take-Suc*[*OF* ‹*j < |vs|*›] *concat-morph*[*symmetric*] **by** *force*
from *this*[*folded* ‹*r · z = vs ! j*›]
have *z · concat* (*drop* (*Suc j*) *vs*) = *concat us2 · s*
unfolding ‹*concat vs = p · concat us · s*› *lassoc* ‹*concat* (*take j vs*) *· r = p · concat us1*› ‹*us = us1 · us2*› *concat-morph*
unfolding *rassoc cancel* **by** *simp*
from *overlap-concat-hd*[*OF* - ‹*us2 ∈ lists C*› ‹*us2 ≠ ε*› ‹*z ≠ ε*› *sufI*[*OF* ‹*r · z = vs ! j*›] *prefI*[*OF this*]]
have *vs ! j = hd us2*
using *nth-in-lists*[*OF* ‹*j < |vs|*› ‹*vs ∈ lists C*›].

thus *False*
unfolding ‹*vs ! j = last us1*› **using** ‹*last us1 ≠ hd us2*› **by** *contradiction*
qed

have *drop j vs ∈ lists C* **and** *take j vs ∈ lists C*
using ‹*vs ∈ lists C*› **by** *inlists*
have *concat us2 · s = concat* (*drop j vs*)
using *arg-cong*[*OF takedrop*[*of j vs*], *of concat*] ‹*concat* (*take j vs*) *· r = p · concat us1*›

unfolding ‹*concat vs* = *p* · *concat us* · *s*› *concat-morph* ‹*r* = ε› *emp-simps* ‹*us* = *us1* · *us2*› **by** *auto*
from *prefI*[*OF this*]
have *us2* ≤*p drop j vs*
using *concat-pref-concat-conv*[*OF* ‹*us2* ∈ *lists* C› ‹*drop j vs* ∈ *lists* C›] **by** *blast*
hence *s*: *concat* (*us2*$^{-1>}$ *drop j vs*) = *s*
using ‹*concat us2* · *s* = *concat* (*drop j vs*)› *concat-morph-lq lqI* **by** *blast*

from ‹*concat* (*take j vs*) · *r* = *p* · *concat us1*›[*unfolded* ‹*r* = ε› *emp-simps*]
have *concat us1* ≤*s concat* (*take j vs*)
by *fastforce*
hence *us1* ≤*s take j vs*
using *suf.concat-pref-concat-conv*[*reversed*, *OF* ‹*us1* ∈ *lists* C› ‹*take j vs* ∈ *lists* C›] **by** *blast*
from *arg-cong*[*OF rq-suf*[*OF this*], *of concat*, *unfolded concat-morph*]
have *p*: *concat* (*take j vs*$^{<-1}$*us1*) = *p*
using *rqI*[*OF* ‹*concat* (*take j vs*) = *p* · *concat us1*›[*symmetric*]]
rq-triv **by** *metis*

have *take j vs*$^{<-1}$*us1* · *us* · *us2*$^{-1>}$*drop j vs* = *vs*
unfolding ‹*us* = *us1* · *us2*› *rassoc lq-pref*[*OF* ‹*us2* ≤*p drop j vs*›]
unfolding *lassoc rq-suf*[*OF* ‹*us1* ≤*s take j vs*›] **by** *simp*

from *that*[*OF p s this*]
show *thesis*.
qed

theorem *prim-morph*:
assumes *ws* ∈ *lists* C
and |*ws*| ≠ *1*
and *primitive ws*
shows *primitive* (*concat ws*)
proof (*rule ccontr*)
have *ws* ∈ *lists* C **and** *ws* · *ws* ∈ *lists* C
using ‹*ws* ∈ *lists* C› **by** *simp-all*
moreover have *1* < *card* (*set ws*) **using** ‹*primitive ws*› ‹|*ws*| ≠ *1*›
by (*rule prim-card-set*)
moreover assume ¬ *primitive* (*concat ws*)
then obtain *k z* **where** *2* ≤ *k* **and** *z* ^@ *k* = *concat ws*
by (*elim not-prim-primroot-expE*)
have *concat* (*ws* · *ws*) = *z* · *concat ws* · *z*^@(*k*−*1*)
unfolding *concat-morph* ‹*z* ^@ *k* = *concat ws*›[*symmetric*] *pow-add*[*symmetric*] *pow-Suc*[*symmetric*]
using ‹*2* ≤ *k*› **by** *simp*
ultimately obtain *ps ss* **where** *concat ps* = *z* **and** *concat ss* = *z*^@(*k*−*1*) **and** *ps* · *ws* · *ss* = *ws* · *ws*
by (*rule fac-concat-fac*)
have *ps* ^@ *k* ∈ *lists* C
using ‹*ps* · *ws* · *ss* = *ws* · *ws*› ‹*ws* · *ws* ∈ *lists* C› **by** *inlists*

moreover have *concat* (*ps* [@] *k*) = *concat ws*
unfolding *concat-pow-list* ‹*concat ps* = *z*› ‹*z* [@] *k* = *concat ws*›**..**
ultimately have *ps* [@] *k* = *ws* **using** ‹*ws* ∈ *lists* C› **by** (*intro is-code*)
show *False*
using *prim-exp-one*[*OF* ‹*primitive ws*› ‹*ps* [@] *k* = *ws*›] ‹2 ≤ *k*› **by** *presburger*
qed

lemma *prim-concat-conv*:
assumes *ws* ∈ *lists* C
and |*ws*| ≠ 1
shows *primitive* (*concat ws*) ⟷ *primitive ws*
using *prim-concat-prim prim-morph*[*OF assms*]**..**

end

3.8 Binary code

We pay a special attention to two element codes. In particular, we show that two words form a code if and only if they do not commute. This means that two words either commute, or do not satisfy any nontrivial relation.

definition *bin-lcp* **where** *bin-lcp x y* = *x*·*y* ∧$_p$ *y*·*x*
definition *bin-lcs* **where** *bin-lcs x y* = *x*·*y* ∧$_s$ *y*·*x*

definition *bin-mismatch* **where** *bin-mismatch x y* = (*x*·*y*)!|*bin-lcp x y*|
definition *bin-mismatch-suf* **where** *bin-mismatch-suf x y* = *bin-mismatch* (*rev y*) (*rev x*)

value[*nbe*] [*0*::*nat*,*1*,*0*]!*3*

lemma *bin-lcs-rev*: *bin-lcs x y* = *rev* (*bin-lcp* (*rev x*) (*rev y*))
unfolding *bin-lcp-def bin-lcs-def longest-common-suffix-def rev-append* **using** *lcp-sym* **by** *fastforce*

lemma *bin-lcp-sym*: *bin-lcp x y* = *bin-lcp y x*
unfolding *bin-lcp-def* **using** *lcp-sym***.**

lemma *bin-mismatch-comm*: (*bin-mismatch x y* = *bin-mismatch y x*) ⟷ (*x* · *y* = *y* · *x*)
unfolding *bin-mismatch-def bin-lcp-def lcp-sym*[*of y* · *x*]
using *lcp-mismatch'*[*of x* · *y y* · *x*, *unfolded comm-comp-eq-conv*[*of x y*]] **by** *fastforce*

lemma *bin-lcp-pref-fst-snd*: *bin-lcp x y* ≤*p x* · *y*
unfolding *bin-lcp-def* **using** *lcp-pref***.**

lemma *bin-lcp-pref-snd-fst*: *bin-lcp x y* ≤*p y* · *x*
using *bin-lcp-pref-fst-snd*[*of y x*, *unfolded bin-lcp-sym*[*of y x*]]**.**

lemma *bin-lcp-bin-lcs* [*reversal-rule*]: *bin-lcp* (*rev x*) (*rev y*) = *rev* (*bin-lcs x y*)
unfolding *bin-lcp-def bin-lcs-def rev-append*[*symmetric*] *lcs-lcp*
lcs-sym[*of x · y*]**..**

lemmas *bin-lcs-sym* = *bin-lcp-sym*[*reversed*]

lemma *bin-lcp-len*: $x \cdot y \neq y \cdot x \Longrightarrow |$*bin-lcp x y*$| < |x \cdot y|$
unfolding *bin-lcp-def*
using *lcp-len′ pref-comm-eq* **by** *blast*

lemmas *bin-lcs-len* = *bin-lcp-len*[*reversed*]

lemma *bin-mismatch-pref-suf′*[*reversal-rule*]:
bin-mismatch (*rev y*) (*rev x*) = *bin-mismatch-suf x y*
unfolding *bin-mismatch-suf-def***..**

3.8.1 Binary code locale

locale *binary-code* =
fixes u_0 u_1
assumes *non-comm*: $u_0 \cdot u_1 \neq u_1 \cdot u_0$

begin

A crucial property of two element codes is the constant decoding delay given by the word α, which is a prefix of any generating word (sufficiently long), while the letter immediately after this common prefix indicates the first element of the decomposition.

definition *uu* **where** *uu a* = (*if a then* u_0 *else* u_1)

lemma *bin-code-set-bool*: {*uu a*,*uu* (¬ *a*)} = {u_0,u_1}
by (*induct a*, *unfold uu-def*, *simp-all add*: *insert-commute*)

lemma *bin-code-set-bool′*: {*uu a*,*uu* (¬ *a*)} = {u_1,u_0}
by (*induct a*, *unfold uu-def*, *simp-all add*: *insert-commute*)

lemma *bin-code-swap*: *binary-code* u_1 u_0
using *binary-code.intro*[*OF non-comm*[*symmetric*]]**.**

lemma *bin-code-bool*: *binary-code* (*uu a*) (*uu* (¬ *a*))
unfolding *uu-def* **by** (*induct a*, *simp-all add*: *bin-code-swap binary-code-axioms*)

lemma *bin-code-neq*: $u_0 \neq u_1$
using *non-comm* **by** *auto*

lemma *bin-code-neq-bool*: *uu a* ≠ *uu* (¬ *a*)
unfolding *uu-def* **by** (*induct a*) (*use bin-code-neq* **in** *fastforce*)+

lemma *bin-fst-nemp*: $u_0 \neq \varepsilon$ **and** *bin-snd-nemp*: $u_1 \neq \varepsilon$ **and** *bin-nemp-bool*: *uu a*

$\neq \varepsilon$
using *non-comm uu-def* **by** *auto*

lemma *bin-not-comp*: $\neg\ u_0 \cdot u_1 \bowtie u_1 \cdot u_0$
using *comm-comp-eq-conv non-comm* **by** *blast*

lemma *bin-not-comp-bool*: $\neg\ (uu\ a \cdot uu\ (\neg\ a) \bowtie uu\ (\neg\ a) \cdot uu\ a)$
unfolding *uu-def* **by** (*induct a*, *use bin-not-comp pref-comp-sym* **in** *auto*)

lemma *bin-not-comp-suf*: $\neg\ u_0 \cdot u_1 \bowtie_s u_1 \cdot u_0$
using *comm-comp-eq-conv-suf non-comm*[*reversed*] **by** *blast*

lemma *bin-not-comp-suf-bool*: $\neg\ (uu\ a \cdot uu\ (\neg\ a) \bowtie_s uu\ (\neg\ a) \cdot uu\ a)$
unfolding *uu-def* **by** (*induct a*, *use bin-not-comp-suf suf-comp-sym* **in** *auto*)

lemma *bin-mismatch-neq*: *bin-mismatch* $u_0\ u_1 \neq$ *bin-mismatch* $u_1\ u_0$
using *non-comm*[*folded bin-mismatch-comm*]**.**

abbreviation *bin-code-lcp* (α) **where** *bin-code-lcp* $\equiv$ *bin-lcp* $u_0\ u_1$
abbreviation *bin-code-lcs* **where** *bin-code-lcs* $\equiv$ *bin-lcs* $u_0\ u_1$
abbreviation *bin-code-mismatch-fst* (c_0) **where** *bin-code-mismatch-fst* $\equiv$ *bin-mismatch* $u_0\ u_1$
abbreviation *bin-code-mismatch-snd* (c_1) **where** *bin-code-mismatch-snd* $\equiv$ *bin-mismatch* $u_1\ u_0$

definition *cc* **where** *cc a* = (*if a then* c_0 *else* c_1)

lemmas *bin-lcp-swap* = *bin-lcp-sym*[*of* $u_0\ u_1$, *symmetric*] **and**
bin-lcp-pref = *bin-lcp-pref-fst-snd*[*of* $u_0\ u_1$] **and**
bin-lcp-pref′ = *bin-lcp-pref-snd-fst*[*of* $u_0\ u_1$] **and**
bin-lcp-short = *bin-lcp-len*[*OF non-comm*, *unfolded lenmorph*]

lemmas *bin-code-simps* = *cc-def uu-def if-True if-False bool-simps*

lemma *bin-lcp-bool*: *bin-lcp* (*uu a*) (*uu* ($\neg$ *a*)) = *bin-code-lcp*
unfolding *uu-def* **by** (*induct a*, *simp-all add*: *bin-lcp-swap*)

lemma *bin-lcp-spref*: $\alpha <p\ u_0 \cdot u_1$
using *bin-lcp-pref bin-lcp-pref′ bin-not-comp* **by** *fastforce*

lemma *bin-lcp-spref′*: $\alpha <p\ u_1 \cdot u_0$
using *bin-lcp-pref bin-lcp-pref′ bin-not-comp* **by** *fastforce*

lemma *bin-lcp-spref-bool*: $\alpha <p\ uu\ a \cdot uu\ (\neg\ a)$
unfolding *uu-def* **by** (*induct a*, *use bin-lcp-spref bin-lcp-spref′* **in** *auto*)

lemma *bin-mismatch-bool′*: $\alpha \cdot [cc\ a] \leq p\ uu\ a \cdot uu\ (\neg\ a)$
using *add-nth-pref*[*OF bin-lcp-spref-bool*, *of a*]

unfolding *uu-def cc-def bin-mismatch-def bin-lcp-bool bin-lcp-swap*
by (*induct a*) *simp-all*

lemma *bin-mismatch-bool*: $\alpha \cdot [cc\ a] \leq p\ uu\ a \cdot \alpha$
proof–
from *pref-prolong*[*OF bin-mismatch-bool'*, *OF triv-pref*]
have $\alpha \cdot [cc\ a] \leq p\ uu\ a \cdot (uu\ (\neg\ a) \cdot uu\ a)$
by *blast*
from *pref-prod-pref-short*[*OF this bin-lcp-pref-snd-fst*, *unfolded bin-lcp-bool len-morph sing-len*]
show *?thesis*
using *nemp-len*[*OF bin-nemp-bool*, *of a*] **by** *linarith*
qed

lemmas *bin-fst-mismatch* = *bin-mismatch-bool*[*of True*, *unfolded bin-code-simps*]
and
bin-fst-mismatch' = *bin-mismatch-bool'*[*of True*, *unfolded bin-code-simps*] **and**
bin-snd-mismatch = *bin-mismatch-bool*[*of False*, *unfolded bin-code-simps*] **and**
bin-snd-mismatch' = *bin-mismatch-bool'*[*of False*, *unfolded bin-code-simps*]

lemma *bin-lcp-pref-all*: *xs* ∈ *lists* {u_0,u_1} ⟹ $\alpha \leq p$ *concat xs* · α
proof (*induct xs*)
case (*Cons a xs*)
have *a* ∈ {u_0,u_1} **and** *xs* ∈ *lists* {u_0, u_1}
using ‹*a* # *xs* ∈ *lists* {u_0, u_1}› **by** *simp-all*
show *?case*
proof (*rule two-elemP*[*OF* ‹*a* ∈ {u_0,u_1}›], *simp-all*)
show $\alpha \leq p\ u_0$ · *concat xs* · α
using *pref-extD*[*OF bin-fst-mismatch*] *Cons.hyps*[*OF* ‹*xs* ∈ *lists* {u_0, u_1}›] *pref-prolong* **by** *blast*
next
show $\alpha \leq p\ u_1$ · *concat xs* · α
using *pref-extD*[*OF bin-snd-mismatch*] *Cons.hyps*[*OF* ‹*xs* ∈ *lists* {u_0, u_1}›] *pref-prolong* **by** *blast*
qed
qed *simp*

lemma *bin-lcp-pref-all-hull*: *w* ∈ ⟨{u_0,u_1}⟩ ⟹ $\alpha \leq p\ w \cdot \alpha$
using *bin-lcp-pref-all* **using** *hull-concat-listsE* **by** *metis*

lemma *bin-lcp-mismatch-pref-all-bool*: **assumes** $q \leq p\ w$ **and** $w \in \langle\{uu\ b, uu\ (\neg\ b)\}\rangle$ **and** $|\alpha| < |uu\ a \cdot q|$
shows $\alpha \cdot [cc\ a] \leq p\ uu\ a \cdot q$
proof–
have *aux*: $uu\ a \cdot w \cdot \alpha = (uu\ a \cdot q) \cdot (q^{-1>} w \cdot \alpha)$ {*uu b*,*uu* (¬ *b*)} = {u_0,u_1}
using *lq-pref*[*OF* ‹*q* ≤*p w*›] *bin-code-set-bool* **by** *force+*
have $|\alpha \cdot [cc\ a]| \leq |uu\ a \cdot q|$
using ‹$|\alpha| < |uu\ a \cdot q|$› **by** *auto*
thus *?thesis*

using *pref-prolong*[*OF bin-mismatch-bool bin-lcp-pref-all-hull*[*OF* ‹$w \in \langle\{uu\ b, uu\ (\neg\ b)\}\rangle$›[*unfolded aux*]], *of a*]
unfolding *aux* **by** *blast*
qed

lemmas *bin-lcp-mismatch-pref-all-fst* = *bin-lcp-mismatch-pref-all-bool*[*of - - True True*, *unfolded bin-code-simps*] **and**
bin-lcp-mismatch-pref-all-snd = *bin-lcp-mismatch-pref-all-bool*[*of - - True False*, *unfolded bin-code-simps*]

lemma *bin-lcp-pref-all-len*: **assumes** $q \leq p\ w$ **and** $w \in \langle\{u_0, u_1\}\rangle$ **and** $|\alpha| \leq |q|$
shows $\alpha \leq p\ q$
using *bin-lcp-pref-all-hull*[*OF* ‹$w \in \langle\{u_0, u_1\}\rangle$›] *pref-ext*[*OF* ‹$q \leq p\ w$›] *prefix-length-prefix*[*OF - -* ‹$|bin\text{-}code\text{-}lcp| \leq |q|$›] **by** *blast*

lemma *bin-mismatch-all-bool*: **assumes** *xs* ∈ *lists* $\{uu\ b, uu\ (\neg\ b)\}$ **shows** $\alpha \cdot [cc\ a] \leq p\ (uu\ a) \cdot concat\ xs \cdot \alpha$
using *pref-prolong*[*OF bin-mismatch-bool bin-lcp-pref-all*, *of xs a*] *assms* **unfolding** *bin-code-set-bool*[*of b*]**.**

lemmas *bin-fst-mismatch-all* = *bin-mismatch-all-bool*[*of - True True*, *unfolded bin-code-simps*] **and**
bin-snd-mismatch-all = *bin-mismatch-all-bool*[*of - True False*, *unfolded bin-code-simps*]

lemma *bin-mismatch-all-hull-bool*: **assumes** $w \in \langle\{uu\ b, uu\ (\neg\ b)\}\rangle$ **shows** $\alpha \cdot [cc\ a] \leq p\ uu\ a \cdot w \cdot \alpha$
using *bin-mismatch-all-bool hull-concat-listsE*[*OF assms*] **by** *metis*

lemmas *bin-fst-mismatch-all-hull* = *bin-mismatch-all-hull-bool*[*of - True True*, *unfolded bin-code-simps*] **and**
bin-snd-mismatch-all-hull = *bin-mismatch-all-hull-bool*[*of - True False*, *unfolded bin-code-simps*]

lemma *bin-mismatch-all-len-bool*: **assumes** $q \leq p\ uu\ a \cdot w$ **and** $w \in \langle\{uu\ b, uu\ (\neg\ b)\}\rangle$ **and** $|\alpha| < |q|$
shows $\alpha \cdot [cc\ a] \leq p\ q$
proof−
have $|\alpha \cdot [cc\ a]| \leq |uu\ a \cdot w|\ |\alpha \cdot [cc\ a]| \leq |q|$
using *less-le-trans*[*OF* ‹$|\alpha| < |q|$› *pref-len*[*OF* ‹$q \leq p\ uu\ a \cdot w$›]] ‹$|\alpha| < |q|$› **by** *force*+
from *pref-prod-le*[*OF bin-mismatch-all-hull-bool*[*OF assms*(*2*), *unfolded lassoc*], *OF this*(*1*)]
show *?thesis*
by (*rule prefix-length-prefix*) *fact*+
qed

lemmas *bin-fst-mismatch-all-len* = *bin-mismatch-all-len-bool*[*of - True - True*, *unfolded bin-code-simps*] **and**

bin-snd-mismatch-all-len = *bin-mismatch-all-len-bool*[*of* - *False* - *True*, *unfolded bin-code-simps*]

lemma *bin-code-delay*: **assumes** $|\alpha| \leq |q_0|$ **and** $|\alpha| \leq |q_1|$ **and**
$q_0 \leq p\ u_0 \cdot w_0$ **and** $q_1 \leq p\ u_1 \cdot w_1$ **and**
$w_0 \in \langle\{u_0, u_1\}\rangle$ **and** $w_1 \in \langle\{u_0, u_1\}\rangle$
shows $q_0 \wedge_p q_1 = \alpha$
proof−
have *p1*: $\alpha \cdot [c_0] \leq p\ u_0 \cdot w_0 \cdot \alpha$
using *assms*(*5*) **using** *bin-fst-mismatch-all-hull* **by** *auto*
have *p2*: $\alpha \cdot [c_1] \leq p\ u_1 \cdot w_1 \cdot \alpha$
using *assms*(*6*) **using** *bin-snd-mismatch-all-hull* **by** *auto*
have *lcp*: $u_0 \cdot w_0 \cdot \alpha \wedge_p u_1 \cdot w_1 \cdot \alpha = \alpha$
using *lcp-first-mismatch-pref*[*OF p1 p2 bin-mismatch-neq*]**.**
from *lcp-extend-eq*[*of* $q_0\ u_0 \cdot w_0 \cdot \alpha\ q_1\ u_1 \cdot w_1 \cdot \alpha$,
unfolded this,*OF* - - *assms*(*1*−*2*)]
show *?thesis*
using *pref-ext*[*OF* $\langle q_0 \leq p\ u_0 \cdot w_0\rangle$] *pref-ext*[*OF* $\langle q_1 \leq p\ u_1 \cdot w_1\rangle$] **by** *force*
qed

lemma *hd-lq-mismatch-fst*: *hd* $(\alpha^{-1>}(u_0 \cdot \alpha)) = c_0$
using *hd-lq-conv-nth*[*OF prefix-snocD*[*OF bin-fst-mismatch*]] *bin-fst-mismatch*
by (*auto simp add*: *prefix-def*)

lemma *hd-lq-mismatch-snd*: *hd* $(\alpha^{-1>}(u_1 \cdot \alpha)) = c_1$
using *hd-lq-conv-nth*[*OF prefix-snocD*[*OF bin-snd-mismatch*]] *bin-snd-mismatch*
by (*auto simp add*: *prefix-def*)

lemma *hds-bin-mismatch-neq*: *hd* $(\alpha^{-1>}(u_0 \cdot \alpha)) \neq$ *hd* $(\alpha^{-1>}(u_1 \cdot \alpha))$
unfolding *hd-lq-mismatch-fst hd-lq-mismatch-snd*
using *bin-mismatch-neq***.**

lemma *bin-lcp-fst-pow-pref*: **assumes** $0 < k$ **shows** $\alpha \cdot [c_0] \leq p\ u_0{}^{@}k \cdot u_1 \cdot z$
using *assms*
proof (*induct k rule*: *nat-induct-non-zero*)
case *1*
then show *?case*
unfolding *pow-list-1* **using** *pref-prolong*[*OF bin-fst-mismatch′ triv-pref*]**.**
next
case (*Suc n*)
show *?case*
unfolding *pow-Suc rassoc*
by (*rule pref-prolong*[*OF bin-fst-mismatch*])
(*use append-prefixD*[*OF Suc.hyps*(*2*)] **in** *blast*)
qed

lemmas *bin-lcp-snd-pow-pref* = *binary-code.bin-lcp-fst-pow-pref*[*OF bin-code-swap*, *unfolded bin-lcp-swap*]

lemma *bin-lcp-fst-lcp*: $\alpha \leq p\ u_0 \cdot \alpha$ **and** *bin-lcp-snd-lcp*: $\alpha \leq p\ u_1 \cdot \alpha$
using *pref-extD*[*OF bin-fst-mismatch*] *pref-extD*[*OF bin-snd-mismatch*].

lemma *bin-lcp-pref-all-set*: **assumes** *set ws* = $\{u_0,u_1\}$
shows $\alpha \leq p$ *concat ws*
proof–
have *ws* ∈ *lists* $\{u_0, u_1\}$
using *assms* **by** *blast*
have $|u_0| + |u_1| \leq |$*concat ws*$|$
using *assms two-in-set-concat-len*[*OF bin-code-neq*] **by** *simp*
with *pref-prod-le*[*OF bin-lcp-pref-all*[*OF* ‹*ws* ∈ *lists* $\{u_0, u_1\}$›]] *bin-lcp-short*
show *?thesis*
by *simp*
qed

lemma *bin-lcp-conjug-morph*:
assumes $u \in \langle\{u_0,u_1\}\rangle$ **and** $v \in \langle\{u_0,u_1\}\rangle$
shows $\alpha^{-1>}(u \cdot \alpha) \cdot \alpha^{-1>}(v \cdot \alpha) = \alpha^{-1>}((u \cdot v) \cdot \alpha)$
unfolding *lq-reassoc*[*OF bin-lcp-pref-all-hull*[*OF* ‹$u \in \langle\{u_0,u_1\}\rangle$›]] *rassoc*
lq-pref[*OF bin-lcp-pref-all-hull*[*OF* ‹$v \in \langle\{u_0,u_1\}\rangle$›]]..

lemma *lcp-bin-conjug-prim-iff*:
set ws = $\{u_0,u_1\} \Longrightarrow$ *primitive* $(\alpha^{-1>}$(*concat ws*$) \cdot \alpha) \longleftrightarrow$ *primitive* (*concat ws*)
using *conjug-prim-iff*[*OF root-conjug*[*OF pref-ext*[*OF bin-lcp-pref-all-set*]], *symmetric*]
unfolding *lq-reassoc*[*OF bin-lcp-pref-all-set*] **by** *simp*

lemma *bin-lcp-conjug-inj-on*: *inj-on* $(\lambda u.\ \alpha^{-1>}(u \cdot \alpha))\ \langle\{u_0,u_1\}\rangle$
unfolding *inj-on-def* **using** *bin-lcp-pref-all-hull cancel-right lq-pref*
by *metis*

lemma *bin-code-lcp-marked*: **assumes** *us* ∈ *lists* $\{u_0,u_1\}$ **and** *vs* ∈ *lists* $\{u_0,u_1\}$ **and** *hd us* ≠ *hd vs*
shows *concat us* $\cdot\ \alpha \wedge_p$ *concat vs* $\cdot\ \alpha = \alpha$
proof (*cases us* = $\varepsilon \vee$ *vs* = ε)
assume *us* = $\varepsilon \vee$ *vs* = ε
thus *?thesis*
using *append-self-conv2 assms*(*1*) *assms*(*2*) *bin-lcp-pref-all concat.simps*(*1*) *lcp-pref-conv lcp-sym* **by** *metis*
next
assume ¬ (*us* = $\varepsilon \vee$ *vs* = ε) **hence** *us* ≠ ε **and** *vs* ≠ ε **by** *blast*+
have *spec-case*: *concat us* $\cdot\ \alpha \wedge_p$ *concat vs* $\cdot\ \alpha = \alpha$ **if** *us* ∈ *lists* $\{u_0,u_1\}$ **and** *vs* ∈ *lists* $\{u_0,u_1\}$ **and** *hd us* = u_0 **and** *hd vs* = u_1 **and** *us* ≠ ε **and** *vs* ≠ ε **for** *us vs*
proof–
have *concat us* = $u_0 \cdot$ *concat* (*tl us*)
unfolding *hd-concat-tl*[*OF* ‹*us* ≠ ε›, *symmetric*] ‹*hd us* = u_0›..
from *bin-fst-mismatch-all*[*OF tl-in-lists*[*OF* ‹*us* ∈ *lists* $\{u_0,u_1\}$›], *folded rassoc this*]
have *pref1*: $\alpha \cdot [c_0] \leq p$ *concat us* $\cdot\ \alpha$.

have *concat vs* = u_1 · *concat* (*tl vs*)
unfolding *hd-concat-tl*[*OF* ‹*vs* ≠ ε›, *symmetric*] ‹*hd vs* = u_1›**..**
from *bin-snd-mismatch-all*[*OF tl-in-lists*[*OF* ‹*vs* ∈ *lists* {u_0,u_1}›], *folded rassoc this*]
have *pref2*: α · [c_1] ≤*p* *concat vs* · α**.**
show *?thesis*
using *lcp-first-mismatch-pref*[*OF pref1 pref2 bin-mismatch-neq*]**.**
qed
have *hd us* ∈ {u_0,u_1} **and** *hd vs* ∈ {u_0,u_1} **using**
lists-hd-in-set[*OF* ‹*us* ≠ ε› ‹*us* ∈ *lists* {u_0, u_1}›] *lists-hd-in-set*[*OF* ‹*vs* ≠ ε› ‹*vs* ∈ *lists* {u_0, u_1}›]**.**
then consider *hd us* = u_0 ∧ *hd vs* = u_1 | *hd us* = u_1 ∧ *hd vs* = u_0
using ‹*hd us* ≠ *hd vs*› **by** *fastforce*
then show *?thesis*
using *spec-case*[*rule-format*] ‹*us* ≠ ε› ‹*vs* ≠ ε› *assms lcp-sym* **by** *metis*
qed

— ALT PROOF
lemma **assumes** *us* ∈ *lists* {u_0,u_1} **and** *vs* ∈ *lists* {u_0,u_1} **and** *hd us* ≠ *hd vs*
shows *concat us* · α ∧*p* *concat vs* · α = α
using *assms*
proof (*induct us vs rule*: *list-induct2'*)
case (*2 x xs*)
show *?case*
using *bin-lcp-pref-all*[*OF* ‹*x* # *xs* ∈ *lists* {u_0, u_1}›, *folded lcp-pref-conv*, *unfolded lcp-sym*[*of* α]] **by** *simp*
next
case (*3 y ys*)
show *?case*
using *bin-lcp-pref-all*[*OF* ‹*y* # *ys* ∈ *lists* {u_0, u_1}›, *folded lcp-pref-conv*] **by** *simp*
next
case (*4 x xs y ys*)
interpret *i*: *binary-code x y*
using *4.prems*(*1*) *4.prems*(*2*) *4.prems*(*3*) *non-comm binary-code.intro* **by** *auto*
have *alph*: {u_0,u_1} = {*x*,*y*}
using *4.prems*(*1*) *4.prems*(*2*) *4.prems*(*3*) **by** *auto*
from *disjE*[*OF this*[*unfolded doubleton-eq-iff*]]
have *i.bin-code-lcp* = α
using *i.bin-lcp-swap*[*symmetric*] **by** *blast*
have *c0*: *i.bin-code-lcp* · [*i.bin-code-mismatch-fst*] ≤*p* *x* · *concat xs* · *i.bin-code-lcp*
using *i.bin-lcp-pref-all*[*of xs*] ‹*x* # *xs* ∈ *lists* {u_0, u_1}›[*unfolded Cons-in-lists-iff alph*]
pref-prolong[*OF i.bin-fst-mismatch*] **by** *blast*
have *c1*: *i.bin-code-lcp* · [*i.bin-code-mismatch-snd*] ≤*p* *y* · *concat ys* · *i.bin-code-lcp*
using *pref-prolong*[*OF conjunct2*[*OF* ‹*y* # *ys* ∈ *lists* {u_0, u_1}›[*unfolded Cons-in-lists-iff alph*],
THEN i.bin-snd-mismatch-all[*of ys*]], *OF self-pref*]**.**
have *i.bin-code-lcp*·[*i.bin-code-mismatch-fst*] ∧*p* *i.bin-code-lcp*·[*i.bin-code-mismatch-snd*]

$= i.bin\text{-}code\text{-}lcp$
by (*simp add*: *i.bin-mismatch-neq lcp-first-mismatch'*)
from *lcp-rulers*[*OF c0 c1*, *unfolded this*, *unfolded bin-lcp-swap*]
show *?case*
unfolding *concat.simps*(*2*) *rassoc* **using** *i.bin-mismatch-neq*
‹*i.bin-code-lcp* $= \alpha$› **by** *force*
qed *simp*

lemma *bin-code-lcp-concat*: **assumes** $us \in lists\ \{u_0,u_1\}$ **and** $vs \in lists\ \{u_0,u_1\}$ **and** $\neg\ us \bowtie vs$
shows $concat\ us \cdot \alpha \wedge_p concat\ vs \cdot \alpha = concat\ (us \wedge_p vs) \cdot \alpha$
proof–
obtain *us' vs'* **where** *us*: $(us \wedge_p vs) \cdot us' = us$ **and** *vs*: $(us \wedge_p vs) \cdot vs' = vs$ **and** $us' \neq \varepsilon$ **and** $vs' \neq \varepsilon$ **and** $hd\ us' \neq hd\ vs'$
using *lcp-mismatchE*[*OF* ‹$\neg\ us \bowtie vs$›].
have *cu*: $concat\ us \cdot \alpha = concat\ (us \wedge_p vs) \cdot concat\ us' \cdot \alpha$
unfolding *lassoc concat-morph*[*symmetric*] *us*..
have *cv*: $concat\ vs \cdot \alpha = concat\ (us \wedge_p vs) \cdot concat\ vs' \cdot \alpha$
unfolding *lassoc concat-morph*[*symmetric*] *vs*..
have $us' \in lists\ \{u_0,u_1\}$
using ‹$us \in lists\ \{u_0,u_1\}$› *us* **by** *inlists*
have $vs' \in lists\ \{u_0,u_1\}$
using ‹$vs \in lists\ \{u_0,u_1\}$› *vs* **by** *inlists*
show $concat\ us \cdot \alpha \wedge_p concat\ vs \cdot \alpha = concat\ (us \wedge_p vs) \cdot \alpha$
unfolding *cu cv*
using *bin-code-lcp-marked*[*OF* ‹$us' \in lists\ \{u_0,u_1\}$› ‹$vs' \in lists\ \{u_0,u_1\}$› ‹$hd\ us' \neq hd\ vs'$›]
unfolding *lcp-ext-left* **by** *fast*
qed

lemma *bin-code-lcp-concat'*: **assumes** $us \in lists\ \{u_0,u_1\}$ **and** $vs \in lists\ \{u_0,u_1\}$ **and** $\neg\ concat\ us \bowtie concat\ vs$
shows $concat\ us \wedge_p concat\ vs = concat\ (us \wedge_p vs) \cdot \alpha$
using *bin-code-lcp-concat*[*OF assms*(*1–2*)] *assms*(*3*) *lcp-ext-right-conv pref-concat-pref prefix-comparable-def* **by** *metis*

lemma *bin-lcp-pows*: $0 < k \Longrightarrow 0 < l \Longrightarrow {u_0}^{@}k \cdot u_1 \cdot z \wedge_p {u_1}^{@}l \cdot u_0 \cdot z' = \alpha$
using *lcp-first-mismatch-pref*[*OF bin-lcp-fst-pow-pref bin-lcp-snd-pow-pref bin-mismatch-neq*].

theorem *bin-code*: **assumes** $us \in lists\ \{u_0,u_1\}$ **and** $vs \in lists\ \{u_0,u_1\}$ **and** $concat\ us = concat\ vs$
shows $us = vs$
using *assms*
proof (*induct us vs rule*: *list-induct2'*)
case (*4 x xs y ys*)
then show *?case*
proof–
have $x = y$
using *bin-code-lcp-marked*[*OF* ‹$x \# xs \in lists\ \{u_0,\ u_1\}$› ‹$y \# ys \in lists\ \{u_0,$

$u_1\}\rangle]$ ‹*y # ys ∈ lists* $\{u_0, u_1\}\rangle$ *non-comm*
unfolding ‹*concat* (*x # xs*) = *concat* (*y # ys*)› **unfolding** *concat.simps*(*2*) *lcp-self list.sel*(*1*)
by *auto*
thus *x # xs* = *y # ys*
using *4.hyps* ‹*concat* (*x # xs*) = *concat* (*y # ys*)›[*unfolded concat.simps*(*2*) ‹*x* = *y*›, *unfolded cancel*]
‹*y # ys ∈ lists* $\{u_0, u_1\}\rangle$[*unfolded Cons-in-lists-iff*] ‹*x # xs ∈ lists* $\{u_0, u_1\}\rangle$[*unfolded Cons-in-lists-iff*]
by *simp*
qed
qed (*auto simp*: *bin-fst-nemp bin-snd-nemp*)

lemma *code-bin-roots*: *binary-code* (*ϱ* u_0) (*ϱ* u_1)
using *non-comm comm-primroot-conv′* **by** (*unfold-locales*) *blast*

sublocale *code* $\{u_0,u_1\}$
using *bin-code* **by** *unfold-locales*

lemma *primroot-dec*: (*Dec* {*ϱ* u_0, *ϱ* u_1} u_0) = $[ϱ\ u_0]^{@}e_{ϱ}\ u_0$ (*Dec* {*ϱ* u_0, *ϱ* u_1} u_1) = $[ϱ\ u_1]^{@}e_{ϱ}\ u_1$
proof−
interpret *rs*: *binary-code ϱ* u_0 *ϱ* u_1
by (*simp add*: *code-bin-roots*)
from *primroot-exp-eq*
have *concat* ($[ϱ\ u_0]^{@}e_{ϱ}\ u_0$) = u_0 *concat* ($[ϱ\ u_1]^{@}e_{ϱ}\ u_1$) = u_1
by *force*+
from *rs.code-unique-dec*[*OF - this*(*1*)] *rs.code-unique-dec*[*OF - this*(*2*)]
show (*Dec* {*ϱ* u_0, *ϱ* u_1} u_0) = $[ϱ\ u_0]^{@}e_{ϱ}\ u_0$ (*Dec* {*ϱ* u_0, *ϱ* u_1} u_1) = $[ϱ\ u_1]^{@}e_{ϱ}\ u_1$
by (*simp-all add*: *sing-pow-lists*)
qed

lemma *bin-code-prefs*: **assumes** $w \in \langle\{u_0,u_1\}\rangle$ **and** $p \le_p w\ w' \in \langle\{u_0,u_1\}\rangle$ **and** $|u_1| \le |p|$
shows ¬ $u_0 \cdot\ p \le_p u_1 \cdot w'$
proof
assume *contr*: $u_0 \cdot p \le_p u_1 \cdot w'$
have $|\alpha| < |u_0 \cdot p|$
using ‹$|u_1| \le |p|$› *bin-lcp-short* **by** *auto*
hence $\alpha \cdot [c_0] \le_p u_0 \cdot p$
using ‹$p \le_p w$› ‹$w \in \langle\{u_0,u_1\}\rangle$› *bin-lcp-mismatch-pref-all-fst* **by** *auto*
from *pref-ext*[*OF pref-trans*[*OF this contr*], *unfolded rassoc*]
have $\alpha \cdot [c_0] \le_p u_1 \cdot w' \cdot \alpha$.
from *bin-mismatch-neq same-sing-pref*[*OF bin-snd-mismatch-all-hull*[*OF* ‹$w' \in \langle\{u_0,u_1\}\rangle$›] *this*]
show *False*
by *simp*
qed

lemma *bin-code-rev*: *binary-code* (*rev* u_0) (*rev* u_1)
by (*unfold-locales*, *unfold comm-rev-iff*, *simp add*: *non-comm*)

lemma *bin-lcp-pows-lcp*: $0 < k \Longrightarrow 0 < l \Longrightarrow {u_0}^{@}k \cdot {u_1}^{@}l \wedge_p {u_1}^{@}l \cdot {u_0}^{@}k = u_0 \cdot u_1 \wedge_p u_1 \cdot u_0$
using *bin-lcp-pows* **unfolding** *bin-lcp-def* **using** *pow-pos* **by** *metis*

lemma *bin-mismatch*: $u_0 \cdot \alpha \wedge_p u_1 \cdot \alpha = \alpha$
using *lcp-first-mismatch-pref*[*OF bin-fst-mismatch bin-snd-mismatch bin-mismatch-neq*].

lemma *not-comp-bin-fst-snd*: $\neg\ u_0 \cdot \alpha \bowtie u_1 \cdot\ \alpha$
using *ruler-comp*[*OF bin-fst-mismatch bin-snd-mismatch*] *bin-mismatch-neq*
unfolding *prefix-comparable-def pref-cancel-conv* **by** *force*

theorem *bin-bounded-delay*: **assumes** $z \leq p\ u_0 \cdot w_0$ **and** $z \leq p\ u_1 \cdot w_1$
and $w_0 \in \langle\{u_0,u_1\}\rangle$ **and** $w_1 \in \langle\{u_0,u_1\}\rangle$
shows $|z| \leq |\alpha|$
proof (*rule leI*, *rule notI*)
assume $|\alpha| < |z|$
hence $|\alpha \cdot [a]| \leq |z|$ **for** *a*
unfolding *lenmorph sing-len* **by** *simp*
have $z \leq p\ u_0 \cdot w_0 \cdot \alpha$ **and** $z \leq p\ u_1 \cdot w_1 \cdot \alpha$
using *pref-prolong*[*OF* ‹$z \leq p\ u_0 \cdot w_0$› *triv-pref*] *pref-prolong*[*OF* ‹$z \leq p\ u_1 \cdot w_1$› *triv-pref*].
have $\alpha \cdot [c_0] \leq p\ u_0 \cdot w_0 \cdot \alpha$ **and** $\alpha \cdot [c_1] \leq p\ u_1 \cdot w_1 \cdot \alpha$
using *bin-fst-mismatch-all-hull*[*OF* ‹$w_0 \in \langle\{u_0,u_1\}\rangle$›] *bin-snd-mismatch-all-hull*[*OF* ‹$w_1 \in \langle\{u_0,u_1\}\rangle$›].
from ‹$z \leq p\ u_0 \cdot w_0 \cdot \alpha$› ‹$\alpha \cdot [c_0] \leq p\ u_0 \cdot w_0 \cdot \alpha$› ‹$|\alpha \cdot [c_0]| \leq |z|$›
have $\alpha \cdot [c_0] \leq p\ z$
using *prefix-length-prefix* **by** *blast*
from ‹$z \leq p\ u_1 \cdot w_1 \cdot \alpha$› ‹$\alpha \cdot [c_1] \leq p\ u_1 \cdot w_1 \cdot \alpha$› ‹$|\alpha \cdot [c_1]| \leq |z|$›
have $\alpha \cdot [c_1] \leq p\ z$
using *prefix-length-prefix* **by** *blast*
from ‹$\alpha \cdot [c_1] \leq p\ z$› ‹$\alpha \cdot [c_0] \leq p\ z$› *bin-mismatch-neq*
show *False*
unfolding *prefix-def* **by** *force*
qed

thm *binary-code.bin-lcp-pows-lcp*

lemma *prim-roots-lcp*: *bin-lcp* $(\varrho\ u_0)$ $(\varrho\ u_1)$ $= \alpha$
proof–
obtain *k* **where** $\varrho\ {u_0}^{@}k = u_0\ 0 < k$
using *primroot-expE*.
obtain *m* **where** $\varrho\ {u_1}^{@}m = u_1\ 0 < m$
using *primroot-expE*.
have $\varrho\ u_0 \cdot \varrho\ u_1 \neq \varrho\ u_1 \cdot \varrho\ u_0$

using *non-comm*[*unfolded comm-primroot-conv'*[*of* u_0]]**.**
then interpret r: *binary-code* ϱ u_0 ϱ u_1 **by** *unfold-locales*
from *r.bin-lcp-pows-lcp*[*OF* ‹$0 < k$› ‹$0 < m$›, *unfolded* ‹ϱ $u_1{}^{@}m = u_1$› ‹ϱ $u_0{}^{@}k = u_0$›, *symmetric*]
show *?thesis*
unfolding *bin-lcp-def***.**
qed

lemma *bin-roots-decompose*:
Dec {ϱ u_0, u_1} u_0 = [ϱ u_0]$^{@}e_{\varrho}$ u_0
Dec {ϱ u_0, u_1} u_1 = [u_1]
Dec {u_0, ϱ u_1} u_1 = [ϱ u_1]$^{@}e_{\varrho}$ u_1
Dec {u_0, ϱ u_1} u_0 = [u_0]
Dec {u_0,u_1} u_0 = [u_0]
Dec {u_0,u_1} u_1 = [u_1]
proof–
show *Dec* {u_0,u_1} u_0 = [u_0] *Dec* {u_0,u_1} u_1 = [u_1]
using *code-el-dec* **by** *simp-all*
interpret r: *binary-code* ϱ u_0 u_1
using *non-comm* **unfolding** *binary-code-def* **using** *comm-primroot-conv*[*of* u_1 u_0]
by *presburger*
interpret r': *binary-code* u_0 ϱ u_1
using *non-comm* **unfolding** *binary-code-def* **using** *comm-primroot-conv*[*of* u_0 u_1]
by *presburger*
from *r.code-el-dec*
show *Dec* {ϱ u_0, u_1} u_1 = [u_1]
by *force*
from *r'.code-el-dec*
show *Dec* {u_0, ϱ u_1} u_0 = [u_0]
by *force*
show *Dec* {ϱ u_0, u_1} u_0 = [ϱ u_0]$^{@}e_{\varrho}$ u_0
using *r.code-unique-dec'*[*OF sing-pow-lists*[*of* ϱ u_0 {ϱ u_0, u_1} e_{ϱ} u_0]]
unfolding *concat-pow-list concat-sing' primroot-exp-eq* **by** *simp*
show *Dec* {u_0, ϱ u_1} u_1 = [ϱ u_1] $^{@}$ e_{ϱ} u_1
using *r'.code-unique-dec'*[*OF sing-pow-lists*[*of* ϱ u_1 {u_0, ϱ u_1} e_{ϱ} u_1]]
unfolding *concat-pow-list concat-sing' primroot-exp-eq* **by** *simp*
qed

lemma *ref-fst-sq*: *Ref* {ϱ u_0, u_1}[u_0,u_0] = [ϱ u_0]$^{@}$(e_{ϱ} u_0 * *2*)
unfolding *refine-def pow-mult pow-list-2*
using *bin-roots-decompose* **by** *simp*

lemma *ref-fst-pow*: *Ref* {ϱ u_0, u_1}[u_0]$^{@}k$ = [ϱ u_0]$^{@}$(e_{ϱ} u_0 * k)
unfolding *refine-def pow-mult pow-list-2*
using *bin-roots-decompose* **by** *simp*

lemma *bin-code-concat-len*: **assumes** *ws* ∈ *lists* {u_0,u_1}

shows $|concat\ ws| = count\text{-}list\ ws\ u_0 * |u_0| + count\text{-}list\ ws\ u_1 * |u_1|$
using *bin-len-concat*[*OF bin-code-neq assms*]**.**

Maximal r-prefixes

lemma *bin-lcp-per-root-max-pref-short*: **assumes** $\alpha <p\ u_0 \cdot u_1 \wedge_p r \cdot u_0 \cdot u_1$ **and** $r \neq \varepsilon$ **and** $q \leq p\ w$ **and** $w \in \langle\{u_0, u_1\}\rangle$
shows $u_1 \cdot q \wedge_p r \cdot u_1 \cdot q = take\ |u_1 \cdot q|\ \alpha$
proof−
have $q \bowtie \alpha$
using *bin-lcp-pref-all-hull*[*OF* ‹$w \in \langle\{u_0, u_1\}\rangle$›] *ruler-comp*[*OF* ‹$q \leq p\ w$›, *of* α $w \cdot \alpha$] **by** *blast*
hence *comp1*: $u_1 \cdot q \bowtie \alpha \cdot [c_1]$
using *ruler-comp*[*OF self-pref bin-snd-mismatch*, *of* $u_1 \cdot q$] **unfolding** *comp-cancel*
by *blast*

from *add-nth-pref*[*OF assms*(*1*), *THEN pref-lcp-pref*] *bin-fst-mismatch′*
have $(u_0 \cdot u_1 \wedge_p r \cdot u_0 \cdot u_1)\ !\ |\alpha| = c_0$
using *same-sing-pref* **by** *fast*

from *add-nth-pref*[*OF assms*(*1*), *unfolded this*]
have $\alpha \cdot [c_0] \leq p\ r \cdot u_0 \cdot u_1$
by *force*

have *len*: $|\alpha \cdot [c_0]| \leq |r \cdot \alpha|$
using *nemp-len*[*OF* ‹$r \neq \varepsilon$›] **unfolding** *lenmorph sing-len* **by** *linarith*

have *comp2*: $r \cdot u_1 \cdot q \bowtie \alpha \cdot [c_0]$
proof(*rule ruler-comp*[*OF - - comp-refl*])
show $r \cdot u_1 \cdot q \leq p\ r \cdot u_1 \cdot w \cdot \alpha$
using ‹$q \leq p\ w$› **by** *fastforce*
show $\alpha \cdot [c_0] \leq p\ r \cdot u_1 \cdot w \cdot \alpha$
proof(*rule pref-prolong*)
show $\alpha \cdot [c_0] \leq p\ r \cdot \alpha$
using ‹$\alpha \cdot [c_0] \leq p\ r \cdot u_0 \cdot u_1$› *bin-lcp-pref len pref-prod-pref-short* **by** *blast*
show $\alpha \leq p\ u_1 \cdot w \cdot \alpha$
using ‹$w \in \langle\{u_0, u_1\}\rangle$› *bin-lcp-pref-all-hull*[*of* $u_1 \cdot w$] **by** *auto*
qed
qed

have *min*: $(min\ |u_1 \cdot q|\ |r \cdot u_1 \cdot q|) = |u_1 \cdot q|$
unfolding *lenmorph* **by** *simp*

show *?thesis*
using *bin-mismatch-neq double-ruler*[*OF comp1 comp2*,*unfolded min*]
by (*simp add*: *lcp-mismatch-eq-len mismatch-incopm*)
qed

lemma *bin-per-root-max-pref-short*: **assumes** $(u_0 \cdot u_1) <p\ r \cdot u_0 \cdot u_1$ **and** $q \leq p$

w **and** $w \in \langle\{u_0, u_1\}\rangle$
shows $u_1 \cdot q \wedge_p r \cdot u_1 \cdot q = take\ |u_1 \cdot q|\ \alpha$
proof (*rule bin-lcp-per-root-max-pref-short*[*OF - - assms*(*2*−*3*)])
show $\alpha <p\ u_0 \cdot u_1 \wedge_p r \cdot u_0 \cdot u_1$
unfolding *lcp.absorb3*[*OF assms*(*1*)] **using** *bin-fst-mismatch'*[*THEN prefix-snocD*].
qed (*use assms*(*1*) **in** *blast*)

lemma *bin-root-max-pref-long*: **assumes** $r \cdot u_0 \cdot u_1 = u_0 \cdot u_1 \cdot r$ **and** $q \leq p\ w$ **and** $w \in \langle\{u_0, u_1\}\rangle$ **and** $|\alpha| \leq |q|$
shows $u_0 \cdot \alpha \leq p\ u_0 \cdot q \wedge_p r \cdot u_0 \cdot q$
proof (*rule pref-pref-lcp*)
have *len*: $|u_0 \cdot \alpha| \leq |r \cdot u_0 \cdot \alpha|$
by *simp*
from *bin-lcp-pref-all-len*[*OF assms*(*2*−*4*)]
show $u_0 \cdot \alpha \leq p\ u_0 \cdot q$
unfolding *pref-cancel-conv*.
have $u_0 \cdot \alpha \leq p\ r \cdot u_0 \cdot \alpha$
proof(*rule ruler-le*[*OF - - len*])
show $u_0 \cdot \alpha \leq p\ (r \cdot u_0 \cdot u_1) \cdot u_0 \cdot u_1$
unfolding *assms*(*1*) **unfolding** *rassoc pref-cancel-conv assms*(*1*)
using *pref-ext*[*OF pref-ext*[*OF bin-lcp-pref'*], *unfolded rassoc*].
show $r \cdot u_0 \cdot \alpha \leq p\ (r \cdot u_0 \cdot u_1) \cdot u_0 \cdot u_1$
unfolding *rassoc pref-cancel-conv* **using** *pref-ext*[*OF bin-lcp-pref'*, *unfolded rassoc*].
qed
from *pref-prolong*[*OF this*[*unfolded lassoc*], *OF* ‹$\alpha \leq p\ q$›, *unfolded rassoc*]
show $u_0 \cdot \alpha \leq p\ r \cdot u_0 \cdot q$.
qed

lemma *per-root-lcp-per-root*: $u_0 \cdot u_1 <p\ r \cdot u_0 \cdot u_1 \Longrightarrow \alpha \cdot [c_0] \leq p\ r \cdot \alpha$
using *per-root-pref-sing*[*OF - bin-fst-mismatch'*].

lemma *per-root-bin-fst-snd-lcp*: **assumes** $u_0 \cdot u_1 <p\ r \cdot u_0 \cdot u_1$ **and**
$q \leq p\ w$ **and** $w \in \langle\{u_0,u_1\}\rangle$ **and** $|\alpha| < |u_1 \cdot q|$
$q' \leq p\ w'$ **and** $w' \in \langle\{u_0,u_1\}\rangle$ **and** $|\alpha| \leq |q'|$
shows $u_1 \cdot q \wedge_p r \cdot q' = \alpha$
proof−
have *pref1*: $\alpha \cdot [c_1] \leq p\ u_1 \cdot q$
using ‹$|\alpha| < |u_1 \cdot q|$› ‹$q \leq p\ w$› *bin-snd-mismatch-all-len*[*of* $u_1 \cdot q$, *OF -* ‹$w \in \langle\{u_0,u_1\}\rangle$›]
unfolding *pref-cancel-conv* **by** *blast*

have $\alpha \leq p\ q'$
using ‹$|\alpha| \leq |q'|$› ‹$q' \leq p\ w'$› ‹$w' \in \langle\{u_0,u_1\}\rangle$› *bin-lcp-pref-all-len* **by** *blast*
have *pref2*: $\alpha \cdot [c_0] \leq p\ r \cdot \alpha$
using *assms*(*1*) *per-root-lcp-per-root* **by** *auto*
hence *pref2*: $\alpha \cdot [c_0] \leq p\ r \cdot q'$
using ‹$\alpha \leq p\ q'$› *pref-prolong* **by** *blast*

show *?thesis*
using *lcp-first-mismatch-pref*[*OF pref1 pref2 bin-mismatch-neq*[*symmetric*]]**.**

qed

end

lemmas allowing to translate properties of binary code to its roots

named-theorems *bin-code-primroots*
lemma *bin-lcp-eq-primroots* [*bin-code-primroots*]: **assumes** $x \cdot y \neq y \cdot x$
shows *bin-lcp* (ϱ *x*) (ϱ *y*) = *bin-lcp x y*
using *binary-code.prim-roots-lcp*[*OF binary-code.intro*[*OF assms*]]**.**

lemma *bin-lcs-eq-primroots* [*bin-code-primroots*]: **assumes** $x \cdot y \neq y \cdot x$
shows *bin-lcs* (ϱ *x*) (ϱ *y*) = *bin-lcs x y*
using *bin-lcp-eq-primroots*[*reversed*, *OF assms*[*symmetric*]]**.**

lemma *bin-mismatch-fst-eq-primroots* [*bin-code-primroots*]: **assumes** $x \cdot y \neq y \cdot x$
shows *bin-mismatch* (ϱ *x*) (ϱ *y*) = *bin-mismatch x y*
proof−
interpret *binary-code x y*
using *assms* **by** *unfold-locales*
interpret *r*: *binary-code* ϱ *x* ϱ *y*
using *assms* **by** *unfold-locales* (*simp add*: *comm-primroot-conv'*[*of x*])
have *r.bin-code-lcp* = *bin-code-lcp*
using *prim-roots-lcp***.**
have *bin-lcp* (ϱ *x*) (ϱ *y*) · [*bin-mismatch* (ϱ *x*) (ϱ *y*)] = *bin-lcp x y* · [*bin-mismatch x y*]
proof (*rule ruler-eq-len*[*OF - bin-fst-mismatch*], *unfold prim-roots-lcp*[*symmetric*])
show *r.bin-code-lcp* · [*r.bin-code-mismatch-fst*] ≤*p x* · *r.bin-code-lcp*
by (*subst* (*3*) *pop-primroot*[*of x*], *unfold rassoc*, *rule r.bin-fst-mismatch-all-hull*, *blast*)
qed *force*
then show *?thesis*
unfolding *bin-lcp-eq-primroots*[*OF assms*] *cancel* **by** *blast*
qed

lemmas *bin-mismatch-snd-eq-primroots*[*bin-code-primroots*] = *bin-mismatch-fst-eq-primroots*[*OF not-sym*] **and**
bin-mismatch-suf-fst-eq-primroots[*bin-code-primroots*] = *bin-mismatch-fst-eq-primroots*[*reversed*]
and
bin-mismatch-suf-snd-primroots[*bin-code-primroots*] = *bin-mismatch-fst-eq-primroots*[*reversed*, *OF not-sym*]

lemma *bin-lcp-eq-primroots'* [*bin-code-primroots*]: $x \cdot y \neq y \cdot x \Longrightarrow \varrho\ x \cdot \varrho\ y \wedge_p \varrho\ y \cdot \varrho\ x = x \cdot y \wedge_p y \cdot x$
using *bin-lcp-eq-primroots* **unfolding** *bin-lcp-def*.

lemmas *no-comm-bin-code* = *binary-code.bin-code*[*unfolded binary-code-def*]

theorem *bin-code-code*[*intro*]: **assumes** $u \cdot v \neq v \cdot u$ **shows** *code* $\{u, v\}$
unfolding *code-def* **using** *no-comm-bin-code*[*OF assms*] **by** *blast*

lemma *code-bin-code*: $u \neq v \Longrightarrow$ *code* $\{u,v\} \Longrightarrow u \cdot v \neq v \cdot u$
using *code.code-comm-eq*[*of* $\{u,v\}$ *u v*] **by** *blast*

lemma *lcp-roots-lcp*: $x \cdot y \neq y \cdot x \Longrightarrow x \cdot y \wedge_p y \cdot x = \varrho\ x \cdot \varrho\ y \wedge_p \varrho\ y \cdot \varrho\ x$
using *binary-code.prim-roots-lcp*[*unfolded binary-code-def bin-lcp-def*, *symmetric*].

lemma *sing-gen-primroot* [*simp*]: $u \in \langle\{\varrho\ u\}\rangle$
unfolding *sing-gen-pow-conv* **by** *simp*

lemma *sing-gen-pref-cancel* [*elim*]: $u \cdot v \in \langle\{r\}\rangle \Longrightarrow u \in \langle\{r\}\rangle \Longrightarrow v \in \langle\{r\}\rangle$
using *exp-pref-cancel*[*of - r v*] **unfolding** *sing-gen-pow-ex-conv* **by** *blast*

lemma *sing-gen-suf-cancel* [*elim*]: $u \cdot v \in \langle\{r\}\rangle \Longrightarrow v \in \langle\{r\}\rangle \Longrightarrow u \in \langle\{r\}\rangle$
using *exp-suf-cancel*[*of u - r*] **unfolding** *sing-gen-pow-ex-conv* **by** *blast*

lemma *prim-comm-root*[*elim*]: **assumes** *primitive r* **and** $u \cdot r = r \cdot u$ **shows** $u \in \langle\{r\}\rangle$
using ‹$u \cdot r = r \cdot u$›[*unfolded comm*] *prim-exp-eq*[*OF* ‹*primitive r*›] *pow-sing-gen* **by** *metis*

lemma *prim-root-drop-exp*[*elim*]: **assumes** $u^{@}k \in \langle\{r\}\rangle$ **and** $0 < k$ **and** *primitive r*
shows $u \in \langle\{r\}\rangle$
using *pow-list-comm-comm*[*of k u - r*, *OF* ‹$0 < k$›, *THEN prim-comm-root*[*OF* ‹*primitive r*›]]
‹$u^{@}k \in \langle\{r\}\rangle$› *sing-gen-power* **by** *blast*

lemma *per-root-trans*[*intro*]: **assumes** $w <p\ u \cdot w$ **and** $u \in \langle\{t\}\rangle$ **shows** $w <p\ t \cdot w$
by (*rule sing-genE*[*OF* ‹$u \in \langle\{t\}\rangle$› *per-root-drop-exp*]) (*use* ‹$w <p\ u \cdot w$› **in** *blast*)

lemma *per-root-trans'*[*intro*]: $w \leq p\ u \cdot w \Longrightarrow u \in \langle\{r\}\rangle \Longrightarrow u \neq \varepsilon \Longrightarrow w \leq p\ r \cdot w$
using *per-root-trans sprefD1 per-rootI* **by** *metis*

lemmas *per-root-trans-suf'*[*intro*] = *per-root-trans'*[*reversed*]

lemma *per-root-pref*: $w \neq \varepsilon \Longrightarrow w \in \langle\{r\}\rangle \Longrightarrow r \leq p\ w$

by (*rule hull.cases*) *blast+*

lemmas *per-root-suf* = *per-root-pref*[*reversed*]

lemma *comm-primrootE*: **assumes** $x \cdot y = y \cdot x$
obtains t **where** $x \in \langle\{t\}\rangle$ **and** $y \in \langle\{t\}\rangle$ **and** *primitive* t
using *comm-primroots assms prim-sing primroot-prim*
emp-in sing-gen-primroot **by** *metis*

lemma *root-trans*[*trans*]: $[\![v \in \langle\{u\}\rangle;\ u \in \langle\{t\}\rangle]\!] \Longrightarrow v \in \langle\{t\}\rangle$
by (*metis sing-gen-pow-ex-conv pow-mult*)

lemma *root-rev-iff*[*reversal-rule*]: $((rev\ u) \in \langle\{rev\ t\}\rangle) \longleftrightarrow (u \in \langle\{t\}\rangle)$
unfolding *sing-gen-pow-ex-conv rev-pow*[*symmetric*] **by** *blast*

3.9 Two words hull (not necessarily a code)

lemma *bin-lists-len-count*: **assumes** $x \neq y$ **and** $ws \in lists\ \{x,y\}$ **shows**
count-list ws x + *count-list ws y* = $|ws|$
proof−
have *finite* $\{x,y\}$ **by** *simp*
have *set ws* $\subseteq \{x,y\}$ **using** ‹*ws* ∈ *lists*{*x*,*y*}› **by** *blast*
show *?thesis*
using *sum-count-set*[*OF* ‹*set ws* ⊆ {*x*,*y*}› ‹*finite* {*x*,*y*}›] ‹*x* ≠ *y*› **by** *simp*
qed

lemma *two-elem-first-block*: **assumes** $w \in \langle\{u,v\}\rangle$
obtains m **where** $u^{@}m \cdot v \leq p\ w \vee w = u^{@}m$
using *assms*
proof (*induction rule*: *hull.induct*)
case (*prod-cl w1 w2*)
show *?case*
proof (*rule two-elem-cases*[*OF* ‹*w1* ∈ {*u*,*v*}›])
assume *w1* = *u*
from *prod-cl.IH*[*OF prod-cl.prems*, *of Suc*, *unfolded this pow-Suc*]
show *thesis*
by *simp*
next
assume *w1* = *v*
from *prod-cl.IH*[*OF prod-cl.prems*, *of* λ *-.0*, *unfolded pow-zero this*]
show *thesis*
by *simp*
qed
qed *force*

lemmas *two-elem-last-block* = *two-elem-first-block*[*reversed*]

lemma *two-elem-pref*: **assumes** $v \leq p\ u \cdot p$ **and** $p \in \langle\{u,v\}\rangle$

shows $v \leq_p u \cdot v$
proof (*rule two-elem-first-block*[*OF* ‹$p \in \langle\{u,v\}\rangle$›], *erule disjE*)
show $v \leq_p u \cdot v$ **if** $u \mathbin{@} m \cdot v \leq_p p$ **for** m
proof−
have $u^{@}(Suc\ m) \cdot v \leq_p u \cdot p$
unfolding *pow-Suc rassoc pref-cancel-conv* **by** *fact*
from *ruler-le*[*OF* ‹$v \leq_p u \cdot p$› *this*]
have $v \leq_p u \mathbin{@} Suc\ m \cdot v$
unfolding *lenmorph* **by** *linarith*
from *per-drop-exp*[*OF zero-less-Suc this*]
show $v \leq_p u \cdot v$**.**
qed
show $v \leq_p u \cdot v$ **if** $p = u \mathbin{@} m$ **for** m
using ‹$v \leq_p u \cdot p$›[*unfolded that*, *folded pow-Suc*]
by (*rule pref-pow-root*)
qed

lemmas *two-elem-suf* = *two-elem-pref*[*reversed*]

lemma *gen-drop-exp*: **assumes** $p \in \langle\{u,v^{@}(Suc\ k)\}\rangle$ **shows** $p \in \langle\{u,v\}\rangle$
by (*rule hull.induct*[*OF assms*], *simp*, *blast*)

lemma *gen-drop-exp-pos*: **assumes** $p \in \langle\{u,v^{@}k\}\rangle$ $0 < k$ **shows** $p \in \langle\{u,v\}\rangle$
using *gen-drop-exp*[*of - - k−1*, *unfolded Suc-minus-pos*[*OF* ‹$0 < k$›], *OF* ‹$p \in \langle\{u,v^{@}k\}\rangle$›]**.**

lemma *gen-prim*: $p \in \langle\{u,v\}\rangle \Longrightarrow p \in \langle\{u,\varrho\ v\}\rangle$
using *gen-drop-exp-pos primroot-expE* **by** *metis*

lemma *roots-hull*: **assumes** $w \in \langle\{u^{@}k,v^{@}m\}\rangle$ **shows** $w \in \langle\{u,v\}\rangle$
proof−
have $u^{@}k \in \langle\{u,v\}\rangle$ **and** $v^{@}m \in \langle\{u,v\}\rangle$
by (*simp-all add*: *gen-in power-in*)
hence $\{u^{@}k,v^{@}m\} \subseteq \langle\{u,v\}\rangle$
by *blast*
from *hull-mono′*[*OF this*]
show $w \in \langle\{u,v\}\rangle$
using ‹$w \in \langle\{u^{@}k,v^{@}m\}\rangle$› **by** *blast*
qed

lemma *in-hull-primroots*: $w \in \langle\{x,\ y\}\rangle \Longrightarrow w \in \langle\{\varrho\ x,\ \varrho\ y\}\rangle$
using *roots-hull*[*of w* $e_\varrho\ x\ \varrho\ x$ $e_\varrho\ y\ \varrho\ y$] **unfolding** *primroot-exp-eq***.**

lemma *roots-hull-sub*: $\langle\{u^{@}k,v^{@}m\}\rangle \subseteq \langle\{u,v\}\rangle$
by (*rule subsetI*; *rule roots-hull*)

lemma *primroot-gen*[*simp*,*intro*]: $v \in \langle\{u,\ \varrho\ v\}\rangle$
using *power-in*[*of* $\varrho\ v$ $\{u,\varrho\ v\}$]
by (*cases* $v = \varepsilon$, *simp*) (*metis primroot-expE gen-in insert-iff*)

lemma *primroot-gen'*[*simp*, *intro*]: $u \in \langle \{\varrho\ u,\ v\}\rangle$
using *primroot-gen insert-commute* **by** *metis*

lemma *lists-lists-gen-primroots* [*intro*]: $u \in lists\ \{x,\ y\} \Longrightarrow u \in lists\ \langle \{\varrho\ x,\ \varrho\ y\}\rangle$
using *lists-mono*[*OF genset-sub*] **by** *fast*

lemma *dec-primroot-bin-sing* [*intro*]: **assumes** $a \in \{x,y\}\ x \cdot y \neq y \cdot x$
shows *Dec* $\{\varrho\ x,\ \varrho\ y\}\ a = [\varrho\ a]^{@} e_{\varrho}\ a$
using *assms* **by** (*auto intro*: *binary-code.primroot-dec*[*OF binary-code.intro*[*OF* ‹$x \cdot y \neq y \cdot x$›]])

lemma *ref-primroot-bin-sing*: **assumes** $a \in \{x,y\}\ x \cdot y \neq y \cdot x$
shows *Ref* $\{\varrho\ x,\ \varrho\ y\}\ [a] = [\varrho\ a]^{@} e_{\varrho}\ a$
by (*rule code.code-unique-ref*[*of* $\{\varrho\ x,\ \varrho\ y\}$ [*a*], *unfolded dec-primroot-bin-sing*[*OF assms*] *concat-simps*])
(*use assms* **in** *blast*, *use assms* **in** *force*)

lemma *lists-sub-mono-gen*: $ws \in lists\ S \Longrightarrow S \subseteq \langle G\rangle \Longrightarrow ws \in lists\ \langle G\rangle$
using *sub-lists-mono*.

3.9.1 Binary Mismatch tools

definition *bin-mismatch-hard* :: *'a list* ⇒ *'a list* ⇒ *'a list* ⇒ *bool* **where**
bin-mismatch-hard x y w ≡ ∃ *k*. $\varrho\ x \cdot (\varrho\ x)^{@} k \cdot \varrho\ y \leq_p w$

lemma *bin-mismatch-hard-def'*: **assumes** $x \cdot y \neq y \cdot x$ *bin-mismatch-hard x y w*
shows *bin-lcp x y* · [*bin-mismatch x y*] ≤*p w*
proof−
from ‹$x \cdot y \neq y \cdot x$›
interpret *binary-code* $\varrho\ x\ \varrho\ y$
by *unfold-locales blast*
find-theorems *bin-lcp ?x ?y* · [*bin-mismatch ?x ?y*]
from *bin-lcp-fst-pow-pref*[*OF zero-less-Suc*, *of* - ε, *unfolded emp-simps*] ‹*bin-mismatch-hard x y w*›[*unfolded bin-mismatch-hard-def lassoc pow-Suc*[*symmetric*]]
show *?thesis*
unfolding *bin-code-primroots*[*OF* ‹$x \cdot y \neq y \cdot x$›] **using** *pref-trans* **by** *blast*
qed

definition *bin-mismatch-pref* :: *'a list* ⇒ *'a list* ⇒ *'a list* ⇒ *bool* **where**
bin-mismatch-pref x y w ≡ ∃ *k*. $\varrho\ x^{@} k \cdot \varrho\ y \leq_p w$

lemma *bm-pref-letter*: **assumes** $x \cdot y \neq y \cdot x$ **and** *bin-mismatch-pref x y* ($w \cdot \varrho\ y$)
shows *bin-lcp x y* · [*bin-mismatch x y*] ≤*p x* · *w* · *bin-lcp x y* (**is** *?α* · [*?c*] ≤*p x* · *w* · *?α*)
proof−
from *binary-code.code-bin-roots*[*OF binary-code.intro*, *OF* ‹$x \cdot y \neq y \cdot x$›]
interpret *binary-code* $\varrho\ x\ \varrho\ y$.

write *bin-code-lcp* (α) **and** *bin-code-mismatch-fst* (c_x) **and** *bin-code-mismatch-snd* (c_y)
have $x \neq \varepsilon$
using ‹$x \cdot y \neq y \cdot x$› **by** *blast*
from *exE*[*OF assms*(*2*)[*unfolded bin-mismatch-pref-def*]]
obtain k **where** *pref-x*: $\varrho\ x \ ^{@}\ (e_\varrho\ x + k) \cdot \varrho\ y \leq p\ x \cdot w \cdot \varrho\ y$
unfolding *pow-add primroot-exp-eq rassoc pref-cancel-conv*.
have *mismatch-x*: $\alpha \cdot [c_x] \leq p\ \varrho\ x \ ^{@}\ (e_\varrho\ x + k) \cdot \varrho\ y$
using *bin-lcp-fst-pow-pref*[*of* $e_\varrho\ x + k\ \varepsilon$, *unfolded emp-simps*]
‹$x \neq \varepsilon$›[*folded primroot-exp-zero-conv*] **by** *blast*
have $\alpha \cdot [c_x] \leq p\ x \cdot w \cdot \alpha$
by (*rule ruler-le*[*OF pref-trans*[*OF mismatch-x pref-ext*[*OF pref-x*, *unfolded rassoc*]]],
unfold pref-cancel-conv, *rule bin-lcp-pref'*)
(*unfold pref-cancel-conv lenmorph sing-len*, *use nemp-len*[*OF* ‹$x \neq \varepsilon$›] **in** *linarith*)
then show *?thesis*
unfolding *bin-code-primroots*[*OF* ‹$x \cdot y \neq y \cdot x$›].
qed

— Binary mismatch elims
named-theorems *bm-elims*

lemma *bm-eq1* [*bm-elims*]: **assumes** $x \cdot w1 = y \cdot w2$ **and** *bin-mismatch-pref x y* ($w1 \cdot \varrho\ y$) **and** *bin-mismatch-pref y x* ($w2 \cdot \varrho\ x$)
shows $x \cdot y = y \cdot x$
proof(*rule nemp-comm*, *rule ccontr*)
assume *a*: $x \cdot y \neq y \cdot x$
from *binary-code.code-bin-roots*[*OF binary-code.intro*, *OF* ‹$x \cdot y \neq y \cdot x$›]
interpret *binary-code* $\varrho\ x\ \varrho\ y$.
from *bm-pref-letter*[*OF a assms*(*2*), *unfolded lassoc* ‹$x \cdot w1 = y \cdot w2$›]
bm-pref-letter[*OF a*[*symmetric*] *assms*(*3*), *unfolded lassoc*]
show *False*
unfolding *bin-lcp-sym*[*of y x*] *bin-code-primroots*[*OF a*, *symmetric*]
using *same-sing-pref bin-mismatch-neq* **by** *fast*
qed

lemma *bm-eq2* [*bm-elims*]: **assumes** $\varrho\ x \cdot w1 = y \cdot w2$ **and** *bin-mismatch-pref x y* ($w1 \cdot \varrho\ y$) **and** *bin-mismatch-pref y x* ($w2 \cdot \varrho\ x$)
shows $x \cdot y = y \cdot x$
using *bm-eq1*[*OF* ‹$\varrho\ x \cdot w1 = y \cdot w2$›] *assms*(*2−3*) **unfolding** *bin-mismatch-pref-def primroot-idemp*
comm-primroot-conv[*of* $\varrho\ x\ y$] **using** *comm-primroot-conv'*[*of x y*, *symmetric*] **by** *argo*

lemma *bm-eq3* [*bm-elims*]: **assumes** $x \cdot w1 = \varrho\ y \cdot w2$ **and** *bin-mismatch-pref x y* ($w1 \cdot \varrho\ y$) **and** *bin-mismatch-pref y x* ($w2 \cdot \varrho\ x$)
shows $x \cdot y = y \cdot x$
using *bm-eq1*[*OF* ‹$x \cdot w1 = \varrho\ y \cdot w2$›] *assms*(*2−3*) **unfolding** *bin-mismatch-pref-def primroot-idemp*

comm-primroot-conv[*of* ϱ *x y*] **using** *comm-primroot-conv*[*of x y*, *symmetric*] **by** *blast*

lemma *bm-eq4* [*bm-elims*]: **assumes** $\varrho\ x \cdot w1 = \varrho\ y \cdot w2$ **and** *bin-mismatch-pref x y* $(w1 \cdot \varrho\ y)$ **and** *bin-mismatch-pref y x* $(w2 \cdot \varrho\ x)$
shows $x \cdot y = y \cdot x$
using *bm-eq3*[*OF* ‹$\varrho\ x \cdot w1 = \varrho\ y \cdot w2$›, *symmetric*] *assms*(*2*−*3*) **unfolding** *bin-mismatch-pref-def primroot-idemp comm-primroot-conv'*[*of x y*, *symmetric*] **using** *comm-primroot-conv*[*of y x*] **by** *presburger*

lemma *bm-hard-lcp* [*bm-elims*]: **assumes** $x \cdot y \neq y \cdot x$ **and** *bin-mismatch-hard x y w1* **and** *bin-mismatch-hard y x w2*
shows $w1 \wedge_p w2 = x \cdot y \wedge_p y \cdot x$
proof−
interpret *binary-code* $\varrho\ x\ \varrho\ y$
using ‹$x \cdot y \neq y \cdot x$› **by** *unfold-locales blast*
show *?thesis*
using *bin-mismatch-hard-def'*[*OF assms*(*1*−*2*)] *bin-mismatch-hard-def'*[*OF assms*(*1*)[*symmetric*] *assms*(*3*)]
unfolding *bin-code-primroots*[*OF assms*(*1*), *symmetric*] *bin-lcp-def*[*symmetric*] *bin-lcp-sym*[*of y*]
prefix-def rassoc bin-mismatch-hard-def
using *lcp-first-mismatch*[*OF bin-mismatch-neq*] **by** *blast*
qed

lemma *bin-mismatch-pref-ext*: *bin-mismatch-pref x y w1* $\Longrightarrow$ *bin-mismatch-pref x y* $(w1 \cdot z)$
unfolding *bin-mismatch-pref-def* **using** *pref-ext* **by** *blast*

lemma *bm-pref1* [*bm-elims*]: **assumes** $x \cdot w1 \leq_p y \cdot w2$ **and** *bin-mismatch-pref x y w1*
and *bin-mismatch-pref y x* $(w2 \cdot \varrho\ x)$
shows $x \cdot y = y \cdot x$
using *bm-eq1*[*OF lq-pref*[*OF* ‹$x \cdot w1 \leq_p y \cdot w2$›, *unfolded rassoc*]]
bin-mismatch-pref-ext[*OF assms*(*2*)] *assms*(*3*) **unfolding** *rassoc* **by** *presburger*

lemma *bm-pref2* [*bm-elims*]: **assumes** $\varrho\ x \cdot w1 \leq_p y \cdot w2$ **and** *bin-mismatch-pref x y w1*
and *bin-mismatch-pref y x* $(w2 \cdot \varrho\ x)$
shows $x \cdot y = y \cdot x$
using *bm-eq2*[*OF lq-pref*[*OF* ‹$\varrho\ x \cdot w1 \leq_p y \cdot w2$›, *unfolded rassoc*]] *bin-mismatch-pref-ext*[*OF assms*(*2*)]
assms(*3*) **unfolding** *rassoc* **by** *presburger*

lemma *bm-pref3* [*bm-elims*]: **assumes** $x \cdot w1 \leq_p \varrho\ y \cdot w2$ **and** *bin-mismatch-pref x y w1*
and *bin-mismatch-pref y x* $(w2 \cdot \varrho\ x)$
shows $x \cdot y = y \cdot x$
using *bm-eq3*[*OF lq-pref*[*OF* ‹$x \cdot w1 \leq_p \varrho\ y \cdot w2$›, *unfolded rassoc*]] *bin-mismatch-pref-ext*[*OF*

assms(*2*)]
assms(*3*) **unfolding** *rassoc* **by** *presburger*

lemma *bm-pref4* [*bm-elims*]: **assumes** $\varrho\ x \cdot w1 \leq p\ \varrho\ y \cdot w2$ **and** *bin-mismatch-pref x y w1*
and *bin-mismatch-pref y x* $(w2 \cdot \varrho\ x)$
shows $x \cdot y = y \cdot x$
using *bm-eq4*[*OF lq-pref*[*OF* ‹$\varrho\ x \cdot w1 \leq p\ \varrho\ y \cdot w2$›, *unfolded rassoc*]] *bin-mismatch-pref-ext*[*OF assms*(*2*)] *assms*(*3*) **unfolding** *rassoc* **by** *presburger*

lemmas [*bm-elims*] = *bm-elims*[*symmetric*]

— Binary mismatch predicate evaluation

named-theorems *bm-simps*
lemma *bm-mismatch-rhoI1* [*bm-simps*]: **assumes** *bin-mismatch-pref x y w* **shows** *bin-mismatch-hard x y* $(\varrho\ x \cdot w)$
using *assms* **unfolding** *bin-mismatch-pref-def bin-mismatch-hard-def*
by *blast*

lemma *bm-mismatchI1* [*bm-simps*]: **assumes** *bin-mismatch-pref x y w* **shows** *bin-mismatch-hard x y* $(x \cdot w)$
proof−
from *assms*[*unfolded bin-mismatch-pref-def*]
obtain k **where** $\varrho\ x\ ^{@}\ k \cdot \varrho\ y \leq p\ w$
by *blast*
have $\varrho\ x \cdot \varrho\ x\ ^{@}\ (e_\varrho\ x - 1 + k) \cdot \varrho\ y \leq p\ x \cdot w$
by (*subst* (*4*) *pop-primroot*[*of x*], *unfold pow-add rassoc pref-cancel-conv*) *fact*
then show *?thesis*
unfolding *bin-mismatch-hard-def* **by** *blast*
qed

lemma *bm-mismatch-rhoI2′* [*bm-simps*]: **assumes** *bin-mismatch-pref x y w* **shows** *bin-mismatch-pref x y* $(\varrho\ x^{@}k \cdot w)$
unfolding *bin-mismatch-pref-def*
proof (*rule exE*[*OF assms*[*unfolded bin-mismatch-pref-def*]])
fix m
assume $\varrho\ x\ ^{@}\ m \cdot \varrho\ y \leq p\ w$
show $\exists ka.\ \varrho\ x\ ^{@}\ ka \cdot \varrho\ y \leq p\ \varrho\ x\ ^{@}\ k \cdot w$
by (*rule exI*[*of - k+m*], *unfold pow-add rassoc pref-cancel-conv*) *fact*
qed

lemma *bm-mismatch-rhoI2* [*bm-simps*]: **assumes** *bin-mismatch-pref x y w* **shows** *bin-mismatch-pref x y* $(\varrho\ x \cdot w)$
using *bm-mismatch-rhoI2′*[*OF assms*, *of 1*, *unfolded pow-list-1*].

lemma *bm-mismatchI2* [*bm-simps*]: **assumes** *bin-mismatch-pref x y w* **shows** *bin-mismatch-pref x y* $(x \cdot w)$
using *bm-mismatch-rhoI2′*[*OF assms*, *of* e_ϱ *x*, *unfolded primroot-exp-eq*].

lemma *bm-mismatchI2′* [*bm-simps*]: **assumes** *bin-mismatch-pref x y w* **shows** *bin-mismatch-pref x y* ($x^{@}k \cdot w$)
using *bm-mismatch-rhoI2′*[*OF assms*, *of* e_ϱ $x * k$, *unfolded pow-mult primroot-exp-eq*].

lemma [*bm-simps*]: *bin-mismatch-pref x y* ($y \cdot v$)
unfolding *bin-mismatch-pref-def* **using** *append-Nil*[*of* ϱ *y*, *folded pow-zero*[*of* ϱ *x*]]
pref-ext[*OF primroot-pref*[*of y*]] **by** *metis*

lemma [*bm-simps*]: *bin-mismatch-pref x y y*
unfolding *bin-mismatch-pref-def* **using** *append-Nil*[*of* ϱ *y*, *folded pow-zero*[*of* ϱ *x*]] *primroot-pref*[*of y*] **by** *metis*

lemma [*bm-simps*]: **assumes** *w1* $\in \langle\{x,y\}\rangle$ *bin-mismatch-pref x y w*
shows *bin-mismatch-pref x y* (*w1* $\cdot$ *w*)
proof−
obtain *k* **where** $\varrho\ x^{@}k \cdot \varrho\ y \leq_p w$
using ‹*bin-mismatch-pref x y w*›[*unfolded bin-mismatch-pref-def*] **by** *blast*
obtain *m* **where** *or*: $x\ ^{@}\ m \cdot y \leq_p w1 \vee w1 = x\ ^{@}\ m$
using *two-elem-first-block*[*OF* ‹*w1* $\in \langle\{x,y\}\rangle$›] **by** *blast*
show *bin-mismatch-pref x y* (*w1* $\cdot$ *w*)
unfolding *bin-mismatch-pref-def*
proof (*rule disjE*[*OF or*])
assume $x\ ^{@}\ m \cdot y \leq_p w1$
from *pref-shorten*[*OF primroot-pref this*]
have $\varrho\ x^{@}(e_\varrho\ x * m) \cdot \varrho\ \ y \leq_p w1$
unfolding *pow-mult primroot-exp-eq*.
from *pref-ext*[*OF this*]
show $\exists k.\ \varrho\ x\ ^{@}\ k \cdot \varrho\ y \leq_p w1 \cdot w$
by *blast*
next
assume $w1 = x\ ^{@}\ m$
have $\varrho\ x^{@}(e_\varrho\ x * m + k) \cdot \varrho\ y \leq_p w1 \cdot w$
unfolding *pow-add pow-mult primroot-exp-eq* ‹$w1 = x\ ^{@}\ m$› *pref-cancel-conv*
using ‹$\varrho\ x^{@}k \cdot \varrho\ y \leq_p w$› **by** *fastforce*
then show $\exists k.\ \varrho\ x\ ^{@}\ k \cdot \varrho\ y \leq_p w1 \cdot w$
by *blast*
qed
qed

lemma *bm-pref-triv-rho* [*bm-simps*]: *bin-mismatch-pref x y* ($\varrho\ y \cdot w$)
unfolding *bin-mismatch-pref-def* **using** *append-Nil*[*of* ϱ *y*,*folded pow-zero*[*of* ϱ *x*]] **by** *blast*

lemma *bm-pref-fst-rho* [*bm-simps*]: *bin-mismatch-pref x y* ($\varrho\ y$)
unfolding *bin-mismatch-pref-def* **using** *append-Nil*[*of* ϱ *y*,*folded pow-zero*[*of* ϱ *x*]] *self-pref*[*of* ϱ *y*] **by** *metis*

— Binary hull membership evaluation

lemma[*bm-simps*]: $x \in \langle\{x,y\}\rangle$
  by *blast*
lemma[*bm-simps*]: $y \in \langle\{x,y\}\rangle$
  by *blast*
lemma[*bm-simps*]: $w \in \langle\{x,y\}\rangle \longleftrightarrow w \in \langle\{y,x\}\rangle$
  by (*simp add*: *insert-commute*)

lemmas[*bm-simps*] = *hull-closed power-in*
lemmas [*bm-simps*] = *lcp-ext-left*

— the method setup

method *mismatch0* =
  (*insert method-facts*, *use nothing* **in**
    ‹(
      ((*simp only*: *shifts bm-simps*)*?*,
      (*elim bm-elims*);(*simp-all only*: *shifts bm-simps*))
)›
)

lemmas *bm-simps-rev* = *bm-simps*[*reversed*]
lemmas *bm-elims-rev* = *bm-elims*[*reversed*]

method *mismatch-rev* =
  (*insert method-facts*, *use nothing* **in**
    ‹(
      ((*simp only*: *shifts-rev bm-simps-rev*)*?*,
      (*elim bm-elims-rev*);(*simp-all only*: *shifts-rev bm-simps-rev*))
)›
)

method *mismatch* =
  (*insert method-facts*, *use nothing* **in**
    ‹(*mismatch0*;*fail*|*mismatch-rev*)›
)

find-theorems *name*: *bm-elims ?x* $\wedge_p$ *?y*

Mismatch method demonstrations

lemma assumes $x \cdot y \cdot z = y \cdot y \cdot x \cdot v$ **shows** $x \cdot y = y \cdot x$
  using *assms* **by** *mismatch*

lemma assumes $y \cdot x \cdot x \cdot y \cdot z = (y \cdot x \cdot y) \cdot y \cdot x \cdot v$ **shows** $x \cdot y = y \cdot x$ —
cancel
  using *assms* **by** *mismatch*

lemma $y \cdot y \cdot x \cdot v = x \cdot x \cdot y \cdot z \Longrightarrow x \cdot y = y \cdot x$ — swap
by *mismatch*

lemma $\varrho\ x \cdot \varrho\ x^{@}k \cdot y = y \cdot w \cdot \varrho\ x \Longrightarrow w \in \langle\{x,y\}\rangle \Longrightarrow x \cdot y = y \cdot x$ — primitive root and hull
by *mismatch*

lemma $(r \cdot q) \cdot (r \cdot q)\ ^{@}\ t \cdot r \cdot (q \cdot r \cdot r \cdot q)\ ^{@}\ k = ((r \cdot q)\ ^{@}\ t \cdot r \cdot (q \cdot r \cdot r \cdot q)\ ^{@}\ k) \cdot r \cdot q \Longrightarrow$
$0 < k \Longrightarrow r \cdot q = q \cdot r$ — power
by *mismatch*

lemma $((u \cdot v)\ ^{@}\ m \cdot u) \cdot v \cdot (u \cdot v)\ ^{@}\ k = (v \cdot (u \cdot v)\ ^{@}\ k) \cdot (u \cdot v)\ ^{@}\ m \cdot u \Longrightarrow u \cdot v = v \cdot u$
by *mismatch*

lemma $w1 \in \langle\{x,y\}\rangle \Longrightarrow y \cdot x \cdot w2 \cdot z = x \cdot w1 \Longrightarrow x \cdot y = y \cdot x$
by *mismatch*

lemma $x \cdot y \cdot u \cdot x \cdot y = y \cdot v \cdot y \cdot x \Longrightarrow x \cdot y = y \cdot x$ — reverse mismatch
apply *mismatch-rev*.

lemma $z \cdot x \cdot y \cdot x \cdot x = v \cdot x \cdot y \cdot y \Longrightarrow y \cdot x = x \cdot y$ — reverse mismatch
by *mismatch*

lemma $k \neq 0 \Longrightarrow j \neq 0 \Longrightarrow (x\ ^{@}\ j \cdot y\ ^{@}\ k) \cdot y = y^{@}k \cdot x\ ^{@}\ j \cdot y\ ^{@}\ (k - 1) \Longrightarrow x \cdot y = y \cdot x$
by *mismatch*

— prefixes
lemma $\varrho\ x \cdot \varrho\ x^{@}k \cdot y \leq p\ y \cdot w \cdot \varrho\ x \Longrightarrow w \in \langle\{x,y\}\rangle \Longrightarrow x \cdot y = y \cdot x$
by *mismatch*

lemma $y \cdot x \leq p\ x^{@}k \cdot x \cdot y \cdot w \Longrightarrow x \cdot y = y \cdot x$
by *mismatch*

lemma $0 < k \Longrightarrow y \cdot x \leq p\ x^{@}k \cdot y \cdot w \Longrightarrow x \cdot y = y \cdot x$
by *mismatch*

lemma $0 < k \Longrightarrow 0 < l \Longrightarrow y^{@}l \cdot x \leq p\ x^{@}k \cdot y \cdot w \Longrightarrow x \cdot y = y \cdot x$
by *mismatch*

lemma $0 < k \Longrightarrow 1 < l \Longrightarrow y^{@}l \cdot x \leq p\ y \cdot x^{@}k \cdot y \cdot w \Longrightarrow x \cdot y = y \cdot x$
by *mismatch*

lemma $0 < k \Longrightarrow m < l \Longrightarrow y^{@}l \cdot x \leq p\ y^{@}m \cdot x^{@}k \cdot y \cdot w \Longrightarrow x \cdot y = y \cdot x$
by *mismatch*

lemma assumes $0 < k\ m < l\ y^{@}l \cdot x \leq_p y^{@}m \cdot x^{@}k \cdot y \cdot w$ **shows** $x \cdot y = y \cdot x$
using *assms* **by** *mismatch*

lemma $w1 \in \langle\{x,y\}\rangle \Longrightarrow w2 \in \langle\{x,y\}\rangle \Longrightarrow x \cdot w2 \cdot y \cdot z = y \cdot w1 \cdot x \cdot v \Longrightarrow x \cdot y = y \cdot x$
by *mismatch*

— suffixes

lemma assumes $x \cdot x \cdot y \cdot y \cdot y \cdot y \leq_s z \cdot y \cdot y \cdot x \cdot x$ **shows** $x \cdot y = y \cdot x$
using *assms* **by** *mismatch*

lemma $u \cdot v\ ^{@}\ k \cdot v \leq_s v \cdot w \cdot u \Longrightarrow w \in \langle\{u, v\}\rangle \Longrightarrow u \cdot v = v \cdot u$
by *mismatch*

lemma $u \cdot \varrho\ v\ ^{@}\ k \cdot \varrho\ v \leq_s \varrho\ v \cdot w \cdot u \Longrightarrow w \in \langle\{u, v\}\rangle \Longrightarrow u \cdot v = v \cdot u$
by *mismatch*

lemma $u \cdot v\ ^{@}\ k \cdot \varrho\ v \leq_s \varrho\ v \cdot w \cdot u \Longrightarrow w \in \langle\{u, v\}\rangle \Longrightarrow u \cdot v = v \cdot u$
by *mismatch*

lemma $2 \leq j \Longrightarrow q \cdot p \cdot q \leq_s (q \cdot p)\ ^{@}\ et' \cdot q\ ^{@}\ j \Longrightarrow p \cdot q = q \cdot p$
by *mismatch*

lemma $w1 \in \langle\{x,y\}\rangle \Longrightarrow w2 \in \langle\{x,y\}\rangle \Longrightarrow x \cdot y \cdot w2 \cdot x \leq_s x \cdot w1 \cdot y \Longrightarrow x \cdot y = y \cdot x$
by *mismatch*

lemma $w \in \langle\{x,y\}\rangle \Longrightarrow w' \in \langle\{x,y\}\rangle \Longrightarrow$ *bin-mismatch-pref* $x\ \ y\ (w \cdot w \cdot y)$
by (*simp add*: *bm-simps*)

— Known issue: In he next example, the method fails because the assumption simplifies the goal; on the other hand, avoiding simplification would fail to use the positivity of k
lemma $x \cdot y = y^{@}k \Longrightarrow 0 < k \Longrightarrow x \cdot y = y \cdot x$
oops

— least common prefix

lemma assumes $x \cdot y \neq y \cdot x$
shows $x \cdot x \cdot y \wedge_p y \cdot y \cdot x = (x \cdot y \wedge_p y \cdot x)$
using *assms* **by** *mismatch*

lemma assumes $x \cdot y \neq y \cdot x$
shows $w \cdot z \cdot x \cdot x \cdot y \wedge_p w \cdot z \cdot y \cdot y \cdot x = (w \cdot z) \cdot (x \cdot y \wedge_p y \cdot x)$
using *assms* **by** *mismatch*

lemma assumes $x \cdot y \neq y \cdot x$

shows $y \cdot y \cdot x \wedge_p x \cdot x \cdot y = (x \cdot y \wedge_p y \cdot x)$
using *assms*[*symmetric*] **unfolding** *lcp-sym*[*of* $x \cdot y$] — lcp mismatch is order sensitive
by *mismatch*

— least common suffix

lemma assumes $x \cdot y \neq y \cdot x$
shows $x \cdot x \cdot y \wedge_s y \cdot y \cdot x = (x \cdot y \wedge_s y \cdot x)$
using *assms* **by** *mismatch*

lemma assumes $x \cdot y \neq y \cdot x$
shows $x \cdot x \cdot y \cdot z \cdot y \wedge_s y \cdot y \cdot x \cdot z \cdot y = (x \cdot y \wedge_s y \cdot x) \cdot (z \cdot y)$
using *assms* **by** *mismatch*

3.9.2 Applied mismatch

lemma assumes $x^{@}j = y^{@}k$ $0 < j$ **shows** $x \cdot y = y \cdot x$
using *arg-cong*[*OF assms*(*1*), *of* λ *z*. $z \cdot y$] ‹$0 < j$›
by *mismatch*

lemma *pows-comm-comm*: **assumes** $u^{@}k \cdot v^{@}m = u^{@}l \cdot v^{@}n$ $k \neq l$ **shows** $u \cdot v = v \cdot u$
proof–
have *aux*: $u^{@}k \cdot v^{@}m \cdot v \cdot u = u^{@}l \cdot v^{@}n \cdot v \cdot u \Longrightarrow k \neq l \Longrightarrow u \cdot v = v \cdot u$
proof (*induct k l rule*: *diff-induct*) **qed** (*mismatch*, *mismatch*, *fastforce*)
— the third goal has nothing to do with mismatch, therefore trying to make mismatch+ succeed is misleading
from *this*[*unfolded lassoc cancel-right*, *OF assms*]
show $u \cdot v = v \cdot u$**.**
qed

— The following is needed in binary interpretations
lemma *sq-not-pref-mesosome-cover*: **assumes** *eq*: $x \cdot x \cdot concat\ ws = p \cdot x \cdot y^{@}k \cdot x \cdot s$ **and** $ws \in lists\ \{x,y\}$ $p <p\ x\ p \neq \varepsilon$ ‹$1 < k$›
shows $x \cdot y = y \ \cdot x$
proof–
define z **where** $z = (p \cdot x)\ ^{-1>}\ (x \cdot x)$
have $p \cdot x <p\ x \cdot x$
by (*rule ruler-less*[*of* - ‹$x \cdot x \cdot concat\ ws$›], *simp add*: *eq*, *simp-all add*: *spref-len*[*OF* ‹$p <p\ x$›])
from *mid-sq-primroot*[*OF this*, *folded z-def*]
have $(\varrho\ x)^{@}e_\varrho\ z \ \cdot concat\ ws = y^{@}k \cdot x \cdot s$
using *eq* **unfolding** *lassoc* ‹$x \cdot x = p \cdot x \cdot \varrho\ x\ ^{@}\ e_\varrho\ z$› **unfolding** *rassoc cancel* **by** *blast*
thus $x \cdot y = y \ \cdot x$
using *concat-in-hull'*[*OF* ‹$ws \in lists\ \{x,y\}$›]
unfolding *pow-pos*[*OF* ‹$0 < e_\varrho\ z$›] *pop-pow-2*[*OF* ‹$1 < k$›, *symmetric*] *rassoc*
by *mismatch*

qed

lemma (**in** *binary-code*) *bin-mismatch-pows*: $u_0 \,^{@}\, Suc\ k \cdot u_1 \cdot z \neq u_1 \,^{@}\, Suc\ l \cdot u_0 \cdot z'$
proof
  assume $u_0 \,^{@}\, Suc\ k \cdot u_1 \cdot z = u_1 \,^{@}\, Suc\ l \cdot u_0 \cdot z'$
  then have $u_0 \cdot u_1 = u_1 \cdot u_0$
    by *mismatch*
  then show *False*
    using *non-comm* **by** *contradiction*
qed

3.10 Free hull

While not every set G of generators is a code, there is a unique minimal free monoid containing it, called the *free hull* of G. It can be defined inductively using the property known as the *stability condition*.

inductive-set *free-hull* :: *'a list set* ⇒ *'a list set* ($\langle\text{-}\rangle_F$)
  for G **where**
    $\varepsilon \in \langle G\rangle_F$
  | *free-gen-in*: $w \in G \Longrightarrow w \in \langle G\rangle_F$
  | *w1* $\in \langle G\rangle_F \Longrightarrow$ *w2* $\in \langle G\rangle_F \Longrightarrow$ *w1* $\cdot$ *w2* $\in \langle G\rangle_F$
  | *stability*: $p \in \langle G\rangle_F \Longrightarrow q \in \langle G\rangle_F \Longrightarrow p \cdot w \in \langle G\rangle_F \Longrightarrow w \cdot q \in \langle G\rangle_F \Longrightarrow w \in \langle G\rangle_F$ — the stability condition

lemmas [*intro*] = *free-hull.intros*

The defined set indeed is a hull.

lemma *free-hull-hull*[*simp*]: $\langle\langle G\rangle_F\rangle = \langle G\rangle_F$
  by (*intro antisym subsetI*, *rule hull.induct*) *blast+*

The free hull is always (non-strictly) larger than the hull.

lemma *hull-sub-free-hull*: $\langle G\rangle \subseteq \langle G\rangle_F$
proof
  fix *x* **assume** $x \in \langle G\rangle$
  then show $x \in \langle G\rangle_F$
    using *free-hull.intros*(*3*)
      gen-monoid-induct[*of x G* λ *x*. $x \in \langle G\rangle_F$, *OF* ‹$x \in \langle G\rangle$› *free-hull.intros*(*1*)[*of G*] *free-hull.intros*(*2*)]
    by *auto*
qed

On the other hand, it can be proved that the *free basis*, defined as the basis of the free hull, has a (non-strictly) smaller cardinality than the ordinary basis. (See the AFP theory Combinatorics-Words-Graph-Lemma.Graph-Lemma)

definition *free-basis* :: *'a list set* ⇒ *'a list set* ($\mathfrak{B}_F$ - [*54*] *55*)
  where *free-basis* $G \equiv \mathfrak{B}\ \langle G\rangle_F$

lemma *basis-gen-hull-free*: $\langle \mathfrak{B}_F\ G\rangle = \langle G\rangle_F$
unfolding *free-basis-def basis-gen-hull free-hull-hull*..

lemma *genset-sub-free*: $G \subseteq \langle G\rangle_F$
by (*simp add*: *free-hull.free-gen-in subsetI*)

We have developed two points of view on freeness:

- inductive point of view: to satisfy the stability condition;
- being generated by a code.

We now show their equivalence

First, basis of a free hull is a code.

lemma *free-basis-code*[*simp*]: *code* $(\mathfrak{B}_F\ G)$
proof (*rule code.intro*)
fix *xs ys*
show *xs* $\in$ *lists* $(\mathfrak{B}_F\ G) \Longrightarrow$ *ys* $\in$ *lists* $(\mathfrak{B}_F\ G) \Longrightarrow$ *concat xs* = *concat ys* $\Longrightarrow$ *xs* = *ys*
proof(*induction xs ys rule*: *list-induct2'*)
case (*2 x xs*)
have $x = \varepsilon$
using ‹*concat* $(x \# xs)$ = *concat* ε› **by** *force*
have $x \in \mathfrak{B}\ \langle G\rangle_F$
using ‹$x \# xs \in$ *lists* $(\mathfrak{B}_F\ G)$› **unfolding** *free-basis-def* **by** *force*
from *emp-not-basis*[*OF this*]
have $x \neq \varepsilon$.
then show *?case*
using ‹$x = \varepsilon$› **by** *contradiction*
next
case (*3 y ys*)
have $y = \varepsilon$
using ‹*concat* ε = *concat* $(y \# ys)$› **by** *force*
have $y \in \mathfrak{B}\ \langle G\rangle_F$
using ‹$y \# ys \in$ *lists* $(\mathfrak{B}_F\ G)$› **unfolding** *free-basis-def* **by** *force*
from *emp-not-basis*[*OF this*]
have $y \neq \varepsilon$.
then show *?case*
using ‹$y = \varepsilon$› **by** *contradiction*
next
case (*4 x xs y ys*)
show *?case*
proof (*unfold list.inject*, *rule conjI*)
have *in-free-hull*: $x \in \langle G\rangle_F\ y \in \langle G\rangle_F$ *concat xs* $\in \langle G\rangle_F$ *concat ys* $\in \langle G\rangle_F$ **and**
simple: $x \in U\ \langle G\rangle_F\ y \in U\ \langle G\rangle_F$ **and**
in-hull: $x \in \langle\langle G\rangle_F\rangle\ y \in \langle\langle G\rangle_F\rangle$ **and**

nemp: $x \neq \varepsilon\ y \neq \varepsilon$
using ‹$x \# xs \in lists\ (\mathfrak{B}_F\ G)$› ‹$y \# ys \in lists\ (\mathfrak{B}_F\ G)$› *free-basis-def emp-not-basis* **unfolding** *basis-def basis-gen-hull-free*[*symmetric*] **by** *auto*
have $x \cdot concat\ xs = y \cdot concat\ ys$
using ‹$concat\ (x \# xs) = concat\ (y \# ys)$› **by** *simp*
from *eqd-or*[*OF this*]
obtain t **where** *or*: $x \cdot t = y \wedge t \cdot concat\ ys = concat\ xs \vee y \cdot t = x \wedge t \cdot concat\ xs = concat\ ys$
by *blast*
have $t \in \langle G \rangle_F$
by (*rule disjE*[*OF or*])
(*use stability*[*of x G concat ys t*] *stability*[*of y G concat xs t*] *in-free-hull*
in *fastforce*)+
hence $t = \varepsilon$
using *or simple free-hull-hull in-hull nemp ungen-dec-triv* **by** *metis*
thus $x = y$
using *or* **by** *blast*
then show $xs = ys$
using *4.IH* ‹$x \# xs \in lists\ (\mathfrak{B}_F\ G)$› ‹$y \# ys \in lists\ (\mathfrak{B}_F\ G)$› ‹$concat\ (x \# xs) = concat\ (y \# ys)$›
by *auto*
qed
qed *simp*
qed

lemma *gen-in-free-hull*: $x \in G \Longrightarrow x \in \langle \mathfrak{B}_F\ G \rangle$
using *free-hull.free-gen-in*[*folded basis-gen-hull-free*]**.**

Second, a code generates its free hull.

lemma (**in** *code*) *code-gen-free-hull*: $\langle \mathcal{C} \rangle_F = \langle \mathcal{C} \rangle$
proof
show $\langle \mathcal{C} \rangle \subseteq \langle \mathcal{C} \rangle_F$
using *hull-mono*[*of* $\mathcal{C}$ $\langle \mathcal{C} \rangle_F$] *free-gen-in*[*of - C*] *subsetI*[*of* $\mathcal{C}$ $\langle \mathcal{C} \rangle_F$]
unfolding *free-hull-hull* **by** *fast*
show $\langle \mathcal{C} \rangle_F \subseteq \langle \mathcal{C} \rangle$
proof
fix x **assume** $x \in \langle \mathcal{C} \rangle_F$
thm *emp-in*
show $x \in \langle \mathcal{C} \rangle$
proof(*rule free-hull.induct*[*OF* ‹$x \in \langle \mathcal{C} \rangle_F$›])
show $w \in \langle \mathcal{C} \rangle$ **if** *elems*: $p \in \langle \mathcal{C} \rangle\ q \in \langle \mathcal{C} \rangle\ p \cdot w \in \langle \mathcal{C} \rangle\ w \cdot q \in \langle \mathcal{C} \rangle$ **for** $p\ q\ w$
proof–
have *eq*: $(Dec\ \mathcal{C}\ p) \cdot (Dec\ \mathcal{C}\ w \cdot q) = (Dec\ \mathcal{C}\ p \cdot w) \cdot (Dec\ \mathcal{C}\ q)$
using *code-dec-morph*[*OF* ‹$p \in \langle \mathcal{C} \rangle$› ‹$w \cdot q \in \langle \mathcal{C} \rangle$›, *unfolded lassoc*]
unfolding *code-dec-morph*[*OF* ‹$p \cdot w \in \langle \mathcal{C} \rangle$› ‹$q \in \langle \mathcal{C} \rangle$›, *symmetric*]**.**
from *eqd-or*[*OF this*]
obtain *ts* **where** *or'*: $(Dec\ \mathcal{C}\ p) \cdot ts = Dec\ \mathcal{C}\ p \cdot w \vee (Dec\ \mathcal{C}\ p \cdot w) \cdot ts = Dec\ \mathcal{C}\ p$
by *blast*

```
      hence concat ((Dec C p) · ts) = concat (Dec C p · w) ∨
          concat((Dec C p · w) · ts) = concat (Dec C p)
        by presburger
      hence concat ts = w
          unfolding concat-dec[OF ‹p · w ∈ ⟨C⟩›] concat-dec[OF ‹p ∈ ⟨C⟩›]
concat-morph
        rassoc by blast
      have ts ∈ lists C
        using dec-in-lists[OF ‹p · w ∈ ⟨C⟩›] dec-in-lists[OF ‹p ∈ ⟨C⟩›]
        or′ append-in-lists-conv[of - ts C] by metis
      thus w ∈ ⟨C⟩
        using ‹concat ts = w› by blast
    qed
  qed (simp-all add: ‹x ∈ ⟨C⟩_F› hull-closed gen-in)
 qed
qed
```

That is, a code is its own free basis

```
lemma (in code) code-free-basis: C = 𝔅_F C
 using basis-of-hull[of C, unfolded code-gen-free-hull[symmetric]
    code-is-basis, symmetric] unfolding free-basis-def.
```

This allows to use the introduction rules of the free hull to prove one of the basic characterizations of the code, called the stability condition

```
lemma (in code) stability: p ∈ ⟨C⟩ ⟹ q ∈ ⟨C⟩ ⟹ p · w ∈ ⟨C⟩ ⟹ w · q ∈ ⟨C⟩
⟹ w ∈ ⟨C⟩
 unfolding code-gen-free-hull[symmetric] using free-hull.intros(4) by auto
```

Moreover, the free hull of G is the smallest code-generated hull containing G. In other words, the term free hull is appropriate.

First, several intuitive monotonicity and closure results.

```
lemma free-hull-mono: assumes G ⊆ H shows ⟨G⟩_F ⊆ ⟨H⟩_F
proof
 fix x assume x ∈ ⟨G⟩_F
 have el: ⋀ w. w ∈ G ⟹ w ∈ ⟨H⟩_F
   using ‹G ⊆ H› free-hull.free-gen-in by auto
 show x ∈ ⟨H⟩_F
   by (rule free-hull.induct[of x G]) (auto simp add: ‹x ∈ ⟨G⟩_F› el)
qed

lemma free-hull-idem: ⟨⟨G⟩_F⟩_F = ⟨G⟩_F
proof
 show ⟨⟨G⟩_F⟩_F ⊆ ⟨G⟩_F
 proof
   fix x assume x ∈ ⟨⟨G⟩_F⟩_F
   show x ∈ ⟨G⟩_F
   proof (rule free-hull.induct[of x ⟨G⟩_F])
```

show $\bigwedge p\ q\ w.\ p \in \langle G\rangle_F \Longrightarrow\ q \in \langle G\rangle_F \Longrightarrow\ p \cdot w \in \langle G\rangle_F \Longrightarrow\ w \cdot q \in \langle G\rangle_F \Longrightarrow w \in \langle G\rangle_F$
using *free-hull.intros*(*4*) **by** *auto*
qed (*simp-all add*: ‹$x \in \langle\langle G\rangle_F\rangle_F$› *free-hull.intros*(*1*), *simp add*: *free-hull.intros*(*2*), *simp add*: *free-hull.intros*(*3*))
qed
next
show $\langle G\rangle_F \subseteq \langle\langle G\rangle_F\rangle_F$
using *free-hull-hull hull-sub-free-hull* **by** *auto*
qed

lemma *hull-gen-free-hull*: $\langle\langle G\rangle\rangle_F = \langle G\rangle_F$
proof
show $\langle\langle G\rangle\rangle_F \subseteq \langle G\rangle_F$
using *free-hull-idem free-hull-mono hull-sub-free-hull* **by** *metis*
next
show $\langle G\rangle_F \subseteq \langle\langle G\rangle\rangle_F$
by (*simp add*: *free-hull-mono*)
qed

Code is also the free basis of its hull.

lemma (**in** *code*) *code-free-basis-hull*: $\mathcal{C} = \mathfrak{B}_F\ \langle\mathcal{C}\rangle$
unfolding *free-basis-def* **using** *code-free-basis*[*unfolded free-basis-def*]
unfolding *hull-gen-free-hull*.

The minimality of the free hull easily follows.

theorem (**in** *code*) *free-hull-min*: **assumes** $G \subseteq \langle\mathcal{C}\rangle$ **shows** $\langle G\rangle_F \subseteq \langle\mathcal{C}\rangle$
using *free-hull-mono*[*OF* ‹$G \subseteq \langle\mathcal{C}\rangle$›] **unfolding** *hull-gen-free-hull*
unfolding *code-gen-free-hull*.

theorem *free-hull-inter*: $\langle G\rangle_F = \bigcap\ \{M.\ G \subseteq M \wedge M = \langle M\rangle_F\}$
proof
have $X \in \{M.\ G \subseteq M \wedge M = \langle M\rangle_F\} \Longrightarrow \langle G\rangle_F \subseteq X$ **for** X
unfolding *mem-Collect-eq*[*of* - $\lambda\ M.\ G \subseteq M \wedge M = \langle M\rangle_F$]
using *free-hull-mono*[*of G X*] **by** *simp*
from *Inter-greatest*[*of* $\{M.\ G \subseteq M \wedge M = \langle M\rangle_F\}$, *OF this*]
show $\langle G\rangle_F \subseteq \bigcap\ \{M.\ G \subseteq M \wedge M = \langle M\rangle_F\}$
by *blast*
next
show $\bigcap\ \{M.\ G \subseteq M \wedge M = \langle M\rangle_F\} \subseteq \langle G\rangle_F$
by (*simp add*: *Inter-lower free-hull-idem genset-sub-free*)
qed

Decomposition into the free basis is a morphism.

lemma *free-basis-dec-morph*: $u \in \langle G\rangle_F \Longrightarrow v \in \langle G\rangle_F \Longrightarrow$
$Dec\ (\mathfrak{B}_F\ G)\ (u \cdot v) = (Dec\ (\mathfrak{B}_F\ G)\ u) \cdot (Dec\ (\mathfrak{B}_F\ G)\ v)$
using *code.code-dec-morph*[*OF free-basis-code*, *of u G v*, *symmetric*, *unfolded basis-gen-hull-free*[*of G*]].

3.11 Reversing hulls and decompositions

lemma *basis-rev-commute*[*reversal-rule*]: $\mathfrak{B}\ (rev\ `\ G) = rev\ `\ (\mathfrak{B}\ G)$
proof
have $\langle rev\ `\ \mathfrak{B}\ G\rangle = \langle rev\ `\ G\rangle$ **and** $*$: $\langle rev\ `\ \mathfrak{B}\ (rev\ `\ G)\rangle = \langle rev\ `\ rev\ `G\rangle$
unfolding *rev-hull*[*symmetric*] *basis-gen-hull* **by** *blast*+
from *gen-basis-sub*[*OF this*(*1*)]
show $\mathfrak{B}\ (rev\ `\ G) \subseteq rev\ `\ \mathfrak{B}\ G$**.**
from *image-mono*[*OF gen-basis-sub*[*OF* $*$], *of rev*]
show $rev\ `\ (\mathfrak{B}\ G) \subseteq \mathfrak{B}\ (rev\ `\ G)$
unfolding *rev-rev-image-eq***.**
qed

lemma *rev-free-hull-comm*: $\langle rev\ `\ X\rangle_F = rev\ `\ \langle X\rangle_F$
proof−
have $rev\ `\ \langle X\rangle_F \subseteq \langle rev\ `\ X\rangle_F$ **for** X :: *'a list set*
proof
fix x **assume** $x \in rev\ `\ \langle X\rangle_F$
hence $rev\ x \in \langle X\rangle_F$
by (*simp add*: *rev-in-conv*)
have $rev\ x \in rev\ `\ \langle rev\ `\ X\rangle_F$
by (*induct rule*: *free-hull.induct*[*OF* ‹$rev\ x \in \langle X\rangle_F$›])
(*auto simp add*: *rev-in-conv*[*symmetric*])
then show $x \in \langle rev\ `\ X\rangle_F$
by *blast*
qed
from *this*
image-mono[*OF this*[*of rev* ` *X*, *unfolded rev-rev-image-eq*], *of rev*, *unfolded rev-rev-image-eq*]
show $\langle rev\ `\ X\rangle_F = rev\ `\ \langle X\rangle_F$
by *blast*
qed

lemma *free-basis-rev-commute* [*reversal-rule*]: $\mathfrak{B}_F\ rev\ `\ X = rev\ `\ (\mathfrak{B}_F\ X)$
unfolding *free-basis-def basis-rev-commute free-basis-def rev-free-hull-comm***..**

lemma *rev-dec*[*reversal-rule*]: **assumes** $x \in \langle X\rangle_F$ **shows** $Dec\ rev\ `\ (\mathfrak{B}_F\ X)\ (rev\ x) = map\ rev\ (rev\ (Dec\ (\mathfrak{B}_F\ X)\ x))$
proof−
have $x \in \langle \mathfrak{B}_F\ X\rangle$
using ‹$x \in \langle X\rangle_F$› **by** (*simp add*: *basis-gen-hull-free*)
from *concat-dec*[*OF this*]
have $concat\ (map\ rev\ (rev\ (Dec\ \mathfrak{B}_F\ X\ x))) = rev\ x$
unfolding *rev-concat*[*symmetric*] **by** *blast*
from *rev-image-eqI*[*OF rev-in-lists*[*OF dec-in-lists*[*OF* ‹$x \in \langle \mathfrak{B}_F\ X\rangle$›]], *of* - *map rev*]
have $map\ rev\ (rev\ (Dec\ \mathfrak{B}_F\ X\ x)) \in lists\ (rev\ `\ (\mathfrak{B}_F\ X))$
unfolding *lists-image* **by** *blast*
from *code.code-unique-dec'*[*OF code.code-rev-code*[*OF free-basis-code*] *this*]

show *?thesis*
unfolding ‹*concat* (*map rev* (*rev* (*Dec* $\mathfrak{B}_F$ *X x*))) = *rev x*›.
qed

lemma *rev-hd-dec-last-eq*[*reversal-rule*]: **assumes** $x \in X$ **and** $x \neq \varepsilon$ **shows**
rev (*hd* (*Dec* (*rev* ‘ ($\mathfrak{B}_F$ *X*)) (*rev x*))) = *last* (*Dec* $\mathfrak{B}_F$ *X x*)
proof−
have *rev* (*Dec* $\mathfrak{B}_F$ *X x*) ≠ ε
using ‹$x \in X$› *basis-gen-hull-free dec-nemp'*[*OF* ‹$x \neq \varepsilon$›] **by** *blast*
show *?thesis*
unfolding *hd-rev rev-dec*[*OF free-gen-in*[*OF* ‹$x \in X$›]] *hd-map*[*OF* ‹*rev* (*Dec* $\mathfrak{B}_F$ *X x*) ≠ ε›]
by *simp*
qed

lemma *rev-hd-dec-last-eq'*[*reversal-rule*]: **assumes** $x \in X$ **and** $x \neq \varepsilon$ **shows**
(*hd* (*Dec* (*rev* ‘ ($\mathfrak{B}_F$ *X*)) (*rev x*))) = *rev* (*last* (*Dec* $\mathfrak{B}_F$ *X x*))
using *assms*(*1*) *assms*(*2*) *rev-hd-dec-last-eq rev-swap* **by** *blast*

3.12 Lists as the free hull of singletons

A crucial property of free monoids of words is that they can be seen as lists over the free basis, instead as lists over the original alphabet.

abbreviation *sings* **where** *sings B* ≡ {[*b*] | *b*. *b* ∈ *B*}

term *Set.filter P A*

lemma *sings-image*: *sings B* = (λ *x*. [*x*]) ‘ *B*
using *Setcompr-eq-image*.

lemma *lists-sing-map-concat-ident*: *xs* ∈ *lists* (*sings B*) ⟹ *xs* = *map* (λ *x*. [*x*]) (*concat xs*)
by (*induct xs*, *simp*, *auto*)

lemma *code-sings*: *code* (*sings B*)
proof
fix *xs ys* **assume** *xs*: *xs* ∈ *lists* (*sings B*) **and** *ys*: *ys* ∈ *lists* (*sings B*)
and *eq*: *concat xs* = *concat ys*
from *lists-sing-map-concat-ident*[*OF xs*, *unfolded eq*]
show *xs* = *ys* **unfolding** *lists-sing-map-concat-ident*[*OF ys*, *symmetric*].
qed

lemma *sings-gen-lists*: ⟨*sings B*⟩ = *lists B*
unfolding *hull-concat-lists*
proof(*intro equalityI subsetI*, *standard*)
fix *xs*
show *xs* ∈ *concat* ‘ *lists* (*sings B*) ⟹ ∀ *x*∈*set xs*. *x* ∈ *B*
by *force*

assume *xs* ∈ *lists B*
hence *map* (λ*x*. *x* # ε) *xs* ∈ *lists* (*sings B*)
by *force*
from *imageI*[*OF this*, *of concat*]
show *xs* ∈ *concat* ‘ *lists* (*sings B*)
unfolding *concat-map-sing-ident*[*of xs*].
qed

lemma *sing-gen-lists*: *lists* {*x*} = ⟨{[*x*]}⟩
using *sings-gen-lists*[*of* {*x*}] **by** *simp*

lemma *bin-gen-lists*: *lists* {*x*, *y*} = ⟨{[*x*],[*y*]}⟩
using *sings-gen-lists*[*of* {*x*,*y*}] **unfolding** *Setcompr-eq-image* **by** *simp*

lemma *sings B* = $\mathfrak{B}_F$ (*lists B*)
using *code.code-free-basis-hull*[*OF code-sings*, *of B*, *unfolded sings-gen-lists*].

lemma *map-sings*: *xs* ∈ *lists B* ⟹ *map* (λ*x*. *x* # ε) *xs* ∈ *lists* (*sings B*)
by (*induct xs*) *auto*

lemma *dec-sings*: *xs* ∈ *lists B* ⟹ *Dec* (*sings B*) *xs* = *map* (λ *x*. [*x*]) *xs*
using *code.code-unique-dec′*[*OF code-sings*, *of map* (λ *x*. [*x*]) *xs B*, *OF map-sings*]
unfolding *concat-map-sing-ident*.

lemma *sing-lists-exp*: **assumes** *ws* ∈ *lists* {*a*}
obtains *k* **where** *ws* = [*a*]^@*k*
using *sing-set-wordE*[*OF assms*[*folded in-lists-conv-set-subset*]].

lemma *sing-lists-exp-len*: *ws* ∈ *lists* {*a*} ⟷ [*a*]^@|*ws*| = *ws*
by (*induct ws*, *auto*)

lemma *sing-lists-exp-count*: *ws* ∈ *lists* {*a*} ⟷ [*a*]^@(*count-list ws a*) = *ws*
by (*standard*, *induct ws*, *force*, *simp*)
(*use sing-pow-lists*[*OF singletonI*, *of count-list ws a a*] **in** *argo*)

lemma *sing-set-pow-count-list*: *set ws* ⊆ {*a*} ⟷ [*a*]^@(*count-list ws a*) = *ws*
unfolding *in-lists-conv-set-subset* **using** *sing-lists-exp-count*.

lemma *count-le-length-iff*: *set ws* ⊆ {*a*} ⟷ *count-list ws a* = |*ws*|
proof
show *set ws* ⊆ {*a*} ⟹ *count-list ws a* = |*ws*|
unfolding *sing-set-pow-count-list* **using**
lenarg[*of* [*a*] ^@ *count-list ws a ws*, *unfolded pow-len sing-len*] **by** *force*
next
assume *count-list ws a* = |*ws*|
then show *set ws* ⊆ {*a*}
proof (*induct ws*, *force*)
case (*Cons b ws*)
have *b* = *a*

using *Cons.prems count-le-length*[*of ws a*] *impossible-Cons*[*of b*#*ws*]
unfolding *count-list.simps*(*2*) **by** *metis*
have *count-list ws a* = |*ws*|
using *Cons.prems* **unfolding** ‹*b* = *a*› **by** *force*
from *Cons.hyps*[*OF this*]
show *set* (*b* # *ws*) ⊆ {*a*}
unfolding ‹*b* = *a*› **by** *simp*
qed
qed

lemma *sing-set-pow*: *set ws* ⊆ {*a*} ⟷ [*a*]^@|*ws*| = *ws*
using *sing-lists-exp-len*[*unfolded sing-lists-exp-count*]
using *sing-set-pow-count-list*[*unfolded count-le-length-iff*]
unfolding *count-le-length-iff* **by** *fast*

lemma *count-sing-exp*[*simp*]: *count-list* ([*a*]^@*k*) *a* = *k*
by (*induct k*, *simp-all*)

lemma *count-sing-exp′*[*simp*]: *count-list* ([*a*]) *a* = *1*
by *simp*

lemma *count-sing-distinct*[*simp*]: *a* ≠ *b* ⟹ *count-list* ([*a*]^@*k*) *b* = *0*
by (*induct k*, *simp*, *auto*)

lemma *count-sing-distinct′*[*simp*]: *a* ≠ *b* ⟹ *count-list* ([*a*]) *b* = *0*
by *simp*

lemma *sing-letter-imp-prim*: **assumes** *count-list w a* = *1* **shows** *primitive w*
proof
fix *r k*
assume *r* ^@ *k* = *w*
have *count-list w a* = *k* ∗ *count-list r a*
by (*simp only*: *count-list-pow-list flip*: ‹*r* ^@ *k* = *w*›)
then show *k* = *1*
unfolding ‹*count-list w a* = *1*› **by** *simp*
qed

lemma *prim-abk*: *a* ≠ *b* ⟹ *primitive* ([*a*] · [*b*] ^@ *k*)
by (*intro sing-letter-imp-prim*[*of* - *a*]) *simp*

lemma *sing-code*: *x* ≠ *ε* ⟹ *code* {*x*}
proof (*rule code.intro*)
fix *xs ys*
assume *x* ≠ *ε xs* ∈ *lists* {*x*} *ys* ∈ *lists* {*x*} *concat xs* = *concat ys*
from ‹*xs* ∈ *lists* {*x*}›[*unfolded sing-lists-exp-len*, *symmetric*]
‹*ys* ∈ *lists* {*x*}›[*unfolded sing-lists-exp-len*, *symmetric*]
have |*xs*| = |*ys*|
using ‹*concat xs* = *concat ys*› *concat-pow-list-single eq-pow-exp*[*OF* ‹*x* ≠ *ε*›] **by** *metis*

then show *xs = ys*
using ‹*xs = [x]* @ *|xs|*› ‹*ys = [x]* @ *|ys|*› **by** *argo*
qed

lemma *sings-card*: *card A = card (sings A)*
by(*rule bij-betw-same-card*, *rule bij-betwI'*[*of* - *λx. [x]*], *auto*)

lemma *sings-finite*: *finite A = finite (sings A)*
by(*rule bij-betw-finite*, *rule bij-betwI'*[*of* - *λx. [x]*], *auto*)

lemma *sings-conv*: *A = B ⟷ sings A = sings B*
proof(*standard*, *simp*)
have *⋀x A B. sings A = sings B ⟹ x ∈ A ⟹ x ∈ B*
proof–
fix *x :: 'b* **and** *A B*
assume *sings A = sings B x ∈ A*
hence *[x] ∈ sings B*
using ‹*sings A = sings B*› **by** *blast*
thus *x ∈ B*
by *blast*
qed
from *this*[*of A B*] *this*[*of B A*, *OF sym*]
show *sings A = sings B ⟹ A = B*
by *blast*
qed

3.13 Various additional lemmas

3.13.1 Roots of binary set

lemma *two-roots-code*: **assumes** *x ≠ ε* **and** *y ≠ ε* **shows** *code {ϱ x, ϱ y}*
using *assms*
proof (*cases ϱ x = ϱ y*)
assume *ϱ x = ϱ y*
thus *code {ϱ x, ϱ y}* **using** *sing-code*[*OF primroot-nemp*[*OF* ‹*x ≠ ε*›]] **by** *simp*
next
assume *ϱ x ≠ ϱ y*
hence *ϱ x · ϱ y ≠ ϱ y · ϱ x*
using *comm-prim*[*OF primroot-prim*[*OF* ‹*x ≠ ε*›] *primroot-prim*[*OF* ‹*y ≠ ε*›]]
by *blast*
thus *code {ϱ x, ϱ y}*
by (*simp add*: *bin-code-code*)
qed

lemma *primroot-in-set-dec*: **assumes** *x ≠ ε* **and** *y ≠ ε* **shows** *ϱ x ∈ set (Dec {ϱ x, ϱ y} x)*
proof–
obtain *k* **where** *concat ([ϱ x]*@*k) = x 0 < k*
using *primroot-expE*

concat-pow-list-single[*symmetric*, *of* - ϱ *x*] **by** *metis*
from *code.code-unique-dec'*[*OF two-roots-code*[*OF assms*], *of* [ϱ *x*]$^{@}$*k*, *unfolded* ‹*concat* ([ϱ *x*]$^{@}$*k*) = *x*›]
have *Dec* {ϱ *x*, ϱ *y*} *x* = [ϱ *x*]$^{@}$*k*
using *insertI1 sing-pow-lists* **by** *metis*
show *?thesis*
unfolding ‹*Dec* {ϱ *x*, ϱ *y*} *x* = [ϱ *x*]$^{@}$*k*› **using** ‹*0* < *k*› **by** *simp*
qed

lemma *primroot-dec*: **assumes** *x* · *y* ≠ *y* · *x*
shows (*Dec* {ϱ *x*, ϱ *y*} *x*) = [ϱ *x*]$^{@}e_{\varrho}$ *x* (*Dec* {ϱ *x*, ϱ *y*} *y*) = [ϱ *y*]$^{@}e_{\varrho}$ *y*
by (*simp-all add*: *binary-code.intro*[*OF assms*] *binary-code.primroot-dec*)

3.13.2 Other

lemma *bin-count-one-decompose*: **assumes** *ws* ∈ *lists* {*x*,*y*} **and** *x* ≠ *y* **and** *count-list ws y* = *1*
obtains *k m* **where** [*x*]$^{@}$*k* · [*y*] · [*x*]$^{@}$*m* = *ws*
proof–
have ¬ *set ws* ⊆ {*x*}
using *count-sing-distinct*[*OF* ‹*x* ≠ *y*›] ‹*count-list ws y* = *1*› **by** *auto*
from *distinct-letter-in*[*OF this*]
obtain *ws'* *k* *b* **where** [*x*]$^{@}$*k* · [*b*] · *ws'* = *ws* **and** *b* ≠ *x* **by** *blast*
hence *b* = *y*
using ‹*ws* ∈ *lists* {*x*,*y*}› **by** *force*
have *ws'* ∈ *lists* {*x*,*y*}
using ‹*ws* ∈ *lists* {*x*,*y*}›[*folded* ‹[*x*]$^{@}$*k* · [*b*] · *ws'* = *ws*›] **by** *simp*
have *count-list ws' y* = *0*
using *arg-cong*[*OF* ‹[*x*]$^{@}$*k* · [*b*] · *ws'* = *ws*›, *of* λ *x*. *count-list x y*]
unfolding *count-list-append* ‹*count-list ws y* = *1*› ‹*b* = *y*› **by** *force*
hence *set ws'* ⊆ {*x*}
unfolding *count-list-0-iff* **using** ‹*ws'* ∈ *lists* {*x*,*y*}› **by** *blast*
then obtain *m* **where** *ws'* = [*x*]$^{@}$*m*
by *blast*
from *that*[*OF* ‹[*x*]$^{@}$*k* · [*b*] · *ws'* = *ws*›[*unfolded this* ‹*b* = *y*›]]
show *thesis*.
qed

lemma *bin-count-one-conjug*: **assumes** *ws* ∈ *lists* {*x*,*y*} **and** *x* ≠ *y* **and** *count-list ws y* = *1*
shows *ws* ∼ [*x*]$^{@}$(*count-list ws x*) · [*y*]
proof–
obtain *e1 e2* **where** [*x*]$^{@}$*e1* · [*y*] · [*x*]$^{@}$*e2* = *ws*
using *bin-count-one-decompose*[*OF assms*].
from *conjugI'*[*of* [*x*] $^{@}$ *e1* · [*y*] [*x*]$^{@}$*e2*, *unfolded rassoc this*]
have *ws* ∼ [*x*]$^{@}$(*e2* + *e1*) · [*y*]
unfolding *pow-add rassoc*.
moreover have *count-list* ([*x*]$^{@}$(*e2* + *e1*) · [*y*]) *x* = *e2* + *e1*
using ‹*x* ≠ *y*› **by** *simp*

ultimately show *?thesis*
by (*simp add*: *count-list-conjug*)
qed

lemma *bin-prim-long-set*: **assumes** $ws \in lists\ \{x,y\}$ **and** *primitive ws* **and** $2 \leq |ws|$
shows $set\ ws = \{x,y\}$
proof–
have $\neg\ set\ ws \subseteq \{c\}$ **for** *c*
using ‹*primitive ws*› *pow-nemp-imprim* ‹$2 \leq |ws|$›
sing-lists-exp-len[*folded in-lists-conv-set-subset*] **by** *metis*
then show $set\ ws = \{x,y\}$
unfolding *subset-singleton-iff* **using** ‹$ws \in lists\ \{x,y\}$›[*folded in-lists-conv-set-subset*] *doubleton-subset-cases* **by** *metis*
qed

lemma *bin-prim-long-pref*: **assumes** $ws \in lists\ \{x,y\}$ **and** *primitive ws* **and** $2 \leq |ws|$
obtains ws' **where** $ws \sim ws'$ **and** $[x,y] \leq p\ ws'$
proof–
from *pow-nemp-imprim*[*OF* ‹$2 \leq |ws|$›, *of* $[x]$] *sing-lists-exp-len*[*of ws x*]
have $\neg\ ws \in lists\ \{x\}$
using ‹*primitive ws*› ‹$2 \leq |ws|$› **by** *fastforce*
hence $x \neq y$
using ‹$ws \in lists\ \{x,y\}$› **by** *fastforce*
from *switch-fac*[*OF* ‹$x \neq y$› *bin-prim-long-set*[*OF assms*]]
show *thesis*
using ‹$2 \leq |ws|$› *rotate-into-pos-sq*[*of* ε $[x,y]$ *ws thesis*, *unfolded emp-simps*, *OF* ‹$[x,\ y] \leq f\ ws \cdot ws$› - - *that*, *of id*]
by *force*
qed

end

theory *Morphisms*

imports *CoWBasic Submonoids*

begin

Chapter 4

Morphisms

4.1 One morphism

4.1.1 Morphism, core map and extension

definition *list-extension* :: $('a \Rightarrow 'b\ list) \Rightarrow ('a\ list \Rightarrow 'b\ list)$ $(\text{-}^{\mathcal{L}}\ [1000]\ 1000)$
where $t^{\mathcal{L}} \equiv (\lambda\ x.\ concat\ (map\ t\ x))$

definition *morphism-core* :: $('a\ list \Rightarrow 'b\ list) \Rightarrow ('a\ \Rightarrow 'b\ list)$ $(\text{-}^{\mathcal{C}}\ [1000]\ 1000)$
where *core-def*: $f^{\mathcal{C}} \equiv (\lambda\ x.\ f\ [x])$

lemma *core-sing*: $f^{\mathcal{C}}\ a = f\ [a]$
unfolding *core-def*..

lemma *range-map-core*: $range\ (map\ f^{\mathcal{C}}) = lists\ (range\ f^{\mathcal{C}})$
using *lists-image*[*of* $\lambda x.\ f\ [x]$ *UNIV*, *folded core-def*, *symmetric*]
unfolding *lists-UNIV*.

lemma *map-core-lists*[*simp*]: $(map\ f^{\mathcal{C}}\ w) \in lists\ (range\ f^{\mathcal{C}})$
by *auto*

lemma *comp-core*: $(f \circ g)^{\mathcal{C}} = f \circ g^{\mathcal{C}}$
unfolding *core-def*
by *auto*

lemma *ext-core-id* [*simp*]: $f^{\mathcal{LC}} = f$
unfolding *list-extension-def core-def* **by** *simp*

locale *morphism-on* =
fixes $f :: 'a\ list \Rightarrow 'b\ list$ **and** $A :: 'a\ list\ set$
assumes *morph-on*: $u \in \langle A \rangle \Longrightarrow v \in \langle A \rangle \Longrightarrow f\ (u \cdot v) = f\ u \cdot f\ v$

begin

lemma *emp-to-emp-on*[*simp*]: $f\ \varepsilon = \varepsilon$
using *morph-on*[*of* ε ε] *self-append-conv2*[*of* f ε f ε] **by** *simp*

lemma *emp-to-emp'*: $w = \varepsilon \Longrightarrow f\ w = \varepsilon$
using *morph-on*[*of* ε ε] *self-append-conv2*[*of* f ε f ε] **by** *simp*

lemma *morph-concat-concat-map-on*: $ws \in lists\ \langle A\rangle \Longrightarrow f\ (concat\ ws) = concat\ (map\ f\ ws)$
by (*induct ws*, *simp-all add*: *morph-on hull-closed-lists*)

lemma *hull-im-hull*:
shows $\langle f\ `\ A\rangle = f\ `\ \langle A\rangle$
unfolding *set-eq-iff*
proof
have *elem1*: $x \in \langle f\ `\ A\rangle \longleftrightarrow (\exists\ ws.\ ws \in lists\ A \wedge x = concat\ (map\ f\ ws))$ **for** x
unfolding *hull-concat-lists* **unfolding** *lists-image*
by *blast*
have *elem2*: $x \in f\ `\ \langle A\rangle \longleftrightarrow (\exists\ ws.\ ws \in lists\ A \wedge x = f\ (concat\ ws))$ **for** x
by *blast*
show $x \in \langle f\ `\ A\rangle \longleftrightarrow x \in f\ `\ \langle A\rangle$ **for** x
unfolding *elem1 elem2*
using *morph-concat-concat-map-on* **by** *force*
qed

lemma *inj-basis-to-basis*: **assumes** *inj-on* $f\ \langle A\rangle$
shows $f\ `\ (\mathfrak{B}\ \langle A\rangle) = \mathfrak{B}\ (f`\langle A\rangle)$
proof
interpret *basis*: *morphism-on* $f\ \mathfrak{B}\ \langle A\rangle$
by (*rule morph-on morphism-on.intro*, *unfold basis-gen-hull'*[*of A*])
(*simp only*: *morph-on*)
show $\mathfrak{B}\ (f\ `\ \langle A\rangle) \subseteq f\ `\ \mathfrak{B}\ \langle A\rangle$
using *basis.hull-im-hull* **unfolding** *basis-gen-hull* **unfolding** *gen-idemp* **using** *basis-hull-sub*[*of* $f\ `\ \mathfrak{B}\ \langle A\rangle$] **by** *argo*
show $f\ `\ \mathfrak{B}\ \langle A\rangle \subseteq \mathfrak{B}\ (f\ `\ \langle A\rangle)$
proof
fix x
assume $x \in f\ `\ \mathfrak{B}\ \langle A\rangle$
then obtain y **where** $y \in \mathfrak{B}\ \langle A\rangle$ **and** $x = f\ y$ **by** *blast*
hence $x \in f\ `\ \langle A\rangle$
using *basis-sub-gen* **by** *blast*
from *basis-concat-listsE*[*OF this*]
obtain *xs* **where** $xs \in lists\ \mathfrak{B}\ (f\ `\langle A\rangle)$ **and** $concat\ xs = x$**.**
hence $\varepsilon \notin set\ xs$
using *emp-not-basis* **by** *blast*
have $xs \in lists\ (f\ `\langle A\rangle)$
using ‹$xs \in lists\ \mathfrak{B}\ (f\ `\langle A\rangle)$› *basis-sub-gen* **by** *blast*
then obtain *ys* **where** $map\ f\ ys = xs$ **and** $ys \in lists\ \langle A\rangle$
unfolding *lists-image* **by** *blast*
have $\varepsilon \notin set\ ys$

```
    using emp-to-emp-on ‹ε ∉ set xs›
      imageI[of ε set ys f] unfolding list.set-map[of f ys, unfolded ‹map f ys =
xs›] by presburger
  hence ys ∈ lists (⟨A⟩ − {ε})
    using ‹ys ∈ lists ⟨A⟩› by fast
  have f (concat ys) = x
    unfolding morph-concat-concat-map-on[OF ‹ys ∈ lists ⟨A⟩›] ‹map f ys = xs›
by fact
  from ‹inj-on f ⟨A⟩› this[unfolded ‹x = f y›]
  have concat ys = y
  unfolding inj-on-def using subsetD[OF basis-sub-gen ‹y ∈ 𝔅 ⟨A⟩›] hull-closed-lists[OF
‹ys ∈ lists ⟨A⟩›] by blast
  hence |ys| = 1
    using basisD[OF ‹y ∈ 𝔅 ⟨A⟩›] ungen-dec-triv'[OF ‹ys ∈ lists (⟨A⟩ − {ε})›]
by simp
  hence |xs| = 1
    using ‹map f ys = xs› by fastforce
  with ‹concat xs = x› ‹xs ∈ lists 𝔅 (f ‘⟨A⟩)›
  show x ∈ 𝔅 (f ‘ ⟨A⟩)
    using len-one-concat-in by blast
 qed
qed

lemma inj-code-to-code: assumes inj-on f ⟨A⟩ and code A
        shows code (f ‘ A)
proof
 fix xs ys
 assume xs ∈ lists (f ‘ A) and ys ∈ lists (f ‘ A)
 then obtain xs' ys' where xs' ∈ lists A and xs = map f xs' and ys' ∈ lists A
and ys = map f ys'
  unfolding lists-image by blast
 assume concat xs = concat ys
 hence f (concat xs') = f (concat ys')
  using morph-concat-concat-map-on lists-mono[OF genset-sub]
  ‹xs' ∈ lists A› ‹ys' ∈ lists A›
  unfolding ‹xs = map f xs'› ‹ys = map f ys'› subset-iff by metis
 hence concat xs' = concat ys'
  using ‹inj-on f ⟨A⟩›[unfolded inj-on-def] ‹xs' ∈ lists A› ‹ys' ∈ lists A› by auto
 hence xs' = ys'
  using ‹code A›[unfolded code-def] ‹xs' ∈ lists A› ‹ys' ∈ lists A› by simp
 thus xs = ys
  using ‹xs = map f xs'› ‹ys = map f ys'› by blast
qed

end

locale morphism =
 fixes f :: 'a list ⇒ 'b list
 assumes morph: f (u · v) = f u · f v
```

begin

sublocale *morphism-on f UNIV*
  by (*simp add*: *morph morphism-on.intro*)

lemmas *emp-to-emp* = *emp-to-emp-on*

lemma *pow-morph*: $f\ (x^{@}k) = (f\ x)^{@}k$
  by (*induction k*) (*simp add*: *morph*)+

lemma *rev-map-pow*: $(rev\text{-}map\ f)\ (w^{@}n) = rev\ ((f\ (rev\ w))^{@}n)$
  by (*simp add*: *pow-morph rev-map-arg rev-pow*)

lemma *pop-hd*: $f\ (a\#u) = f\ [a] \cdot f\ u$
  unfolding *hd-word*[*of a u*] **using** *morph*.

lemma *pop-hd-nemp*: $u \neq \varepsilon \Longrightarrow f\ (u) = f\ [hd\ u] \cdot f\ (tl\ u)$
  using *list.exhaust-sel pop-hd*[*of hd u tl u*] **by** *force*

lemma *pop-last-nemp*: $u \neq \varepsilon \Longrightarrow f\ (u) = f\ (butlast\ u) \cdot f\ [last\ u]$
  unfolding *morph*[*symmetric*] *append-butlast-last-id* ..

lemma *pref-mono*: $u \leq p\ v \Longrightarrow f\ u \leq p\ f\ v$
  using *morph* **by** (*auto simp add*: *prefix-def*)

lemma *suf-mono*: $u \leq s\ v \Longrightarrow f\ u \leq s\ f\ v$
  using *morph* **by** (*auto simp add*: *suffix-def*)

lemma *morph-concat-map*: $concat\ (map\ f^{\mathcal{C}}\ x) = f\ x$
  unfolding *core-def*
proof (*induction x*)
  case (*Cons a x*)
  then show *?case*
    unfolding *pop-hd*[*of a x*] **by** *auto*
qed *simp*

lemma *morph-concat-map'*: $(\lambda\ x.\ concat\ (map\ f^{\mathcal{C}}\ x)) = f$
  using *morph-concat-map* **by** *simp*

lemma *morph-to-concat*:
  obtains *xs* **where** $xs \in lists\ (range\ f^{\mathcal{C}})$ **and** $f\ x = concat\ xs$
proof−
  have $map\ f^{\mathcal{C}}\ x \in lists\ (range\ f^{\mathcal{C}})$
    by *fastforce*
      from *that*[*OF this morph-concat-map*[*symmetric*]]
  show *thesis*.
qed

lemma *range-hull*: $range\ f = \langle (range\ f^{\mathcal{C}}) \rangle$

using *arg-cong*[*OF range-map-core*[*of f*], *of image concat*, *unfolded image-comp*, *folded hull-concat-lists*] *morph-concat-map* **by** *auto*

lemma *im-in-hull*: $f\ w \in \langle (range\ f^{\mathcal{C}}) \rangle$
using *range-hull* **by** *blast*

lemma *core-ext-id*: $f^{\mathcal{C}\mathcal{L}} = f$
using *morph-concat-map* **unfolding** *list-extension-def core-def* **by** *simp*

lemma *rev-map-morph*: *morphism* (*rev-map f*)
by (*standard*, *auto simp add*: *rev-map-def morph*)

lemma *morph-rev-len*: $|f\ (rev\ u)| = |f\ u|$
proof (*induction u*)
case (*Cons a u*)
then show *?case*
unfolding *rev.simps*(*2*) *pop-hd*[*of a u*] *morph lenmorph* **by** *force*
qed *simp*

lemma *rev-map-len*: $|rev\text{-}map\ f\ u| = |f\ u|$
unfolding *rev-map-def*
by (*simp add*: *morph-rev-len*)

lemma *in-set-morph-len*: **assumes** $a \in set\ w$ **shows** $|f\ [a]| \leq |f\ w|$
proof–
from *split-listE*[*OF assms*]
obtain *p s* **where** $w = p \cdot [a] \cdot s$**.**
from *lenarg*[*OF arg-cong*[*of - - f*, *OF this*], *unfolded morph lenmorph*]
show *?thesis* **by** *linarith*
qed

lemma *morph-lq-comm*: $u \leq p\ v \Longrightarrow f\ (u^{-1>}v) = (f\ u)^{-1>}(f\ v)$
using *morph* **by** (*auto simp add*: *prefix-def*)

lemma *morph-rq-comm*: **assumes** $v \leq s\ u$
shows $f\ (u^{<-1}v) = (f\ u)^{<-1}(f\ v)$
using *arg-cong*[*OF rq-suf*[*OF* ‹$v \leq s\ u$›], *of f*, *unfolded morph*, *THEN rqI*, *symmetric*]**.**

lemma *code-set-morph*: **assumes** *c*: *code* ($f^{\mathcal{C}}$ ‘(*set* ($u \cdot v$))) **and** *i*: *inj-on* $f^{\mathcal{C}}$ (*set* ($u \cdot v$))
and $f\ u = f\ v$
shows $u = v$
proof–
let *?C* = $f^{\mathcal{C}}$ ‘(*set* ($u \cdot v$))
interpret *code ?C*
using *c* **by** *blast*
have (*map* $f^{\mathcal{C}}$ *u*) ∈ *lists ?C* **and** (*map* $f^{\mathcal{C}}$ *v*) ∈ *lists ?C*
by (*simp-all add*: *in-listsI*)

from *is-code*[*OF this* ‹*f u* = *f v*›[*folded morph-concat-map*]]
show *u* = *v*
using *inj-on-map-lists*[*OF i*] **unfolding** *inj-on-def*
by (*simp add*: *in-listsI*)
qed

lemma *morph-concat-concat-map*: *f* (*concat ws*) = *concat* (*map f ws*)
by (*induct ws*, *simp-all add*: *morph*)

lemma *morph-on-all*: *morphism-on f A*
unfolding *morphism-on-def* **using** *morph* **by** *blast*

lemma *noner-sings-conv*: (∀ *w*. *w* = *ε* ⟷ *f w* = *ε*) ⟷ (∀ *a*. *f* [*a*] ≠ *ε*)
by (*rule iffI*, *blast*)
(*metis Nil-is-append-conv emp-to-emp′ hd-tlE pop-hd*)

lemma *fac-mono*: *u* ≤*f w* ⟹ *f u* ≤*f f w*
using *morph* **by** *fastforce*

lemma *set-core-set*: *set* (*f w*) = ⋃ (*set* ‘ *f*$^{\mathcal{C}}$ ‘ (*set w*))
unfolding *list.set-map*[*symmetric*]
unfolding *image-set*[*of set* (*map f*$^{\mathcal{C}}$ *w*), *symmetric*]
unfolding *morph-concat-map*[*symmetric*, *of w*]
using *set-concat*.

end

lemma *morph-map*: *morphism* (*map f*)
by (*simp add*: *morphism-def*)

lemma *list-ext-morph*: *morphism t*$^{\mathcal{L}}$
unfolding *list-extension-def* **by** (*simp add*: *morphism-def*)

lemma *ext-def-on-set*[*intro*]: (⋀ *a*. *a* ∈ *set u* ⟹ *g a* = *f a*) ⟹ *g*$^{\mathcal{L}}$ *u* = *f*$^{\mathcal{L}}$ *u*
unfolding *list-extension-def* **using** *map-ext* **by** *metis*

lemma *morph-def-on-set*: *morphism f* ⟹ *morphism g* ⟹ (⋀ *a*. *a* ∈ *set u* ⟹ *g*$^{\mathcal{C}}$ *a* = *f*$^{\mathcal{C}}$ *a*) ⟹ *g u* = *f u*
using *ext-def-on-set morphism.core-ext-id* **by** *metis*

lemma *morph-compose*: *morphism f* ⟹ *morphism g* ⟹ *morphism* (*f* ∘ *g*)
by (*simp add*: *morphism-def*)

4.1.2 periodic morphism

locale *periodic-morphism* = *morphism* +
assumes *ims-comm*: ⋀ *u v*. *f u* · *f v* = *f v* · *f u* **and**
not-triv-emp: ¬ (∀ *c*. *f* [*c*] = *ε*)
begin

lemma *per-morph-root-ex*:
 $\exists\ r.\ \forall\ u.\ \exists\ n.\ f\ u = r^{@} n \wedge primitive\ r$
proof−
 obtain *c root n* **where** $f[c] = root^{@} n$ **and** $root = \varrho\ (f\ [c])$ **and** $f\ [c] \neq \varepsilon$
  using *primroot-expE not-triv-emp* **by** *metis*
 have $\exists\ n.\ f\ u = root^{@} n$ **for** *u*
  using *comm-primroot-exp*[*OF* ‹$f\ [c] \neq \varepsilon$›, *OF ims-comm*, *folded* ‹$root = \varrho\ (f\ [c])$›] **by** *metis*
 thus *?thesis*
  using ‹$root = \varrho\ (f\ [c])$› ‹$f\ [c] \neq \varepsilon$› **by** *auto*
qed

definition *mroot* **where** $mroot \equiv (SOME\ r.\ (\forall\ u.\ \exists\ n.\ f\ u = r^{@} n) \wedge primitive\ r)$
definition *im-exp* :: *'a list* ⇒ *nat* **where** $im\text{-}exp\ u \equiv (SOME\ n.\ f\ u = mroot^{@} n)$

lemma *per-morph-rootI*: $\forall\ u.\ \exists\ n.\ f\ u = mroot^{@} n$ **and**
 per-morph-root-prim: *primitive mroot*
 using *per-morph-root-ex exE-some*[*of* $\lambda\ r.\ \forall u.\ \exists n.\ f\ u = r\ ^{@}\ n \wedge primitive\ r$, *of mroot*]
 unfolding *mroot-def* **by** *auto*

lemma *per-morph-im-exp*:
  $f\ u = mroot^{@} im\text{-}exp\ u$
 unfolding *im-exp-def*
 using *per-morph-rootI tfl-some* **by** *meson*

lemma *mroot-primroot*: **assumes** $f\ u \neq \varepsilon$
 shows $mroot = \varrho\ (f\ u)$
 using *per-morph-rootI primroot-unique*[*OF* ‹$f\ u \neq \varepsilon$› *per-morph-root-prim*, *sym-metric*] **by** *blast*

interpretation *mirror*: *periodic-morphism rev-map f*
proof
 show $rev\text{-}map\ f\ (u \cdot v) = rev\text{-}map\ f\ u \cdot rev\text{-}map\ f\ v$ **for** *u v*
  using *morphism.morph*[*OF rev-map-morph*]**.**
 show $rev\text{-}map\ f\ u \cdot rev\text{-}map\ f\ v = rev\text{-}map\ f\ v \cdot rev\text{-}map\ f\ u$ **for** *u v*
  unfolding *comm-rev-iff ims-comm rev-map-arg*..
 show $\neg\ (\forall\, c.\ rev\text{-}map\ f\ [c] = \varepsilon)$
  using *not-triv-emp* **unfolding** *rev-map-sing* **by** *blast*
qed

lemma *per-morph-rev-map*: $rev\text{-}map\ f\ u = rev\ (f\ u)$
proof (*induct u*, *simp*)
 case (*Cons a u*)
 show *?case*
  unfolding *hd-word*[*of a u*] *morph morphism.morph*[*OF rev-map-morph*] *rev-append Cons.hyps*
   ims-comm cancel-right rev-map-sing..

qed

lemma *mroot-rev*: *mirror.mroot* = *rev mroot*
proof−
obtain *u* **where** $f\ u \neq \varepsilon$
using *not-triv-emp* **by** *auto*
hence *rev* $(f\ u) \neq \varepsilon$
by *blast*
show *?thesis*
unfolding *per-morph-rev-map mirror.mroot-primroot*[*unfolded per-morph-rev-map*, *OF* ‹*rev* $(f\ u) \neq \varepsilon$›]
mroot-primroot[*OF* ‹$f\ u \neq \varepsilon$›] *primroot-rev*..
qed

end

4.1.3 Non-erasing morphism

locale *nonerasing-morphism* = *morphism* +
assumes *nonerasing*: $f\ w = \varepsilon \Longrightarrow w = \varepsilon$
begin

lemma *core-nemp*: $f^{\mathcal{C}}\ a \neq \varepsilon$
unfolding *core-def* **using** *nonerasing not-Cons-self2* **by** *blast*

lemma *nemp-to-nemp*: $w \neq \varepsilon \Longrightarrow f\ w \neq \varepsilon$
using *nonerasing* **by** *blast*

lemma *sing-to-nemp*: $f\ [a] \neq \varepsilon$
by (*simp add*: *nemp-to-nemp*)

lemma *pref-morph-pref-eq*: $u \leq p\ v \Longrightarrow f\ v \leq p\ f\ u \Longrightarrow u = v$
using *nonerasing morph*[*of u* $u^{-1>}v$] **unfolding** *prefix-def* **by** *fastforce*

lemma *comm-eq-im-eq*:
$u \cdot v = v \cdot u \Longrightarrow f\ u = f\ v \Longrightarrow u = v$
by (*elim ruler-eqE*)
(*simp-all add*: *pref-morph-pref-eq pref-morph-pref-eq*[*symmetric*])

lemma *comm-eq-im-iff* :
assumes $u \cdot v = v \cdot u$
shows $f\ u = f\ v \longleftrightarrow u = v$
using *comm-eq-im-eq*[*OF* ‹$u \cdot v = v \cdot u$›] **by** *blast*

lemma *rev-map-nonerasing*: *nonerasing-morphism* (*rev-map f*)
proof
show *rev-map f* $(u \cdot v)$ = *rev-map f u* · *rev-map f v* **for** *u v*
by (*simp add*: *morphism.morph rev-map-morph*)
show *rev-map f* $w = \varepsilon \Longrightarrow w = \varepsilon$ **for** *w*

```
    using nonerasing[of rev w]
    unfolding rev-map-arg rev-is-Nil-conv.
qed

lemma first-of-first: (f (a # ws))!0 = f [a]!0
  unfolding pop-hd[of a ws] using  hd-prod[of f[a] f ws, OF
      nonerasing[of [a], THEN contrapos-nn[OF not-Cons-self2[of a ε], of ‹f (a #
ε) = ε›]]].

lemma hd-im-hd-hd: assumes u ≠ ε shows hd (f u) = hd (f [hd u])
  unfolding hd-append2[OF sing-to-nemp] pop-hd-nemp[OF ‹u ≠ ε›]..

lemma ssuf-mono: u <s v ⟹ f u <s f v
  by (elim strict-suffixE')
  (use morph sing-to-nemp ssufI1 suf-nemp in metis)

lemma im-len-le: |u| ≤ |f u|
proof (induct u)
  case (Cons a u)
  show ?case
    unfolding hd-word[of a u] morph lenmorph sing-len
     by (rule add-mono[OF - ‹|u| ≤ |f u|›], use nemp-le-len[OF sing-to-nemp] in
force)
qed simp

lemma im-len-eq-iff: |u| = |f u| ⟷ (∀ c. c ∈ set u ⟶ |f [c]| = 1)
proof (induct u)
  case (Cons a u)
  show ?case
  proof
    assume |a # u| = |f (a # u)|
    from this[unfolded  hd-word[of a u] morph lenmorph sing-len]
    have |f [a]| = 1 and |u| = |f u|
      unfolding sing-len[of a, symmetric] using  im-len-le[of [a]] im-len-le[of u]
by auto
    from this(2)[unfolded Cons.hyps] this(1)
    show ∀ c. c ∈ set (a # u) ⟶ |f [c]| = 1 by auto
  next
    assume ∀ c. c ∈ set (a # u) ⟶ |f [c]| = 1
    hence all: ∀ c. c ∈ set u ⟶ |f [c]| = 1 and |f [a]| = 1
      by simp-all
    show |a # u| = |f (a # u)|
      unfolding hd-word[of a u] morph lenmorph sing-len ‹|f [a]| = 1› all[folded
Cons.hyps]..
  qed
qed simp

lemma im-len-less: a ∈ set u ⟹ |f [a]| ≠ 1 ⟹ |u| < |f u|
  using im-len-le im-len-eq-iff order-le-neq-trans by auto
```

end

lemma (**in** *morphism*) *nonerI*[*intro*]: **assumes** ($\bigwedge$ *a*. $f^{\mathcal{C}}$ *a* $\neq$ ε)
shows *nonerasing-morphism f*
proof
from *assms*[*unfolded core-def*] *noner-sings-conv*
show $\bigwedge$*w*. *f w* = ε $\Longrightarrow$ *w* = ε **by** *presburger*
qed

lemma (**in** *morphism*) *prim-morph-noner*:
assumes *prim-morph*: $\bigwedge$*u*. 2 $\leq$ |*u*| $\Longrightarrow$ *primitive u* $\Longrightarrow$ *primitive* (*f u*)
and *non-single-dom*: $\exists$ *a b* :: '*a*. *a* $\neq$ *b*
shows *nonerasing-morphism f*
proof (*intro nonerI notI*)
fix *a*
assume $f^{\mathcal{C}}$ *a* = ε
obtain *c d* :: '*a* **where** *c* $\neq$ *d*
using *non-single-dom* **by** *blast*
then obtain *b* **where** *a* $\neq$ *b*
by (*cases a* = *c*) *simp-all*
then have $\neg$ *primitive* (*f* ([*a*] · [*b*] · [*b*]))
using ‹$f^{\mathcal{C}}$ *a* = ε› **unfolding** *morph*
by (*simp add*: *core-def eq-append-not-prim*)
have *primitive* ([*a*] · [*b*] · [*b*])
using *prim-abk*[*OF* ‹*a* $\neq$ *b*›, *of 2*]
by (*simp add*: *pow-list-2*)
from *prim-morph*[*OF - this*] ‹$\neg$ *primitive* (*f* ([*a*] · [*b*] · [*b*]))›
show *False*
by *simp*
qed

4.1.4 Code morphism

The term "Code morphism" is equivalent to "injective morphism".

Note that this is not equivalent to *code* (*range* $f^{\mathcal{C}}$), since the core can be not injective.

lemma (**in** *morphism*) *code-core-range-inj*: *inj f* $\longleftrightarrow$ *code* (*range* $f^{\mathcal{C}}$) $\wedge$ *inj* $f^{\mathcal{C}}$
proof
assume *inj f*
show *code* (*range* $f^{\mathcal{C}}$) $\wedge$ *inj* $f^{\mathcal{C}}$
proof
show *inj* $f^{\mathcal{C}}$
using ‹*inj f*› **unfolding** *inj-on-def core-def* **by** *blast*
show *code* (*range* $f^{\mathcal{C}}$)
proof
show

$xs \in lists\ (range\ f^{\mathcal{C}}) \Longrightarrow ys \in lists\ (range\ f^{\mathcal{C}}) \Longrightarrow concat\ xs = concat\ ys \Longrightarrow xs = ys$ **for** *xs ys*
unfolding *range-map-core*[*symmetric*] **using** ‹*inj f*›[*unfolded inj-on-def core-def*] *morph-concat-map*
by *force*
qed
qed
next
assume *code* (*range* $f^{\mathcal{C}}$) $\wedge$ *inj* $f^{\mathcal{C}}$ **hence** *code* (*range* $f^{\mathcal{C}}$) **and** *inj* $f^{\mathcal{C}}$ **by** *blast*+
show *inj f*
proof
fix *x y* **assume** *f x* = *f y*
with *code.is-code*[*OF* ‹*code* (*range* $f^{\mathcal{C}}$)›, *folded range-map-core*, *OF rangeI rangeI*, *unfolded morph-concat-map*]
have *map* $f^{\mathcal{C}}$ *x* = *map* $f^{\mathcal{C}}$ *y* **by** *blast*
with ‹*inj* $f^{\mathcal{C}}$›
show *x* = *y* **by** *simp*
qed
qed

locale *code-morphism* = *morphism f* **for** *f* +
assumes *code-morph*: *inj f*

begin

lemma *inj-core*: *inj* $f^{\mathcal{C}}$
using *code-morph* **unfolding** *core-def inj-on-def* **by** *blast*

lemma *sing-im-core*: *f* [*a*] ∈ (*range* $f^{\mathcal{C}}$)
unfolding *core-def* **by** *simp*

lemma *code-im*: *code* (*range* $f^{\mathcal{C}}$)
using *code-morph morph-concat-map* **unfolding** *inj-on-def code-def core-def*
unfolding *lists-image lists-UNIV* **by** *fastforce*

sublocale *code range* $f^{\mathcal{C}}$
using *code-im*.

sublocale *nonerasing-morphism*
by (*rule nonerI*, *simp add*: *nemp*)

lemma *code-morph-code*: **assumes** *f r* = *f s* **shows** *r* = *s*
proof−
from *code.is-code*[*OF code-im*, *of map* $f^{\mathcal{C}}$ *r map* $f^{\mathcal{C}}$ *s*]
have *map* $f^{\mathcal{C}}$ *r* = *map* $f^{\mathcal{C}}$ *s*
unfolding *morph-concat-map* **using** *range-map-core assms* **by** *blast*
thus *r* = *s*
unfolding *inj-map-eq-map*[*OF inj-core*].

qed

lemma *code-morph-bij*: *bij-betw f UNIV* $\langle$(*range* $f^{\mathcal{C}}$)$\rangle$
unfolding *bij-betw-def*
by (*rule disjE*, *simp-all add*: *range-hull*)
(*rule injI*, *simp add*: *code-morph-code*)

lemma *code-morphism-rev-map*: *code-morphism* (*rev-map f*)
unfolding *code-morphism-def code-morphism-axioms-def*
proof (*rule conjI*)
show *inj* (*rev-map f*)
using *code-morph*
unfolding *inj-def rev-map-arg rev-is-rev-conv*
using *rev-is-rev-conv* **by** *blast*
qed (*simp add*: *rev-map-morph*)

lemma *morph-on-inj-on*:
morphism-on f A inj-on f A
using *morph code-morph-code* **unfolding** *morphism-on-def inj-on-def*
by *blast+*

end

lemma (**in** *morphism*) *code-morphismI*: *inj f* $\Longrightarrow$ *code-morphism f*
by *unfold-locales*

lemma (**in** *nonerasing-morphism*) *code-morphismI'* :
assumes *comm*: $\bigwedge u\ v.\ f\ u = f\ v \Longrightarrow u \cdot v = v \cdot u$
shows *code-morphism f*
proof (*unfold-locales*, *intro injI*)
fix *u v*
assume *f u* = *f v*
then have $u \cdot v = v \cdot u$
by (*fact comm*)
from *comm-eq-im-eq*[*OF this* ‹*f u* = *f v*›]
show *u* = *v*.
qed

4.1.5 Prefix code morphism

locale *prefix-code-morphism* = *nonerasing-morphism* +
assumes
pref-free: $f^{\mathcal{C}}\ a \leq p\ f^{\mathcal{C}}\ b \Longrightarrow a = b$

begin

interpretation *prefrange*: *prefix-code* (*range* $f^{\mathcal{C}}$)
by (*unfold-locales*, *unfold image-iff*)
(*use core-nemp* **in** *metis*, *use pref-free* **in** *fast*)

lemma *inj-core*: *inj* $f^{\mathcal{C}}$
unfolding *inj-on-def* **using** *pref-free* **by** *force*

sublocale *code-morphism*
proof
show *inj f*
proof (*rule injI*)
fix *x y*
assume *f x* = *f y*
hence *map* $f^{\mathcal{C}}$ *x* = *map* $f^{\mathcal{C}}$ *y*
using *prefrange.is-code*[*folded range-map-core*, *of map* $f^{\mathcal{C}}$ *x map* $f^{\mathcal{C}}$ *y*]
unfolding *morph-concat-map* **by** *fast*
with *inj-core*[*folded inj-map*[*of* $f^{\mathcal{C}}$], *unfolded inj-on-def*]
show *x* = *y*
by *fast*
qed
qed

lemma *pref-free-morph*: **assumes** *f r* ≤*p f s* **shows** *r* ≤*p s*
using *assms*
proof (*induction r s rule*: *list-induct2′*)
case (*2 x xs*)
then show *?case*
using *emp-to-emp nonerasing prefix-bot.extremum-unique* **by** *auto*
next
case (*4 x xs y ys*)
then show *?case*
proof−
have $f^{\mathcal{C}}$ *x* ≤*p* $f^{\mathcal{C}}$ *y* · *f ys*
unfolding *core-def* **using** *4.prems*[*unfolded pop-hd*[*of x xs*] *pop-hd*[*of y ys*],
THEN append-prefixD].
from *ruler-pref′*[*OF this*] *prefrange.pref-free*[*OF rangeI rangeI*] *inj-core*
have *x* = *y*
unfolding *inj-on-def* **by** *fastforce*
show *?case*
using *4.IH 4.prems* **unfolding** *pop-hd*[*of x xs*] *pop-hd*[*of y ys*]
unfolding ‹*x* = *y*› **by** *fastforce*
qed
qed *simp-all*

end

4.1.6 Marked morphism

locale *marked-morphism* = *nonerasing-morphism* +
assumes
marked-core: *hd* ($f^{\mathcal{C}}$ *a*) = *hd* ($f^{\mathcal{C}}$ *b*) ⟹ *a* = *b*

begin

lemma *marked-im*: *marked-code* (*range* $f^{\mathcal{C}}$)
by (*unfold-locales*, *unfold image-iff*)
(*use marked-core core-nemp* **in** *metis*)+

interpretation *marked-code* (*range* $f^{\mathcal{C}}$)
using *marked-im*.

lemmas *marked-morph* = *marked-core*[*unfolded core-sing*]

sublocale *prefix-code-morphism*
by (*unfold-locales*, *simp-all add*: *core-nemp marked-core pref-hd-eq*)

lemma *hd-im-eq-hd-eq*: **assumes** $u \neq \varepsilon$ **and** $v \neq \varepsilon$ **and** *hd* (*f u*) = *hd* (*f v*)
shows *hd u* = *hd v*
using *marked-morph*[*OF* ‹*hd* (*f u*) = *hd* (*f v*)›[*unfolded hd-im-hd-hd*[*OF* ‹*u* ≠ *ε*›] *hd-im-hd-hd*[*OF* ‹*v* ≠ *ε*›]]].

lemma *marked-morph-lcp*: *f* (*r* $\wedge_p$ *s*) = *f r* $\wedge_p$ *f s*
by (*rule marked-concat-lcp*[*of map* $f^{\mathcal{C}}$ *r map* $f^{\mathcal{C}}$ *s*, *unfolded map-lcp-conv*[*OF inj-core*] *morph-concat-map*]) *simp-all*

lemma *marked-inj-map*: *inj e* ⟹ *marked-morphism* ((*map e*) ∘ *f*)
unfolding *inj-on-def*
by *unfold-locales*
(*simp add*: *morph*, *simp add*: *code-morph-code*, *simp add*: *core-def core-nemp nemp-to-nemp marked-core list.map-sel*(*1*) *sing-to-nemp*)

end

thm *morphism.nonerI*

lemma (**in** *morphism*) *marked-morphismI*:
(⋀ *a*. *f*[*a*] ≠ *ε*) ⟹ (⋀ *a b*. *a* ≠ *b*) ⟹ *hd* (*f*[*a*]) ≠ *hd* (*f*[*b*]) ⟹ *marked-morphism f*
by *unfold-locales presburger*+

4.1.7 Image length

definition *max-image-length*:: ('*a list* ⇒ '*b list*) ⇒ *nat* (⌈-⌉)
where *max-image-length f* = *Max* (*length*‘(*range* $f^{\mathcal{C}}$))

definition *min-image-length*::('*a list* ⇒ '*b list*) ⇒ *nat* (⌊-⌋)
where *min-image-length f* = *Min* (*length*‘(*range* $f^{\mathcal{C}}$))

lemma *max-im-len-id*: ⌈*id*::('*a list* ⇒ '*a list*)⌉ = *1* **and** *min-im-len-id*: ⌊*id*::('*a list* ⇒ '*a list*)⌋ = *1*
proof–

have *a1*: *length ‘ range* $(\lambda x.\ [x]) = \{1\}$
by *force*
show $\lceil id::('a\ list \Rightarrow 'a\ list)\rceil = 1$ **and** $\lfloor id::('a\ list \Rightarrow 'a\ list)\rfloor = 1$
unfolding *max-image-length-def min-image-length-def core-def id-apply a1*
by *force+*
qed

context *morphism*
begin

lemma *max-im-len-le*: *finite* (*length‘range* $f^{\mathcal{C}}$) $\Longrightarrow |f\ z| \leq |z| * \lceil f \rceil$
proof(*induction z*)
case (*Cons a z*)
have $|f\ [a]| \in$ *length‘*(*range* $f^{\mathcal{C}}$)
by (*simp add*: *core-def*)
hence $|f\ [a]| \leq \lceil f \rceil$
unfolding *max-image-length-def*
using *Cons.prems Max.coboundedI* **by** *metis*
thus *?case*
unfolding *hd-word*[*of a z*] *morph*[*of* [*a*] *z*]
unfolding *lenmorph*
using *Cons.IH*[*OF Cons.prems*] **by** *auto*
qed *simp*

lemma *max-im-len-le-sing*: **assumes** *finite* (*length‘range* $f^{\mathcal{C}}$)
shows $|f\ [a]| \leq \lceil f \rceil$
using *max-im-len-le*[*OF assms*, *of* [*a*]]
unfolding *mult-1 sing-len*.

lemma *min-im-len-ge*: *finite* (*length‘range* $f^{\mathcal{C}}$) $\Longrightarrow\ |z| * \lfloor f \rfloor \leq |f\ z|$
proof(*induction z*)
case (*Cons a z*)
have $|f\ [a]| \in$ *length‘*(*range* $f^{\mathcal{C}}$)
by (*simp add*: *core-def*)
hence $\lfloor f \rfloor \leq |f\ [a]|$
unfolding *min-image-length-def*
by (*meson Cons.prems Min-le*)
thus *?case*
unfolding *hd-word*[*of a z*] *morph*[*of* [*a*] *z*]
unfolding *lenmorph*
using *Cons.IH*[*OF Cons.prems*] **by** *auto*
qed *simp*

lemma *max-im-len-comp-le*: **assumes** *finite-f*: *finite* (*length‘range* $f^{\mathcal{C}}$) **and**
finite-g: *finite* (*length‘range* $g^{\mathcal{C}}$) **and** *morphism g*
shows *finite* (*length ‘ range* $(g \circ f)^{\mathcal{C}}$) $\lceil g \circ f \rceil \leq \lceil f \rceil * \lceil g \rceil$
proof−
interpret *mg*: *morphism g*
by (*simp add*: ‹*morphism g*›)

have $|g\ (f\ [x])| \leq \lceil f \rceil * \lceil g \rceil$ **for** x
proof−
have $|f\ [x]| \leq \lceil f \rceil$
using *finite-f max-im-len-le-sing* **by** *presburger*
thus $|g\ (f\ [x])| \leq \lceil f \rceil * \lceil g \rceil$
by (*meson finite-g le-trans mg.max-im-len-le mult-le-cancel2*)
qed
hence $|(g\ o\ f)^{\mathcal{C}}\ x| \leq \lceil f \rceil * \lceil g \rceil$ **for** x
by (*simp add*: *core-sing*)
hence $l \in$ *length* ‘ *range* $(g \circ f)^{\mathcal{C}} \Longrightarrow l \leq \lceil f \rceil * \lceil g \rceil$ **for** l
by *blast*
thus *finite* (*length* ‘ *range* $(g \circ f)^{\mathcal{C}}$)
using *finite-nat-set-iff-bounded-le* **by** *metis*
from *Max.boundedI*[*OF this*]
show $\lceil g\ o\ f \rceil \leq \lceil f \rceil * \lceil g \rceil$
using ‹$\bigwedge l.\ l \in$ *length* ‘ *range* $(g \circ f)^{\mathcal{C}} \Longrightarrow l \leq \lceil f \rceil * \lceil g \rceil$›
unfolding *max-image-length-def*
by *blast*
qed

lemma *max-im-len-emp*: **assumes** *finite* (*length* ‘ *range* $f^{\mathcal{C}}$)
shows $\lceil f \rceil = 0 \longleftrightarrow (f = (\lambda w.\ \varepsilon))$
proof
show $\lceil f \rceil = 0 \Longrightarrow f = (\lambda w.\ \varepsilon)$
using *max-im-len-le*[*OF assms*] **by** *fastforce*
show $f = (\lambda w.\ \varepsilon) \Longrightarrow \lceil f \rceil = 0$
unfolding *max-image-length-def core-def* **by** *force*
qed

lemmas *max-im-len-le-dom* = *max-im-len-le*[*OF finite-imageI*, *OF finite-imageI*]
and
max-im-len-le-sing-dom = *max-im-len-le-sing*[*OF finite-imageI*, *OF finite-imageI*]
and
min-im-len-ge-dom = *min-im-len-ge*[*OF finite-imageI*, *OF finite-imageI*] **and**
max-im-len-comp-le-dom = *max-im-len-comp-le*[*OF finite-imageI*, *OF finite-imageI*]
and
max-im-len-emp-dom = *max-im-len-emp*[*OF finite-imageI*, *OF finite-imageI*]

lemma *Cons-im*: $f\ (x\#xs) = f^{\mathcal{C}}\ x \cdot f\ xs$
unfolding *core-sing* **using** *pop-hd*.

end

4.1.8 Endomorphism

locale *endomorphism* = *morphism f* **for** f:: $'a\ list \Rightarrow 'a\ list$
begin

lemma *pow-endomorphism*: *endomorphism* $(f^{\frown\frown}k)$
by (*unfold-locales*, *induction k*) (*simp-all add*: *power.power.power-0 morph*)

interpretation *pow-endm*: *endomorphism* $(f^{\frown\frown}k)$
using *pow-endomorphism* **by** *blast*

lemmas *pow-morphism* = *pow-endm.morphism-axioms* **and**
pow-morph = *pow-endm.morph* **and**
pow-emp-to-emp = *pow-endm.emp-to-emp*

lemma *pow-sets-im*: *set w* = *set v* $\Longrightarrow$ *set* $((f^{\frown\frown}k)\ w)$ = *set* $((f^{\frown\frown}k)\ v)$
by(*induct k*, *auto simp add*: *power.power.power-0 set-core-set*)

lemma *fin-len-ran-pow*: *finite* (*length* ‘ *range* $f^{\mathcal{C}}$) $\Longrightarrow$ *finite* (*length* ‘ *range* $(f^{\frown\frown}k)^{\mathcal{C}}$)
proof(*induction k*)
case *0*
have (*w*::*'a list*) $\in$ *range* ($\lambda a.$ [*a*]) $\Longrightarrow$ |*w*| = *1* **for** *w*
by *force*
thus *?case*
unfolding *funpow-0 core-def*
using *finite-nat-set-iff-bounded-le* **by** *auto*
next
case (*Suc k*)
show *?case*
using *pow-endm.max-im-len-comp-le*(*1*)[*of* - *f*, *folded funpow.simps*(*2*), *OF Suc.IH*, *OF Suc.prems Suc.prems morphism-axioms*].
qed

lemma *max-im-len-pow-le*: **assumes** *finite* (*length* ‘ *range* $f^{\mathcal{C}}$) **shows** $\lceil f^{\frown\frown}k\rceil \leq \lceil f\rceil\ ^\frown k$
proof(*induction k*)
have *funpow-1*: $f^{\frown\frown}1 = f$ **by** *simp*
case (*Suc k*)
show *?case*
using *mult-le-mono2*[*OF Suc.IH*[*OF Suc.prems*], *of* $\lceil f \frown 1\rceil$] *pow-endm.max-im-len-comp-le*(*2*)[*OF fin-len-ran-pow*, *OF* ‹*finite* (*length* ‘ *range* $f^{\mathcal{C}}$)› ‹*finite* (*length* ‘ *range* $f^{\mathcal{C}}$)› *morphism-axioms*]
unfolding *compow-Suc funpow-1 comp-apply*
unfolding *power-class.power.power-Suc*
unfolding *mult.commute*[*of* $\lceil f\rceil$]
using *dual-order.trans* **by** *blast*
qed (*simp add*: *max-im-len-id*[*unfolded id-def*])

lemma *max-im-len-pow-le'*: *finite* (*length* ‘ *range* $f^{\mathcal{C}}$) $\Longrightarrow$ $|(f^{\frown\frown}k)\ w| \leq |w| * \lceil f\rceil\ ^\frown k$
using *fin-len-ran-pow le-trans max-im-len-pow-le mult-le-mono2 pow-endm.max-im-len-le*

by *meson*

lemmas *max-im-len-pow-le-dom* = *max-im-len-pow-le*[*OF finite-imageI*, *OF finite-imageI*] **and**
max-im-len-pow-le'-dom = *max-im-len-pow-le'*[*OF finite-imageI*, *OF finite-imageI*]

lemma *funpow-nonerasing-morphism*: **assumes** *nonerasing-morphism f*
shows *nonerasing-morphism* (*f*^^*k*)
proof(*unfold-locales*, *induction k*)
case (*Suc k*)
then show *?case*
using *nonerasing-morphism.nonerasing*[*OF assms*]
unfolding *compow-Suc'* **by** *blast*
qed *simp*

lemma *im-len-pow-mono*: **assumes** *nonerasing-morphism f i* ≤ *j*
shows (|(*f*^^*i*) *w*| ≤ |(*f*^^*j*) *w*|)
using *nonerasing-morphism.im-len-le*[*OF funpow-nonerasing-morphism*[*of j*−*i*], *OF* ‹*nonerasing-morphism f*›, *of* (*f*^^*i*) *w*]
using *funpow-add*[*unfolded comp-apply*, *of j*−*i i f*]
unfolding *diff-add*[*OF* ‹*i* ≤ *j*›]
by *simp*

lemma *fac-mono-pow*: *u* ≤*f* (*f*^^*k*) *w* ⟹ (*f*^^*l*) *u* ≤*f* (*f*^^(*l*+*k*)) *w*
by (*simp add*: *funpow-add pow-endm.fac-mono*)

lemma *rev-map-endomorph*: *endomorphism* (*rev-map f*)
by (*simp add*: *endomorphism.intro rev-map-morph*)

end

4.1.9 Decomposition into primitive roots

definition *root-refine* (*Ref*$_\varrho$ - [*55*] *56*) **where** *root-refine* = (λ *x*. [*ϱ x*]$^{@}$*e*$_\varrho$ *x*)$^{\mathcal{L}}$

lemma *root-ref-morph*: *morphism root-refine*
unfolding *root-refine-def* **using** *list-ext-morph*.

thm *morphism.morph-concat-concat-map*

interpretation *rr-morph*: *morphism root-refine*
using *root-ref-morph* **by** *blast*

lemma *root-ref-def*: *Ref*$_\varrho$ *us* = *concat* (*map* (λ *x*. [*ϱ x*]$^{@}$*e*$_\varrho$ *x*) *us*)
unfolding *root-refine-def list-extension-def*..

lemma *root-ref-refines*: *concat* (*Ref*$_\varrho$ *ws*) = *concat ws*

proof (*induct ws*)
case (*Cons a ws*)
show *?case*
unfolding *rr-morph.Cons-im concat-morph concat.simps Cons.hyps*
by (*simp add*: *root-refine-def*)
qed *simp*

lemmas *root-ref-emp* = *rr-morph.emp-to-emp*

lemma $Ref_\varrho\ [\varepsilon,\varepsilon] = \varepsilon$
unfolding *root-ref-def* **by** *force*

lemma *root-ref-empty-conv*: $Ref_\varrho\ us = \varepsilon \longleftrightarrow set\ us \subseteq \{\varepsilon\}$
proof
show $set\ us \subseteq \{\varepsilon\} \implies Ref_\varrho\ us = \varepsilon$
unfolding *root-ref-def* **by** *auto*
show $Ref_\varrho\ us = \varepsilon \implies set\ us \subseteq \{\varepsilon\}$
unfolding *root-ref-def*
proof (*induction us*, *force*)
case (*Cons a us*)
hence $[\varrho\ a]^{@}\ e_\varrho\ a = \varepsilon$ **and** *hyp*: $concat\ (map\ (\lambda x.\ [\varrho\ x]^{@}\ e_\varrho\ x)\ us) = \varepsilon$
unfolding *list.simps concat.simps* **by** *fast+*
hence $a = \varepsilon$
by *blast*
from *Cons.IH*[*OF hyp*]
obtain k **where** $[\varepsilon]^{@}k = us$
by *blast*
show *?case*
using ‹$a = \varepsilon$› ‹$set\ us \subseteq \{\varepsilon\}$› **by** *simp*
qed
qed

lemma *root-ref-hd*: **assumes** $x \neq \varepsilon$
shows $hd\ (Ref_\varrho\ (x\#xs)) = \varrho\ x$
unfolding *root-ref-def* **by** (*simp-all add*: *hd-append* ‹$x \neq \varepsilon$› *hd-sing-pow*)

lemma *root-ref-injective*:
assumes
emp-not-in: $\varepsilon \notin \mathcal{C}$ **and**
comm-eq: $\bigwedge x\ y.\ x \in \mathcal{C} \implies y \in \mathcal{C} \implies x \cdot y = y \cdot x \implies x = y$ **and**
$us \in lists\ \mathcal{C}\ vs \in lists\ \mathcal{C}$
shows $Ref_\varrho\ us = Ref_\varrho\ vs \implies us = vs$
using *assms*(*3*−*4*)
proof (*induction rule*: *list-induct2'*)
case (*4 x xs y ys*)
show *?case*
proof (*rule arg-cong2*[*of x y xs ys*])
have $x \neq \varepsilon\ y \neq \varepsilon\ x \in \mathcal{C}\ y \in \mathcal{C}\ xs \in lists\ \mathcal{C}\ ys \in lists\ \mathcal{C}$
using ‹$x\ \#\ xs \in lists\ \mathcal{C}$› ‹$y\ \#\ ys \in lists\ \mathcal{C}$› *emp-not-in* **by** *auto*

```
    from root-ref-hd[OF ‹x ≠ ε›, of xs] root-ref-hd[OF ‹y ≠ ε›, of ys]
    have ϱ x = ϱ y
      unfolding 4.prems(1) by presburger
    from same-primroots-comm[OF this, THEN comm-eq[OF ‹x ∈ C› ‹y ∈ C›]]
    show x = y.
    from 4.prems(1)[unfolded this] 4.IH[OF - ‹xs ∈ lists C› ‹ys ∈ lists C›]
    show xs = ys
      unfolding rr-morph.Cons-im cancel by blast
  qed
qed (simp-all add: root-ref-def emp-not-in)

lemma (in code) code-root-ref-injective:
  assumes us ∈ lists C vs ∈ lists C
  shows Ref_ϱ us = Ref_ϱ vs ⟹ us = vs
  using root-ref-injective[OF emp-not-in code-comm-eq assms].

find-theorems set ?w = {?a}

lemma (in code) code-roots-non-overlapping: non-overlapping ((λ x. [ϱ x]^@(e_ϱ x))
‘ C)
proof
  show nemp-dec: ε ∉ (λx. [ϱ x] @ e_ϱ x) ‘ C
  proof
    assume ε ∈ (λx. [ϱ x] @ e_ϱ x) ‘ C
    from this[unfolded image-iff]
    obtain u where u ∈ C and ε = [ϱ u] @ e_ϱ u
      by blast
    from arg-cong[OF this(2), of concat]
    show False
      unfolding concat.simps(1) concat-pow-list-single primroot-exp-eq
      using emp-not-in ‹u ∈ C› by blast
  qed
  fix us vs
  assume us′: us ∈ (λx. [ϱ x] @ e_ϱ x) ‘ C and vs′: vs ∈ (λx. [ϱ x] @ e_ϱ x) ‘ C
  have us ≠ ε vs ≠ ε
    using us′ vs′ nemp-dec by blast+
  from us′[unfolded image-iff]
  obtain u where u ∈ C and us: us = [ϱ u] @ e_ϱ u and  0 < e_ϱ u
    using ‹us ≠ ε› by fastforce
  from vs′[unfolded image-iff]
  obtain v where v ∈ C and vs: vs = [ϱ v] @ e_ϱ v and 0 < e_ϱ v
    using ‹vs ≠ ε› by fastforce
  have set us = {ϱ u}
    using us ‹0 < e_ϱ u› ..
  have set vs = {ϱ v}
    using vs ‹0 < e_ϱ v› ..
  show us = vs if zs ≤p us and zs ≤s vs and zs ≠ ε for zs
  proof–
    have set zs = {ϱ u}
```

using *set-mono-prefix*[*OF* ‹*zs* ≤*p us*›] ‹*zs* ≠ *ε*› **unfolding** ‹*set us* = {*ϱ u*}›
by *auto*
have *set zs* = {*ϱ v*}
using *set-mono-suffix*[*OF* ‹*zs* ≤*s vs*›] ‹*zs* ≠ *ε*› **unfolding** ‹*set vs* = {*ϱ v*}›
by *auto*
hence *ϱ u* = *ϱ v*
unfolding ‹*set zs* = {*ϱ u*}› **by** *simp*
from *same-primroots-comm*[*OF this*]
have *u* = *v*
using *code-comm-eq* [*OF* ‹*u* ∈ *C*› ‹*v* ∈ *C*›] **by** *blast*
thus *us* = *vs*
unfolding ‹*us* = [*ϱ u*] @ e_{ϱ} *u*› ‹*vs* = [*ϱ v*] @ e_{ϱ} *v*› **by** *blast*
qed
show *us* = *vs* **if** *us* ≤*f vs*
proof−
from ‹*set us* = {*ϱ u*}› ‹*set vs* = {*ϱ v*}›
have *ϱ u* = *ϱ v*
using *set-mono-sublist*[*OF* ‹*us* ≤*f vs*›] **by** *force*
from *same-primroots-comm*[*OF this*]
have *u* = *v*
using *code-comm-eq* [*OF* ‹*u* ∈ *C*› ‹*v* ∈ *C*›] **by** *blast*
thus *us* = *vs*
unfolding ‹*us* = [*ϱ u*] @ e_{ϱ} *u*› ‹*vs* = [*ϱ v*] @ e_{ϱ} *v*› **by** *blast*
qed
qed

lemma (**in** *binary-code*) *bin-roots-sings-code*: *non-overlapping* {*Dec* {*ϱ* u_0, *ϱ* u_1} u_0, *Dec* {*ϱ* u_0, *ϱ* u_1} u_1}
using *code-roots-non-overlapping* **unfolding** *primroot-dec* **by** *force*

theorem (**in** *code*) *roots-prim-morph*:
assumes *ws* ∈ *lists C*
and |*ws*| ≠ *1*
and *primitive ws*
shows *primitive* (*Ref*$_{\varrho}$ *ws*)
proof−
let *?R* = λ *x*. [*ϱ x*]@(e_{ϱ} *x*)
interpret *rc*: *non-overlapping ?R* ‘ *C*
using *code-roots-non-overlapping*.

show *?thesis*
unfolding *root-ref-def*
proof (*rule rc.prim-morph*)
show *primitive* (*map ?R ws*)
using *inj-map-prim*[*OF root-dec-inj-on*
‹*ws* ∈ *lists C*› ‹*primitive ws*›].
show *map ?R ws* ∈ *lists* (*?R* ‘ *C*)
using ‹*ws* ∈ *lists C*› *lists-image*[*of ?R C*] **by** *force*
show |*map* (λ*x*. [*ϱ x*] @ e_{ϱ} *x*) *ws*| ≠ *1*

using ‹$|ws| \neq 1$› **by** *simp*
qed
qed

4.2 Primitivity preserving morphisms

locale *primitivity-preserving-morphism* = *nonerasing-morphism* +
assumes *prim-morph* : $2 \leq |u| \Longrightarrow$ *primitive* $u \Longrightarrow$ *primitive* $(f\ u)$
begin

sublocale *code-morphism*
proof (*rule code-morphismI'*, *rule nemp-comm*)
fix $u\ v$
assume $u \neq \varepsilon$ **and** $v \neq \varepsilon$ **and** $f\ u = f\ v$
then have $2 \leq |u \cdot v|$ **and** $2 \leq |u \cdot v \cdot v|$
by (*simp-all flip*: *len-nemp-conv*)
moreover have ¬ *primitive* $(f\ (u \cdot v))$ **and** ¬ *primitive* $(f\ (u \cdot v \cdot v))$
using *pow-nemp-imprim*[*of 2*] *pow-nemp-imprim*[*of 3*] **unfolding** *numeral-nat*
by (*simp-all add*: *morph* ‹$f\ u = f\ v$›) *assumption*+
ultimately have ¬ *primitive* $(u \cdot v)$ **and** ¬ *primitive* $(u \cdot v \cdot v)$
by (*intro notI*; *elim prim-morph*[*rotated*, *elim-format*], *blast*+)+
then show $u \cdot v = v \cdot u$
by (*fact imprim-ext-suf-comm*)
qed

lemmas *code-morph* = *code-morph*

end

4.3 Two morphisms

Solutions and the coincidence pairs are defined for any two mappings

4.3.1 Solutions

definition *minimal-solution* :: *'a list* ⇒ (*'a list* ⇒ *'b list*) ⇒ (*'a list* ⇒ *'b list*) ⇒ *bool*
(- ∈ - =$_M$ - [*80*,*80*,*80*] *51*)
where *min-sol-def*: *minimal-solution* $s\ g\ h \equiv s \neq \varepsilon \wedge g\ s = h\ s$
$\wedge\ (\forall\ s'.\ s' \neq \varepsilon \wedge s' \leq p\ s \wedge g\ s' = h\ s' \longrightarrow s' = s)$

lemma *min-solD*: $s \in g =_M h \Longrightarrow g\ s = h\ s$
using *min-sol-def* **by** *blast*

lemma *min-solD'*: $s \in g =_M h \Longrightarrow s \neq \varepsilon$
using *min-sol-def* **by** *blast*

lemma *min-solD-min*: $s \in g =_M h \Longrightarrow p \neq \varepsilon \Longrightarrow p \leq p\ s \Longrightarrow g\ p = h\ p \Longrightarrow p = s$
by (*simp add*: *min-sol-def*)

lemma *min-solI*[*intro*]: $s \neq \varepsilon \Longrightarrow g\ s = h\ s \Longrightarrow (\bigwedge s'.\ s' \leq p\ s \Longrightarrow s' \neq \varepsilon \Longrightarrow g\ s' = h\ s' \Longrightarrow s' = s) \Longrightarrow s \in g =_M h$
using *min-sol-def* **by** *metis*

lemma *min-sol-sym-iff*: $s \in g =_M h \longleftrightarrow s \in h =_M g$
unfolding *min-sol-def eq-commute*[*of g - h -*] **by** *blast*

lemma *min-sol-sym*[*sym*]: $s \in g =_M h \Longrightarrow s \in h =_M g$
unfolding *min-sol-def eq-commute*[*of g -*].

lemma *min-sol-prefE*:
assumes $g\ r = h\ r$ **and** $r \neq \varepsilon$
obtains e **where** $e \in g =_M h$ **and** $e \leq p\ r$
proof−
let *?min* = $\lambda\ n.$ *take* $n\ r \neq \varepsilon \wedge g\ (take\ n\ r) = h\ (take\ n\ r)$
have *?min* $|r|$
using *assms* **by** *force*
define n **where** $n = (LEAST\ n.$ *?min* $n)$
define e **where** $e = take\ n\ r$
from *Least-le*[*of ?min*, *folded n-def*, *OF* ‹*?min* $|r|$›]
have $n = |e|$
unfolding *e-def* **by** *simp*
show *thesis*
proof (*rule that*)
show $e \leq p\ r$
unfolding *e-def* **using** *take-is-prefix* **by** *blast*
show $e \in g =_M h$
proof (*rule min-solI*)
from *LeastI*[*of ?min*, *OF* ‹*?min* $|r|$›, *folded n-def e-def*]
show $e \neq \varepsilon$ **and** $g\ e = h\ e$
by *blast*+
show *min*: $s = e$ **if** $s \leq p\ e\ s \neq \varepsilon\ g\ s = h\ s$ **for** s
proof−
have $|s| \leq |e|$
using *pref-len*[*OF* ‹$s \leq p\ e$›].
hence *take* $|s|\ r = s$
using ‹$s \leq p\ e$› *pref-take* **unfolding** *e-def* **by** *fastforce*
from *not-less-Least*[*of* $|s|$ *?min*, *folded e-def n-def*, *unfolded this*]
show $s = e$
using *that leI long-pref* **unfolding** ‹$n = |e|$› **by** *fast*
qed
qed
qed
qed

4.3.2 Coincidence pairs

definition *coincidence-set* :: (*'a list* ⇒ *'b list*) ⇒ (*'a list* ⇒ *'b list*) ⇒ (*'a list* × *'a list*) *set* (𝔈)
where *coincidence-set g h* ≡ {(*r*,*s*). *g r* = *h s*}

lemma *coin-set-eq*: (*g* ∘ *fst*)`(𝔈 *g h*) = (*h* ∘ *snd*)`(𝔈 *g h*)
unfolding *coincidence-set-def comp-apply* **using** *Product-Type.Collect-case-prodD*[*of* - λ *x y*. *g x* = *h y*] *image-cong* **by** *auto*

lemma *coin-setD*: *pair* ∈ 𝔈 *g h* ⟹ *g* (*fst pair*) = *h* (*snd pair*)
unfolding *coincidence-set-def* **by** *force*

lemma *coin-setD-iff*: *pair* ∈ 𝔈 *g h* ⟷ *g* (*fst pair*) = *h* (*snd pair*)
unfolding *coincidence-set-def* **by** *force*

lemma *coin-set-sym*: *fst*`(𝔈 *g h*) = *snd* `(𝔈 *h g*)
unfolding *coincidence-set-def*
by (*rule set-eqI*) (*auto simp add*: *image-iff*, *metis*)

lemma *coin-set-inter-fst*: (*g* ∘ *fst*)`(𝔈 *g h*) = *range g* ∩ *range h*
proof
show (*g* ∘ *fst*) ` 𝔈 *g h* ⊆ *range g* ∩ *range h*
proof
fix *x* **assume** *x* ∈ (*g* ∘ *fst*) ` 𝔈 *g h*
then obtain *pair* **where** *x* = *g* (*fst pair*) **and** *pair* ∈ 𝔈 *g h*
by *force*
from *this*(*1*)[*unfolded coin-setD*[*OF this*(*2*)]] *this*(*1*)
show *x* ∈ *range g* ∩ *range h* **by** *blast*
qed
next
show *range g* ∩ *range h* ⊆ (*g* ∘ *fst*) ` 𝔈 *g h*
proof
fix *x* **assume** *x* ∈ *range g* ∩ *range h*
then obtain *r s* **where** *g r* = *h s* **and** *x* = *g r* **by** *blast*
hence (*r*,*s*) ∈ 𝔈 *g h*
unfolding *coincidence-set-def* **by** *blast*
thus *x* ∈ (*g* ∘ *fst*) ` 𝔈 *g h*
unfolding ‹*x* = *g r*› **by** *force*
qed
qed

lemmas *coin-set-inter-snd* = *coin-set-inter-fst*[*unfolded coin-set-eq*]

definition *minimal-coincidence* :: (*'a list* ⇒ *'b list*) ⇒ *'a list* ⇒ (*'a list* ⇒ *'b list*) ⇒ *'a list* ⇒ *bool* ((- -) =$_m$ (- -) [*80*,*81*,*80*,*81*] *51*)
where *min-coin-def*: *minimal-coincidence g r h s* ≡ *r* ≠ ε ∧ *s* ≠ ε ∧ *g r* = *h s* ∧ (∀ *r' s'*. *r'* ≤*p r* ∧ *r'* ≠ ε ∧ *s'* ≤*p s* ∧ *s'* ≠ ε ∧ *g r'* = *h s'* ⟶ *r'* = *r* ∧ *s'* = *s*)

definition *min-coincidence-set* :: (*'a list* ⇒ *'b list*) ⇒ (*'a list* ⇒ *'b list*) ⇒ (*'a list*

$\times$ *'a list*) *set* ($\mathfrak{C}_m$)
where *min-coincidence-set g h* $\equiv$ ({(*r*,*s*) . *g r* $=_m$ *h s*})

lemma *min-coin-minD*: *g r* $=_m$ *h s* $\Longrightarrow$ *r'* $\leq_p$ *r* $\Longrightarrow$ *r'* $\neq \varepsilon \Longrightarrow$ *s'* $\leq_p$ *s* $\Longrightarrow$ *s'* $\neq \varepsilon \Longrightarrow$ *g r'* = *h s'* $\Longrightarrow$ *r'* = *r* $\wedge$ *s'* = *s*
unfolding *min-coin-def* **by** *blast*

lemma *min-coin-setD*: *p* $\in \mathfrak{C}_m$ *g h* $\Longrightarrow$ *g* (*fst p*) $=_m$ *h* (*snd p*)
unfolding *min-coincidence-set-def* **by** *force*

lemma *min-coinD*: *g r* $=_m$ *h s* $\Longrightarrow$ *g r* = *h s*
using *min-coin-def* **by** *blast*

lemma *min-coinD'*: *g r* $=_m$ *h s* $\Longrightarrow$ *r* $\neq \varepsilon \wedge$ *s* $\neq \varepsilon$
using *min-coin-def* **by** *blast*

lemma *min-coin-sub*: $\mathfrak{C}_m$ *g h* $\subseteq \mathfrak{C}$ *g h*
unfolding *coincidence-set-def min-coincidence-set-def*
using *min-coinD* **by** *blast*

lemma *min-coin-defI*: **assumes** *r* $\neq \varepsilon$ **and** *s* $\neq \varepsilon$ **and** *g r* = *h s* **and**
($\bigwedge$ *r' s'*. *r'* $\leq_p$ *r* $\Longrightarrow$ *r'* $\neq \varepsilon \Longrightarrow$ *s'* $\leq_p$ *s* $\Longrightarrow$ *s'* $\neq \varepsilon \Longrightarrow$ *g r'* = *h s'* $\Longrightarrow$ *r'* = *r* $\wedge$ *s'* = *s*)
shows *g r* $=_m$ *h s*
unfolding *min-coin-def*[*rule-format*] **using** *assms* **by** *auto*

lemma *min-coin-sym*[*sym*]: *g r* $=_m$ *h s* $\Longrightarrow$ *h s* $=_m$ *g r*
unfolding *min-coin-def eq-commute*[*of g - h -*] **by** *blast*

lemma *min-coin-sym-iff*: *g r* $=_m$ *h s* $\longleftrightarrow$ *h s* $=_m$ *g r*
using *min-coin-sym* **by** *auto*

lemma *min-coin-set-sym*: *fst*`($\mathfrak{C}_m$ *g h*) = *snd* `($\mathfrak{C}_m$ *h g*)
unfolding *min-coincidence-set-def image-iff*
by (*rule set-eqI*, *rule iffI*) (*simp-all add*: *image-iff min-coin-sym-iff*)

4.3.3 Basics

locale *two-morphisms* = *g*: *morphism g* + *h*: *morphism h* **for** *g h* :: *'a list* $\Rightarrow$ *'b list*

begin

lemma *def-on-sings*:
assumes $\bigwedge$*a*. *a* $\in$ *set u* $\Longrightarrow$ *g* [*a*] = *h* [*a*]
shows *g u* = *h u*
using *assms*
proof (*induct u*)
next

case (*Cons a u*)
then show *?case*
unfolding *g.pop-hd*[*of a u*] *h.pop-hd*[*of a u*] **using** *assms* **by** *simp*
qed *simp*

lemma *def-on-sings-eq*:
assumes $\bigwedge a.\ g\ [a] = h\ [a]$
shows $g = h$
using *def-on-sings*[*OF assms*]
by (*simp add*: *ext*)

lemma *ims-prefs-comp*:
assumes $u \leq p\ u'$ **and** $v \leq p\ v'$ **and** $g\ u' \bowtie h\ v'$ **shows** $g\ u \bowtie h\ v$
using *ruler-comp*[*OF g.pref-mono h.pref-mono*, *OF assms*].

lemma *ims-sufs-comp*:
assumes $u \leq s\ u'$ **and** $v \leq s\ v'$ **and** $g\ u' \bowtie_s h\ v'$ **shows** $g\ u \bowtie_s h\ v$
using *suf-ruler-comp*[*OF g.suf-mono h.suf-mono*, *OF assms*].

lemma *ims-hd-eq-comp*:
assumes $u \neq \varepsilon$ **and** $g\ u = h\ u$ **shows** $g\ [hd\ u] \bowtie h\ [hd\ u]$
using *ims-prefs-comp*[*OF hd-pref*[*OF* ‹$u \neq \varepsilon$›] *hd-pref*[*OF* ‹$u \neq \varepsilon$›]]
unfolding ‹$g\ u = h\ u$› **by** *blast*

lemma *ims-last-eq-suf-comp*:
assumes $u \neq \varepsilon$ **and** $g\ u = h\ u$ **shows** $g\ [last\ u] \bowtie_s h\ [last\ u]$
using *ims-sufs-comp*[*OF hd-pref*[*reversed*, *OF* ‹$u \neq \varepsilon$›] *hd-pref*[*reversed*, *OF* ‹$u \neq \varepsilon$›]]
unfolding ‹$g\ u = h\ u$› **using** *comp-refl*[*reversed*] **by** *blast*

lemma *len-im-le*:
assumes $(\bigwedge a.\ a \in set\ s \Longrightarrow |g\ [a]| \leq |h\ [a]|)$
shows $|g\ s| \leq |h\ s|$
using *assms* **proof** (*induction s*)
case (*Cons a s*)
have *IH-prem*: $\bigwedge a.\ a \in set\ s \Longrightarrow |g\ [a]| \leq |h\ [a]|$ **using** *Cons.prems* **by** *simp*
show $|g\ (a\ \#\ s)| \leq |h\ (a\ \#\ s)|$
unfolding *g.pop-hd*[*of - s*] *h.pop-hd*[*of - s*] *lenmorph*
using *Cons.prems*[*of a*, *simplified*] *Cons.IH*[*OF IH-prem*]
by (*rule add-le-mono*)
qed *simp*

lemma *len-im-less*:
assumes $\bigwedge a.\ a \in set\ s \Longrightarrow |g\ [a]| \leq |h\ [a]|$ **and**
$b \in set\ s$ **and** $|g\ [b]| < |h\ [b]|$
shows $|g\ s| < |h\ s|$
using *assms* **proof** (*induction s arbitrary*: *b*)
case (*Cons a s*)
have *IH-prem*: $\bigwedge a.\ a \in set\ s \Longrightarrow |g\ [a]| \leq |h\ [a]|$ **using** *Cons.prems*(*1*)[*OF*

list.set-intros(2)].
  note *split = g.pop-hd[of - s] h.pop-hd[of - s] lenmorph*
  show $|g\ (a\ \#\ s)| < |h\ (a\ \#\ s)|$
  proof (*cases*)
    assume $a = b$ **show** $|g\ (a\ \#\ s)| < |h\ (a\ \#\ s)|$
      unfolding *split* ‹$a = b$› **using** ‹$|g\ [b]| < |h\ [b]|$› *len-im-le[OF IH-prem]*
      by (*rule add-less-le-mono*)
  next
    assume $a \neq b$
    then have $b \in set\ s$ **using** ‹$b \in set\ (a\ \#\ s)$› **by** *simp*
    show $|g\ (a\ \#\ s)| < |h\ (a\ \#\ s)|$
      unfolding *split* **using** *Cons.prems(1)[OF list.set-intros(1)]*
        Cons.IH[OF IH-prem ‹$b \in set\ s$› ‹$|g\ [b]| < |h\ [b]|$›]
      by (*rule add-le-less-mono*)
  qed
qed *simp*

lemma *solution-eq-len-eq*:
  assumes $g\ s = h\ s$ **and** $\bigwedge a.\ a \in set\ s \Longrightarrow |g\ [a]| = |h\ [a]|$
  shows $\bigwedge a.\ a \in set\ s \Longrightarrow g\ [a] = h\ [a]$
using *assms* **proof** (*induction s*)
  case (*Cons b s*)
  have *nemp*: $b\ \#\ s \neq \varepsilon$ **using** *list.distinct(2)*.
  from *ims-hd-eq-comp[OF nemp* ‹$g\ (b\ \#\ s) = h\ (b\ \#\ s)$›] *Cons.prems(3)[OF list.set-intros(1)]*
  have $*$: $g\ [b] = h\ [b]$ **unfolding** *list.sel(1)* **by** (*fact pref-comp-eq*)
  moreover have $g\ s = h\ s$
    using ‹$g\ (b\ \#\ s) = h\ (b\ \#\ s)$›
    unfolding *g.pop-hd-nemp[OF nemp] h.pop-hd-nemp[OF nemp] list.sel* $*$ **..**
  from *Cons.IH[OF - this Cons.prems(3)[OF list.set-intros(2)]]*
  have $a \in set\ s \Longrightarrow g\ [a] = h\ [a]$ **for** *a*.
  ultimately show $\bigwedge a.\ a \in set\ (b\ \#\ s) \Longrightarrow g\ [a] = h\ [a]$ **by** *auto*
qed *auto*

lemma *rev-maps*: *two-morphisms* (*rev-map g*) (*rev-map h*)
  using *g.rev-map-morph h.rev-map-morph* **by** (*intro two-morphisms.intro*)

lemma *min-solD-min-suf*: **assumes** $sol \in g =_M h$ **and** $s \neq \varepsilon$ $s \leq s\ sol$ **and** $g\ s = h\ s$
  shows $s = sol$
proof (*rule ccontr*)
  assume $s \neq sol$
  from *sufE[OF* ‹$s \leq s\ sol$›]
  obtain *y* **where** $sol = y \cdot s$.
  hence $y \neq \varepsilon$
    using ‹$s \neq sol$› **by** *force*
  have $g\ y = h\ y$
    using *min-solD[OF* ‹$sol \in g =_M h$›, *unfolded* ‹$sol = y \cdot s$›]
    unfolding *g.morph h.morph* ‹$g\ s = h\ s$› **by** *blast*

from *min-solD-min*[*OF* ‹*sol* ∈ *g* =$_M$ *h*› ‹*y* ≠ *ε*› - *this*]
have *y* = *sol*
using ‹*sol* = *y* · *s*› **by** *blast*
thus *False*
using ‹*sol* = *y* · *s*› ‹*s* ≠ *ε*› **by** *fast*
qed

lemma *min-sol-rev*[*reversal-rule*]:
assumes *s* ∈ *g* =$_M$ *h*
shows (*rev s*) ∈ (*rev-map g*) =$_M$ (*rev-map h*)
unfolding *min-sol-def*[*of* - *rev-map g rev-map h*, *reversed*]
using *min-solD*[*OF assms*] *min-solD'*[*OF assms*] *min-solD-min-suf*[*OF assms*]
by *blast*

lemma *coin-set-lists-concat*: *ps* ∈ *lists* (𝔈 *g h*) ⟹ *g* (*concat* (*map fst ps*)) = *h* (*concat* (*map snd ps*))
unfolding *coincidence-set-def*
by (*induct ps*, *simp*, *auto simp add*: *g.morph h.morph*)

lemma *coin-set-hull*: ⟨*snd* ‘(𝔈 *g h*)⟩ = *snd* ‘(𝔈 *g h*)
proof (*rule equalityI*, *rule subsetI*)
fix *x* **assume** *x* ∈ ⟨*snd* ‘ 𝔈 *g h*⟩
then obtain *xs* **where** *xs* ∈ *lists* (*snd* ‘ 𝔈 *g h*) **and** *concat xs* = *x*
using *hull-concat-lists0* **by** *blast*
then obtain *ps* **where** *ps* ∈ *lists* (𝔈 *g h*) **and** *map snd ps* = *xs*
unfolding *image-iff lists-image* **by** *blast*
from *coin-set-lists-concat*[*OF this*(*1*), *unfolded this*(*2*) ‹*concat xs* = *x*›]
show *x* ∈ *snd* ‘ 𝔈 *g h*
unfolding *coincidence-set-def* **by** *force*
qed *simp*

lemma *min-sol-sufE*:
assumes *g r* = *h r* **and** *r* ≠ *ε*
obtains *e* **where** *e* ∈ *g* =$_M$ *h* **and** *e* ≤*s r*
using *assms*
proof (*induction* |*r*| *arbitrary*: *r thesis rule*: *less-induct*)
case *less*
then show *thesis*
proof–
from *min-sol-prefE*[*of g r h*, *OF* ‹*g r* = *h r*› ‹*r* ≠ *ε*›]
obtain *p* **where** *p* ∈ *g* =$_M$ *h* **and** *p* ≤*p r*.
show *thesis*
proof (*cases p* = *r*, (*use less.prems*(*1*)[*OF* ‹*p* ∈ *g* =$_M$ *h*›] **in** *fast*))
assume *p* ≠ *r*
from *prefE*[*OF* ‹*p* ≤*p r*›]
obtain *r'* **where** *r* = *p* · *r'*.
have *g r'* = *h r'*
using ‹*g r* = *h r*›[*unfolded* ‹*r* = *p* · *r'*› *g.morph h.morph min-solD*[*OF* ‹*p* ∈ *g* =$_M$ *h*›] *cancel*].

```
      from ‹p ≠ r› ‹r = p · r'›
      have r' ≠ ε by fast
      from min-solD'[OF ‹p ∈ g =M h›] ‹r = p · r'›
      have |r'| < |r| by fastforce
      from less.hyps[OF this - ‹g r' = h r'› ‹r' ≠ ε›]
      obtain e where e ∈ g =M h e ≤s r'.
      from less.prems(1)[OF this(1), unfolded ‹r = p · r'›, OF suf-ext, OF this(2)]
      show thesis.
    qed
  qed
qed

lemma min-sol-primitive: assumes sol ∈ g =M h shows primitive sol
proof (rule ccontr)
  have sol ≠ ε
    using assms min-sol-def by auto
  assume ¬ primitive sol
  from not-prim-primroot-expE[OF this]
  obtain k where (ϱ sol)^@k = sol 2 ≤ k.
  hence 0 < k by linarith
  note  min-solD[OF assms]
  have g (ϱ sol) = h (ϱ sol)
    by (rule pow-eq-eq[OF - ‹0 < k›])
    (unfold g.pow-morph[of k ϱ sol, symmetric] h.pow-morph[of k ϱ sol, symmetric]
‹(ϱ sol)^@k = sol›, fact)
  thus False
     using ‹¬ primitive sol›  min-solD-min[OF ‹sol ∈ g =M h› primroot-nemp
primroot-pref] ‹sol ≠ ε›
    unfolding prim-primroot-conv[OF ‹sol ≠ ε›, symmetric] by blast
qed

lemma prim-sol-two-sols:
  assumes g u = h u and ¬ u ∈ g =M h and primitive u
  obtains s1 s2 where s1 ∈ g =M h and s2 ∈ g =M h and s1 ≠ s2
proof−
  show thesis
    using assms
  proof (induction |u| arbitrary: u rule: less-induct)
    case less
    then show ?case
    proof−
      obtain s1 where s1 ∈ g =M h and s1 ≤p u
       using min-sol-prefE[of g u h, OF ‹g u = h u› prim-nemp[OF ‹primitive u›]].
      obtain u' where s1 · u' = u
        using ‹s1 ≤p u› unfolding prefix-def by blast
      have g u' = h u'
        using ‹g u = h u›[folded ‹s1 · u' = u›]
        unfolding g.morph h.morph  min-solD[OF ‹s1 ∈ g =M h›] cancel.
      have u' ≠ ε
```

```
      using ‹s1 ∈ g =_M h› ‹¬ u ∈ g =_M h›[folded ‹s1 · u' = u›] by force
    obtain exp where (ϱ u')^@exp = u' 0 < exp
      using primroot-expE.
     from pow-eq-eq[of exp g (ϱ u') h (ϱ u'), folded g.pow-morph h.pow-morph,
unfolded this(1), OF ‹g u' = h u'› ‹0 < exp›]
    have g (ϱ u') = h (ϱ u').
    have |ϱ u'| < |u|
      using add-strict-increasing[OF nemp-len [OF min-solD'[OF ‹s1 ∈ g =_M
h›]] primroot-len-le[OF ‹u' ≠ ε›]]
      unfolding lenarg[OF ‹s1 · u' = u›, unfolded lenmorph].
    show thesis
    proof (cases)
      assume ϱ u' ∈ g =_M h
      have ϱ u' ≠ s1
          using ‹primitive u›[folded ‹s1 · u' = u›]  comm-not-prim[OF prim-
root-nemp[OF ‹u' ≠ ε›] ‹u' ≠ ε›  comm-primroot[symmetric]] by fast
      from that[OF ‹ϱ u' ∈ g =_M h› ‹s1 ∈ g =_M h› this]
      show thesis.
    next
      assume ¬ ϱ u' ∈ g =_M h
      from less.hyps[OF ‹|ϱ u'| < |u|› ‹g (ϱ u') = h (ϱ u')› this]
      show thesis
        using ‹u' ≠ ε› by blast
    qed
   qed
  qed
qed

lemma prim-sols-two-sols:
  assumes g r = h r and g s = h s and primitive s and primitive r and r ≠ s
  obtains s1 s2 where s1 ∈ g =_M h and s2 ∈ g =_M h and s1 ≠ s2
  using prim-sol-two-sols assms by blast

end
```

4.3.4 Factor intepretation of morphic images

```
context morphism
begin

lemma image-fac-interp': assumes w ≤f f z w ≠ ε
  obtains p w-pred s where w-pred ≤f z p w s ∼_I (map f^C w-pred)
proof–
  let ?fzs = map f^C z
  have w ≤f concat (map f^C z)
    by (simp add: assms(1) morph-concat-map)

  from fac-fac-interpE'[OF ‹w ≤f concat (map f^C z)› ‹w ≠ ε›]
  obtain p s ws where p w s ∼_I ws ws ≤f ?fzs
```

by *blast*

obtain *w-pred* **where** $ws = map\ f^{\mathcal{C}}\ w\text{-}pred\ w\text{-}pred \leq_f z$
using ‹$ws \leq_f map\ f^{\mathcal{C}}\ z$› *sublist-map-rightE* **by** *blast*

show *?thesis*
using ‹$p\ w\ s \sim_{\mathcal{I}} ws$› ‹$w\text{-}pred \leq_f z$› ‹$ws = map\ f^{\mathcal{C}}\ w\text{-}pred$› *that* **by** *blast*
qed

lemma *image-fac-interp*: **assumes** $u{\cdot}w{\cdot}v = f\ z\ w \neq \varepsilon$
obtains *p w-pred s u-pred v-pred* **where**
$u\text{-}pred{\cdot}w\text{-}pred{\cdot}v\text{-}pred = z\ p\ w\ s \sim_{\mathcal{I}} (map\ f^{\mathcal{C}}\ w\text{-}pred)$
$u = (f\ u\text{-}pred){\cdot}p\ v = s{\cdot}(f\ v\text{-}pred)$
proof–
let *?fzs* $= map\ f^{\mathcal{C}}\ z$

have $u{\cdot}w{\cdot}v = concat\ (map\ f^{\mathcal{C}}\ z)$
by (*simp add*: *assms*(*1*) *morph-concat-map*)

from *fac-fac-interpE*[*OF* ‹$u{\cdot}w{\cdot}v = concat\ (map\ f^{\mathcal{C}}\ z)$› ‹$w \neq \varepsilon$›]
obtain *ps ss p s ws* **where** $p\ w\ s \sim_{\mathcal{I}} ws\ ps{\cdot}ws{\cdot}ss = ?fzs\ concat\ ps \cdot p = u\ \ s \cdot concat\ ss = v$
by *metis*

obtain *w-pred u-pred v-pred* **where** $ws = map\ f^{\mathcal{C}}\ w\text{-}pred\ ps = map\ f^{\mathcal{C}}\ u\text{-}pred$
$ss = map\ f^{\mathcal{C}}\ v\text{-}pred\ u\text{-}pred{\cdot}w\text{-}pred{\cdot}v\text{-}pred = z$
using ‹$ps \cdot ws \cdot ss = map\ f^{\mathcal{C}}\ z$›[*unfolded append-eq-map-conv*]
by *blast*

show *?thesis*
using ‹$concat\ ps \cdot p = u$› ‹$p\ w\ s \sim_{\mathcal{I}} ws$› ‹$ps = map\ f^{\mathcal{C}}\ u\text{-}pred$› ‹$s \cdot concat\ ss = v$› ‹$ss = map\ f^{\mathcal{C}}\ v\text{-}pred$› ‹$u\text{-}pred \cdot w\text{-}pred \cdot v\text{-}pred = z$› ‹$ws = map\ f^{\mathcal{C}}\ w\text{-}pred$› *morph-concat-map that* **by** *blast*
qed

lemma *image-fac-interp-mid*: **assumes** $p\ w\ s \sim_{\mathcal{I}} map\ f^{\mathcal{C}}\ w\text{-}pred\ 2 \leq |w\text{-}pred|$
obtains *pw sw* **where**
$w = pw \cdot (f\ (butlast\ (tl\ w\text{-}pred))) \cdot sw\ p{\cdot}pw = f\ [hd\ w\text{-}pred]\ sw{\cdot}s = f\ [last\ w\text{-}pred]$
proof–

note *fac-interpD*[*OF* ‹$p\ w\ s \sim_{\mathcal{I}} map\ f^{\mathcal{C}}\ w\text{-}pred$›, *unfolded morph-concat-map*]
note $butl = hd\text{-}middle\text{-}last$[*OF* ‹$2 \leq |w\text{-}pred|$›[*folded One-less-Two-le-iff*]]

have $w\text{-}pred \neq \varepsilon$
using *assms*(*2*) **by** *force*

obtain pw' **where**
$p \cdot pw' = hd\ (map\ f^{\mathcal{C}}\ w\text{-}pred)$

using *sprefE*[*OF* ‹*p* <*p hd* (*map* $f^{\mathcal{C}}$ *w-pred*)›] *prefixE* **by** *metis*
hence *pw'*: $p \cdot pw' = f\ [hd\ w\text{-}pred]$
unfolding *core-def*
unfolding *hd-map*[*OF* ‹*w-pred* ≠ ε›, *of* $f^{\mathcal{C}}$, *unfolded core-def*].

obtain *sw'* **where**
$sw' \cdot s = last\ (map\ f^{\mathcal{C}}\ (w\text{-}pred))$
using *sprefE*[*reversed*, *OF* ‹*s* <*s last* (*map* $f^{\mathcal{C}}$ *w-pred*)›] *suffix-def* **by** *metis*
hence *sw'* : $sw' \cdot s = f\ [last\ (w\text{-}pred)]$
unfolding *core-def*
unfolding *last-map*[*OF* ‹*w-pred* ≠ ε›, *of* $f^{\mathcal{C}}$, *unfolded core-def*].

have $w = pw' \cdot f\ (butlast\ (tl\ w\text{-}pred)) \cdot sw'$
using ‹$p \cdot w \cdot s = f$ *w-pred*›[*unfolded arg-cong*[*OF butl*[*symmetric*], *of f*]]
unfolding *morph*
unfolding *pw'*[*symmetric*] *sw'*[*symmetric*]
by *simp*
thus *?thesis*
using *pw' sw' that* **by** *blast*
qed

end

4.3.5 Two nonerasing morphisms

Minimal coincidence pairs and minimal solutions make good sense for nonerasing morphisms only.

locale *two-nonerasing-morphisms* = *two-morphisms* +
g: *nonerasing-morphism g* +
h: *nonerasing-morphism h*

begin

thm *g.morph*
thm *g.emp-to-emp*

lemma *two-nonerasing-morphisms-swap*: *two-nonerasing-morphisms h g*
by *unfold-locales*

lemma *noner-eq-emp-iff*: $g\ u = h\ v \Longrightarrow u = \varepsilon \longleftrightarrow v = \varepsilon$
by (*metis g.emp-to-emp g.nonerasing h.emp-to-emp h.nonerasing*)

lemma *min-coin-rev*:
assumes $g\ r =_m h\ s$
shows (*rev-map g*) (*rev r*) $=_m$ (*rev-map h*) (*rev s*)
proof (*rule min-coin-defI*)
show *rev r* ≠ ε **and** *rev s* ≠ ε
using *min-coinD'*[*OF* ‹$g\ r =_m h\ s$›] **by** *simp-all*
show *rev-map g* (*rev r*) = *rev-map h* (*rev s*)

unfolding *rev-map-def* **using** *min-coinD*[*OF* ‹*g r* =$_m$ *h s*›] **by** *auto*
next
fix *r′ s′* **assume** *r′* ≤*p rev r r′* ≠ *ε s′* ≤*p rev s s′* ≠ *ε rev-map g r′* = *rev-map h s′*
then obtain *r″ s″* **where** *r″*· *rev r′* = *r* **and** *s″*· *rev s′* = *s*
using ‹*s′* ≤*p rev s*› ‹*r′* ≤*p rev r*›
unfolding *pref-rev-suf-iff rev-rev-ident* **using** *sufD* **by** (*auto simp add*: *suffix-def*)
have *g* (*rev r′*) = *h* (*rev s′*)
using ‹*rev-map g r′* = *rev-map h s′*›[*unfolded rev-map-def rev-is-rev-conv*] **by** *simp*
hence *g r″* = *h s″*
using *min-coinD*[*OF* ‹*g r* =$_m$ *h s*›, *folded* ‹*r″*· *rev r′* = *r*› ‹*s″*· *rev s′* = *s*›, *unfolded g.morph h.morph*] **by** *simp*
have *r″* ≠ *r*
using ‹*r′* ≤*p rev r*› ‹*r′* ≠ *ε*› ‹*r″* · *rev r′* = *r*› **by** *auto*
hence *r″* = *ε* ∨ *s″* = *ε*
using *min-coin-minD*[*OF* ‹*g r* =$_m$ *h s*› - - - ‹*g r″* = *h s″*›, *THEN conjunct1*] ‹*r″*· *rev r′* = *r*› ‹*s″*· *rev s′* = *s*› ‹*g r″* = *h s″*› **by** *blast*
hence *r″* = *ε* **and** *s″* = *ε*
using *noner-eq-emp-iff*[*OF* ‹*g r″* = *h s″*›] **by** *force*+
thus *r′* = *rev r* ∧ *s′* = *rev s*
using ‹*r″*· *rev r′* = *r*› ‹*s″*· *rev s′* = *s*› **by** *auto*
qed

lemma *min-coin-pref-eq*:
assumes *g e* =$_m$ *h f* **and** *g e′* = *h f′* **and** *e′* ≤*p e* **and** *e′* ≠ *ε* **and** *f′* ⋈ *f*
shows *e′* = *e* **and** *f′* = *f*
proof−
have *f* ≠ *ε* **and** *g e* = *h f*
using ‹*g e* =$_m$ *h f*›[*unfolded min-coin-def*] **by** *blast*+
have *f′* ≠ *ε*
using ‹*g e′* = *h f′*› ‹*e′* ≠ *ε*› **by** (*simp add*: *noner-eq-emp-iff*)
from *g.pref-mono*[*OF* ‹*e′* ≤*p e*›, *unfolded* ‹*g e* = *h f*› ‹*g e′* = *h f′*›]
have *f′* ≤*p f*
using *pref-compE*[*OF* ‹*f′* ⋈ *f*›] ‹*f′* ≠ *ε*› *h.pref-mono h.pref-morph-pref-eq* **by** *metis*
with ‹*g e* =$_m$ *h f*›[*unfolded min-coin-def*]
show *e′* = *e* **and** *f′* = *f*
using ‹*g e′* = *h f′*› ‹*e′* ≤*p e*› ‹*e′* ≠ *ε*› ‹*f′* ≠ *ε*› **by** *blast*+
qed

lemma *min-coin-prefE*:
assumes *g r* = *h s* **and** *r* ≠ *ε*
obtains *e f* **where** *g e* =$_m$ *h f* **and** *e* ≤*p r* **and** *f* ≤*p s* **and** *hd e* = *hd r*
proof−
define *P* **where** *P* = (*λ k*. ∃ *e f*. *g e* = *h f* ∧ *e* ≠ *ε* ∧ *e* ≤*p r* ∧ *f* ≤*p s* ∧ |*e*| = *k*)
define *d* **where** *d* = (*LEAST k*. *P k*)

```
  obtain e f where g e = h f and e ≠ ε and  e ≤p r and  f ≤p s and |e| = d
    using ‹g r = h s› LeastI[of P |r|]  P-def assms(2) d-def by blast
  hence f ≠ ε
    using noner-eq-emp-iff by blast
  have r' ≤p e ⟹ r'≠ ε⟹ s' ≤p f ⟹ s' ≠ ε ⟹ g r' = h s' ⟹ r' = e ∧ s'
= f for r' s'
  proof–
    assume r' ≤p e and r' ≠ ε and s' ≤p f and s' ≠ ε and g r' = h s'
    hence P |r'|
      unfolding P-def using ‹e ≤p r› ‹f ≤p s› ‹r' ≠ ε›
          pref-trans[OF ‹r' ≤p e› ‹e ≤p r›]
          pref-trans[OF ‹s' ≤p f› ‹f ≤p s›] by blast
    from Least-le[of P, OF this, folded ‹|e| = d› d-def]
    have r' = e
      using  long-pref[OF ‹r' ≤p e›] by blast
    from ‹g e = h f›[folded this, unfolded ‹g r' = h s'›] this
    show  ?thesis
      using ‹s' ≤p f› h.pref-morph-pref-eq
      by simp
  qed
  hence g e =m h f
    unfolding min-coin-def using ‹e ≠ ε› ‹f ≠ ε› ‹g e = h f› by blast
  from that[OF this ‹e ≤p r› ‹f ≤p s› pref-hd-eq[OF ‹e ≤p r› ‹e ≠ ε›]]
  show thesis.
qed

lemma min-coin-dec: assumes g e = h f
  obtains ps where concat (map fst ps) = e and concat (map snd ps) = f and
    ⋀ p. p ∈ set ps ⟹ g (fst p) =m h (snd p)
  using assms
proof (induct |e| arbitrary: e f thesis rule: less-induct)
  case less
  then show ?case
  proof–
    show thesis
    proof (cases e = ε)
      assume e = ε
      hence f = ε using ‹g e = h f›
        using noner-eq-emp-iff by auto
      from less.prems(1)[of ε] ‹e = ε› ‹f = ε›
      show thesis by simp
    next
      assume e ≠ ε
      from min-coin-prefE[OF ‹g e  = h f› this]
     obtain e1 e2 f1 f2 where g e1 =m h f1 and e1 · e2 = e and f1 · f2 = f and
e1 ≠ ε and f1 ≠ ε
        using min-coinD' prefD by metis
      have g e2 = h f2
        using ‹g e  = h f›[folded ‹e1 · e2 = e› ‹f1 · f2 = f›, unfolded g.morph
```

h.morph min-coinD[*OF* ‹*g e*$_1$ =$_m$ *h f*$_1$›] *cancel*].
have |*e*$_2$| < |*e*|
using *lenarg*[*OF* ‹*e*$_1$ · *e*$_2$ = *e*›] *nemp-len*[*OF* ‹*e*$_1$ ≠ ε›] **unfolding** *lenmorph*
by *linarith*
from *less.hyps*[*OF* ‹|*e*$_2$| < |*e*|› - ‹*g e*$_2$ = *h f*$_2$›]
obtain *ps′* **where** *concat* (*map fst ps′*) = *e*$_2$ **and** *concat* (*map snd ps′*) = *f*$_2$
and ⋀*p*. *p* ∈ *set ps′* ⟹ *g* (*fst p*) =$_m$ *h* (*snd p*)
by *blast*
show *thesis*
proof(*rule less.prems*(*1*)[*of* (*e*$_1$,*f*$_1$)#*ps′*])
show *concat* (*map fst* ((*e*$_1$, *f*$_1$) # *ps′*)) = *e*
using ‹*concat* (*map fst ps′*) = *e*$_2$› ‹*e*$_1$ · *e*$_2$ = *e*› **by** *simp*
show *concat* (*map snd* ((*e*$_1$, *f*$_1$) # *ps′*)) = *f*
using ‹*concat* (*map snd ps′*) = *f*$_2$› ‹*f*$_1$ · *f*$_2$ = *f*› **by** *simp*
show ⋀*p*. *p* ∈ *set* ((*e*$_1$, *f*$_1$) # *ps′*) ⟹ *g* (*fst p*) =$_m$ *h* (*snd p*)
using ‹⋀*p*. *p* ∈ *set ps′* ⟹ *g* (*fst p*) =$_m$ *h* (*snd p*)› ‹*g e*$_1$ =$_m$ *h f*$_1$› **by** *auto*
qed
qed
qed
qed

lemma *min-coin-code*:
assumes *xs* ∈ *lists* (𝔈$_m$ *g h*) **and** *ys* ∈ *lists* (𝔈$_m$ *g h*) **and**
concat (*map fst xs*) = *concat* (*map fst ys*) **and**
concat (*map snd xs*) = *concat* (*map snd ys*)
shows *xs* = *ys*
using *assms*
proof (*induction xs ys rule*: *list-induct2′*)
case (*2 x xs*)
then show *?case*
using *min-coin-setD*[*THEN min-coinD′*, *of x g h*] *listsE*[*OF* ‹*x* # *xs* ∈ *lists*
(𝔈$_m$ *g h*)›] **by** *force*
next
case (*3 y ys*)
then show *?case*
using *min-coin-setD*[*of y g h*, *THEN min-coinD′*] *listsE*[*OF* ‹*y* # *ys* ∈ *lists*
(𝔈$_m$ *g h*)›] **by** *auto*
next
case (*4 x xs y ys*)
then show *?case*
proof–
have *concat* (*map fst* (*x*#*xs*)) = *fst x* · *concat* (*map fst xs*)
concat (*map fst* (*y*#*ys*)) = *fst y* · *concat* (*map fst ys*)
concat (*map snd* (*x*#*xs*)) = *snd x* · *concat* (*map snd xs*)
concat (*map snd* (*y*#*ys*)) = *snd y* · *concat* (*map snd ys*)
by *auto*
from *eqd-comp*[*OF* ‹*concat* (*map fst* (*x*#*xs*)) = *concat* (*map fst* (*y*#*ys*))›[*unfolded*
this]] *eqd-comp*[*OF* ‹*concat* (*map snd* (*x*#*xs*)) = *concat* (*map snd* (*y*#*ys*))›[*unfolded*
this]]

```
    have fst x ⋈ fst y and snd x ⋈ snd y.
    have g (fst y) =_m h (snd y) and g (fst x) =_m h (snd x)
      by (use min-coin-setD listsE[OF ‹y # ys ∈ lists (𝔈_m g h)›] in blast)
         (use min-coin-setD listsE[OF ‹x # xs ∈ lists (𝔈_m g h)›] in blast)
    from min-coin-pref-eq[OF this(1) min-coinD[OF this(2)]] - - ‹snd x ⋈ snd y›]
         min-coin-pref-eq[OF this(2) min-coinD[OF this(1)]] - - pref-comp-sym[OF
‹snd x ⋈ snd y›]] min-coinD'[OF this(1)] min-coinD'[OF this(2)]
    have fst x = fst y and snd x = snd y
      using pref-compE[OF ‹fst x ⋈ fst y›] by metis+
    hence eq: concat (map fst xs) = concat (map fst ys)
              concat (map snd xs) = concat (map snd ys)
      using 4.prems(3–4) by fastforce+
    have xs ∈ lists (𝔈_m g h) ys ∈ lists (𝔈_m g h)
      using 4.prems(1–2) by fastforce+
    from 4.IH(1)[OF this eq] prod-eqI[OF ‹fst x = fst y› ‹snd x = snd y›]
    show x # xs = y # ys
      by blast
  qed
qed simp

lemma coin-closed: ps ∈ lists (𝔈 g h) ⟹ (concat (map fst ps), concat (map snd
ps)) ∈ 𝔈 g h
  unfolding coincidence-set-def
  by (induct ps, simp, auto simp add: g.morph h.morph)

lemma min-coin-gen-snd: ⟨snd ‘ (𝔈_m g h)⟩ = snd ‘(𝔈 g h)
proof
  show ⟨snd ‘ 𝔈_m g h⟩ ⊆ snd ‘ 𝔈 g h
  proof
    fix x assume x ∈ ⟨snd ‘ 𝔈_m g h⟩
    then obtain xs where xs ∈ lists (snd ‘ 𝔈_m g h) and x = concat xs
      using hull-concat-lists0 by blast
    then obtain ps where ps ∈ lists (𝔈_m g h) and xs = map snd ps
      unfolding lists-image image-iff by blast
    from min-coin-sub coin-closed this(1)
    have (concat (map fst ps), x) ∈ 𝔈 g h
      unfolding ‹x = concat xs› ‹xs = map snd ps› by fast
    thus x ∈ snd ‘ 𝔈 g h by force
  qed
  show snd ‘ 𝔈 g h ⊆ ⟨snd ‘ 𝔈_m g h⟩
  proof
    fix x assume x ∈ snd ‘ 𝔈 g h
    then obtain r where g r = h x
      unfolding image-iff coincidence-set-def by force
    from min-coin-dec[OF this]
    obtain ps where concat (map snd ps) = x and ⋀p. p ∈ set ps ⟹ g (fst p)
=_m h (snd p) by metis
    thus x ∈ ⟨snd ‘ 𝔈_m g h⟩
      unfolding min-coincidence-set-def image-def by fastforce
```

qed
qed

lemma *min-coin-gen-fst*: $\langle fst\ `\ (\mathfrak{C}_m\ g\ h)\rangle = fst\ `(\mathfrak{C}\ g\ h)$
using *two-nonerasing-morphisms.min-coin-gen-snd*[*folded coin-set-sym min-coin-set-sym*, *OF two-nonerasing-morphisms-swap*].

lemma *min-coin-code-snd*:
assumes *code-morphism g* **shows** *code* (*snd* ` ($\mathfrak{C}_m$ *g h*))
proof
fix *xs ys* **assume** *xs* ∈ *lists* (*snd* ` $\mathfrak{C}_m$ *g h*) **and** *ys* ∈ *lists* (*snd* ` $\mathfrak{C}_m$ *g h*)
then obtain *psx psy* **where** *psx* ∈ *lists* ($\mathfrak{C}_m$ *g h*) **and** *xs* = *map snd psx* **and**
psy ∈ *lists* ($\mathfrak{C}_m$ *g h*) **and** *ys* = *map snd psy*
unfolding *image-iff lists-image* **by** *blast+*
have *eq1*: *g* (*concat* (*map fst psx*)) = *h* (*concat xs*)
using ‹*psx* ∈ *lists* ($\mathfrak{C}_m$ *g h*)› ‹*xs* = *map snd psx*› *min-coin-sub*[*of g h*]
coin-set-lists-concat **by** *fastforce*
have *eq2*: *g* (*concat* (*map fst psy*)) = *h* (*concat ys*)
using ‹*psy* ∈ *lists* ($\mathfrak{C}_m$ *g h*)› ‹*ys* = *map snd psy*› *min-coin-sub*[*of g h*]
coin-set-lists-concat **by** *fastforce*
assume *concat xs* = *concat ys*
from *arg-cong*[*OF this*, *of h*, *folded eq1 eq2*]
have *concat* (*map fst psx*) = *concat* (*map fst psy*)
using *code-morphism.code-morph-code*[*OF* ‹*code-morphism g*›] **by** *auto*
have *concat* (*map snd psx*) = *concat* (*map snd psy*)
using ‹*concat xs* = *concat ys*› ‹*xs* = *map snd psx*› ‹*ys* = *map snd psy*› **by** *auto*
from *min-coin-code*[*OF* ‹*psx* ∈ *lists* ($\mathfrak{C}_m$ *g h*)› ‹*psy* ∈ *lists* ($\mathfrak{C}_m$ *g h*)› ‹*concat* (*map fst psx*) = *concat* (*map fst psy*)› *this*]
show *xs* = *ys*
unfolding ‹*xs* = *map snd psx*› ‹*ys* = *map snd psy*› **by** *blast*
qed

lemma *min-coin-code-fst*:
code-morphism h ⟹ *code* (*fst* ` ($\mathfrak{C}_m$ *g h*))
using *two-nonerasing-morphisms.min-coin-code-snd*[*OF two-nonerasing-morphisms-swap*, *folded min-coin-set-sym*].

lemma *min-coin-basis-snd*:
assumes *code-morphism g*
shows $\mathfrak{B}$ (*snd* `($\mathfrak{C}$ *g h*)) = *snd* ` ($\mathfrak{C}_m$ *g h*)
unfolding *min-coin-gen-snd*[*symmetric*] *basis-of-hull*
using *min-coin-code-snd*[*OF assms*] *code.code-is-basis* **by** *blast*

lemma *min-coin-basis-fst*:
code-morphism h ⟹ $\mathfrak{B}$ (*fst* `($\mathfrak{C}$ *g h*)) = *fst* ` ($\mathfrak{C}_m$ *g h*)
using *two-nonerasing-morphisms.min-coin-basis-snd*[*folded coin-set-sym min-coin-set-sym*, *OF two-nonerasing-morphisms-swap*].

lemma *sol-im-len-less*: **assumes** *g u* = *h u* **and** *g* ≠ *h* **and** *set u* = *UNIV*

shows $|u| < |g\ u|$
proof (*rule ccontr*)
assume $\neg\ |u| < |g\ u|$
hence $|u| = |g\ u|$ **and** $|u| = |h\ u|$
unfolding ‹$g\ u = h\ u$› **using** *h.im-len-le le-neq-implies-less* **by** *blast+*
from *this(1)*[*unfolded g.im-len-eq-iff*] *this(2)*[*unfolded h.im-len-eq-iff*] ‹*set u = UNIV*›
have $|g\ [c]| = 1$ **and** $|h\ [c]| = 1$ **for** *c* **by** *blast+*
hence $g = h$
using *solution-eq-len-eq*[*OF* ‹$g\ u = h\ u$›, *THEN def-on-sings-eq*, *unfolded* ‹*set u = UNIV*›, *OF - UNIV-I*]
by *force*
thus *False* **using** ‹$g \neq h$› **by** *contradiction*
qed

end

locale *two-code-morphisms* = *g*: *code-morphism g* + *h*: *code-morphism h*
for *g h* :: *'a list* ⇒ *'b list*

begin

sublocale *two-nonerasing-morphisms g h*
by *unfold-locales*

lemmas *code-morphs* = *g.code-morphism-axioms h.code-morphism-axioms*

lemma *revs-two-code-morphisms*: *two-code-morphisms* (*rev-map g*) (*rev-map h*)
by (*simp add*: *g.code-morphism-rev-map h.code-morphism-rev-map two-code-morphisms.intro*)

lemma *min-coin-im-basis*:
$\mathfrak{B}\ (h\text{`}\ (snd\ \text{`}(\mathfrak{C}\ g\ h))) = h\ \text{`}\ snd\ \text{`}\ (\mathfrak{C}_m\ g\ h)$
$\mathfrak{B}\ (g\text{`}\ (fst\ \text{`}(\mathfrak{C}\ g\ h))) = g\ \text{`}\ fst\ \text{`}\ (\mathfrak{C}_m\ g\ h)$
proof−
thm *morphism-on.inj-basis-to-basis*
code-morphism.morph-on-inj-on
min-coin-basis-snd

note *basis-morph-swap* = *morphism-on.inj-basis-to-basis*[*OF code-morphism.morph-on-inj-on*, *symmetric*]
thm *basis-morph-swap*
coin-set-hull
basis-morph-swap[*OF code-morphs(2) code-morphs(2)*, *of snd* ‘ $\mathfrak{C}$ *g h*, *unfolded coin-set-hull*]
show $\mathfrak{B}\ (h\ \text{`}\ snd\ \text{`}\ \mathfrak{C}\ g\ h) = h\ \text{`}\ snd\ \text{`}\ \mathfrak{C}_m\ g\ h$
unfolding *basis-morph-swap*[*OF code-morphs(2) code-morphs(2)*, *of snd* ‘ $\mathfrak{C}$ *g h*, *unfolded coin-set-hull*]
unfolding *min-coin-basis-snd*[*OF code-morphs(1)*]**..**

interpret *swap*: *two-code-morphisms h g*
using *two-code-morphisms-def code-morphs* **by** *blast*

thm *basis-morph-swap*[*OF code-morphs*(*1*) *code-morphs*(*1*), *of fst* ‘ $\mathfrak{C}$ *g h*]
swap.coin-set-hull
coin-set-hull
coin-set-sym
swap.coin-set-hull[*folded coin-set-sym*]
basis-morph-swap[*OF code-morphs*(*1*) *code-morphs*(*1*), *of fst* ‘ $\mathfrak{C}$ *g h*, *unfolded swap.coin-set-hull*[*folded coin-set-sym*]]
min-coin-basis-fst

show $\mathfrak{B}$ (*g* ‘ *fst* ‘ $\mathfrak{C}$ *g h*) = *g* ‘ *fst* ‘ $\mathfrak{C}_m$ *g h*
unfolding *basis-morph-swap*[*OF code-morphs*(*1*) *code-morphs*(*1*), *of fst* ‘ $\mathfrak{C}$ *g h*, *unfolded swap.coin-set-hull*[*folded coin-set-sym*]]
unfolding *min-coin-basis-fst*[*OF code-morphs*(*2*)]
unfolding *min-coin-gen-fst*..
qed

lemma *range-inter-basis-snd*:
shows $\mathfrak{B}$ (*range g* ∩ *range h*) = *h* ‘ (*snd* ‘ $\mathfrak{C}_m$ *g h*)
$\mathfrak{B}$ (*range g* ∩ *range h*) = *g* ‘ (*fst* ‘ $\mathfrak{C}_m$ *g h*)
proof–
show $\mathfrak{B}$ (*range g* ∩ *range h*) = *h* ‘ (*snd* ‘ $\mathfrak{C}_m$ *g h*)
unfolding *coin-set-inter-snd*[*folded image-comp*, *symmetric*]
using *min-coin-im-basis*(*1*).
show $\mathfrak{B}$ (*range g* ∩ *range h*) = *g* ‘ (*fst* ‘ $\mathfrak{C}_m$ *g h*)
unfolding *coin-set-inter-fst*[*folded image-comp*, *symmetric*]
using *min-coin-im-basis*(*2*).
qed

lemma *range-inter-code*:
shows *code* $\mathfrak{B}$ (*range g* ∩ *range h*)
unfolding *range-inter-basis-snd*
thm *morphism-on.inj-code-to-code*
by (*rule morphism-on.inj-code-to-code*)
(*simp-all add*: *h.morph-on-all h.morph-on-inj-on*(*2*) *code-morphs*(*1*) *min-coin-code-snd*)

end

4.3.6 Two marked morphisms

locale *two-marked-morphisms* = *two-nonerasing-morphisms* +
g: *marked-morphism g* + *h*: *marked-morphism h*

begin

sublocale *revs*: *two-code-morphisms g h*
by (*simp add*: *g.code-morphism-axioms h.code-morphism-axioms two-code-morphisms.intro*)

lemmas *ne-g = g.nonerasing* **and**
ne-h = h.nonerasing

lemma *unique-continuation*:
$z \cdot g\ r = z' \cdot h\ s \Longrightarrow\ z \cdot g\ r' = z' \cdot h\ s' \Longrightarrow z \cdot g\ (r \wedge_p r') = z' \cdot h\ (s \wedge_p s')$
using *lcp-ext-left g.marked-morph-lcp h.marked-morph-lcp* **by** *metis*

lemmas *empty-sol = noner-eq-emp-iff*

lemma *comparable-min-sol-eq*: **assumes** $r \leq_p r'$ **and** $g\ r =_m h\ s$ **and** $g\ r' =_m h\ s'$
shows $r = r'$ **and** $s = s'$
proof–
have $s \leq_p s'$
using *g.pref-mono*[*OF* ‹$r \leq_p r'$›]
h.pref-free-morph
unfolding *min-coinD*[*OF* ‹$g\ r =_m h\ s$›] *min-coinD*[*OF* ‹$g\ r' =_m h\ s'$›] **by** *simp*
thus $r = r'$**and** $s = s'$
using ‹$g\ r' =_m h\ s'$›[*unfolded min-coin-def*] *min-coinD*[*OF* ‹$g\ r =_m h\ s$›] *min-coinD'*[*OF* ‹$g\ r =_m h\ s$›] ‹$r \leq_p r'$›
by *force+*
qed

lemma *first-letter-determines*:
assumes $g\ e =_m h\ f$ **and** $g\ e' = h\ f'$ **and** $hd\ e = hd\ e'$ **and** $e' \neq \varepsilon$
shows $e \leq_p e'$ **and** $f \leq_p f'$
proof–
have $g\ (e \wedge_p e') = h\ (f \wedge_p f')$
using *unique-continuation*[*of* $\varepsilon\ e\ \varepsilon\ f\ e'\ f'$, *unfolded emp-simps*, *OF min-coinD*[*OF*‹$g\ e =_m h\ f$›] ‹$g\ e' = h\ f'$›].
have $e \neq \varepsilon$
using ‹$g\ e =_m h\ f$› *min-coinD'* **by** *auto*
hence *eq1*: $e = [hd\ e] \cdot tl\ e$ **and** *eq2*: $e' = [hd\ e'] \cdot tl\ e'$ **using** ‹$e' \neq \varepsilon$› **by** *simp+*
from *lcp-ext-left*[*of* $[hd\ e]\ tl\ e\ tl\ e'$, *folded eq1 eq2*[*folded* ‹$hd\ e = hd\ e'$›]]
have $e \wedge_p e' \neq \varepsilon$ **by** *force*
from *this*
have $f \wedge_p f' \neq \varepsilon$
using ‹$g\ (e \wedge_p e') = h\ (f \wedge_p f')$› *g.nonerasing h.emp-to-emp* **by** *force*
show $e \leq_p e'$ **and** $f \leq_p f'$
using *min-coin-minD*[*OF assms(1) lcp-pref* ‹$e \wedge_p e' \neq \varepsilon$› *lcp-pref* ‹$f \wedge_p f' \neq \varepsilon$› ‹$g\ (e \wedge_p e') = h\ (f \wedge_p f')$›,
unfolded lcp-sym[*of e*] *lcp-sym*[*of f*]] *lcp-pref* **by** *metis+*
qed

corollary *first-letter-determines'*:
assumes $g\ e =_m h\ f$ **and** $g\ e' =_m h\ f'$ **and** $hd\ e = hd\ e'$
shows $e = e'$ **and** $f = f'$
proof–

have $e \neq \varepsilon$ **and** $e' \neq \varepsilon$
using ‹$g\ e =_m h\ f$› ‹$g\ e' =_m h\ f'$› *min-coinD'* **by** *blast+*
have $g\ e' = h\ f'$ **and** $g\ e = h\ f$
using ‹$g\ e =_m h\ f$› ‹$g\ e' =_m h\ f'$› *min-coinD* **by** *blast+*
show $e = e'$ **and** $f = f'$
using *first-letter-determines*[*OF* ‹$g\ e =_m h\ f$› ‹$g\ e' = h\ f'$› ‹$hd\ e = hd\ e'$› ‹$e' \neq \varepsilon$›]
first-letter-determines[*OF* ‹$g\ e' =_m h\ f'$› ‹$g\ e = h\ f$› ‹$hd\ e = hd\ e'$›[*symmetric*] ‹$e \neq \varepsilon$›]
by *force+*
qed

lemma *first-letter-determines-sol*: **assumes** $r \in g =_M h$ **and** $s \in g =_M h$ **and** $hd\ r = hd\ s$
shows $r = s$
proof–
have $r \wedge_p s \neq \varepsilon$
using *nemp-lcp-distinct-hd*[*OF min-solD'*[*OF* ‹$r \in g =_M h$›] *min-solD'*[*OF* ‹$s \in g =_M h$›]] ‹$hd\ r = hd\ s$›
by *blast*
have $g\ r = h\ r$ **and** $g\ s = h\ s$
using *min-solD*[*OF* ‹$r \in g =_M h$›] *min-solD*[*OF* ‹$s \in g =_M h$›].
have $g\ (r \wedge_p s) = h\ (r \wedge_p s)$
unfolding ‹$g\ r = h\ r$› ‹$g\ s = h\ s$› *g.marked-morph-lcp h.marked-morph-lcp*..
from *min-solD-min*[*OF* ‹$r \in g =_M h$› ‹$r \wedge_p s \neq \varepsilon$› *lcp-pref this*] *min-solD-min*[*OF* ‹$s \in g =_M h$› ‹$r \wedge_p s \neq \varepsilon$› *lcp-pref' this*]
show $r = s$ **by** *force*
qed

definition *pre-block* :: $'a \Rightarrow 'a\ list \times 'a\ list$
where *pre-block a* = (*THE p*. (*g* (*fst p*)) =$_m$ *h* (*snd p*)) ∧ *hd* (*fst p*) = *a*)
— *pre-block a* may not be a block, if no such exists

definition *blockP* :: $'a \Rightarrow bool$
where *blockP a* ≡ *g* (*fst* (*pre-block a*)) =$_m$ *h* (*snd* (*pre-block a*)) ∧ *hd* (*fst* (*pre-block a*)) = *a*
— Predicate: the *pre-block* of the letter *a* is indeed a block

lemma *pre-blockI*: *g u* =$_m$ *h v* ⟹ *pre-block* (*hd u*) = (*u*,*v*)
unfolding *pre-block-def*
proof (*rule the-equality*)
show ⋀*p*. *g u* =$_m$ *h v* ⟹ *g* (*fst p*) =$_m$ *h* (*snd p*) ∧ *hd* (*fst p*) = *hd u* ⟹ *p* = (*u*, *v*)
using *first-letter-determines'* **by** *force*
qed *simp*

lemma *blockI*: **assumes** $g\ u =_m h\ v$ **and** $hd\ u = a$
shows *blockP a*
proof–

from *pre-blockI*[*OF* ‹$g\ u =_m h\ v$›, *unfolded* ‹$hd\ u = a$›]
show *blockP a*
unfolding *blockP-def* **using** *assms* **by** *simp*
qed

lemma *hd-im-comm-eq-aux*:
assumes $g\ w = h\ w$ **and** $w \neq \varepsilon$ **and** *comm*: $g^{\mathcal{C}}\ (hd\ w) \cdot h^{\mathcal{C}}(hd\ w) = h^{\mathcal{C}}\ (hd\ w) \cdot g^{\mathcal{C}}(hd\ w)$ **and** *len*: $|g^{\mathcal{C}}\ (hd\ w)| \leq |h^{\mathcal{C}}(hd\ w)|$
shows $g^{\mathcal{C}}\ (hd\ w) = h^{\mathcal{C}}\ (hd\ w)$
proof(*cases set* $w \subseteq \{hd\ w\}$)
assume *set* $w \subseteq \{hd\ w\}$
then obtain l **where** $w = [hd\ w]^{@}l$
by *blast*
from *nemp-exp-pos*[*OF* ‹$w \neq \varepsilon$›, *of* l $[hd\ w]$, *folded this*]
have $0 < l$
by *fast*
from ‹$g\ w = h\ w$›
have $(g\ [hd\ w])^{@}l = (h\ [hd\ w])^{@}l$
unfolding *g.pow-morph*[*symmetric*] *h.pow-morph*[*symmetric*] ‹$w = [hd\ w]^{@}l$›[*symmetric*]**.**
with ‹$0 < l$›
have $g\ [hd\ w] = h\ [hd\ w]$
using *pow-eq-eq* **by** *blast*
thus $g^{\mathcal{C}}\ (hd\ w) = h^{\mathcal{C}}\ (hd\ w)$
unfolding *core-def***.**
next
assume $\neg$ *set* $w \subseteq \{hd\ w\}$
from *distinct-letter-in-hd*[*OF this*]
obtain $b\ l\ w'$ **where** $[hd\ w]^{@}l \cdot [b] \cdot w' = w$ **and** $b \neq hd\ w$ **and** $l \neq 0$**.**
from *commE*[*OF comm*]
obtain $t\ m\ k$ **where** $g^{\mathcal{C}}\ (hd\ w) = t^{@}m$ **and** $h^{\mathcal{C}}\ (hd\ w) = t^{@}k$**.**
have $|t| \neq 0$ **and** $t \neq \varepsilon$ **and** $m \neq 0$
using ‹$g^{\mathcal{C}}\ (hd\ w) = t^{@}m$› *g.core-nemp pow-zero*[*of t*] **by** (*fastforce*, *fastforce*, *metis*)
with *lenarg*[*OF* ‹$g^{\mathcal{C}}\ (hd\ w) = t^{@}m$›] *lenarg*[*OF* ‹$h^{\mathcal{C}}\ (hd\ w) = t^{@}k$›]
have $m \leq k$
unfolding *pow-len lenmorph* **using** *len* **by** *auto*
have $m = k$
proof(*rule ccontr*)
assume $m \neq k$ **hence** $m < k$ **using** ‹$m \leq k$› **by** *simp*
have $0 < k*l-m*l$
using ‹$m < k$› ‹$l \neq 0$› **by** *force*
have $g\ w = t^{@}(m*l) \cdot g\ [b] \cdot g\ w'$
unfolding *arg-cong*[*OF* ‹$[hd\ w]^{@}l \cdot [b] \cdot w' = w$›, *of g*, *symmetric*] *g.morph g.pow-morph* ‹$g^{\mathcal{C}}\ (hd\ w) = t^{@}m$›[*unfolded core-def*] *pow-mult*[*symmetric*]**..**
moreover have $h\ w = t^{@}(k*l) \cdot h\ [b] \cdot h\ w'$
unfolding *arg-cong*[*OF* ‹$[hd\ w]^{@}l \cdot [b] \cdot w' = w$›, *of h*, *symmetric*] *h.morph h.pow-morph* ‹$h^{\mathcal{C}}\ (hd\ w) = t^{@}k$›[*unfolded core-def*] *pow-mult*[*symmetric*]**..**
ultimately have $*$: $g\ [b] \cdot\ g\ w' = t^{@}(k*l-m*l) \cdot h\ [b] \cdot h\ w'$
using ‹$g\ w = h\ w$› *pop-pow-cancel*[*OF - mult-le-mono1*[*OF* ‹$m \leq k$›]]

unfolding *g.morph*[*symmetric*] *h.morph*[*symmetric*] **by** *metis*
have $t^{@}(k*l-m*l) \neq \varepsilon$
using *gr-implies-not0*[*OF* ‹$0 < k * l - m * l$›] **unfolding** *nemp-emp-pow*[*OF* ‹$t \neq \varepsilon$›].
have *hd* $(t^{@}(k*l-m*l)) =$ *hd t*
using *hd-append2*[*OF* ‹$t \neq \varepsilon$›] **unfolding** *pow-pos*[*OF* ‹$0 < k * l - m * l$›].
have *hd t* = *hd* (*g* [*b*])
using *hd-append2*[*OF g.sing-to-nemp*[*of b*], *of g w'*]
unfolding *hd-append2*[*of - h* [*b*] · *h w'*, *OF* ‹$t^{@}(k*l-m*l) \neq \varepsilon$›, *folded* ∗] ‹*hd* $(t^{@}(k*l-m*l))$ = *hd t*›.
have *hd t* = *hd* (*g* [*hd w*])
using *g.hd-im-hd-hd*[*OF* ‹$w \neq \varepsilon$›, *unfolded* ‹$g^{\mathcal{C}}$ (*hd w*) = t $^{@}$ m›[*unfolded core-def*]]
hd-append2[*OF* ‹$t \neq \varepsilon$›, *of* $t^{@}(m-1)$, *unfolded pow-Suc*, *folded pow-Suc*[*of m−1 t*, *unfolded Suc-minus*[*OF* ‹$m \neq 0$›]]]
g.hd-im-hd-hd[*OF* ‹$w \neq \varepsilon$›] **by** *force*
thus *False*
unfolding ‹*hd t* = *hd* (*g* [*b*])› **using** ‹$b \neq$ *hd w*› *g.marked-morph* **by** *blast*
qed
show $g^{\mathcal{C}}$ (*hd w*) = $h^{\mathcal{C}}$ (*hd w*)
unfolding ‹$g^{\mathcal{C}}$ (*hd w*) = $t^{@}m$› ‹$h^{\mathcal{C}}$ (*hd w*) = $t^{@}k$› ‹$m = k$›..
qed

lemma *hd-im-comm-eq*:
assumes *g w* = *h w* **and** $w \neq \varepsilon$ **and** *comm*: $g^{\mathcal{C}}(hd\ w) \cdot h^{\mathcal{C}}(hd\ w) = h^{\mathcal{C}}(hd\ w) \cdot g^{\mathcal{C}}(hd\ w)$
shows $g^{\mathcal{C}}$ (*hd w*) = $h^{\mathcal{C}}$ (*hd w*)
proof−
interpret *swap*: *two-marked-morphisms h g* **by** *unfold-locales*
show $g^{\mathcal{C}}$ (*hd w*) = $h^{\mathcal{C}}$ (*hd w*)
using *hd-im-comm-eq-aux*[*OF assms*] *swap.hd-im-comm-eq-aux*[*OF assms*(*1*)[*symmetric*] *assms*(*2*) *assms*(*3*)[*symmetric*], *symmetric*]
by *force*
qed

lemma *block-ex*: **assumes** *g u* $=_m$ *h v* **shows** *blockP* (*hd u*)
unfolding *blockP-def* **using** *pre-blockI*[*OF assms*] *assms* **by** *simp*

lemma *sol-block-ex*: **assumes** *g u* = *h v* **and** $u \neq \varepsilon$ **shows** *blockP* (*hd u*)
using *min-coin-prefE*[*OF assms*] *block-ex* **by** *metis*

— Successor morphisms

definition *suc-fst* **where** *suc-fst* ≡ λ *x*. *concat*(*map* (λ *y*. *fst* (*pre-block y*)) *x*)
definition *suc-snd* **where** *suc-snd* ≡ λ *x*. *concat*(*map* (λ *y*. *snd* (*pre-block y*)) *x*)

lemma *blockP-D*: *blockP a* ⟹ *g* (*suc-fst* [*a*]) $=_m$ *h* (*suc-snd* [*a*])
unfolding *blockP-def suc-fst-def suc-snd-def* **by** *simp*

lemma *blockP-D-hd*: *blockP a* $\Longrightarrow$ *hd* (*suc-fst* [*a*]) = *a*
unfolding *blockP-def suc-fst-def* **by** *simp*

abbreviation *blocks* $\tau \equiv (\forall\ a.\ a \in set\ \tau \longrightarrow blockP\ a)$

sublocale *sucs*: *two-morphisms suc-fst suc-snd*
by (*standard*) (*simp-all add*: *suc-fst-def suc-snd-def*)

lemma *blockP-D-hd-hd*: **assumes** *blockP a*
shows *hd* ($h^{\mathcal{C}}$ (*hd* (*suc-snd* [*a*]))) = *hd* ($g^{\mathcal{C}}$ *a*)
proof−
from *hd-tlE*[*OF conjunct2*[*OF min-coinD'*[*OF blockP-D*[*OF* ‹*blockP a*›]]]]
obtain *b* **where** *hd* (*suc-snd* [*a*]) = *b* **by** *blast*
have *suc-fst* [*a*] ≠ *ε* **and** *suc-snd* [*a*] ≠ *ε*
using *min-coinD'*[*OF blockP-D*[*OF* ‹*blockP a*›]] **by** *blast*+
from *g.hd-im-hd-hd*[*OF this*(*1*)] *h.hd-im-hd-hd*[*OF this*(*2*)]
have *hd* ($h^{\mathcal{C}}$ (*hd* (*suc-snd* [*a*]))) = *hd* ($g^{\mathcal{C}}$ (*hd* (*suc-fst* [*a*])))
unfolding *core-def min-coinD*[*OF blockP-D*[*OF* ‹*blockP a*›]] **by** *argo*
thus *?thesis*
unfolding *blockP-D-hd*[*OF assms*].
qed

lemma *suc-morph-sings*: **assumes** *g e* =$_m$ *h f*
shows *suc-fst* [*hd e*] = *e* **and** *suc-snd* [*hd e*] = *f*
unfolding *suc-fst-def suc-snd-def* **using** *pre-blockI*[*OF assms*] **by** *simp-all*

lemma *blocks-eq*: *blocks* $\tau \Longrightarrow$ *g* (*suc-fst* τ) = *h* (*suc-snd* τ)
proof (*induct* τ)
case (*Cons a* τ)
have *blocks* τ **and** *blockP a*
using ‹*blocks* (*a* # τ)› **by** *simp-all*
from *Cons.hyps*[*OF this*(*1*)]
show *?case*
unfolding *sucs.g.pop-hd*[*of a* τ] *sucs.h.pop-hd*[*of a* τ] *g.morph h.morph*
using *min-coinD*[*OF blockP-D*, *OF* ‹*blockP a*›] **by** *simp*
qed *simp*

lemma *suc-eq'*: **assumes** $\bigwedge$ *a. blockP a* **shows** *g*(*suc-fst w*) = *h*(*suc-snd w*)
by (*simp add*: *assms blocks-eq*)

lemma *suc-eq*: **assumes** *all-blocks*: $\bigwedge$ *a. blockP a* **shows** *g* ∘ *suc-fst* = *h* ∘ *suc-snd*
using *suc-eq'*[*OF assms*] **by** *fastforce*

lemma *block-eq*: *blockP a* $\Longrightarrow$ *g* (*suc-fst* [*a*]) = *h* (*suc-snd* [*a*])
using *blockP-D min-coinD* **by** *blast*

lemma *blocks-inj-suc*: **assumes** *blocks* τ **shows** *inj-on suc-fst*$^{\mathcal{C}}$ (*set* τ)

unfolding *inj-on-def core-def* **using** *blockP-D-hd*[*OF* ‹*blocks τ*›[*rule-format*]]
by *metis*

lemma *blocks-inj-suc′*: **assumes** *blocks τ* **shows** *inj-on suc-snd*$^{\mathcal{C}}$ (*set τ*)
using *g.marked-core blockP-D-hd-hd*[*OF* ‹*blocks τ*›[*rule-format*]]
unfolding *inj-on-def core-def* **by** *metis*

lemma *blocks-marked-code*: **assumes** *blocks τ*
shows *marked-code* (*suc-fst*$^{\mathcal{C}}$ ‘(*set τ*))
proof
show *ε* ∉ *suc-fst*$^{\mathcal{C}}$ ‘ *set τ*
unfolding *core-def image-iff* **using** *min-coinD′*[*OF blockP-D*[*OF* ‹*blocks τ*›[*rule-format*]]]
by *fastforce*
show ⋀*u v*. *u* ∈ *suc-fst*$^{\mathcal{C}}$ ‘ *set τ* ⟹
v ∈ *suc-fst*$^{\mathcal{C}}$ ‘ *set τ* ⟹ *hd u* = *hd v* ⟹ *u* = *v*
using *blockP-D-hd*[*OF* ‹*blocks τ*›[*rule-format*]] **unfolding** *core-def* **by** *fastforce*
qed

lemma *blocks-marked-code′*: **assumes** *all-blocks*: ⋀ *a*. *a* ∈ *set τ* ⟹ *blockP a*
shows *marked-code* (*suc-snd*$^{\mathcal{C}}$ ‘(*set τ*))
proof
show *ε* ∉ *suc-snd*$^{\mathcal{C}}$ ‘ *set τ*
unfolding *core-def image-iff* **using** *min-coinD′*[*OF all-blocks*[*THEN blockP-D*]]
by *fastforce*
show *u* = *v* **if** *u* ∈ *suc-snd*$^{\mathcal{C}}$ ‘ *set τ* **and** *v* ∈ *suc-snd*$^{\mathcal{C}}$ ‘ *set τ* **and** *hd u* = *hd v*
for *u v*
proof−
obtain *a b* **where** *u* = *suc-snd* [*a*] **and** *v* = *suc-snd* [*b*] **and** *a* ∈ *set τ* **and** *b* ∈ *set τ*
using ‹*v* ∈ *suc-snd*$^{\mathcal{C}}$ ‘ *set τ*› ‹*u* ∈ *suc-snd*$^{\mathcal{C}}$ ‘ *set τ*› **unfolding** *core-def* **by** *fast*+
from *g.marked-core*[*of a b*,
folded blockP-D-hd-hd[*OF all-blocks*, *OF* ‹*a* ∈ *set τ*›] *blockP-D-hd-hd*[*OF all-blocks*, *OF* ‹*b* ∈ *set τ*›]
this(*1*−*2*) ‹*hd u* = *hd v*›,*OF refl*]
show *u* = *v*
unfolding ‹*u* = *suc-snd* [*a*]› ‹*v* = *suc-snd* [*b*]› **by** *blast*
qed
qed

lemma *sucs-marked-morphs*: **assumes** *all-blocks*: ⋀ *a*. *blockP a*
shows *two-marked-morphisms suc-fst suc-snd*
proof
show *hd* (*suc-fst*$^{\mathcal{C}}$ *a*) = *hd* (*suc-fst*$^{\mathcal{C}}$ *b*) ⟹ *a* = *b* **for** *a b*
using *blockP-D-hd*[*OF all-blocks*] **unfolding** *core-def* **by** *force*
show *hd* (*suc-snd*$^{\mathcal{C}}$ *a*) = *hd* (*suc-snd*$^{\mathcal{C}}$ *b*) ⟹ *a* = *b* **for** *a b*
using *blockP-D-hd-hd*[*OF all-blocks*, *folded core-sing*] *g.marked-core* **by** *metis*
show *suc-fst w* = *ε* ⟹ *w* = *ε* **for** *w*
using *blockP-D*[*OF assms*, *THEN min-coinD′*] *sucs.g.noner-sings-conv* **by** *blast*

show *suc-snd* $w = \varepsilon \Longrightarrow w = \varepsilon$ **for** w
using *blockP-D*[*OF assms*(*1*), *THEN min-coinD'*] *sucs.h.noner-sings-conv* **by** *blast*
qed

lemma *pre-blocks-range*: $\{(e,f).\ g\ e =_m h\ f\ \} \subseteq$ *range pre-block*
using *pre-blockI case-prodE* **by** *blast*

corollary *card-blocks*: **assumes** *finite* (*UNIV* :: *'a set*) **shows** *card* $\{(e,f).\ g\ e =_m h\ f\ \} \leq$ *card* (*UNIV* :: *'a set*)
using *le-trans*[*OF card-mono*[*OF finite-imageI pre-blocks-range*, *OF assms*] *card-image-le*[*of - pre-block*, *OF assms*]]**.**

lemma *block-decomposition*: **assumes** $g\ e = h\ f$
obtains τ **where** *suc-fst* $\tau = e$ **and** *suc-snd* $\tau = f$ **and** *blocks* τ
using *assms*
proof (*induct* $|e|$ *arbitrary*: *e f thesis rule*: *less-induct*)
case *less*
show *?case*
proof (*cases* $e = \varepsilon$)
assume $e = \varepsilon$
hence $f = \varepsilon$
using *less.hyps empty-sol*[*OF* ‹$g\ e = h\ f$›] **by** *blast*
hence *suc-fst* $\varepsilon = e$ **and** *suc-snd* $\varepsilon = f$
unfolding *suc-fst-def suc-snd-def* **by** (*simp add*: ‹$e = \varepsilon$›)+
from *less.prems*(*1*)[*OF this*]
show *thesis*
by *simp*
next
assume $e \neq \varepsilon$
from *min-coin-prefE*[*OF* ‹$g\ e = h\ f$› *this*]
obtain $e_1\ e_2\ f_1\ f_2$
where $g\ e_1 =_m h\ f_1$ **and** $e_1 \cdot e_2 = e$ **and** $f_1 \cdot f_2 = f$ **and** $e_1 \neq \varepsilon$ **and** $f_1 \neq \varepsilon$
by (*metis min-coinD' prefD*)
from ‹$g\ e = h\ f$›[*folded* ‹$e_1 \cdot e_2 = e$› ‹$f_1 \cdot f_2 = f$›, *unfolded g.morph h.morph*]
have $g\ e_2 = h\ f_2$
using *min-coinD*[*OF* ‹$g\ e_1 =_m h\ f_1$›] **by** *simp*
have $|e_2| < |e|$
using ‹$e_1 \cdot e_2 = e$› ‹$e_1 \neq \varepsilon$› **by** *auto*
from *less.hyps*[*OF this* - ‹$g\ e_2 = h\ f_2$›]
obtain τ' **where** *suc-fst* $\tau' = e_2$ **and** *suc-snd* $\tau' = f_2$ **and** *blocks* τ'**.**
have *suc-fst* [*hd e*] = e_1 **and** *suc-snd* [*hd e*] = f_1
using *suc-morph-sings* ‹$e_1 \cdot e_2 = e$› ‹$g\ e_1 =_m h\ f_1$› ‹$e_1 \neq \varepsilon$› **by** *auto*
hence *suc-fst* (*hd e* # τ') = e **and** *suc-snd* (*hd e* # τ') = f
using ‹$e_1 \cdot e_2 = e$› ‹$f_1 \cdot f_2 = f$›
unfolding *hd-word*[*of hd e* τ'] *sucs.g.morph sucs.h.morph* ‹*suc-fst* $\tau' = e_2$› ‹*suc-snd* $\tau' = f_2$› ‹*suc-fst* [*hd e*] = e_1› ‹*suc-snd* [*hd e*] = f_1› **by** *force*+
have *blocks* (*hd e* # τ')
using ‹*blocks* τ'› ‹$e_1 \cdot e_2 = e$› ‹$e_1 \neq \varepsilon$› ‹$g\ e_1 =_m h\ f_1$› *block-ex* **by** *auto*

from *less.prems*(*1*)[*OF* - - *this*]
show *thesis*
by (*simp add*: ‹*suc-fst* (*hd e* # τ') = *e*› ‹*suc-snd* (*hd e* # τ') = *f*›)
qed
qed

lemma *block-decomposition-unique*: **assumes** *g e* = *h f* **and**
suc-fst τ = *e* **and** *suc-fst* τ' = *e* **and** *blocks* τ **and** *blocks* τ' **shows** $\tau = \tau'$
proof–
let *?C* = *suc-fst*$^{\mathcal{C}}$ ‘(*set* (τ · τ'))
have *blocks* (τ · τ')
using ‹*blocks* τ› ‹*blocks* τ'› **by** *auto*
interpret *marked-code ?C*
by (*rule blocks-marked-code*) (*simp add*: ‹*blocks* (τ · τ')›)
have *inj-on suc-fst*$^{\mathcal{C}}$ (*set* (τ · τ'))
using ‹*blocks* (τ · τ')› *blocks-inj-suc* **by** *blast*
from *sucs.g.code-set-morph*[*OF code-axioms this* ‹*suc-fst* τ = *e*›[*folded* ‹*suc-fst* τ' = *e*›]]
show $\tau = \tau'$.
qed

lemma *block-decomposition-unique'*: **assumes** *g e* = *h f* **and**
suc-snd τ = *f* **and** *suc-snd* τ' = *f* **and** *blocks* τ **and** *blocks* τ'
shows $\tau = \tau'$
proof–
have *suc-fst* τ = *e* **and** *suc-fst* τ' = *e*
using *assms blocks-eq g.code-morph-code* **by** *presburger*+
from *block-decomposition-unique*[*OF assms*(*1*) *this assms*(*4*–*5*)]
show $\tau = \tau'$.
qed

lemma *comm-sings-block*: **assumes** *g*[*a*] · *h*[*b*] = *h*[*b*] · *g*[*a*]
obtains *m n* **where** *suc-fst* [*a*] = [*a*]@*Suc m* **and** *suc-snd* [*a*] = [*b*]@*Suc n*
proof–
have [*a*] @ |*h* [*b*]| ≠ ε
using *nemp-len*[*OF h.sing-to-nemp*, *of b*] **by** *simp*
obtain *e f* **where** *g e* =$_m$ *h f* **and** *e* ≤*p* [*a*] @ |*h* [*b*]| **and** *f* ≤*p* [*b*] @ |*g* [*a*]|
using *min-coin-prefE*[*OF comm-common-pow-list-iff*[*OF assms*,*folded g.pow-morph h.pow-morph*]
‹[*a*] @ |*h* [*b*]| ≠ ε›, *of thesis*] **by** *blast*
note *e* = *pref-sing-pow*[*OF* ‹*e* ≤*p* [*a*] @ |*h* [*b*]|›]
note *f* = *pref-sing-pow*[*OF* ‹*f* ≤*p* [*b*] @ |*g* [*a*]|›]
from *min-coinD'*[*OF* ‹*g e* =$_m$ *h f*›]
have *exps*: |*e*| = *Suc* (|*e*| − *1*) |*f*| = *Suc* (|*f*| − *1*)
by *auto*
have *hd e* = *a*
using ‹*e* = [*a*] @ |*e*|›[*unfolded pow-Suc*[*of* |*e*| − *1* [*a*], *folded* ‹|*e*| = *Suc* (|*e*| − *1*)›], *folded hd-word*[*of a* [*a*] @ (|*e*| − *1*)]]
list.sel(*1*)[*of a* [*a*] @ (|*e*| − *1*)] **by** *argo*

from *that suc-morph-sings*[*OF* ‹*g e* =$_m$ *h f*›, *unfolded this*] *e f exps*
show *thesis*
by *metis*
qed

— a variant of successor morphisms: target alphabet encoded to be the same as for original morphisms

definition *sucs-encoding* **where** *sucs-encoding* = (λ *a*. *hd* (*g* [*a*]))
definition *sucs-decoding* **where** *sucs-decoding* = (λ *a*. *SOME c*. *hd* (*g*[*c*]) = *a*)

lemma *sucs-encoding-inv*: *sucs-decoding* ∘ *sucs-encoding* = *id*
by (*rule ext*)
(*unfold sucs-encoding-def sucs-decoding-def comp-apply id-apply*, *use g.marked-core*[*unfolded core-def*] **in** *blast*)

lemma *encoding-inj*: *inj sucs-encoding*
unfolding *sucs-encoding-def inj-on-def* **using** *g.marked-core*[*unfolded core-def*]
by *blast*

lemma *map-encoding-inj*: *inj* (*map sucs-encoding*)
using *encoding-inj* **by** *simp*

definition *suc-fst′* **where** *suc-fst′* = (*map sucs-encoding*) ∘ *suc-fst*
definition *suc-snd′* **where** *suc-snd′* = (*map sucs-encoding*) ∘ *suc-snd*

lemma *encoded-sucs-eq-conv*: *suc-fst w* = *suc-snd w′* ⟷ *suc-fst′ w* = *suc-snd′ w′*
unfolding *suc-fst′-def suc-snd′-def* **using** *encoding-inj* **by** *force*

lemma *encoded-sucs-eq-conv′*: *suc-fst* = *suc-snd* ⟷ *suc-fst′* = *suc-snd′*
unfolding *suc-fst′-def suc-snd′-def* **using** *inj-comp-eq*[*OF map-encoding-inj*] **by** *blast*

lemma *encoded-sucs*: **assumes** ⋀ *c*. *blockP c* **shows** *two-marked-morphisms suc-fst′ suc-snd′*
unfolding *suc-fst′-def suc-snd′-def*
proof−
from *sucs-marked-morphs*[*OF assms*]
interpret *sucs*: *two-marked-morphisms suc-fst suc-snd*
by *force*
interpret *nonerasing-morphism* (*map sucs-encoding*) ∘ *suc-fst*
unfolding *comp-apply*
by *unfold-locales*
(*use sucs.g.morph sucs.g.nemp-to-nemp* **in** *auto*)
interpret *nonerasing-morphism* (*map sucs-encoding*) ∘ *suc-snd*
by *unfold-locales*
(*use sucs.h.morph sucs.h.nemp-to-nemp* **in** *auto*)

interpret *marked-morphism* (*map sucs-encoding*) ∘ *suc-fst*
proof
show *hd* ((*map sucs-encoding* ∘ *suc-fst*)$^{\mathcal{C}}$ *a*) = *hd* ((*map sucs-encoding* ∘ *suc-fst*)$^{\mathcal{C}}$ *b*) ⟹ *a* = *b* **for** *a b*
unfolding *comp-apply core-def sucs-encoding-def hd-map*[*OF sucs.g.sing-to-nemp*]
unfolding *blockP-D-hd*[*OF assms*] **using** *g.marked-morph*.
qed
interpret *marked-morphism* (*map sucs-encoding*) ∘ *suc-snd*
proof
show *hd* ((*map sucs-encoding* ∘ *suc-snd*)$^{\mathcal{C}}$ *a*) = *hd* ((*map sucs-encoding* ∘ *suc-snd*)$^{\mathcal{C}}$ *b*) ⟹ *a* = *b* **for** *a b*
unfolding *comp-apply core-def sucs-encoding-def hd-map*[*OF sucs.h.sing-to-nemp*]
using *g.marked-morph*[*THEN sucs.h.marked-morph*].
qed
show *two-marked-morphisms* ((*map sucs-encoding*) ∘ *suc-fst*) ((*map sucs-encoding*) ∘ *suc-snd*)..
qed

lemma *encoded-sucs-len*: |*suc-fst w*| = |*suc-fst′ w*| **and** |*suc-snd w*| = |*suc-snd′ w*|
unfolding *suc-fst′-def suc-snd′-def sucs-encoding-def comp-apply* **by** *force*+

end

end

theory *Periodicity-Lemma*
imports *CoWBasic*
begin

Chapter 5

The periodicity Lemma

The periodicity Lemma says that if a sufficiently long word has two periods p and q, then the period can be refined to *gcd p q*. The consequence is equivalent to the fact that the corresponding periodic roots commute. "Sufficiently long" here means at least $p + q - gcd\ p\ q$. It is also known as the Fine and Wilf theorem due to its authors [3].

If we relax the requirement to $p + q$, then the claim becomes easy, and it is proved in theory *Combinatorics-Words.CoWBasic* as *two-pers*: $[\![w \leq p\ u \cdot w;\ w \leq p\ v \cdot w;\ |u| + |v| \leq |w|]\!] \Longrightarrow u \cdot v = v \cdot u$.

theorem *per-lemma-relaxed*:
assumes *period w p* **and** *period w q* **and** $p + q \leq |w|$
shows $(take\ p\ w) \cdot (take\ q\ w) = (take\ q\ w) \cdot (take\ p\ w)$
using *two-pers*[*OF*
‹*period w p*›[*unfolded period-def*[*of w p*]]
‹*period w q*›[*unfolded period-def*[*of w q*]], *unfolded*
take-len[*OF add-leD1*[*OF* ‹$p + q \leq |w|$›]]
take-len[*OF add-leD2*[*OF* ‹$p + q \leq |w|$›]], *OF* ‹$p + q \leq |w|$›].

5.1 Main claim

We first formulate the claim of the periodicity lemma in terms of commutation of two periodic roots. For trivial reasons we can also drop the requirement that the roots are nonempty.

The proof is by induction which mimics the Euclidean algorithm. The step is given by the following lemma:

lemma *per-lemma-step*:
fixes *u v w* :: *'a list*
defines $w' \equiv u \cdot w$ **and** $v' \equiv u \cdot v$
shows $w' \leq p\ u \cdot w' \wedge w' \leq p\ v' \cdot w' \longleftrightarrow\ w \leq p\ u \cdot w \wedge w \leq p\ v \cdot w$
proof
assume *a1*: $w \leq p\ u \cdot w \wedge w \leq p\ v \cdot w$

```
  show w' ≤p u · w' ∧ w' ≤p v' · w'
    unfolding w'-def v'-def
    by (rule conjI, unfold pref-cancel-conv rassoc) (use a1 pref-prolong in blast)+
next
  assume a2: w' ≤p u · w' ∧ w' ≤p v' · w'
  hence w ≤p u · w w ≤p v · w'
    unfolding w'-def v'-def
    by force+
  then show w ≤p u · w ∧ w ≤p v · w
    unfolding w'-def
    using pref-prod-pref by blast
qed

theorem per-lemma-comm:
  assumes w ≤p r · w and w ≤p s · w
    and len: |r| + |s| − (gcd |r| |s|) ≤ |w|
  shows r · s = s · r
  using assms
proof (induction |s| + |s| + |r| arbitrary: w r s rule: less-induct)
  case less
  consider (empty) s = ε | (short)  |r| < |s| | (step) s ≠ ε ∧ |s| ≤ |r| by force
  then show ?case
  proof (cases)
    case (empty)
    thus r · s = s · r
      by fastforce
  next
    case (short)
    show r · s = s · r
    proof (rule less.hyps[symmetric])
      show |r| + |r| + |s| < |s| + |s| + |r|
        using short by simp
      show |s| + |r| − (gcd |s| |r|) ≤ |w|
        unfolding gcd.commute[of |s|] add.commute[of |s|] by fact
    qed fact+
   next
    case (step)
    hence  s ≠ ε and |s| ≤ |r|
      by blast+

   from le-add-diff[OF gcd-le2-nat[OF ‹s ≠ ε›[folded length-0-conv], of |r|], unfolded
gcd.commute[of |r|], of |r|]
    have |r| ≤ |w|
    using ‹|r| + |s| − (gcd |r| |s|) ≤ |w|›[unfolded gcd.commute[of |r|] add.commute[of
|r|]] order.trans by blast
    hence |s| ≤ |w|
      using ‹|s| ≤ |r|› order.trans by blast
    from pref-prod-long[OF  ‹w ≤p s · w› this]
    have s ≤p r
```

using *prefix-order.trans*[*OF* - ‹$w \le_p r \cdot w$›] *pref-prod-le* ‹$|s| \le |r|$›
by *blast*
hence r: $s \cdot s^{-1>} r = r$ **and** w': $s \cdot s^{-1>} w = w$
using ‹$s \le_p w$› **by** *simp-all*

— induction step
from *per-lemma-step*[*of* s $s^{-1>} w$ $s^{-1>} r$, *unfolded* w' r]
have *new-pers*: $s^{-1>} w \le_p s^{-1>} r \cdot s^{-1>} w$ $s^{-1>} w \le_p s \cdot s^{-1>} w$
using ‹$w \le_p s \cdot w$› ‹$w \le_p r \cdot w$› **unfolding** w' **by** *blast+*

have *ind-len*: $|s^{-1>} r| + |s| - (gcd\ |s^{-1>} r|\ |s|) \le |s^{-1>} w|$
using ‹$|r| + |s| - (gcd\ |r|\ |s|) \le |w|$›[*folded lenarg*[*OF* ‹$s \cdot s^{-1>} w = w$›]]
unfolding *pref-gcd-lq*[*OF* ‹$s \le_p r$›, *unfolded gcd.commute*[*of* $|s|$]] *lenmorph lq-short-len*[*OF* ‹$s \le_p r$›, *unfolded add.commute*[*of* $|s|$]] **by** *force*

have $s \cdot s^{-1>} r = s^{-1>} r \cdot s$
using *less.hyps*[*OF* - *new-pers ind-len*] ‹$s \le_p r$›
unfolding *prefix-def*
by *force*

thus $r \cdot s = s \cdot r$
using ‹$s \le_p r$› **by** (*fastforce simp add*: *prefix-def*)
qed
qed

We can now prove the numeric version.

theorem *per-lemma*: **assumes** *period* w p **and** *period* w q **and** *len*: $p + q - gcd\ p\ q \le |w|$
shows *period* w $(gcd\ p\ q)$
proof (*cases* $p = 0 \vee q = 0 \vee w = \varepsilon$, *use assms*(*1–2*) **in** *force*)
assume *pos*: $\neg\ (p = 0 \vee q = 0 \vee w = \varepsilon)$
have *takep*: $w \le_p (take\ p\ w) \cdot w$ **and** *takeq*: $w \le_p (take\ q\ w) \cdot w$
using ‹*period* w p› ‹*period* w q› *periodD* **by** *blast+*
have $p \le |w|$
using *per-lemma-len-le*[*OF len*] *pos* **by** *blast*
from *take-len*[*OF this*]
have *lenp*: $|take\ p\ w| = p$.
have *lenq*: $|take\ q\ w| = q$
using *gcd-le1-pos*[*of* p q] *pos len take-len*[*of* q w] **by** *fastforce*
obtain t k m **where** *take* p $w = t^{@} k$ **and** *take* q $w = t^{@} m$ **and** $t \neq \varepsilon$
using *commE*[*OF per-lemma-comm*[*OF takep takeq*, *unfolded lenp lenq*, *OF len*]].
have $w <_p t \cdot w$
using ‹$t \neq \varepsilon$› ‹*take* q $w = t\ ^{@}\ m$› *conjug-pow per-rootI pos root-comm-root take-eq-Nil takeq* **by** *metis*
have *period* w $|t|$
using ‹$w <_p t \cdot w$› *periodI-pref'* *strict-prefix-def* **by** *metis*
have $|t|$ *dvd* $(gcd\ p\ q)$
using *lenp*[*unfolded* ‹*take* p $w = t^{@} k$›] *lenq*[*unfolded* ‹*take* q $w = t^{@} m$›]

nemp-len[*OF* ‹$t \neq \varepsilon$›]
unfolding *lenmorph pow-len* **by** *force*
from *dvd-div-mult-self*[*OF this*]
have *gcd p q div* $|t| * |t| = gcd\ p\ q$.
have $gcd\ p\ q \neq 0$
using *pos* **by** *auto*
from *this*[*folded dvd-div-eq-0-iff*[*OF* ‹$|t|$ *dvd* $(gcd\ p\ q)$›]]
show *period w* $(gcd\ p\ q)$
using *per-mult*[*OF* ‹*period w* $|t|$›, *of gcd p q div* $|t|$, *unfolded dvd-div-mult-self*[*OF* ‹$|t|$ *dvd* $(gcd\ p\ q)$›]] **by** *blast*
qed

5.2 Optimality of the bound by construction of the Fine and Wilf word.

FW-word (where FW stands for Fine and Wilf) yields a word which shows the optimality of the bound in the periodicity lemma.

definition *fw-per k d* ≡ $[0..<d]^{@}k \cdot [0..<(d-1)]$
definition *fw-base k d* ≡ *fw-per k d* · [*d*] · *fw-per k d*

fun *FW-word* :: *nat* ⇒ *nat* ⇒ *nat list* **where**
FW-word-def: *FW-word p q* =
— symmetry (*if* $q < p$ *then FW-word q p else*
— artificial value *if* $p = 0$ *then* ε *else*
— artificial value *if p dvd q then* ε *else*
— base case *if* $gcd\ p\ q = q - p$ *then fw-base* $(p\ div\ (gcd\ p\ q) - 1)\ (gcd\ p\ q)$
— step *else* $(take\ p\ (FW\text{-}word\ p\ (q-p))) \cdot FW\text{-}word\ p\ (q-p))$

lemma *FW-sym*: *FW-word p q* = *FW-word q p*
by (*cases rule*: *linorder-cases*[*of p q*]) *simp*+

lemma *FW-dvd*: **assumes** *p dvd q* **shows** *FW-word p q* = ε
proof (*cases* $q = 0$, *use FW-word-def*[*of p q*] **in** *force*)
assume $q \neq 0$
hence ¬ $q < p$
using *assms* **by** *auto*
thus *?thesis*
using *FW-word-def* ‹*p dvd q*› **by** *auto*
qed

lemma *upt-minus-pref*: $[i..< j-d] \leq_p [i..< j]$
by (*rule le-cases*[*of j d*], *force*)
(*use upt-add-eq-append*[*of i j*−*d d*] **in** *fastforce*)

lemma *fw-per-pref*: *fw-per k d* ≤*p take d* (*fw-per k d*) · (*fw-per k d*)
proof−
consider $k = 0 \vee d = 0 \mid 0 < k \wedge 0 < d$

```
    by fastforce
  thus ?thesis
  proof(cases)
    assume k = 0 ∨ d = 0
    then show ?thesis
      unfolding fw-per-def by fastforce
  next
    assume 0 < k ∧ 0 < d
    hence 0 < k 0 < d
      by blast+
    hence nemp: [0..<d] @ k ≠ ε
      using emp-pow-pos-emp[OF - ‹0 < k›, of [0..<d]]
      unfolding upt-eq-Nil-conv[of 0 d] by blast
    have t: take d (fw-per k d) = [0..<d]
      unfolding fw-per-def pow-pos[OF ‹0 < k›]
        butlast-append if-not-P[OF nemp] by auto
    show ?thesis
      unfolding t unfolding fw-per-def
      unfolding lassoc pow-comm[symmetric]
      unfolding rassoc pref-cancel-conv
      using upt-minus-pref by simp
  qed
qed

lemma fw-per-len: shows |fw-per k d| = (k+1)*d − 1
 unfolding fw-per-def lenmorph pow-len
  by (cases d = 0) fastforce+

lemma fw-base-len: assumes 0 < d shows |fw-base k d| = (k+1)*d + (k+1)*d
− 1
  unfolding fw-base-def lenmorph fw-per-len sing-len using assms by simp

lemma fw-base-per1: assumes 0 < d
  shows period (fw-base k d) ((k+1)*d)
proof−
  have take: take ((k+1)*d) (fw-base k d) = fw-per k d · [d]
    unfolding fw-base-def lassoc using fw-per-len ‹0 < d› by force
  show period (fw-base k d) ((k+1)*d)
    unfolding period-def take unfolding fw-base-def by force
qed

lemma fw-base-per2: assumes 0 < d
  shows (fw-base k d) <p take ((k+1)*d + d) (fw-base k d) · (fw-base k d)
proof (rule per-rootI)
  show take ((k+1)*d + d) (fw-base k d) ≠ ε
    using ‹0 < d› unfolding fw-base-def by simp
  have t1: take ((k+1)*d) (fw-base k d) = fw-per k d · [d]
    unfolding fw-base-def lassoc using fw-per-len ‹0 < d› by force
  have |fw-per k d · [d]| = (k+1)*d
```

```
    using fw-per-len unfolding lenmorph sing-len using ‹0 < d› by simp
  hence d: drop ((k+1)*d) (fw-per k d · [d] · fw-per k d) = fw-per k d
    unfolding lassoc using drop-pref by metis
  show fw-base k d ≤p take (((k+1)*d) + d) (fw-base k d) · fw-base k d
    unfolding take-add t1 unfolding fw-base-def rassoc pref-cancel-conv d
    using fw-per-pref pref-prolong triv-pref by meson
qed

lemma fw-base-not-per: assumes 0 < d 0 < k
  shows ¬ period (fw-base k d) d (is ¬ period ?w d)
proof
  have u: take d ?w = [0..< d]
    unfolding fw-base-def fw-per-def pow-pos[OF ‹0 < k›] by force
  have suc: [0..<d] = [0..<d−1]·[d−1]
    using ‹0 < d› upt-Suc-append[of 0 d−1] by fastforce
  assume period ?w d
  hence ?w ≤p [0..< d] · ?w
    unfolding period-def u[symmetric] by blast
  hence fw-per k d · [d] ≤p [0..< d] · fw-per k d · [d]
    unfolding fw-base-def lassoc
    using  pref-cancel-right by blast
  hence  [0..<d−1] · [d] ≤p [0..<d] · [0..<d−1] · [d]
    unfolding u fw-per-def pow-pos[OF ‹0 < k›]
    unfolding lassoc pow-comm[symmetric] by fastforce
  from pref-prod-eq[OF this]
  have eq: [0..<d−1] · [d] = [0..<d]
    using ‹0 < d› by simp
  thus False
    using ‹0 < d› unfolding suc cancel by fastforce
qed

lemma fw-per-per: assumes fw-per k d ≠ ε
  shows (fw-per k d) <p take d (fw-per k d) · (fw-per k d)
  unfolding period-def
proof (rule per-rootI[OF fw-per-pref])
  show take d (fw-per k d) ≠ ε
  using assms unfolding take-eq-Nil fw-per-def by fastforce
qed

lemma fw-per-nth: assumes i < |fw-per k d|
  shows (fw-per k d)!i = i mod d
proof (cases k = 0)
  assume k = 0
  then show ?thesis
  using assms unfolding fw-per-def by force
next
  assume k ≠ 0
  hence 0 < k
    by blast
```

```
  have fw-per k d ≠ ε
    using assms by fastforce
  hence 0 < d
    unfolding fw-per-def by force
  have period (fw-per k d) d
    unfolding period-def using fw-per-per[OF ‹fw-per k d ≠ ε›] by blast
  from period-mod[OF this assms]
  have mod: fw-per k d ! i = fw-per k d ! (i mod d).
  show ?thesis
    using assms nth-upt[of 0 i mod d d, unfolded add-0] mod-less-divisor[OF ‹0 <
d›]
    unfolding mod
    unfolding fw-per-def pow-pos[OF ‹0 < k›] rassoc
   using length-upt[of 0 d, unfolded diff-zero] unfolding nth-append by presburger
qed

lemma fw-base-nth: assumes i < |fw-base k d|
  shows (fw-base k d)!i = (if i = |fw-per k d| then d else i mod d)
proof–
  note formula = nth-append[of fw-per k d · [d] fw-per k d i, unfolded rassoc, folded
fw-base-def]
  show ?thesis
  proof (rule linorder-cases[of i  |fw-per k d|])
    assume i = |fw-per k d|
    hence  i < |fw-per k d · [d]|
      by auto
    show ?thesis
     unfolding if-P[OF ‹i = |fw-per k d|›] formula if-P[OF ‹i < |fw-per k d · [d]|›]
      unfolding ‹i = |fw-per k d|› by simp
  next
    assume i < |fw-per k d|
    hence  i < |fw-per k d · [d]| i ≠ |fw-per k d|
      by auto
    show ?thesis
      unfolding formula if-not-P[OF ‹i ≠ |fw-per k d|›] if-P[OF ‹i < |fw-per k d ·
[d]|›]
      using nth-append[of fw-per k d [d] i]
      unfolding if-P[OF ‹i < |fw-per k d|›] fw-per-nth[OF ‹i < |fw-per k d|›].
  next
    assume |fw-per k d| < i
    hence  ¬ i < |fw-per k d · [d]| i ≠ |fw-per k d| |fw-per k d · [d]| ≤ i
      by auto
    have diff-less: i − |fw-per k d · [d]| < |fw-per k d|
      using diff-less-mono[OF assms ‹|fw-per k d · [d]| ≤ i›]
      unfolding fw-base-def by force
    have d ≠ 0
    proof
      assume d = 0
      show False
```

```
        using assms ‹|fw-per k d| < i› unfolding fw-base-def fw-per-def pow-len
lenmorph
         sing-len ‹d = 0› by force
    qed
    hence (k + 1) * d - 1 + 1 = (k+1)*d
      by simp
    have (k+1)*d ≤ i
      using ‹|fw-per k d| < i› unfolding fw-per-len by force
    show ?thesis
     unfolding formula if-not-P[OF ‹i ≠ |fw-per k d|›] if-not-P[OF ‹¬ i < |fw-per
k d · [d]|›]
      fw-per-nth[OF diff-less]
      unfolding lenmorph fw-per-len sing-len ‹(k + 1) * d - 1 + 1 = (k+1)*d›
      using mod-mult-self3[of k+1 d i - (k + 1) * d]
      unfolding le-add-diff-inverse[OF ‹(k+1)*d ≤ i›] by force
  qed
qed

lemma fw-base-match: assumes i < |fw-base k d| j < |fw-base k d| i ≠ j and
     eq: (fw-base k d)!i = (fw-base k d)!j
    shows i mod d = j mod d and i ≠ |fw-per k d| and j ≠ |fw-per k d|
proof (atomize (full), cases)
  assume d = 0
  hence fw-base k d = [0]
    unfolding fw-base-def fw-per-def
    by simp
  have i = j
    using assms(1-3) unfolding ‹fw-base k d = [0]› sing-len by blast
  thus i mod d = j mod d ∧ i ≠ |fw-per k d| ∧ j ≠ |fw-per k d|
    using ‹i ≠ j› by blast
next
  assume d ≠ 0
  hence i mod d ≠ d for i
    using mod-less-divisor[of d i] by force
 with eq[unfolded fw-base-nth[OF ‹i < |fw-base k d|›] fw-base-nth[OF ‹j < |fw-base
k d|›]]
  show i mod d = j mod d ∧ i ≠ |fw-per k d| ∧ j ≠ |fw-per k d|
    using ‹i ≠ j› by metis
qed

— A numeric formulation of the induction step
lemma ext-per-sum: assumes period w p and period w q and  p ≤ |w|
  shows period ((take p w) · w) (p+q)
proof–
  have sum: take (p + q) (take p w · w) = take p w · take q w
    unfolding take-add  by (simp add: ‹p ≤ |w|›)
  show ?thesis
    unfolding period-def sum rassoc
    using assms(1,2) per-lemma-step by auto
```

qed

lemma *drop-per-diff*: **assumes** *period w p* **and** *period w q* **and** $p < q$ **and** $p < |w|$
shows *period* (*drop p w*) (*q*−*p*)
proof−
have *nemp*: *take* (*q* − *p*) (*drop p w*) ≠ *ε*
using ‹*p* < |*w*|› ‹*p* < *q*› **by** *force*
have *t1*: *take p w* · *drop p* (*take q w*) = *take q w*
unfolding *drop-take take-add*[*symmetric*] **using** ‹*p* < *q*› **by** *simp*
from *per-lemma-step*[*of take p w drop p w drop p* (*take q w*),
unfolded append-take-drop-id t1,
unfolded drop-take]
have *drop p w* ≤*p take* (*q* − *p*) (*drop p w*) · *drop p w*
using *assms* **unfolding** *period-def* **using** *sprefD1* **by** *blast*
hence *drop p w* <*p take* (*q* − *p*) (*drop p w*) · *drop p w*
using *nemp* **by** *blast*
thus *?thesis*
unfolding *period-def* **by** *blast*
qed

theorem *fw-word*: **assumes** ¬ *p dvd q* ¬ *q dvd p*
shows |*FW-word p q*| = *p* + *q* − *gcd p q* − *1*
period (*FW-word p q*) *p* **and** *period* (*FW-word p q*) *q*
¬ *period* (*FW-word p q*) (*gcd p q*)
using *assms*
proof (*atomize* (*full*), *induction p* + *p* + *q arbitrary*: *p q rule*: *less-induct*)
case *less*
have *p* ≠ *0*
using ‹¬ *q dvd p*› *dvd-0-right*[*of q*] **by** *meson*
hence *0* < *p*
by *simp*
have *p* ≠ *q*
using ‹¬ *p dvd q*› **by** *auto*
then consider *q* < *p* | *p* < *q*
by *linarith*
then show *?case*
proof (*cases*)
assume *q* < *p*
hence *q* + *q* + *p* < *p* + *p* + *q*
by *simp*
from *less.hyps*[*OF this* ‹¬ *q dvd p*› ‹¬ *p dvd q*›]
show *?case*
unfolding *FW-sym*[*of p q*] *gcd.commute*[*of p q*] *add.commute*[*of p q*] **by** *blast*
next
assume *p* < *q*
— auxiliary
hence ¬ *q* < *p p* + (*q* − *p*) = *q p* ≥ *gcd p q q* − *p* ≥ *1 q* −*p* ≠ *0*
using ‹*p* ≠ *0*› ‹*p* < *q*› **by** *auto*

```
  have q − p ≥ gcd p q
    using ‹p < q› ‹p + (q − p) = q› ‹q −p ≠ 0› gcd-add2 gcd-le2-nat by metis
  hence q ≥ gcd p q + 1
    using ‹0 < p› ‹p + (q − p) = q› ‹p ≥ gcd p q› ‹q − p ≥ 1› by linarith

  let ?w = FW-word p q
  let ?d = gcd p q
  let ?k = p div gcd p q − 1
  let ?dw = [0..<(gcd p q)]
  let ?pd = p div (gcd p q)
  show ?thesis
  proof (cases ?d = q − p)
    assume ?d = q − p
    hence p + ?d = q p + q − ?d = p + p 0 < ?d ?d ≠ q ?d < q 1 ≤ q − ?d
      using ‹p ≠ 0› ‹p < q› by auto
    have ?d ≠ p
      using ‹¬ p dvd q›  ‹gcd p q = q − p› gcd-unique-nat by metis
    have kdp: (?k + 1) * ?d = p
       by (metis ‹p ≠ 0› add.commute dvd-mult-div-cancel gcd-dvd1 less-one
linordered-semidom-class.add-diff-inverse mult.commute mult-zero-right)
    hence 0 < ?k
      using ‹?d ≠ p› add-0[of 1] mult-1 not-gr-zero by metis
    let ?per  = fw-per ?k ?d
    have fw-base: ?w = fw-base ?k ?d
      unfolding FW-word-def[of p q] if-not-P[OF ‹¬ q < p›] if-not-P[OF ‹p ≠
0›]
       if-not-P[OF ‹p ≠ q›] if-P[OF ‹?d = q − p›] using less.prems(1) by argo
    hence fw: ?w = ?per·[?d]·?per
      unfolding fw-base-def.
    show (|?w| = p + q − ?d − 1 ∧
       period ?w p) ∧ period ?w q ∧
       ¬ period ?w ?d
      unfolding conj-assoc[symmetric]
    proof (rule conjI)+
      show |?w| = p + q − ?d − 1
        unfolding fw-base fw-base-len[OF ‹0 < ?d›]
          ‹p + q − ?d = p + p› kdp..
      show period ?w p
        unfolding fw-base using fw-base-per1[OF ‹0 < ?d›, of ?k, unfolded kdp].
      show period ?w q
        unfolding fw-base period-def using
       fw-base-per2[OF ‹0 < ?d›, of ?k, unfolded kdp ‹p + gcd p q = q›] by blast
      show ¬ period ?w ?d
        unfolding fw-base using fw-base-not-per[OF ‹0 < ?d› ‹0 < ?k›].
    qed
  next
    assume gcd p q ≠ q − p
    hence q − p > gcd p q
      using ‹q − p ≥ gcd p q› by force
```

```
      let ?w' = FW-word p (q−p)
     have gcd p (q−p) = ?d
      using  gcd-add2[of p q−p, unfolded le-add-diff-inverse[OF less-imp-le[OF ‹p
< q›]], symmetric].
     have fw: ?w = take p ?w' · ?w'
      using FW-word-def ‹p ≠ 0› ‹p ≠ q› ‹p < q› ‹gcd p q ≠ q − p›
      ‹¬ q < p› ‹¬ p dvd q› by meson
     show (|?w| = p + q − ?d − 1 ∧
        period ?w p) ∧ period ?w q ∧
        ¬ period ?w ?d
      unfolding conj-assoc[symmetric]
     proof (rule conjI)+

      have divhyp1: ¬ p dvd q − p
       using ‹¬ p dvd q› ‹p < q› dvd-minus-self by auto

      have divhyp2: ¬ q − p dvd p
      proof (rule notI)
       assume q − p dvd p
       have q = p + (q − p)
        by (simp add: ‹p < q› less-or-eq-imp-le)
       from gcd-add2[of p q − p, folded  this, unfolded gcd-nat.absorb2[of q − p
p, OF ‹q − p dvd p›]]
       show False
        using ‹gcd p q ≠ q − p› by blast
      qed

      have lenhyp: p + p + (q − p) < p + p + q
       using ‹p < q› ‹p ≠ 0› by linarith

— induction assumption
      have len-w': |?w'| = p + (q − p) − ?d − 1 and period ?w' p and period
?w' (q−p) and
       ¬ period ?w' (gcd p (q−p))
       using less.hyps[OF - divhyp1 divhyp2] lenhyp
       unfolding ‹gcd p (q−p) = ?d› by blast+

      have p ≤ |?w'|
       unfolding  len-w' using ‹q − p > gcd p q› by force
      have ?w' ≠ ε
       using ‹¬ period ?w' (gcd p (q − p))› emp-all-periods by metis
      show ¬ period ?w ?d
       using ‹¬ period (FW-word p (q −p)) (gcd p (q−p))›
       unfolding ‹gcd p (q−p) = ?d›
      using period-fac-period[of take p ?w' ?w' ε ?d, unfolded append-Nil2, folded
fw] by blast

      show |?w| = p + q − ?d − 1
       unfolding fw lenmorph len-w' take-len[OF ‹p ≤ |?w'|›] ‹p + (q − p) = q›
```

using ‹$q \geq gcd\ p\ q + 1$› **by** *simp*

show *period ?w p*
using *fw ext-per-left*[*OF* ‹*period* (*FW-word p* (*q*−*p*)) *p*› ‹$p \leq |?w'|$›]
by *presburger*

show *period ?w q*
using *ext-per-sum*[*OF* ‹*period ?w' p*› ‹*period ?w'* (*q* − *p*)› ‹$p \leq |?w'|$›, *folded fw*, *unfolded* ‹*p* + (*q*−*p*) = *q*›].

qed
qed
qed
qed

Calculation examples

value *FW-word 0 4*
value *FW-word 3 4*
value *FW-word 4 7*
value *FW-word 3 7*
value *FW-word 5 7*
value *FW-word 5 13*
value *FW-word 4 6*
value *FW-word 12 18*
value *FW-word 6 10*
value *FW-word 10 16*

5.3 Optimality of the Fine and Wilf word.

The *FW-word* is the most general word having the two desired properties. That is, each equality of letters is forced by periods.

lemma *fw-base-mod-aux*: **assumes** (*i* :: *nat*) < *p* + *p* − *1 i* ≠ *p* − *1*
shows *i mod p* < *p* − *1*
proof (*cases p i rule*: *le-less-cases*)
assume *p* ≤ *i*
have *i* − *p* < *p i* − *p* < *p* − *1*
using *diff-less-mono*[*OF* ‹*i* < *p* + *p* − *1*› ‹*p* ≤ *i*›] **unfolding** *diff-cancel2 diff-diff-eq* **by** *simp-all*
from *le-mod-geq*[*OF* ‹*p* ≤ *i*›]
have *i mod p* = *i* − *p*
unfolding *mod-less*[*OF* ‹*i* − *p* < *p*›].
thus *?thesis*
using ‹*i* − *p* < *p* − *1*› **by** *argo*
next
assume *i* < *p*
show *?thesis*
unfolding *mod-less*[*OF* ‹*i* < *p*›]
using ‹*i* < *p*› ‹*i* ≠ *p* − *1*› **by** *linarith*

qed

theorem *fw-minimal*: **assumes** *period w p* **and** *period w q* **and** $|w| = |FW\text{-}word\ p\ q|$ **and**
$i < |w|$ **and** $j < |w|$ **and** $(FW\text{-}word\ p\ q)!i = (FW\text{-}word\ p\ q)!j$
shows $w!i = w!j$
using *assms*
proof (*induct p + p + q arbitrary*: *w i j p q rule*: *less-induct*)
case *less*
— preliminaries
have $FW\text{-}word\ p\ q \neq \varepsilon$
using ‹$j < |w|$› ‹$|w| = |FW\text{-}word\ p\ q|$› *emp-len*[*of FW-word p q*] **by** *linarith+*
have $0 < p$
by (*rule ccontr*) (*use* ‹$FW\text{-}word\ p\ q \neq \varepsilon$› **in** *force*)
have $\neg\ p\ dvd\ q$
using *FW-dvd* ‹$FW\text{-}word\ p\ q \neq \varepsilon$› **by** *blast*
hence $1 < p$
using ‹$0 < p$› *less-one nat-neq-iff one-dvd* **by** *metis*
have $\neg\ q\ dvd\ p$
using *FW-dvd* ‹$FW\text{-}word\ p\ q \neq \varepsilon$›[*unfolded FW-sym*[*of p*]] **by** *blast*
note *fw-word*[*OF* ‹$\neg\ p\ dvd\ q$› ‹$\neg\ q\ dvd\ p$›]
have *w-len*: $|w| = p + q - gcd\ p\ q - 1$
using *fw-word*(*1*)[*OF* ‹$\neg\ p\ dvd\ q$› ‹$\neg\ q\ dvd\ p$›] **unfolding** ‹$|w| = |FW\text{-}word\ p\ q|$›**.**

show $w\ !\ i = w\ !\ j$
proof (*cases i = j*, *blast*)
assume $i \neq j$
show $w\ !\ i = w\ !\ j$
proof (*cases q < p*)
assume $q < p$
then show $w\ !\ i = w\ !\ j$
using *less.hyps*[*OF - less.prems*(*2*,*1*) *less.prems*(*3−6*)[*unfolded FW-sym*[*of p*]]] **by** *simp*
next
assume $\neg\ q < p$
hence $p < q$
using ‹$\neg\ p\ dvd\ q$› *linorder-neqE-nat* **by** *fastforce+*
show $w\ !\ i = w\ !\ j$
proof (*cases gcd p q = q − p*)
assume $gcd\ p\ q = q - p$
hence *base*: $FW\text{-}word\ p\ q = fw\text{-}base\ (p\ div\ (gcd\ p\ q) - 1)\ (gcd\ p\ q)$ — This is the base case.
using *FW-word-def*[*of p q*] ‹$\neg\ q < p$› ‹$0 < p$› ‹$\neg\ p\ dvd\ q$› **by** *presburger*
let *?d* = $gcd\ p\ q$
let *?pref* = $take\ (p-1)\ w$
let *?FW* = $FW\text{-}word\ p\ q$

— Auxiliary facts
from ‹*0 < p*› *dvd-div-eq-0-iff*[*OF gcd-nat.cobounded1*, *of p q*]
have *per-len*: |*fw-per* (*p div ?d − 1*) *?d*| = *p − 1*
unfolding *fw-per-len*
using *dvd-mult-div-cancel*[*OF gcd-nat.cobounded1*, *of p q*, *unfolded mult.commute*[*of ?d*]] **by** *force*
have *w-len′*: |*w*| = *p* + *p* − *1*
unfolding *w-len* ‹*?d = q − p*› **using** ‹*p < q*› **by** *auto*
hence *p* < |*w*| *p* − *1* ≤ |*w*|
using ‹*1 < p*› **by** *simp-all*
have *i mod p mod ?d* = *i mod ?d*
using *mod-mod-cancel*[*OF gcd-nat.cobounded1*].
have *ij-less*: *i* < |*?FW*| *j* < |*?FW*|
using ‹*i* < |*w*|› ‹*j* < |*w*|› **unfolding** ‹|*w*| = |*?FW*|› **by** *force+*
have |*?pref*| = *p* − *1*
using ‹*p* − *1* ≤ |*w*|› **by** *simp*

— Indeces which agree on FW are as follows:
have *mod-d-eq*: *i mod ?d* = *j mod ?d* **and** *not-max*: *i* ≠ *p−1 j* ≠ *p−1*
using *fw-base-match*[*OF ij-less*[*unfolded base*] ‹*i* ≠ *j*› ‹*?FW* ! *i* = *?FW* ! *j*›[*unfolded base*]]
unfolding *per-len* **by** *blast+*
have *i mod p* < *p* − *1 j mod p* < *p* − *1* — Key fact: taking modulo p, we avoid the non-periodic element [d]
using *fw-base-mod-aux* ‹*i* < |*w*|› ‹*j* < |*w*|› *not-max* **unfolding** ‹|*w*| = *p* + *p* − *1*› **by** *blast+*
have *period ?pref ?d* — The prefix of w has a short period: the standard reduction step
proof−
have *drop p w* = *?pref*
proof−
have *drop p w* ≤*p w*
using ‹*period w p*› **unfolding** *per-shift* **by** *blast*
moreover have |*drop p w*| = *p* − *1*
using ‹|*w*| = *p* + *p* − *1*› **unfolding** *length-drop* **by** *force*
ultimately show *?thesis*
using *pref-take-conv* **by** *metis*
qed
show *?thesis*
using *drop-per-diff*[*OF* ‹*period w p*› ‹*period w q*› ‹*p < q*› ‹*p* < |*w*|›]
unfolding ‹*drop p w* = *?pref*› ‹*?d* = *q* − *p*›.
qed

— Therefore letters can be mapped to the first period.
have *mod-pref-reduce*: *?pref* ! (*i mod p*) = *?pref* ! (*i mod ?d*) *?pref* ! (*j mod p*) = *?pref* ! (*j mod ?d*)
using *period-mod*[*OF* ‹*period ?pref ?d*›, *unfolded* ‹|*?pref*| = *p* − *1*›, *OF* ‹*i mod p* < *p−1*›, *unfolded mod-mod-cancel*[*OF gcd-nat.cobounded1*]]
period-mod[*OF* ‹*period ?pref ?d*›, *unfolded* ‹|*?pref*| = *p* − *1*›, *OF* ‹*j mod p*

< p−1›, unfolded mod-mod-cancel[OF gcd-nat.cobounded1]].
have *?pref ! (i mod p) = ?pref ! (j mod p)* — Hence they are the same, because the indeces coincide modulo d

unfolding *mod-pref-reduce mod-d-eq*..
thus *w ! i = w ! j*
unfolding *nth-take[OF ‹i mod p < p−1›, of w] nth-take[OF ‹j mod p < p−1›, of w]* — Since w has period p, equality for the prefix is equality for the whole word
unfolding *period-mod[OF ‹period w p› ‹i < |w|›] period-mod[OF ‹period w p› ‹j < |w|›]*. — Equality in the whole word can be tested modulo p
next
assume *gcd p q ≠ q − p* — Non-base case
hence *step: FW-word p q = take p (FW-word p (q−p)) · FW-word p (q−p)*
unfolding *FW-word-def[of p q]* **using** *‹¬ q < p› ‹0 < p› ‹¬ p dvd q›* **by** *presburger*

have *¬ p dvd (q−p)*
unfolding *dvd-minus-self* **using** *‹¬ q < p› ‹¬ p dvd q›* **by** *blast*
have *¬ (q−p) dvd p*
using *FW-dvd[of q−p p] FW-dvd[of q−p p, THEN Nil-take-Nil]*
‹FW-word p q ≠ ε› **unfolding** *step[unfolded FW-sym[of - q−p]]* **by** *blast*
have *gcd p (q − p) < (q − p)*
using *‹¬ q − p dvd p›[unfolded gcd-nat.absorb-iff2[of q−p p]]*
nat-dvd-not-less[OF ‹p < q›[folded zero-less-diff[of q]], of gcd p (q−p)] **by** *fastforce*
hence *p ≤ |FW-word p (q − p)|*
unfolding *fw-word(1)[OF ‹¬ p dvd (q−p)› ‹¬ (q−p) dvd p›]* **by** *linarith*
from *drop-take-inv[OF this]*
have *fw′: FW-word p (q −p) = drop p (FW-word p q)*
using *step* **by** *metis*
have *p ≤ |drop p w|*
using *‹p ≤ |FW-word p (q − p)|› less.prems(3)* **unfolding** *fw′ length-drop* **by** *argo*
hence *p < |w|*
using *‹0 < p›* **by** *fastforce*

have *FW-word p (q − p) <p FW-word p q*
using *‹period (FW-word p q) p› ‹0 < p› ‹FW-word p q ≠ ε› drop-neq*
unfolding *per-shift fw′* **by** *blast*
from *sprefD1[OF this]*
have *FW-word p (q − p) ! (i mod p) = FW-word p q ! (i mod p) FW-word p (q − p) ! (j mod p) = FW-word p q ! (j mod p)*
using *pref-index[OF ‹FW-word p (q − p) ≤p FW-word p q›] less-le-trans[OF mod-less-divisor[OF ‹0 < p›] ‹p ≤ |FW-word p (q−p)|›]*
by *blast+*
with *period-mod[OF ‹period (FW-word p q) p› ‹j < |w|›[unfolded ‹|w| = |FW-word p q|›]]*
period-mod[OF ‹period (FW-word p q) p› ‹i < |w|›[unfolded ‹|w| =

```
|FW-word p q|›]]
      have reduce-fw: FW-word p (q − p) ! (j mod p) = (FW-word p q) ! j FW-word
p (q − p) ! (i mod p) = (FW-word p q) ! i
        by argo+

       have drop p w <p w
         using ‹period w p› ‹0 < p› ‹p < |w|› unfolding per-shift by fastforce
       from sprefD1[OF this]
        have drop p w ! (i mod p) = w ! (i mod p) drop p w ! (j mod p) = w ! (j
mod p)
       using pref-index[OF ‹drop p w ≤p w›] less-le-trans[OF mod-less-divisor[OF
‹0 < p›] ‹p ≤ |drop p w|›]
        by blast+
       with period-mod[OF ‹period w p› ‹j < |w|›] period-mod[OF ‹period w p› ‹i
< |w|›]
       have reduce-w: w ! j = drop p w ! (j mod p) w ! i = drop p w ! (i mod p)
         by argo+

       show w ! i = w ! j
         unfolding reduce-w
       proof(rule less.hyps[of p q−p drop p w i mod p j mod p])
         show  p + p + (q − p) < p + p + q
           using ‹p < q› ‹0 < p› by linarith
         show period (drop p w) p
           using ‹p < |w|› period-suf-period[OF ‹period w p› suffix-drop, of p]
           less-le-trans[OF ‹0 < p› ‹p ≤ |drop p w|›] by blast
         show period (drop p w) (q − p)
           using  drop-per-diff[OF ‹period w p› ‹period w q› ‹p < q› ‹p < |w|›].
         show |drop p w| = |FW-word p (q − p)|
           using fw' length-drop ‹|w| = |FW-word p q|› by metis
         show i mod p < |drop p w| j mod p < |drop p w|
            using less-le-trans[OF mod-less-divisor[OF ‹0 < p›] ‹p ≤ |drop p w|›]
by blast+
         show FW-word p (q − p) ! (i mod p) = FW-word p (q − p) ! (j mod p)
           unfolding reduce-fw by fact
       qed
     qed
   qed
  qed
qed

theorem fw-minimal-set: assumes period w p and period w q and |w| = |FW-word
p q|
  shows card(set w) ≤ card(set (FW-word p q))
proof−
  let ?f = λ n. (w ! (LEAST k.((FW-word p q)!k = n)))
  have map-f0: ?f ((FW-word p q)!i) = w!i if i < |w| for i
    using fw-minimal[OF assms - that LeastI[of λ k.(FW-word p q ! k = FW-word
p q ! i), OF refl]]
```

Least-le[*of* $\lambda\ k.$(*FW-word p q* ! k = *FW-word p q* ! i), *OF refl*] *that* **by** *linarith*
have *map-f*: *map ?f* (*FW-word p q*) = w
by (*rule nth-equalityI*, *unfold length-map*, *simp add*: *assms*(*3*))
(*use nth-map*[*of* - *FW-word p q ?f*] *map-f0 assms*(*3*) **in** *presburger*)
then have *set w* = *?f* ‘ (*set* (*FW-word p q*))
using *set-map*[*of ?f FW-word p q*] **by** *presburger*
then show *?thesis*
using *card-image-le*[*of set* (*FW-word p q*) *?f*, *OF finite-set*] *fw-minimal*[*OF assms*] **by** *presburger*
qed

5.4 Other variants of the periodicity lemma

periodicity lemma is one of the most frequent tools in Combinatorics on words. Here are some useful variants.

Note that the following lemmas are stronger versions of ⟦*?w* $<_p$ *?p* · *?w*; $<_s$ *?w* (*?w* · *?q*); |*?p*| + |*?q*| ≤ |*?w*|; ⋀*r s k l m*. ⟦*?p* = (*r* · *s*) @ *k*; *?q* = (*s* · *r*) @ *l*; *?w* = (*r* · *s*) @ *m* · *r*; *primitive* (*r* · *s*)⟧ ⟹ *?thesis*⟧ ⟹ *?thesis*
⟦$\leq_f$ *?u* (*?r* @ *?k*); $\leq_f$ *?u* (*?s* @ *?l*); |*?r*| + |*?s*| ≤ |*?u*|⟧ ⟹ *ϱ ?r* ∼ *ϱ ?s*
⟦$\leq_f$ *?u* (*?r* @ *?k*); $\leq_f$ *?u* (*?s* @ *?l*); |*?r*| + |*?s*| ≤ |*?u*|⟧ ⟹ *ϱ ?r* ∼ *ϱ ?s*
⟦$\leq_f$ *?w* (*?u* @ *?n*); $\leq_f$ *?w* (*?v* @ *?m*); *primitive ?u*; *primitive ?v*; |*?u*| + |*?v*| ≤ |*?w*|⟧ ⟹ *?u* ∼ *?v* that have a relaxed length assumption |*p*| + |*q*| ≤ |*w*| instead of |*p*| + |*q*| − *gcd* |*p*| |*q*| ≤ |*w*| (and which follow from the relaxed version of periodicity lemma ⟦*?w* ≤*p* *?u* · *?w*; *?w* ≤*p* *?v* · *?w*; |*?u*| + |*?v*| ≤ |*?w*|⟧ ⟹ *?u* · *?v* = *?v* · *?u*.

lemma *per-lemma-comm-pref*:
assumes $u \leq_p r^{@}k$ $u \leq_p s^{@}l$
and *len*: |*r*| + |*s*| − *gcd* (|*r*|) (|*s*|) ≤ |*u*|
shows $r \cdot s = s \cdot r$
using *pref-pow-root*[*OF assms*(*2*)] *pref-pow-root*[*OF assms*(*1*)] *per-lemma-comm*[*OF* - - *len*] **by** *blast*

lemma *per-lemma-pref-suf-gcd*: **assumes** $w <_p p \cdot w$ **and** $w <_s w \cdot q$ **and**
fw: |*p*| + |*q*| − (*gcd* |*p*| |*q*|) ≤ |*w*|
obtains *r s k l m* **where** $p = (r \cdot s)^{@}k$ **and** $q = (s \cdot r)^{@}l$ **and** $w = (r \cdot s)^{@}m \cdot r$ **and** *primitive* (*r*·*s*)
proof−
let *?q* = $(w \cdot q)^{<-1}w$
have $w <_p$ *?q* · w
using *ssufD1*[*OF* ‹$w <_s w \cdot q$›] *rq-suf*[*symmetric*, *THEN per-rootI*[*OF prefI rq-ssuf-nemp*[*OF* ‹$w <_s w \cdot q$›]]]
by *argo*
have $q \sim$ *?q*
by (*meson assms*(*2*) *conjugI1 conjug-sym rq-suf suffix-order.less-imp-le*)

have *nemps'*: $p \neq \varepsilon$ $?q \neq \varepsilon$
using *assms(1)* ‹$w <p$ $?q \cdot w$› **by** *fastforce+*
have $|p| + |?q| - gcd\ (|p|)\ (|?q|) \leq |w|$ **using** *fw*
unfolding *conjug-len*[*OF* ‹$q \sim ?q$›]**.**
from *per-lemma-comm*[*OF sprefD1*[*OF* ‹$w <p\ p \cdot w$›] *sprefD1*[*OF* ‹$w <p$ $?q \cdot w$›] *this*]
have $p \cdot ?q = ?q \cdot p$**.**
then have $\varrho\ p = \varrho\ ?q$ **using** *comm-primroots*[*OF nemps'*] **by** *force*
hence [*symmetric*]: $\varrho\ q \sim \varrho\ p$
using *conjug-primroot*[*OF* ‹$q \sim (w \cdot q)^{<-1} w$›]
by *argo*
from *conjug-primrootsE*[*OF this*]
obtain $r\ s\ k\ l$ **where**
$p = (r \cdot s)\ ^{@}\ k$ **and**
$q = (s \cdot r)\ ^{@}\ l$ **and**
primitive $(r \cdot s)$**.**
have $w \leq p\ (r \cdot s) \cdot w$
using *assms per-root-drop-exp sprefD1* ‹$p = (r \cdot s)\ ^{@}\ k$›
by *meson*
have $w \leq s\ w \cdot (s \cdot r)$
using *assms(2) per-root-drop-exp*[*reversed*] *ssufD1* ‹$q = (s \cdot r)\ ^{@}\ l$›
by *meson*
have $|r \cdot s| \leq |w|$
using *conjug-nemp-iff*[*OF* ‹$q \sim ?q$›] *dual-order.trans length-0-conv nemps' per-lemma-len-le*[*OF fw*] *primroot-len-le*[*OF nemps'(1)*]
unfolding *primroot-unique*[*OF nemps'(1)* ‹*primitive* $(r \cdot s)$› ‹$p = (r \cdot s)\ ^{@}\ k$›]
by *fastforce*
from *root-suf-conjug*[*OF* ‹*primitive* $(r \cdot s)$› ‹$w \leq p\ (r \cdot s) \cdot w$› ‹$w \leq s\ w \cdot (s \cdot r)$› *this*]
obtain m **where** $w = (r \cdot s)\ ^{@}\ m \cdot r$**.**
from *that*[*OF* ‹$p = (r \cdot s)\ ^{@}\ k$› ‹$q = (s \cdot r)\ ^{@}\ l$› *this* ‹*primitive* $(r \cdot s)$›]
show *?thesis***.**
qed

lemma *fac-two-conjug-primroot-gcd*:
assumes *facs*: $w \leq f\ p^{@}k\ w \leq f\ q^{@}l$ **and** *nemps*: $p \neq \varepsilon\ q \neq \varepsilon$ **and** *len*: $|p| + |q| - gcd\ (|p|)\ (|q|) \leq |w|$
obtains $r\ s\ m$ **where** $\varrho\ p \sim r \cdot s$ **and** $\varrho\ q \sim r \cdot s$ **and** $w = (r \cdot s)^{@}m \cdot r$ **and** *primitive* $(r \cdot s)$
proof −
obtain p' **where** $w <p\ p' \cdot w\ p \sim p'\ p' \neq \varepsilon$
using *conjug-nemp-iff fac-pow-pref-conjug*[*OF facs(1)*] *nemps(1) per-rootI'* **by** *metis*
obtain q' **where** $w <s\ w \cdot q'\ q \sim q'\ q' \neq \varepsilon$
using *fac-pow-pref-conjug*[*reversed*, *OF* ‹$w \leq f\ q^{@}l$›] *conjug-nemp-iff nemps(2) per-rootI'*[*reversed*] **by** *metis*
from *per-lemma-pref-suf-gcd*[*OF* ‹$w <p\ p' \cdot w$› ‹$w <s\ w \cdot q'$›]
obtain $r\ s\ k\ l\ m$ **where**
$p' = (r \cdot s)\ ^{@}\ k$ **and**
$q' = (s \cdot r)\ ^{@}\ l$ **and**

$w = (r \cdot s)\ ^{@}\ m \cdot r$ **and**
$primitive\ (r \cdot s)$
using *len*[*unfolded conjug-len*[*OF* ‹$p \sim p'$›] *conjug-len*[*OF* ‹$q \sim q'$›]]
by *metis*
moreover have $\varrho\ p' = r{\cdot}s$
using ‹$p' = (r \cdot s)\ ^{@}\ k$› ‹$primitive\ (r \cdot s)$› ‹$p' \neq \varepsilon$› *primroot-unique*
by *meson*
hence $\varrho\ p \sim r{\cdot}s$
using *conjug-primroot*[*OF* ‹$p \sim p'$›]
by *simp*
moreover have $\varrho\ q' = s{\cdot}r$
using ‹$q' = (s \cdot r)\ ^{@}\ l$› ‹$primitive\ (r \cdot s)$›[*unfolded conjug-prim-iff'*[*of r*]] ‹$q' \neq \varepsilon$› *primroot-unique*
by *meson*
hence $\varrho\ q \sim s{\cdot}r$
using *conjug-primroot*[*OF* ‹$q \sim q'$›] **by** *simp*
hence $\varrho\ q \sim r{\cdot}s$
using *conjug-trans*[*OF - conjugI'*]
by *meson*
ultimately show *?thesis*
using *that* **by** *blast*
qed

corollary *fac-two-conjug-primroot'-gcd*:
assumes *facs*: $u \leq_f r^{@}k\ u \leq_f s^{@}l$ **and** *nemps*: $r \neq \varepsilon\ s \neq \varepsilon$ **and** *len*: $|r| + |s| - gcd\ (|r|)\ (|s|) \leq |u|$
shows $\varrho\ r \sim \varrho\ s$
using *fac-two-conjug-primroot-gcd*[*OF assms*] *conjug-trans*[*OF - conjug-sym*[*of* ϱ *s*]].

lemma *fac-two-conjug-primroot''-gcd*:
assumes *facs*: $u \leq_f r^{@}k\ u \leq_f s^{@}l$ **and** $u \neq \varepsilon$ **and** *len*: $|r| + |s| - gcd\ (|r|)\ (|s|) \leq |u|$
shows $\varrho\ r \sim \varrho\ s$
proof −
have *nemps*: $r \neq \varepsilon\ s \neq \varepsilon$ **using** *facs* ‹$u \neq \varepsilon$› **by** *force+*
show $conjugate\ (\varrho\ r)\ (\varrho\ s)$ **using** *fac-two-conjug-primroot'-gcd*[*OF facs nemps len*].
qed

lemma *fac-two-prim-conjug-gcd*:
assumes $w \leq_f u^{@}n\ w \leq_f v^{@}m$ *primitive u primitive v* $|u| + |v| - gcd\ (|u|)\ (|v|) \leq |w|$
shows $u \sim v$
using *fac-two-conjug-primroot'-gcd*[*OF assms*(*1–2*) - - *assms*(*5*)] *prim-nemp*[*OF* ‹*primitive u*›] *prim-nemp*[*OF* ‹*primitive v*›]
unfolding *prim-primroot*[*OF* ‹*primitive u*›] *prim-primroot*[*OF* ‹*primitive v*›].

lemma *two-pers-1*:

assumes *pu*: $w \leq_p u \cdot w$ **and** *pv*: $w \leq_p v \cdot w$ **and** *len*: $|u| + |v| - 1 \leq |w|$
shows $u \cdot v = v \cdot u$
proof (*rule nemp-comm*)
assume $u \neq \varepsilon$ $v \neq \varepsilon$
hence $1 \leq gcd\ |u|\ |v|$
using *nemp-len* **by** (*simp add*: *Suc-leI*)
thus *?thesis*
using *per-lemma-comm*[*OF pu pv*] *len* **by** *linarith*
qed

end

theory *Lyndon-Schutzenberger*
imports *Submonoids Periodicity-Lemma*

begin

Chapter 6

Lyndon-Schützenberger Equation

6.1 The original result

The Lyndon-Schützenberger equation is the following equation:

$$x^a y^b = z^c,$$

in this formalization denoted as $x \,^{@}\, a \cdot y \,^{@}\, b = z \,^{@}\, c$.

We formalize here a complete solution of this equation.

The main result, proved by Lyndon and Schützenberger is that the equation has periodic solutions only in free groups if $2 \leq a, b, c$ In this formalization we consider the equation in words only. Then the original result can be formulated as saying that all words x, y and z satisfying the equality ith $2 \leq a, b, c$ pairwise commute.

The result in free groups was first proved in [7]. For words, there are several proofs to be found in the literature (for instance [4, 2]). The presented proof is the authors' proof.

In addition, we give a full parametric solution of the equation for any a, b and c.

6.2 The original result

If x^a or y^b is sufficiently long, then the claim follows from the periodicity Lemma.

lemma *LS-per-lemma-case1*:
  assumes *eq*: $x^{@}a \cdot y^{@}b = z^{@}c$ **and** $0 < a$ **and** $0 < b$ **and** $|z| + |x| - 1 \leq |x^{@}a|$
  shows $x \cdot y = y \cdot x$ **and** $x \cdot z = z \cdot x$
proof (*rule nemp-comm*)

have $x^{@}a \leq p\ (z^{@}c) \cdot x^{@}a\ x\ ^{@}\ a \leq p\ x \cdot x\ ^{@}\ a$
unfolding *eq*[*symmetric*] *shifts-rev* **by** *blast+*
hence $x^{@}a \leq p\ z \cdot x^{@}a$
using *eq pref-pow-root triv-pref* **by** *metis*
from *two-pers-1*[*OF this* ‹$x\ ^{@}\ a \leq p\ x \cdot x\ ^{@}\ a$› ‹$|z| + |x| - 1 \leq |x\ ^{@}\ a|$›, *symmetric*]
show $x \cdot z = z \cdot x$**.**
hence $z^{@}c{\cdot}x^{@}a = x^{@}a{\cdot}z^{@}c$
by (*simp add*: *comm-pows-comm*)
from *this*[*folded eq*, *unfolded rassoc cancel*, *symmetric*]
have $x^{@}a \cdot y^{@}b = y^{@}b \cdot x^{@}a$**.**
from *this*[*unfolded comm-pow-roots*[*OF* ‹$0 < a$› ‹$0 < b$›]]
show $x \cdot y = y \cdot x$**.**
qed

A weaker version will be often more convenient

lemma *LS-per-lemma-case*:
assumes *eq*: $x^{@}a{\cdot}y^{@}b = z^{@}c$ **and** $0 < a$ **and** $0 < b$ **and** $|z| + |x| \leq |x^{@}a|$
shows $x \cdot y = y \cdot x$ **and** $x \cdot z = z \cdot x$
using *LS-per-lemma-case1*[*OF assms*(*1–3*)] *assms*(*4*) **by** *force+*

The most challenging case is when $c = 3$.

lemma *LS-core-case*:
assumes
eq: $x^{@}a \cdot y^{@}b = z^{@}c$ **and**
$2 \leq a$ **and** $2 \leq b$ **and** $2 \leq c$ **and**
$c = 3$ **and**
$b{*}|y| \leq a{*}|x|$ **and** $x \neq \varepsilon$ **and** $y \neq \varepsilon$ **and**
lenx: $a{*}|x| < |z| + |x|$ **and**
leny: $b{*}|y| < |z| + |y|$
shows $x{\cdot}y = y{\cdot}x$
proof–
have $0 < a$ **and** $0 < b$
using ‹$2 \leq a$› ‹$2 \leq b$› **by** *auto*

— We first show that a = 2
have $a{*}|x|{+}b{*}|y| = 3{*}|z|$
using ‹$c = 3$› *eq lenmorph*[*of* $x^{@}a$ $y^{@}b$]
by (*simp add*: *pow-len*)
hence $3{*}|z| \leq a{*}|x| + a{*}|x|$
using ‹$b{*}|y| \leq a{*}|x|$› **by** *simp*
hence $3{*}|z| < 2{*}|z| + 2{*}|x|$
using *lenx* **by** *linarith*
hence $|z| + |x| < 3 * |x|$ **by** *simp*
from *less-trans*[*OF lenx this*, *unfolded mult-less-cancel2*]
have $a = 2$ **using** ‹$2 \leq a$› **by** *force*

hence $|y| \leq |x|$ **using** ‹$b{*}|y| \leq a{*}|x|$› ‹$2 \leq b$›
pow-len[*of 2 x*] *pow-len*[*of b y*]
mult-le-less-imp-less[*of a b* $|x|$ $|y|$] *not-le*

by *auto*
have $x{\cdot}x{\cdot}y^{@}b = z{\cdot}z{\cdot}z$ **using** ‹$2 \leq a$› *eq* ‹$c{=}3$› ‹$a{=}2$›
by (*simp add*: *numeral-2-eq-2 numeral-3-eq-3*)

— Find words u, v, w
have $|z| < |x{\cdot}x|$
using ‹$|z| + |x| < 3 * |x|$› *add.commute* **by** *auto*
from *ruler-le*[*THEN prefD*, *OF triv-pref*[*of z z*·*z*] - *less-imp-le*[*OF this*]]
obtain w **where** $z{\cdot}w = x{\cdot}x$
using *prefI*[*of x*·*x y*$^{@}$*b z*·*z*·*z*, *unfolded rassoc*, *OF* ‹$x{\cdot}x{\cdot}y^{@}b = z{\cdot}z{\cdot}z$›] **by** *fastforce*

have $|x| < |z|$
using ‹$a = 2$› *lenx* **by** *auto*
from *ruler-le*[*THEN prefD*, *OF* - - *less-imp-le*[*OF this*], *of x*·*x*·*y*$^{@}$*b*, *OF triv-pref*, *unfolded* ‹$x{\cdot}x{\cdot}y^{@}b = z{\cdot}z{\cdot}z$›, *OF triv-pref*]
obtain u :: *'a list* **where** $x{\cdot}u{=}z$
by *blast*
have $u \neq \varepsilon$
using ‹$|x| < |z|$› ‹$x{\cdot}u = z$› **by** *auto*
have $x = u{\cdot}w$ **using** ‹$z{\cdot}w = x{\cdot}x$› ‹$x{\cdot}u = z$› **by** *auto*

have $|x{\cdot}x| < |z{\cdot}z|$ **by** (*simp add*: ‹$|x| < |z|$› *add-less-mono*)
from *ruler-le*[*OF triv-pref*[*of x*·*x y*$^{@}$*b*, *unfolded rassoc* ‹$x{\cdot}x{\cdot}y^{@}b = z{\cdot}z{\cdot}z$›, *unfolded lassoc*] *triv-pref*, *OF less-imp-le*[*OF this*]]
have $z{\cdot}w \leq_p z{\cdot}z$
unfolding ‹$z{\cdot}w = x{\cdot}x$›.
obtain v :: *'a list* **where** $w \cdot v = x$
using *lq-pref*[*of w x*]
pref-prod-pref'[*OF pref-cancel*[*OF* ‹$z{\cdot}w \leq_p z{\cdot}z$›, *folded* ‹$x \cdot u = z$›, *unfolded* ‹$x = u \cdot w$› *rassoc*], *folded* ‹$x = u \cdot w$›] **by** *blast*
have $u{\cdot}w{\cdot}v \neq \varepsilon$
by (*simp add*: ‹$u \neq \varepsilon$›)

— Express x, y and z in terms of u, v and w
hence $z = w{\cdot}v{\cdot}u$
using ‹$w \cdot v = x$› ‹$x \cdot u = z$› **by** *auto*
from ‹$x \cdot x \cdot y^{@}b = z \cdot z \cdot z$›[*unfolded this lassoc*, *folded* ‹$z \cdot w = x \cdot x$›, *unfolded this rassoc*]
have $w{\cdot}v \cdot u{\cdot}w \cdot y^{@}b = w{\cdot}v{\cdot}u{\cdot}w{\cdot}v{\cdot}u{\cdot}w{\cdot}v{\cdot}u$.
hence $y^{@}b = v{\cdot}u{\cdot}w{\cdot}v{\cdot}u$
using *pref-cancel* **by** *auto*

— Double period of uwv
from *period-fac-period*[*of v u* · *w* · *v u* |*y*|, *unfolded rassoc*, *folded this*]
have *period* (*u*·*w*·*v*) |*y*|
using *pow-per*[*of b y*].
have $u{\cdot}w{\cdot}v = x \cdot v$
by (*simp add*: ‹$x = u \cdot w$›)
have $u{\cdot}w{\cdot}v = u{\cdot}\ x$

by (*simp add*: ‹$w \cdot v = x$›)
have $u{\cdot}w{\cdot}v <p\ u \cdot (u{\cdot}w{\cdot}v)$
using ‹$u \cdot w \cdot v = u \cdot x$›[*unfolded* ‹$x = u \cdot w$›] ‹$u \neq \varepsilon$› *triv-pref*[*of* $u \cdot u \cdot w\ v$]
by *force*
have *period* $(u{\cdot}w{\cdot}v)$ $|u|$
using ‹$u{\cdot}w{\cdot}v <p\ u \cdot (u{\cdot}w{\cdot}v)$› **by** *auto*

— Common period d
obtain d::*nat* **where** d=*gcd* $|y|$ $|u|$
by *simp*
have $|y| + |u| \leq |u{\cdot}w{\cdot}v|$ **using** ‹$|y| \leq |x|$› *lenmorph* ‹$u{\cdot}w{\cdot}v = u{\cdot}\ x$›
by *simp*
hence *period* $(u{\cdot}w{\cdot}v)$ d
using ‹*period* $(u \cdot w \cdot v)$ $|u|$› ‹*period* $(u \cdot w \cdot v)$ $|y|$› ‹d = *gcd* $|y|$ $|u|$› *two-periods*
by *blast*

— Divisibility
have $v{\cdot}u{\cdot}z{=}y^{@}b$
by (*simp add*: ‹$y^{@}b = v \cdot u \cdot w \cdot v \cdot u$› ‹$z = w \cdot v \cdot u$›)
have $|u| = |v|$
using ‹$x = u \cdot w$› ‹$w \cdot v = x$› *lenmorph*[*of u w*] *lenmorph*[*of w v*] *add.commute*[*of* $|u|$ $|w|$] *add-left-cancel*
by *simp*
hence d *dvd* $|v|$ **using** *gcd-nat.cobounded1*[*of* $|v|$ $|y|$] *gcd.commute*[*of* $|y|$ $|u|$]
by (*simp add*: ‹d = *gcd* $|y|$ $|u|$›)
have d *dvd* $|u|$
by (*simp add*: ‹d = *gcd* $|y|$ $|u|$›)
have $|z| + |u| + |v| = b{*}|y|$
using *lenarg*[*OF* ‹$v{\cdot}u{\cdot}z{=}y^{@}b$›, *unfolded lenmorph pow-len*] **by** *auto*
from *dvd-add-left-iff*[*OF* ‹d *dvd* $|v|$›, *of* $|z|{+}|u|$, *unfolded this dvd-add-left-iff*[*OF* ‹d *dvd* $|u|$›, *of* $|z|$]]
have d *dvd* $|z|$
using ‹d = *gcd* $|y|$ $|u|$› *dvd-mult* **by** *blast*
from *lenarg*[*OF* ‹$z = w \cdot v \cdot u$›, *unfolded lenmorph pow-len*]
have d *dvd* $|w|$
using ‹d *dvd* $|z|$› ‹d *dvd* $|u|$› ‹d *dvd* $|v|$› **by** (*simp add*: *dvd-add-left-iff*)
hence d *dvd* $|x|$
using ‹d *dvd* $|v|$› ‹$w \cdot v = x$› **by** *force*

— x and y commute
have $x \leq p\ u{\cdot}w{\cdot}v$
by (*simp add*: ‹$x = u \cdot w$›)
have *period* x d
using ‹*period* $(u{\cdot}w{\cdot}v)$ d › ‹$x \leq p\ u \cdot w \cdot v$› *period-pref-period* **by** *blast*
hence $x \in \langle\{take\ d\ x\}\rangle$
using ‹d *dvd* $|x|$› **by** (*rule root-divisor*)

hence *period* u d **using** ‹$x = u \cdot w$› *period-pref-period*
using ‹*period* x d› ‹$u \neq \varepsilon$› **by** *blast*

have *take d x = take d u* **using** ‹*u≠ε*› ‹*x = u · w*› *pref-share-take*
by (*simp add*: ‹*d = gcd |y| |u|*›)
from *root-divisor*[*OF* ‹*period u d*› ‹*d dvd |u|*›, *folded this*]
have $u \in \langle\{take\ d\ x\}\rangle$**.**

hence $z \in \langle\{take\ d\ x\}\rangle$
using ‹*x·u=z*› ‹$x \in \langle\{take\ d\ x\}\rangle$›
by *blast*
from *comm-rootI*[*OF* ‹$u \in \langle\{take\ d\ x\}\rangle$› ‹$x \in \langle\{take\ d\ x\}\rangle$›]
have $y^{@}b \in \langle\{take\ d\ x\}\rangle$
using ‹$u \cdot w \cdot v = u \cdot x$›[*unfolded* ‹$u \cdot x = x \cdot u$› ‹$u \cdot w \cdot v = x \cdot v$› *cancel*]
‹$u \in \langle\{take\ d\ x\}\rangle$› ‹$v \cdot u \cdot z = y\ ^{@}\ b$› ‹$x \in \langle\{take\ d\ x\}\rangle$› ‹$z \in \langle\{take\ d\ x\}\rangle$›
hull-closed
by *metis*
from *comm-rootI*[*OF* ‹$x \in \langle\{take\ d\ x\}\rangle$› *this*]
show $x \cdot y = y \cdot x$
using *comm-pow-roots*[*OF* ‹$0 < a$› ‹$0 < b$›, *of x y*] *comm-pow-comm* **by** *meson*
qed

The main proof is by induction on the length of z. It also uses the reverse symmetry of the equation which is exploited by two interpretations of the locale *LS*. Note also that the case $|x^a| < |y^b|$ is solved by using induction on $|z| + |y^b|$ instead of just on $|z|$.

lemma *Lyndon-Schutzenberger′*:
⟦ $x^{@}a{\cdot}y^{@}b = z^{@}c$; $2 \le a$; $2 \le b$; $2 \le c$ ⟧
⟹ $x{\cdot}y = y{\cdot}x$
proof (*induction* $|z| + b{*}\ |y|$ *arbitrary*: *x y z a b c* *rule*: *less-induct*)
case *less*

have $0 < a$ **and** $0 < b$
using ‹$2 \le a$› ‹$2 \le b$› **by** *auto*

have *LSrev-eq*: *rev y* $^{@}$ *b · rev x* $^{@}$ *a = rev z* $^{@}$ *c*
using ‹$x^{@}a{\cdot}y^{@}b = z^{@}c$›
unfolding *rev-append*[*symmetric*] *rev-pow*[*symmetric*]
by *blast*

have *leneq*: $a * |x| + b{*}|y| = c * |z|$
using *lenarg*[*OF* ‹$x^{@}a{\cdot}y^{@}b = z^{@}c$›] **unfolding** *pow-len lenmorph*.

show $x \cdot y = y \cdot x$
proof (*rule nemp-comm*)
assume $x \neq \varepsilon$ **and** $y \neq \varepsilon$ **and** $x \neq y$
show $x \cdot y = y \cdot x$
proof (*cases* $|x\ ^{@}\ a| < |y\ ^{@}\ b|$)

— WLOG assumption
assume $|x^{@}a| < |y^{@}b|$

have $|rev\ z| + a* |rev\ x| < |z| + b* |y|$ **using** ‹$|x^{@}a| < |y^{@}b|$›
by (*simp add*: *pow-len*)
from *less.hyps*[*OF this LSrev-eq* ‹$2 \leq b$› ‹$2 \leq a$› ‹$2 \leq c$›, *symmetric*]
show $x \cdot y = y \cdot x$
unfolding *rev-append*[*symmetric*] *rev-is-rev-conv* **by** *simp*
next
assume $\neg |x^{@}a| < |y^{@}b|$ **hence** $|y^{@}b| \leq |x^{@}a|$ **by** *force*
— case solved by the periodicity lemma
note *minus* = *Suc-minus2*[*OF* ‹$2 \leq a$›] *Suc-minus2*[*OF* ‹$2 \leq b$›]
consider (*with-periodicity-lemma*)
$|z| + |x| \leq |x\ ^{@}\ Suc(Suc\ (a-2))| \vee |z| + |y| \leq |y\ ^{@}\ Suc(Suc\ (b-2))|$ |
(*without-periodicity-lemma*)
$|x^{@}Suc(Suc\ (a-2))| < |z| + |x|$ **and** $|y^{@}Suc(Suc\ (b-2))| < |z| + |y|$
unfolding *minus*
using *not-le-imp-less* **by** *blast*
thus $x \cdot y = y \cdot x$
proof (*cases*)
case *with-periodicity-lemma*
have $x = \varepsilon \vee rev\ y = \varepsilon \Longrightarrow x \cdot y = y \cdot x$
by *auto*
thus $x \cdot y = y \cdot x$
using *LS-per-lemma-case*[*OF* ‹$x^{@}a \cdot y^{@}b = z^{@}c$› ‹$0 < a$› ‹$0 < b$›]
LS-per-lemma-case[*OF LSrev-eq* ‹$0 < b$› ‹$0 < a$›] *with-periodicity-lemma*[*unfolded minus*]
unfolding *length-rev rev-append*[*symmetric*] *rev-is-rev-conv rev-pow*[*symmetric*]
by *linarith*
next
case *without-periodicity-lemma*
assume *lenx*: $|x^{@}Suc\ (Suc\ (a-2))| < |z| + |x|$ **and** *leny*: $|y^{@}Suc\ (Suc\ (b-2))| < |z| + |y|$
have $Suc\ (Suc\ (a-2)) * |x| + Suc\ (Suc\ (b-2))*|y| < 4 * |z|$
using *lenx leny* **unfolding** *pow-len* **by** *fastforce*
hence $c < 4$ **using** *leneq* **unfolding** *minus* **by** *auto*
consider (*c-is-3*) $c = 3$ | (*c-is-2*) $c = 2$
using ‹$c < 4$› ‹$2 \leq c$› **by** *linarith*
then show $x \cdot y = y \cdot x$
proof (*cases*)
case *c-is-3*
show $x \cdot y = y \cdot x$
using
LS-core-case[*OF* ‹$x^{@}a \cdot y^{@}b = z^{@}c$› ‹$2 \leq a$› ‹$2 \leq b$› ‹$2 \leq c$› ‹$c = 3$› ‹$|y^{@}b| \leq |x^{@}a|$›[*unfolded pow-len*]
- - lenx[*unfolded pow-len minus*] *leny*[*unfolded pow-len minus*]]
‹$x \neq \varepsilon$› ‹$y \neq \varepsilon$›
by *blast*
next
assume $c = 2$
hence *eq2*: $x^{@}a \cdot y^{@}b = z \cdot z$
by (*simp add*: ‹$x^{@}a \cdot y^{@}b = z^{@}c$›)

from *dual-order.trans le-cases*[*of* $|x^{@}a|$ $|z|$ $|z| \leq |x^{@}a|$, *unfolded eq-len-iff*[*OF this*]]
have $|z| \leq |x^{@}a|$
using ‹$|y^{@}b| \leq |x^{@}a|$› **by** *blast*
define a' **where** $a' \equiv a - 1$
have *Suc* $a' = a$ **and** $1 \leq a'$
using ‹$2 \leq a$› **unfolding** a'*-def* **by** *auto*
from *eq2*[*folded* ‹*Suc* $a' = a$›, *unfolded pow-Suc2 rassoc*] *pow-Suc2*[*of* a' x, *unfolded this*, *symmetric*]
have *eq3*: $x \ ^{@} \ a' \cdot x \cdot y \ ^{@} \ b = z \cdot z$ **and** *aa'*:$x \ ^{@} \ a' \cdot x = x \ ^{@} \ a$ **.**
have $|x^{@}a'| < |z|$
using ‹*Suc* $a' = a$› *lenx* **unfolding** *pow-len minus* **by** *fastforce*
hence $|x| < |z|$
using *mult-le-mono*[*of* 1 a' $|z|$ $|x|$, *OF* ‹$1 \leq a'$›, *THEN leD*] **unfolding** *pow-len*
by *linarith*
obtain u w **where** $x^{@}a' \cdot u = z$ **and** $w \cdot y^{@}b = z$
using *eqdE*[*OF eq3*[*unfolded rassoc*] *less-imp-le*[*OF* ‹$|x^{@}a'| < |z|$›], *of thesis*]
eqdE[*OF eq2*[*symmetric*] ‹$|z| \leq |x^{@}a|$›, *of thesis*] **by** *fast*

have $x^{@}a' \cdot x \cdot y^{@}b = x^{@}a' \cdot u \cdot w \cdot y^{@}b$
unfolding *lassoc* ‹$x \ ^{@} \ a' \cdot u = z$› ‹$w \cdot y^{@}b = z$› *aa' eq2 cancel*..
hence $u \cdot w = x$
by *auto*
hence $|w \cdot u| = |x|$
using *swap-len* **by** *blast*

— Induction step: new equation with shorter z
have $w^{@}2 \cdot y^{@}b = (w \cdot u)^{@}a$
unfolding *pow-list-2* **using** ‹$w \cdot y \ ^{@} \ b = z$› ‹$x \ ^{@} \ a' \cdot u = z$› ‹$u \cdot w = x$› *pow-list-slide*[*of* w a' u, *unfolded* ‹*Suc* $a' = a$›] **by** *simp*
from *less.hyps*[*OF - this -* ‹$2 \leq b$› ‹$2 \leq a$›, *unfolded* ‹$|w \cdot u| = |x|$›]
have $y \cdot w = w \cdot y$
using ‹$|x| < |z|$› **by** *force*

have $y \cdot z = z \cdot y$
unfolding ‹$w \cdot y^{@}b = z$›[*symmetric*] *lassoc* ‹$y \cdot w = w \cdot y$›
by (*simp add*: *pow-comm*)
hence $z^{@}c \cdot y^{@}b = y^{@}b \cdot z^{@}c$
by (*simp add*: *comm-pows-comm*)
from *this*[*folded* ‹$x^{@}a \cdot y^{@}b = z^{@}c$›, *unfolded lassoc*]
have $x^{@}a \cdot y^{@}b = y^{@}b \cdot x^{@}a$
using *cancel-right* **by** *blast*
from *this*[*unfolded comm-pow-roots*[*OF* ‹$0 < a$› ‹$0 < b$›]]
show $x \cdot y = y \cdot x$**.**
qed
qed
qed

qed
qed

theorem *Lyndon-Schutzenberger*:
assumes $x^{@}a{\cdot}y^{@}b = z^{@}c$ **and** $2 \leq a$ **and** $2 \leq b$ **and** $2 \leq c$
shows $x{\cdot}y = y{\cdot}x$ **and** $x{\cdot}z = z{\cdot}x$ **and** $y{\cdot}z = z{\cdot}y$
proof−
show $x \cdot y = y \cdot x$
using *Lyndon-Schutzenberger'*[*OF assms*].
have $0 < c$ **and** $0 < b$
using ‹$2 \leq c$› ‹$2 \leq b$› **by** *auto*
have $x \cdot x^{@}a \cdot y^{@}b = x^{@}a \cdot y^{@}b \cdot x$ **and** $y \cdot x^{@}a \cdot y^{@}b = x^{@}a \cdot y^{@}b \cdot y$
unfolding *comm-pow-comm*[*OF* ‹$x \cdot y = y \cdot x$›[*symmetric*], *of b*]
unfolding *lassoc pow-comm comm-pow-comm*[*OF* ‹$x \cdot y = y \cdot x$›, *symmetric*, *of a*] **by** *blast*+
thus $x{\cdot}z = z{\cdot}x$ **and** $y{\cdot}z = z{\cdot}y$
using *comm-drop-exp*[*OF* ‹$0 < c$›] **unfolding** *lassoc* ‹$x^{@}a{\cdot}y^{@}b = z^{@}c$› **by** *metis*+
qed
hide-fact *Lyndon-Schutzenberger' LS-core-case*

6.2.1 Some alternative formulations.

lemma *Lyndon-Schutzenberger-conjug*: **assumes** $u \sim v$ **and** $\neg$ *primitive* $(u \cdot v)$
shows $u \cdot v = v \cdot u$
proof−
obtain $r\ s$ **where** $u = r \cdot s$ **and** $v = s \cdot r$
using ‹$u \sim v$› **by** *blast*
have $u \cdot v \sim r^{@}2 \cdot s^{@}2$
using *conjugI'*[*of* $r \cdot s$ $s\ r$] **unfolding** ‹$u = r \cdot s$› ‹$v = s \cdot r$› *pow-list-2 rassoc*.
hence $\neg$ *primitive* $(r^{@}2 \cdot s^{@}2)$
using ‹$\neg$ *primitive* $(u \cdot v)$› *prim-conjug* **by** *auto*
from *not-prim-primroot-expE*[*OF this*, *of* $r \cdot s = s \cdot r$]
have $r \cdot s = s \cdot r$
using *Lyndon-Schutzenberger*(*1*)[*of 2 r 2 s*, *OF - order.refl order.refl*] **by** *metis*
thus $u \cdot v = v \cdot u$
using ‹$u = r \cdot s$› ‹$v = s \cdot r$› **by** *presburger*
qed

lemma *Lyndon-Schutzenberger-prim*: **assumes** $\neg$ *primitive* x **and** $\neg$ *primitive* y **and** $\neg$ *primitive* $(x \cdot y)$
shows $x \cdot y = y \cdot x$
proof (*rule nemp-comm*)
assume $x \neq \varepsilon$ **and** $y \neq \varepsilon$
from *not-prim-primroot-expE*[*OF* ‹$\neg$ *primitive* y›]
obtain m **where** $\varrho\ y^{@}m = y$ **and** $2 \leq m$.
from *not-prim-primroot-expE*[*OF* ‹$\neg$ *primitive* x›]
obtain k **where** $\varrho\ x^{@}k = x$ **and** $2 \leq k$.
from *not-prim-primroot-expE*[*OF* ‹$\neg$ *primitive* $(x \cdot y)$›]

obtain l **where** $\varrho(x \cdot y)^{@} l = x \cdot y$ **and** $2 \leq l$**.**
from *Lyndon-Schutzenberger(1)*[*of* k ϱ x m ϱ y l ϱ $(x \cdot y)$,
OF - ‹$2 \leq k$› ‹$2 \leq m$› ‹$2 \leq l$›]
show $x \cdot y = y \cdot x$
unfolding ‹ϱ $y^{@} m = y$› ‹ϱ $x^{@} k = x$› ‹$\varrho(x \cdot y)^{@} l = x \cdot y$›
comm-primroot-conv'[*of* x y] **by** *blast*
qed

lemma *Lyndon-Schutzenberger-rotate*: **assumes** $x^{@} c = r\ ^{@}\ k \cdot u^{@} b \cdot r\ ^{@}\ k'$
and $2 \leq b$ **and** $2 \leq c$ **and** $0 < k$ **and** $0 < k'$
shows $u \cdot r = r \cdot u$
proof(*rule comm-drop-exps*)
show $u^{@} b \cdot r^{@}(k' + k) = r^{@}(k' + k) \cdot u^{@} b$
proof(*rule Lyndon-Schutzenberger-prim*)
have $2 \leq (k' + k)$
using ‹$0 < k$› ‹$0 < k'$› **by** *simp*
from *pow-nemp-imprim*[*OF* ‹$2 \leq b$›] *pow-nemp-imprim*[*OF this*]
show ¬ *primitive* $(u^{@} b)$ **and** ¬ *primitive* $(r\ ^{@}\ (k' + k))$
unfolding *Suc-minus2*[*OF* ‹$2 \leq b$›]**.**
from *pow-nemp-imprim*[*OF* ‹$2 \leq c$›]
have ¬ *primitive* $(r\ ^{@}\ k \cdot u^{@} b \cdot r\ ^{@}\ k')$
unfolding *assms(1)*[*symmetric*]**.**
from *this*[*unfolded conjug-prim-iff*[*OF conjugI'*[*of* $r\ ^{@}\ k\ u\ ^{@}\ b \cdot r\ ^{@}\ k'$]] *rassoc*]
show ¬ *primitive* $(u\ ^{@}\ b \cdot r\ ^{@}\ (k' + k))$
unfolding *pow-add*[*symmetric*] **by** *force*
qed
qed (*use assms* **in** *force*)+

6.3 Parametric solution of the equation $x\ ^{@}\ j \cdot y\ ^{@}\ k = z\ ^{@}\ l$

6.3.1 Auxiliary lemmas

lemma *xjy-imprim-len*: **assumes** $x \cdot y \neq y \cdot x$ **and** *eq*: $x^{@} j \cdot y = z^{@} l$ **and** $2 \leq j$ **and** $2 \leq l$
shows $|x^{@} j| < |y| + 2{*}|x|$ **and** $|z| < |x| + |y|$ **and** $|x| < |z|$ **and** $|x^{@} j| < |z| + |x|$
proof−
define j' **where** $j' \equiv j - 2$
have $0 < j$ $j = Suc(Suc\ j')$
unfolding *j'-def* **using** ‹$2 \leq j$› **by** *force*+
from *LS-per-lemma-case*[*of* - - 1, *unfolded pow-list-1*, *OF eq* ‹$0 < j$›]
show $|x^{@} j| < |z| + |x|$
using ‹$x \cdot y \neq y \cdot x$› **by** *linarith*
from *lenarg*[*OF eq*, *unfolded lenmorph*, *unfolded pow-len*]
add-less-mono1[*OF this*, *of* $|y|$, *unfolded pow-len*]
show $|z| < |x| + |y|$
using *mult-le-mono1*[*OF* ‹$2 \leq l$›, *unfolded mult-2*, *of* $|z|$] **by** *force*
with ‹$|x^{@} j| < |z| + |x|$›

show $|x| < |z|$ **and** $|x^{@}j| < |y| + 2*|x|$
unfolding ‹$j = Suc\ (Suc\ j')$› *pow-Suc lenmorph mult-2* **by** *linarith+*
qed

lemma *case-j1k1*: **assumes**
eq: $x{\cdot}y = z^{@}l$ **and**
non-comm: $x \cdot y \neq y \cdot x$ **and**
l-min: $2 \leq l$
obtains $r\ q\ m\ n$ **where**
$x = (r{\cdot}q)^{@}m{\cdot}r$ **and**
$y = q{\cdot}\ (r \cdot q)^{@}n$ **and**
$z = r{\cdot}q$ **and**
$l = m + n + 1$ **and** $r{\cdot}q \neq q{\cdot}r$ **and** $|x| + |y| \geq 4$
proof−
have $0 < l\ y \neq \varepsilon$
using *l-min non-comm* **by** *force+*
from *split-pow*[*OF eq this*]
obtain $r\ q\ m\ n$ **where**
x: $x = (r \cdot q)\ ^{@}\ m \cdot r$ **and**
y: $y = (q \cdot r)^{@}\ n \cdot q$ **and**
z: $z = r \cdot q$ **and**
l: $l = m + n + 1$**.**
from *non-comm*[*unfolded x y*]
have $r \cdot q \neq q \cdot r$
unfolding *shifts*
unfolding *lassoc pow-add*[*symmetric*] *pow-Suc*[*symmetric*] *add.commute*[*of m*]
by *force*
hence $r \neq \varepsilon$ **and** $q \neq \varepsilon$
by *blast+*
have $2 \leq |r \cdot q|$
using *nemp-len*[*OF* ‹$r \neq \varepsilon$›] *nemp-len*[*OF* ‹$q \neq \varepsilon$›]
unfolding *lenmorph* **by** *linarith*
have $|x| + |y| \geq 4$
unfolding *x y lenmorph*[*symmetric*] *shifts*
unfolding *pow-add*[*symmetric*] *lassoc lenmorph*[*of r · q*]
mult-Suc[*symmetric*] *pow-len Suc-eq-plus1 l*[*symmetric*]
using *mult-le-mono*[*OF* ‹$2 \leq l$› ‹$2 \leq |r \cdot q|$›]
by *presburger*
from *that*[*OF x y*[*unfolded shift-pow*] *z l* ‹$r \cdot q \neq q \cdot r$› *this*]
show *thesis***.**
qed

6.3.2 x is longer

We set up a locale representing the Lyndon-Schützenberger Equation with relaxed exponents and a length assumption breaking the symmetry.

locale *LS-len-le* = *binary-code x y* **for** *x y* +
fixes *j k l z*
assumes

y-le-x: $|y| \leq |x|$
and *eq*: $x^{@}j \cdot y^{@}k = z^{@}l$
and *l-min*: $2 \leq l$
and *j-min*: $1 \leq j$
and *k-min*: $1 \leq k$
begin

lemma *jk-small*: **obtains** $j = 1 \mid k = 1$
using *Lyndon-Schutzenberger*(*1*)[*OF eq - - l-min*]
le-neq-implies-less[*OF j-min*]
le-neq-implies-less[*OF k-min*]
non-comm
unfolding *One-less-Two-le-iff*
by *blast*

case $2 \leq j$

lemma *case-j2k1*: **assumes** $2 \leq j$ $k = 1$
obtains r q t **where**
$(r \cdot q)\ ^{@}\ t \cdot r = x$ **and**
$q \cdot r \cdot r \cdot q\ =\ y$ **and**
$(r \cdot q)\ ^{@}\ t \cdot r \cdot r \cdot q = z$ **and** $2 \leq t$
$j = 2$ **and** $l = 2$ **and** $r{\cdot}q \neq q{\cdot}r$ **and**
primitive x **and** *primitive y*
proof–
note $eq' = eq$[*unfolded* ‹$k = 1$› *pow-list-1*]
note *xjy-imprim-len*[*OF non-comm eq*[*unfolded* ‹$k = 1$› *pow-list-1*] ‹$2 \leq j$› *l-min*]

obtain j' **where** $j = Suc\ (Suc\ j')$
using ‹$2 \leq j$› **using** *at-least2-Suc* **by** *metis*
hence $0 < j$ **by** *blast*
from *lenarg*[*OF eq'*, *unfolded lenmorph*, *unfolded pow-len*]
add-less-mono1[*OF* ‹$|x^{@}j| < |z| + |x|$›, *of* $|y|$, *unfolded pow-len*]
have $l{*}|z| < 3{*}|z|$
using ‹$|x| < |z|$› *y-le-x* **by** *linarith*
hence $l = 2$
using *l-min* **by** *simp*
from ‹$|x\ ^{@}\ j| < |z| + |x|$› *add-less-mono1*[*OF* ‹$|z| < |x| + |y|$›, *of* $|x|$] *y-le-x*
have $j' * |x| < |x|$
unfolding ‹$j = Suc\ (Suc\ j')$› *pow-Suc lenmorph pow-len* **by** *linarith*
hence $j = 2$
using ‹$j = Suc\ (Suc\ j')$› **by** *simp*

note *eq*[*unfolded* ‹$k = 1$› *pow-list-1* ‹$j = 2$› ‹$l = 2$› *pow-list-2 rassoc*]
from *eqd*[*OF this less-imp-le*[*OF* ‹$|x| < |z|$›]]
obtain p **where** $x \cdot p = z$ **and** $p \cdot z = x \cdot y$
by *blast*
from *eqd*[*OF* ‹$p \cdot z = x \cdot y$›[*folded* ‹$x \cdot p = z$›, *unfolded lassoc*, *symmetric*]]
obtain s **where** $x \cdot s = p \cdot x$ **and** $s \cdot p = y$

by *auto*
have $p \neq \varepsilon$
using ‹$x \cdot p = z$› ‹$|x| < |z|$› **by** *fastforce*
have $s \neq \varepsilon$
using ‹$p \neq \varepsilon$› ‹$x \cdot s = p \cdot x$› **by** *force*
from *conjug-eqE*[*OF* ‹$x \cdot s = p \cdot x$›[*symmetric*] ‹$p \neq \varepsilon$›]
obtain $r\ q\ t$ **where** $r \cdot q = p$ **and** $q \cdot r = s$ **and** $(r \cdot q)^{@}t{\cdot}r = x$ **and** $q \neq \varepsilon$**.**
note ‹$s \cdot p = y$›[*folded* ‹$q \cdot r = s$› ‹$r \cdot q = p$›, *unfolded rassoc*]
from *y-le-x*[*folded this* ‹$(r \cdot q)^{@}t{\cdot}r = x$›, *unfolded lenmorph pow-len*] *nemp-len*[*OF* ‹$q \neq \varepsilon$›]
add-le-mono1[*OF mult-le-mono1*[*of t 1* $|r| + |q|$, *unfolded mult-1*], *of* $|r|$]
have $2 \leq t$
by *linarith*
from ‹$p \cdot z = x \cdot y$›[*folded* ‹$q \cdot r \cdot r \cdot q = y$› ‹$(r \cdot q)^{@}t{\cdot}r = x$› ‹$r \cdot q = p$›, *symmetric*]
have *z*: $(r \cdot q)\ ^{@}\ t \cdot r \cdot r \cdot q = z$
by *comparison*

— y is primitive due to the Lyndon-Schutzenberger

from *comm-drop-exp*[*OF* ‹$0 < j$›, *of y x j*, *unfolded eq'*]
have *primitive y*
using *Lyndon-Schutzenberger-prim*[*OF pow-nemp-imprim*[*OF* ‹$2 \leq j$›], *of y x*, *unfolded eq'*, *OF - pow-nemp-imprim*[*OF l-min*]] *non-comm* **by** *argo*

hence $q \cdot r \neq r \cdot q$
using ‹$p \neq \varepsilon$› ‹$q \cdot r = s$› ‹$r \cdot q = p$› ‹$s \cdot p = y$› *comm-not-prim*[*OF* ‹$s \neq \varepsilon$› ‹$p \neq \varepsilon$›] **by** *argo*

— primitivity of x using $[\![?u \leq p\ ?p \cdot ?u;\ ?p \neq \varepsilon;\ 2 * |?p| \leq |?u|]\!] \Longrightarrow$ *primitive ?u* $= (?u \cdot ?p \neq ?p \cdot ?u)$

thm *per-le-prim-iff*[*of x p*]
have $x \leq p\ p \cdot x$
unfolding ‹$(r \cdot q)^{@}t{\cdot}r = x$›[*symmetric*] ‹$r \cdot q = p$›[*symmetric*]
by *comparison*
have $2{*}|p| \leq |x|$
unfolding ‹$(r \cdot q)^{@}t{\cdot}r = x$›[*symmetric*] ‹$r \cdot q = p$›[*symmetric*] *lenmorph pow-len*
using *mult-le-mono1*[*OF* ‹$2 \leq t$›, *of* $(|r| + |q|)$] **by** *linarith*
have [*symmetric*]: $p \cdot x \neq x \cdot p$
unfolding ‹$(r \cdot q)^{@}t{\cdot}r = x$›[*symmetric*] ‹$r \cdot q = p$›[*symmetric*] *lassoc pow-comm*[*symmetric*]
unfolding *rassoc cancel* **by** *fact*
with *per-le-prim-iff*[*OF* ‹$x \leq p\ p \cdot x$› ‹$p \neq \varepsilon$› ‹ $2 * |p| \leq |x|$›]
have *primitive x*
by *blast*

from *that*[*OF* ‹$(r \cdot q)^{@}t{\cdot}r = x$› ‹$q \cdot r \cdot r \cdot q = y$› *z* ‹$2 \leq t$›
‹$j = 2$› ‹$l = 2$› ‹$q{\cdot}r \neq r{\cdot}q$›[*symmetric*] ‹*primitive x*› ‹*primitive y*›]
show *thesis***.**

qed

case $j = 1$

lemma *case-j1k2-primitive*: **assumes** $j = 1$ $2 \leq k$
shows *primitive x*
using *Lyndon-Schutzenberger-prim*[*OF - pow-nemp-imprim*
pow-nemp-imprim[*OF l-min*, *of z*, *folded eq*], *OF - ‹2 ≤ k›*]
comm-pow-roots[*of j k x y*] *k-min non-comm*
unfolding *‹j = 1› pow-list-1*
by *linarith*

lemma *case-j1k2-a*: **assumes** $j = 1$ $2 \leq k$ $z \leq_s y^{@}k$
obtains r q t **where**
$x = ((q \cdot r) \cdot (r \cdot (q \cdot r)^{@} t)^{@} (k - 1))^{@} (l - 2) \cdot$
$(((q \cdot r) \cdot (r \cdot (q \cdot r)^{@} t)^{@} (k - 2)) \cdot r) \cdot q$ **and**
$y = r \cdot (q \cdot r)^{@} t$ **and**
$z = (q \cdot r) \cdot (r \cdot (q \cdot r)^{@} t)^{@} (k - 1)$ **and** $‹0 < t›$ **and** $r \cdot q \neq q \cdot r$
proof−
have $z \neq \varepsilon$
using *assms*(1) *bin-fst-nemp eq* **by** *force*
have $0 < k$ $0 < k - 1$
using $‹2 \leq k›$ **by** *linarith*+
have $0 < l$ $0 < l - 1$
using *l-min* **by** *linarith*+

from *LS-per-lemma-case*[*reversed*, *OF eq ‹0 < k›*, *unfolded ‹j = 1›*]
have *perlem*: $|y^{@}k| < |z| + |y|$
using *non-comm*
by *linarith*

obtain v **where** $y^{@}k = v \cdot z$
using $‹z \leq_s y^{@}k›$ *suffix-def* **by** *blast*
have $|v| < |y|$
using *perlem*[*unfolded lenarg*[*OF ‹y^@k = v·z›*] *lenmorph*]
by *simp*
have $v <_p y$
using *prefI*[*OF ‹y^@k = v·z›*[*symmetric*]]
unfolding *pow-pos*[*OF ‹0 < k›*]
using $‹|v| < |y|›$ *less-or-eq-imp-le* **by** *blast*
obtain u **where** $v \cdot u = y$ $u \neq \varepsilon$
using $‹v <_p y›$ *spref-exE* **by** *blast*

have $z = u \cdot y^{@}(k-1)$
using *‹y^@k = v·z›*[*unfolded pow-pos*[*OF ‹0 < k›*],
folded ‹v · u = y›, *unfolded rassoc cancel*,
unfolded ‹v · u = y›, *symmetric*].

note *eq*[*unfolded pow-pos2*[*OF ‹0 < l›*] *‹y^@k = v·z›* *lassoc cancel-right*

```
    ‹j = 1› pow-list-1]

obtain u' where u'·v = y
proof–
  have v ≤s z@(l−1)
    using ‹x · v = z @ (l − 1)› by blast
  moreover have y ≤s z@(l−1)
    unfolding ‹z = u·y@(k−1)› pow-pos2[OF ‹0 < k − 1›]
      pow-pos2[OF ‹0 < l − 1›] lassoc
    by blast
  ultimately have v ≤s y
    using order-less-imp-le[OF ‹|v| < |y|›] suffix-length-suffix by blast
  thus thesis
    using sufD that by blast
qed
hence u' ≠ ε
  using ‹v <p y› by force

from conjugation[OF ‹u'·v = y›[folded ‹v·u = y›] ‹u' ≠ ε›]
obtain r q t where r · q = u' q · r = u (r · q) @ t · r = v
  by blast

have y: y = r · (q · r) @ Suc t
  using ‹u' · v = y›[symmetric, folded ‹(r · q) @ t · r = v› ‹r · q = u'›]
  unfolding rassoc pow-list-slide[symmetric].
have z: z = (q · r) · (r · (q · r) @ Suc t) @ (k − 1)
  using ‹q · r = u› ‹z = u · y @ (k − 1)› y by blast

let ?x = ((q · r) · (r · (q · r) @ Suc t) @ (k − 1)) @ (l − 2) ·
    (((q · r) · (r · (q · r) @ Suc t) @ (k − 2)) · r) · q
have ?x · v = z @ (l − 1)
  unfolding z ‹(r · q) @ t · r = v›[symmetric] pow-pos2[OF ‹0 < k − 1›]
  pow-pos2[OF ‹0 < l − 1›] diff-diff-left nat-1-add-1
  by (simp only: shifts)
from ‹x · v = z @ (l − 1)›[folded this]
have x: x = ?x
  by blast

have z·y ≠ y·z
  using non-comm
  using comm-pow-comm[of z y l, folded eq,
      unfolded rassoc pow-comm, unfolded lassoc cancel-right
      ‹j = 1› pow-list-1]
  by blast
hence r·q ≠ q·r
  unfolding ‹q · r = u› ‹r · q = u'› ‹u'· v  = y›[symmetric]
    ‹z = u · y @ (k − 1)› pow-pos2[OF ‹0 < k›] rassoc
    ‹y @ k = v · z›[unfolded ‹u' · v = y›[symmetric]
      ‹z = u · y @ (k − 1)›, symmetric] cancel-right..
```

show *thesis*
using *that*[*OF x y z - ‹r·q ≠ q·r›*] **by** *blast*
qed

lemma *case-j1k2-b*: **assumes** $j = 1$ $2 \leq k$ $y^{@}k <_s z$
obtains q **where**
$x = (q{\cdot}y^{@}k)^{@}(l{-}1){\cdot}q$ **and**
$z = q{\cdot}y^{@}k$ **and**
$q{\cdot}y \neq y{\cdot}q$
proof–
obtain q **where** $z = q{\cdot}y^{@}k$ $q \neq \varepsilon$
using *ssufD*[*OF ‹y*$^{@}$*k <s z›*]
unfolding *suffix-def*
by *blast*
have $0 < l$ **using** *l-min* **by** *linarith*
have $x = (q{\cdot}y^{@}k)^{@}(l{-}1){\cdot}q$
using *eq*[*unfolded pow-pos2*[*OF ‹0 < l›*] *‹j = 1› pow-list-1*,
unfolded ‹z = q·y$^{@}$*k› lassoc cancel-right*].
have $q{\cdot}y \neq y{\cdot}q$
using
comm-trans[*OF - - ‹q ≠ ε›*, *of y x*] *conjug-pow*[*of y q y k*, *symmetric*]
conjug-pow[*of q · y* $^{@}$ *k q q · y* $^{@}$ *k l−1*] *non-comm*
unfolding *append-same-eq*[*symmetric*, *of ‹(q · y* $^{@}$ *k)* $^{@}$ *(l − 1) · q› ‹q · (q · y* $^{@}$ *k)* $^{@}$ *(l − 1)› q*]
unfolding *‹x = (q · y* $^{@}$ *k)* $^{@}$ *(l − 1) · q› rassoc*
by *argo*
show *?thesis*
using *‹x = (q · y* $^{@}$ *k)* $^{@}$ *(l − 1) · q› ‹z = q · y* $^{@}$ *k› ‹q·y ≠ y·q› that* **by** *blast*
qed

6.3.3 Putting things together

lemma *solution-cases*: **obtains**
$j = 2$ $k = 1$ |
$j = 1$ $2 \leq k$ $z <_s y^{@}k$ |
$j = 1$ $2 \leq k$ $y^{@}k <_s z$ |
$j = 1$ $k = 1$
proof–
have $0 < l$ $0 < l{-}1$
using *l-min* **by** *linarith*+
have $0 < k$
using *k-min* **by** *linarith*
have $0 < j$
using *j-min* **by** *linarith*
have $z \neq \varepsilon$
using *eq nemp-pow-nemp*[*of l z*] *bin-fst-nemp*[*folded pow-list-Nil-iff-Nil*[*OF ‹0 < j›*, *of x*], *THEN pref-nemp*]
by *force*
have $z \neq y^{@}k$

proof
assume $z = y^{@}k$
with *eq*[*unfolded pow-pos2*[*OF* ‹*0* < *l*›], *folded this*, *unfolded cancel-right*]
have $x^{@}j \cdot y^{@}k = y^{@}k \cdot x^{@}j$
using *pow-comm* **by** *auto*
from *comm-drop-exps*[*OF* ‹*0* < *j*› ‹*0* < *k*› *this*]
show *False*
using *non-comm* **by** *blast*
qed
consider
$2 \le j\ k = 1$ |
$j = 1\ 2 \le k$ |
$j = 1\ k = 1$
using *jk-small j-min k-min le-neq-implies-less*
unfolding *One-less-Two-le-iff*[*symmetric*]
by *metis*
moreover consider $z <_s y^{@}k \mid y^{@}k <_s z$
using *suffix-order.less-le*
triv-suf[*of* $y^{@}k\ x^{@}j$, *unfolded eq*, *THEN suf-prod-root*,
THEN ruler-suf⁀] ‹$z \neq y^{@}k$›
by *blast*
moreover consider $j = 1 \mid j = 2$
using *case-j2k1*[*of thesis*] *calculation*(*1*) **by** *blast*
ultimately show *?thesis*
using *that*
by *metis*
qed

theorem *parametric-solutionE*: **obtains**
— case $x \cdot y$
$r\ q\ m\ n$ **where**
$x = (r{\cdot}q)^{@}m{\cdot}r$ **and**
$y = q{\cdot}(r{\cdot}q)^{@}n$ **and**
$z = r{\cdot}q$ **and**
$l = m + n + 1$ **and** $r{\cdot}q \neq q{\cdot}r$
|
— case $x \cdot y\ ^{@}\ k$ with $2 \le k$ and $<_s\ z\ (y\ ^{@}\ k)$
$r\ q\ t$ **where**
$x = ((q \cdot r) \cdot (r \cdot (q \cdot r)\ ^{@}\ t)\ ^{@}\ (k - 1))\ ^{@}\ (l - 2) \cdot$
$(((q \cdot r) \cdot (r \cdot (q \cdot r)\ ^{@}\ t)\ ^{@}\ (k - 2)) \cdot r) \cdot q$ **and**
$y = r \cdot (q \cdot r)\ ^{@}\ t$ **and**
$z = (q \cdot r) \cdot (r \cdot (q \cdot r)\ ^{@}\ t)\ ^{@}\ (k - 1)$ **and**
$0 < t$ **and** $r{\cdot}q \neq q{\cdot}r$
|
— case $x \cdot y\ ^{@}\ k$ with $2 \le k$ and $<_s\ (y\ ^{@}\ k)\ z$
q **where**
$x = (q{\cdot}y^{@}k)^{@}(l{-}1){\cdot}q$ **and**
$z = q{\cdot}y^{@}k$ **and**
$q{\cdot}y \neq y{\cdot}q$

```
|
  — case x @ j · y with 2 ≤ j
  r q t where
  x = (r · q) @ t · r and
  y  =  q · r · r · q and
  z = (r · q) @ t · r · r · q and
  j = 2 and l = 2 and 2 ≤ t and r·q ≠ q·r and
  primitive x and primitive y
proof–
  show ?thesis
    using solution-cases
  proof(cases)
    case 1
    from case-j2k1[OF - ‹k = 1›, of thesis] ‹j = 2›
    show ?thesis
      using that(4) by blast
  next
    case 2
    from case-j1k2-a[OF this(1–2) ssufD1[OF this(3)], of thesis]
    show thesis
      using that(2)
      by blast
  next
    case 3
    from case-j1k2-b[OF this, of thesis]
    show ?thesis
      using that(3) by blast
  next
    case 4
    from case-j1k1[OF eq[unfolded ‹k = 1› ‹j = 1› pow-list-1] non-comm l-min, of
thesis]
    show thesis
      using that(1).
  qed
qed

end
```

Using the solution from locale *LS-len-le*, the following theorem gives the full characterization of the equation in question:

$$x^i y^j = z^\ell$$

```
theorem LS-parametric-solution:
  assumes y-le-x: |y| ≤ |x|
    and j-min: 1 ≤ j  and k-min: 1 ≤ k and l-min: 2 ≤ l
  shows
    x@j · y@k = z@l
  ⟷
```

(∃ r m n t.
x = r^@m ∧ y = r^@n ∧ z = r^@t ∧ m∗j+n∗k=t∗l) — Case A: x,y is not a code
∨ (j = 1 ∧ k = 1) ∧
(∃ r q m n.
x = (r·q)^@m·r ∧ y = q·(r·q)^@n ∧ z = r·q ∧ m + n + 1 = l ∧ r·q ≠ q·r) — Case B
∨ (j = 1 ∧ 2 ≤ k) ∧
(∃ r q.
x = (q·r^@k)^@(l−1)·q ∧ y = r ∧ z = q·r^@k ∧ r·q ≠ q·r) — Case C
∨ (j = 1 ∧ 2 ≤ k) ∧
(∃ r q t. 0 < t ∧
x = ((q · r) · (r · (q · r) ^@ t) ^@ (k − 1)) ^@ (l − 2)·(((q · r) ·
(r · (q · r) ^@ t) ^@ (k − 2)) · r) · q
∧ y = r · (q · r) ^@ t
∧ z = (q · r) · (r · (q · r) ^@ t) ^@ (k − 1)
∧ r·q ≠ q·r) — Case D
∨ (j = 2 ∧ k = 1 ∧ l = 2) ∧
(∃ r q t. 2 ≤ t ∧
x = (r · q) ^@ t · r ∧ y = q · r · r · q
∧ z = (r · q) ^@ t · r · r · q ∧ r·q ≠ q·r) — Case E

(**is** *?eq* =
(*?sol-per* ∨ (*?cond-j1k1* ∧ *?sol-j1k1*) ∨
(*?cond-j1k2* ∧ *?sol-j1k2-b*) ∨
(*?cond-j1k2* ∧ *?sol-j1k2-a*) ∨
(*?cond-j2k1l2* ∧ *?sol-j2k1l2*)))
proof(*rule iffI*)
assume *eq*: x ^@ j · y ^@ k = z ^@ l
show
(*?sol-per* ∨ (*?cond-j1k1* ∧ *?sol-j1k1*) ∨
(*?cond-j1k2* ∧ *?sol-j1k2-b*) ∨
(*?cond-j1k2* ∧ *?sol-j1k2-a*) ∨
(*?cond-j2k1l2* ∧ *?sol-j2k1l2*))
proof(*cases*)
assume *x·y = y·x*
from *comm-primrootE'*[*OF this*]
obtain *r m n* **where** *x = r ^@ m y = r ^@ n primitive r*.

note *eqs = eq*[*unfolded this*, *folded pow-mult*, *folded pow-add*[*of m∗j n∗k r*], *symmetric*]
obtain *t* **where** *z = r ^@ t*
using *l-min pow-list-comm-comm*[*OF - eqs*,
THEN prim-comm-exp[*OF ‹primitive r›*]]
by *auto*
from *eqs*[*unfolded this*, *folded pow-mult*, *symmetric*]
have *m ∗ j + n ∗ k = t ∗ l*
unfolding *prim-nemp*[*OF ‹primitive r›*, *THEN eq-pow-exp*].
hence *?sol-per*

```
    using ‹x = r @ m› ‹y = r @ n› ‹z = r @ t› by blast
  thus ?thesis
    by blast
 next
  assume x·y ≠ y·x
  interpret LS-len-le x y j k l z
    using ‹x @ j · y @ k = z @ l› ‹x·y ≠ y·x› j-min k-min l-min y-le-x
    by(unfold-locales)

  show ?thesis
    using solution-cases
  proof(cases)
    case 1
      from case-j2k1[OF less-or-eq-imp-le[of 2 j] ‹k = 1›, OF disjI2, OF ‹j =
2›[symmetric], of ?sol-j2k1l2 ∧ l = 2]
    have ?sol-j2k1l2 and l = 2
      by auto
    thus ?thesis
      using ‹k = 1› ‹j = 2› by blast
  next
    case 2
    have ?sol-j1k2-a
    using case-j1k2-a[OF ‹j = 1› ‹2 ≤ k› ssufD1[OF ‹z <s y @ k›], of ?sol-j1k2-a]
      unfolding Suc-eq-plus1
      by blast
    thus ?thesis
      using ‹j = 1› ‹2 ≤ k› by blast
  next
    case 3
    with case-j1k2-b[OF this, of ?sol-j1k2-b]
    have ?sol-j1k2-b by metis
    thus ?thesis
      using ‹j = 1› ‹2 ≤ k› by blast
  next
    case 4
     with case-j1k1[OF eq[unfolded ‹k = 1› ‹j = 1› pow-list-1] non-comm l-min,
of ?sol-j1k1]
    have?sol-j1k1
      unfolding Suc-eq-plus1 shift-pow
      by blast
    thus ?thesis
      using ‹j = 1› ‹k = 1› by blast
  qed
 qed
next
 have l ≠ 0 l − 1 ≠ 0
  using l-min by auto
 have k ≠ 0 using k-min by auto
 have j ≠ 0 using j-min by auto
```

assume (*?sol-per* ∨ (*?cond-j1k1* ∧ *?sol-j1k1*) ∨
(*?cond-j1k2* ∧ *?sol-j1k2-b*) ∨
(*?cond-j1k2* ∧ *?sol-j1k2-a*) ∨
(*?cond-j2k1l2* ∧ *?sol-j2k1l2*))
then show *?eq*
proof(*elim disjE conjE exE*)
fix *r m n t*
assume *sol*: $x = r \,^{@}\, m\ y = r \,^{@}\, n\ z = r \,^{@}\, t$
and $m * j + n * k = t * l$
show *?thesis*
unfolding *sol*
unfolding *pow-mult*[*symmetric*] *pow-add*[*symmetric*]
unfolding ‹$m * j + n * k = t * l$›**..**
next
fix *r q m n*
assume $j = 1\ k = 1$ **and** *sol*: $x = (r{\cdot}q)^{@}m{\cdot}r$
$y = q{\cdot}(r{\cdot}q)^{@}n\ z = r{\cdot}q$
and $m + n + 1 = l$
hence *Suc* $(m{+}n) = l$
by *simp*
show *?thesis*
unfolding *sol*
unfolding ‹$j = 1$› ‹$k = 1$› ‹*Suc* $(m + n) = l$›[*symmetric*] *pow-list-1*
unfolding *lassoc pow-Suc pow-add*
unfolding *pow-comm*[*of m*, *symmetric*] *lassoc***..**
next
fix *r q*
assume $j = 1\ 2 \le k$ **and** *sol*: $x = (q \cdot r \,^{@}\, k) \,^{@}\, (l - 1) \cdot q\ y = r\ z = q \cdot r \,^{@}\, k$
have $0 < l$
using ‹$2 \le l$› **by** *force*
show *?thesis*
unfolding *sol* ‹$j = 1$› *pow-list-1*
unfolding *pow-pos2*[*OF* ‹$0 < l$›] *rassoc***..**
next
fix *r q t*
assume $j = 1\ 2 \le k$ **and** *sol*:
$x =$
$((q \cdot r) \cdot (r \cdot (q \cdot r) \,^{@}\, t) \,^{@}\, (k - 1)) \,^{@}\, (l - 2) \cdot$
$(((q \cdot r) \cdot (r \cdot (q \cdot r) \,^{@}\, t) \,^{@}\, (k - 2)) \cdot r) \cdot q$
$y = r \cdot (q \cdot r) \,^{@}\, t$
$z = (q \cdot r) \cdot (r \cdot (q \cdot r) \,^{@}\, t) \,^{@}\, (k - 1)$
$0 < t$
obtain k' **where** *Suc* (*Suc* k') $= k$ **using** *Suc-minus2*[*OF* ‹$2 \le k$›] **by** *blast*
hence *k1*: $k - 1 =$ *Suc* k' **and** *k2*: $k - 2 = k'$ **and** *k*: $k = k' + 2$ **by** *fastforce+*
obtain l' **where** *Suc* (*Suc* l') $= l$ **using** *Suc-minus2*[*OF* ‹$2 \le l$›] **by** *blast*
hence *l2*: $l - 2 = l'$ **and** *l*: $l = l' + 2$ **by** *fastforce+*
show $x \,^{@}\, j \cdot y \,^{@}\, k = z \,^{@}\, l$
unfolding *sol* ‹$j = 1$› *k1 k2 l2* **unfolding** *k l* **by** *comparison*
next

```
    fix r q t
    assume j = 2 k = 1 l = 2 and  sol:
      x = (r · q) @ t · r
      y = q · r · r · q z = (r · q) @ t · r · r · q
      2 ≤ t
    show x @ j · y @ k = z @ l
      unfolding ‹j = 2› ‹k = 1› ‹l = 2› sol pow-list-1 pow-list-2
      by comparison
  qed
qed
```

6.3.4 Uniqueness of the imprimitivity witness

In this section, we show that given a binary code $\{x, y\}$ and two imprimitive words $x \,@\, j \cdot y \,@\, k$ and $x \,@\, j' \cdot y \,@\, k'$ is possible only if the two words are equals, that is, if $j = j'$ and $k = k'$.

```
lemma LS-unique-same: assumes x · y ≠ y · x
  and 0 < j and 0 < k and ¬ primitive(x@j·y@k)
  and 0 < k' and ¬ primitive(x@j·y@k')
shows k = k'
proof(rule ccontr)
  assume k ≠ k'

  define ka where ka = (if k < k' then k else k')
  define ka' where ka' = (if k < k' then k' else k)

  have ka < ka' and ka ≠ ka'
    unfolding ka-def ka'-def using ‹k ≠ k'› by auto
  then obtain dif where [symmetric]: ka + dif = ka' and dif ≠ 0
    using less-imp-add-positive by blast
  have ka ≠ 0 and ka' ≠ 0 and ‹j ≠ 0›
    unfolding ka-def ka'-def using ‹0 < k› ‹0 < k'› ‹0 < j› by force+

  have ¬ primitive(x@j·y@ka) ¬ primitive(x@j·y@ka')
    unfolding ka-def ka'-def using assms(4) assms(6) by presburger+

  have x@j·y@ka' = x@j·y@ka·y@dif
    unfolding pow-add[symmetric] ‹ka' = ka + dif›..

  consider dif = 1 | 2 ≤ dif
    using ‹ka < ka'› ‹ka' = ka + dif› by fastforce
  hence x · y = y · x
  proof(cases)
    assume dif = 1
    define u where u = x@j·y@(ka − 1)
    have ¬ primitive (u · y)
      unfolding u-def rassoc pow-Suc2[symmetric]  Suc-minus[OF ‹ka ≠ 0›] by
fact
    have ¬ primitive (u · y · y)
```

unfolding *u-def rassoc* **using** ‹¬ *primitive*($x^{@}j{\cdot}y^{@}ka'$)›[*unfolded* ‹$x^{@}j{\cdot}y^{@}ka' = x^{@}j{\cdot}y^{@}ka{\cdot}y^{@}dif$› ‹$dif = 1$› *pow-list-1*]
unfolding *pow-Suc2*[*of* $ka - 1$ *y*, *unfolded Suc-minus*[*OF* ‹$ka \neq 0$›]] *rassoc*.
from *imprim-ext-suf-comm*[*OF* ‹¬ *primitive* $(u \cdot y)$› ‹¬ *primitive* $(u \cdot y \cdot y)$›]
have $(x\ ^{@}\ j \cdot y\ ^{@}\ (ka - 1)) \cdot y = y \cdot x\ ^{@}\ j \cdot y\ ^{@}\ (ka - 1)$
unfolding *u-def*.
thus $x \cdot y = y \cdot x$
using ‹$j \neq 0$› **by** *mismatch*
next
assume $2 \leq dif$
hence ¬ *primitive* ($y^{@}dif$)..
from *Lyndon-Schutzenberger-prim*[*OF* ‹¬ *primitive* $(x\ ^{@}\ j \cdot y\ ^{@}\ ka)$› *this* ‹¬ *primitive* $(x\ ^{@}\ j \cdot y\ ^{@}\ ka')$›[*unfolded* ‹$x\ ^{@}\ j \cdot y\ ^{@}\ ka' = x\ ^{@}\ j \cdot y\ ^{@}\ ka{\cdot}\ y^{@}dif$› *lassoc*]]
show $x \cdot y = y \cdot x$
using ‹$dif \neq 0$› ‹$j \neq 0$› **by** *mismatch*
qed
thus *False*
using ‹$x \cdot y \neq y \cdot x$› **by** *blast*
qed

lemma *LS-unique-distinct-le*: **assumes** $x \cdot y \neq y \cdot x$
and $2 \leq j$ **and** ¬ *primitive*($x^{@}j{\cdot}y$)
and $2 \leq k$ **and** ¬ *primitive*($x{\cdot}y^{@}k$)
and $|y| \leq |x|$
shows *False*
proof–
have $0 < k$
using ‹$2 \leq k$› **by** *linarith*
obtain *l z* **where** [*symmetric*]:$z^{@}l = x^{@}j{\cdot}y$ **and** $2 \leq l$
using *not-prim-primroot-expE*[*OF* ‹¬ *primitive*($x^{@}j{\cdot}y$)›].
have $x^{@}j{\cdot}y^{@}1 = z^{@}l$
by (*simp add*: ‹$x\ ^{@}\ j \cdot y = z\ ^{@}\ l$›)
interpret *eq1*: *LS-len-le x y j 1 l z*
using ‹$x \cdot y \neq y \cdot x$› ‹$|y| \leq |x|$› ‹$x^{@}j{\cdot}y^{@}1 = z^{@}l$› ‹$2 \leq l$› ‹$2 \leq j$›
by(*unfold-locales*) *linarith*+

from *eq1.case-j2k1*[*OF* ‹$2 \leq j$› *refl*]
obtain *r q t* **where**
x[*symmetric*]: $(r \cdot q)\ ^{@}\ t \cdot r = x$ **and**
y[*symmetric*]: $q \cdot r \cdot r \cdot q\ =\ y$ **and**
z[*symmetric*]: $(r \cdot q)\ ^{@}\ t \cdot r \cdot r \cdot q = z$ **and**
$2 \leq t$ **and** $j = 2$ **and** $l = 2$ **and** $r{\cdot}q \neq q{\cdot}r$ **and**
primitive x **and** *primitive y*.

have $q{\cdot}r \neq \varepsilon$ $r{\cdot}q \neq \varepsilon$
using *eq1.bin-snd-nemp y* **by** *fastforce*+

obtain z' l' **where** $x{\cdot}y^{@}k = z'^{@}l'$ $2 \leq l'$

```
    using not-prim-primroot-expE[OF ‹¬ primitive (x · y ^@ k)›] by metis
  from ‹x·y^@k = z'^@l'›[unfolded x y, unfolded rassoc, symmetric]
  have z': z' ^@ l' = (r · q) ^@ t · r · (q · r · r · q) ^@ k.
  have 0 < l' and 0 < l' − 1
    using ‹2 ≤ l'› by auto
  have r · q · r · q ≤p x
   using pref-extD[of r·q·r·q (r · q) ^@ (t − 2) · r]
    unfolding  x[folded pop-pow[OF ‹2 ≤ t›], unfolded pow-list-2] rassoc by blast

  have per1: x · q · r ≤p (r · q) · x · q · r
    unfolding x by comparison
  have per2: x · q · r ≤p z' · x · q · r
    by (rule pref-pow-root[of - l'],
    unfold ‹x·y^@k = z'^@l'›[unfolded  y  pow-pos[OF ‹0 < k›], symmetric])
    comparison
  have (r · q) · z' ≠ z' · (r · q)
  proof
    assume (r · q) · z' = z' · (r · q)
    hence (r · q) · z'^@l' = z'^@l' · r · q
      by (simp add: comm-pow-comm)
    from this[unfolded z']
    have r · q = q · r
      using ‹0 < k› by mismatch
    thus False
      using ‹r · q ≠ q · r› by presburger
  qed
  with two-pers[OF per1 per2]
  have |x| ≤ |z'|
    unfolding lenmorph by linarith

  from eqdE[OF ‹x·y^@k = z'^@l'›[unfolded pow-pos[OF ‹0 < l'›]
       pow-pos2[OF ‹0 < l'−1›]] ‹|x| ≤ |z'|› ]
  obtain w where x · w = z' w · z' ^@ (l' − 1 − 1) · z' = y ^@ k.
  from this(1) this(2)[unfolded lassoc]
  have  x ≤f y^@k
    by blast
  hence r·q·r·q ≤f (q·r·r·q)^@k
    unfolding y
    using pref-fac[OF ‹r · q · r · q ≤p x›] by blast

  have |r · q · r · q| = |q · r · r · q|
    by simp
   from xyxy-conj-yxxy[OF fac-pow-len-conjug[OF this ‹r·q·r·q ≤f (q·r·r·q)^@k›],
symmetric]]
  have r · q = q · r.
  thus False
    using ‹r · q ≠ q · r› by blast
qed
```

lemma *LS-unique-distinct*: **assumes** $x \cdot y \neq y \cdot x$
and $2 \leq j$ **and** $\neg$ *primitive*$(x^{@}j{\cdot}y)$
and $2 \leq k$ **and** $\neg$ *primitive*$(x{\cdot}y^{@}k)$
shows *False*
using *LS-unique-distinct-le*[*OF assms*] *LS-unique-distinct-le*[*reversed*, *OF assms*(*1*,*4*−*5*,*2*−*3*)]
by *fastforce*

lemma *LS-unique'*: **assumes** $x \cdot y \neq y \cdot x$
and $0 < j$ **and** $0 < k$ **and** $\neg$ *primitive*$(x^{@}j{\cdot}y^{@}k)$
and $0 < j'$ **and** $0 < k'$ **and** $\neg$ *primitive*$(x^{@}j'{\cdot}y^{@}k')$
shows $k = k'$
proof−
have $j = 1 \vee k = 1$
using *Lyndon-Schutzenberger-prim*[*OF pow-non-prim pow-non-prim*,
OF - - ‹$\neg$ *primitive* $(x \,^{@}\, j \cdot y \,^{@}\, k)$›]
comm-drop-exps[*OF* ‹$0 < j$› ‹$0 < k$›] ‹$x \cdot y \neq y \cdot x$› **by** *blast*
have $j' = 1 \vee k' = 1$
using *Lyndon-Schutzenberger-prim*[*OF pow-non-prim pow-non-prim*,
OF - - ‹$\neg$ *primitive* $(x \,^{@}\, j' \cdot y \,^{@}\, k')$›]
comm-drop-exps[*OF* ‹$0 < j'$› ‹$0 < k'$›] ‹$x \cdot y \neq y \cdot x$› **by** *blast*
show $k = k'$
proof (*cases* $j = j'$)
assume $j = j'$
from *LS-unique-same*[*OF assms*(*1*−*4*,*6*,*7*)[*folded this*]]
show $k = k'$**.**
next
assume $j \neq j'$
show $k = k'$
proof(*rule ccontr*, *cases* $j = 1$)
assume $k \neq k'$ **and** $j = 1$
hence $2 \leq j'$ **and** $k' = 1$ **and** $2 \leq k$
using ‹$j \neq j'$› ‹$0 < j'$› ‹$k \neq k'$› ‹$0 < k$› ‹$j' = 1 \vee k' = 1$› **by** *auto*
from *LS-unique-distinct*[*OF* ‹$x \cdot y \neq y \cdot x$› ‹$2 \leq j'$› - ‹$2 \leq k$›]
show *False*
using ‹$\neg$ *primitive*$(x^{@}j'{\cdot}y^{@}k')$›[*unfolded* ‹$k'{=}1$› *pow-list-1*] ‹$\neg$ *primitive*$(x^{@}j{\cdot}y^{@}k)$›[*unfolded* ‹$j{=}1$› *pow-list-1*]
by *blast*
next
assume $k \neq k'$ **and** $j \neq 1$
hence $2 \leq j$ **and** $k = 1$ **and** $2 \leq k'$ **and** $j' = 1$
using ‹$0 < j$› ‹$j = 1 \vee k = 1$› ‹$0 < k'$› ‹$j' = 1 \vee k' = 1$› **by** *auto*
from *LS-unique-distinct*[*OF* ‹$x \cdot y \neq y \cdot x$› ‹$2 \leq j$› - ‹$2 \leq k'$›]
show *False*
using ‹$\neg$ *primitive*$(x^{@}j'{\cdot}y^{@}k')$›[*unfolded* ‹$j'{=}1$› *pow-list-1*] ‹$\neg$ *primitive*$(x^{@}j{\cdot}y^{@}k)$›[*unfolded* ‹$k{=}1$› *pow-list-1*]
by *blast*
qed
qed
qed

lemma *LS-unique*: **assumes** $x \cdot y \neq y \cdot x$
and $0 < j$ **and** $0 < k$ **and** $\neg$ *primitive*$(x^{@}j{\cdot}y^{@}k)$
and $0 < j'$ **and** $0 < k'$ **and** $\neg$ *primitive*$(x^{@}j'{\cdot}y^{@}k')$
shows $j = j'$ **and** $k = k'$
using *LS-unique'*[*OF* ‹$x \cdot y \neq y \cdot x$›
‹$0 < j$› ‹$0 < k$› ‹$\neg$ *primitive* $(x \,^{@}\, j \cdot y \,^{@}\, k)$›
‹$0 < j'$› ‹$0 < k'$› ‹$\neg$ *primitive* $(x \,^{@}\, j' \cdot y \,^{@}\, k')$›]
LS-unique'[*reversed*, *OF* ‹$x \cdot y \neq y \cdot x$›
‹$0 < k$› ‹$0 < j$› ‹$\neg$ *primitive* $(x \,^{@}\, j \cdot y \,^{@}\, k)$›
‹$0 < k'$› ‹$0 < j'$› ‹$\neg$ *primitive* $(x \,^{@}\, j' \cdot y \,^{@}\, k')$›]
by *blast*+

6.4 The bound on the exponent in Lyndon-Schützenberger equation

lemma (**in** *LS-len-le*) *case-j1k2-exp-le*:
assumes $j = 1$ $2 \leq k$
shows $k{*}|y| + 4 \leq |x| + 2{*}|y|$
proof−
have $x \cdot y \,^{@}\, k = z \,^{@}\, l$ **and** $|y| \neq 0$ **and** $0 < l$
using *eq*[*unfolded* ‹$j = 1$› *cow-simps*] *nemp-len*[*OF bin-snd-nemp*] *l-min*
by *linarith*+

consider $y \,^{@}\, k <_s z \mid z \leq_s y^{@}k$
using *ruler-eq'*[*reversed*,
OF ‹$x \cdot y^{@}k = z^{@}l$›[*symmetric*, *unfolded pow-pos2*[*OF* ‹$0 < l$›]]] **by** *blast*
thus *?thesis*
proof(*cases*)
assume $y^{@}k <_s z$
from *case-j1k2-b*[*OF assms this*]
obtain q **where**
x: $x = (q \cdot y \,^{@}\, k) \,^{@}\, (l - 1) \cdot q$ **and**
$z = q \cdot y \,^{@}\, k$ **and**
$q \cdot y \neq y \cdot q$.
have $1 \leq |q|$
using *nemp-le-len*[*of q*] ‹$q \cdot y \neq y \cdot q$› **by** *blast*

have $|y| \geq 1$
using *bin-snd-nemp nemp-le-len* **by** *blast*

have *lle*: $x \leq (l-1){*}x$ **for** x
using *l-min*
by (*simp add*: *quotient-smaller*)

have $|x| \geq k{*}|y| + 2$
unfolding *x lenmorph pow-len*
using *le-trans*[*OF - lle*, *of* $k * |y| + 1\ |q| + k * |y|$, *THEN add-le-mono*[*OF*

‹*1* ≤ |*q*|›]]
  unfolding *add.commute*[*of* |*q*|]
  using ‹*1* ≤ |*q*|› **by** *auto*
 thus *?thesis*
  using ‹|*y*| ≥ *1*› **by** *linarith*
next
 assume *z* ≤*s* *y* @ *k*
 from *case-j1k2-a*[*OF assms this*]
 obtain *q r t* **where**
  x: *x* = ((*q* · *r*) · (*r* · (*q* · *r*) @ *t*) @ (*k* − *1*)) @ (*l* − *2*) · (((*q* · *r*) · (*r* · (*q* · *r*) @ *t*) @ (*k* − *2*)) · *r*) · *q* **and**
  y = *r* · (*q* · *r*) @ *t* **and**
  z = (*q* · *r*) · (*r* · (*q* · *r*) @ *t*) @ (*k* − *1*) **and**
  0 < *t* **and** *r* · *q* ≠ *q* · *r***.**

 have *q* ≠ *ε* *r* ≠ *ε*
  using ‹*r* · *q* ≠ *q* · *r*› **by** *blast*+
 hence |*q*| ≥ *1* |*r*| ≥ *1*
  using *nemp-le-len* **by** *blast*+
 hence |*q*·*r*| + |*r*| + |*q*| ≥ *4*
  by *simp*

 have |*x*| ≥ |(((*q* · *r*) · (*r* · (*q* · *r*) @ *t*) @ (*k* − *2*)) · *r*) · *q*|
  using *x suf-len′* **by** *blast*
 hence |*x*| ≥ |*q*·*r*| + (*k*−*2*)∗|*y*| + |*r*| + |*q*|
  unfolding ‹*y* = *r* · (*q* · *r*) @ *t*›[*symmetric*]
  by (*simp add*: *pow-len*)
 hence |*x*| ≥ (*k*−*2*)∗|*y*| + *4*
  using ‹*4* ≤ |*q* · *r*| + |*r*| + |*q*|› **by** *linarith*
 thus *?thesis*
  unfolding *add.commute*[*of* |*x*|]
  unfolding *nat-le-add-iff1*[*OF* ‹*2* ≤ *k*›]**.**
qed
qed

lemma (**in** *LS-len-le*) *case-j2k1-exp-le*:
 assumes *2* ≤ *j* *k* = *1*
 shows *j*∗|*x*| + *4* ≤ |*y*| + *2*∗|*x*|
proof−
 from *case-j2k1*[*OF assms*]
 obtain *r q t* **where**
  (*r* · *q*) @ *t* · *r* = *x* **and**
  q · *r* · *r* · *q* = *y* **and**
  (*r* · *q*) @ *t* · *r* · *r* · *q* = *z* **and**
  ‹*2* ≤ *t*› **and** *j* = *2* **and** *l* = *2* **and**
  r · *q* ≠ *q* · *r* **and**
  primitive x **and**
  *primitive y***.**

```
  have |r| ≥ 1 |q| ≥ 1
    using ‹r · q ≠ q · r› nemp-le-len by blast+
  hence |y| ≥ 4
    unfolding ‹q · r · r · q = y›[symmetric] lenmorph
    by linarith
  thus ?thesis
    by (simp add: ‹j = 2›)
qed

theorem LS-exp-le-one:
  assumes eq: x · y @ k = z @ l
      and 2 ≤ l
      and x · y ≠ y · x
      and 1 ≤ k
      shows k*|y| + 4 ≤ |x|+2*|y|
proof–
  have x ≠ ε y ≠ ε x ≠ y |y| ≠ 0 z ≠ ε
    using ‹x · y ≠ y · x› ‹x · y @ k = z @ l› by fastforce+
  have l ≠ 0  ‹1 ≤ l–1›
    using ‹2 ≤ l› by force+

  consider k = 1 | k ≥ 2
    using ‹1 ≤ k› by linarith
  then show ?thesis
  proof(cases)
    assume k=1
    have 4 ≤ |x| + |y|
      using case-j1k1[OF eq[unfolded ‹k = 1› pow-list-1] ‹x · y ≠ y · x› ‹2 ≤ l›]
      by blast
    thus ?thesis
      unfolding ‹k = 1› by force
  next
    assume k ≥ 2
  show ?thesis
  proof(rule le-cases)
    assume |y| ≤ |x|
    then interpret LS-len-le x y 1 k l z
      using assms by unfold-locales (blast, blast, force, blast+)
    from case-j1k2-exp-le[OF refl ‹k ≥ 2›]
    show ?thesis.
  next
    assume |x| ≤ |y|
    have y · x ≠ x · y
      using assms(3) by force
    define z' where z' = rotate |x| z
    hence y@k · x = z' @ l
      using arg-cong[OF assms(1), of λt. rotate |x| t]
      unfolding rotate-append rotate-pow-list-swap
      by blast
```

```
    interpret LS-len-le y x k 1 l z'
      using ‹|x| ≤ |y|› ‹y · x ≠ x · y› ‹y^@k · x = z' @ l› ‹2 ≤ l› ‹1 ≤ k›
      by (unfold-locales) (blast, blast, force, blast+)
    from case-j2k1-exp-le[OF ‹2 ≤ k› refl]
    show ?thesis.
     qed
  qed
qed

lemma LS-exp-le-conv-rat:
  fixes x y k::'a::linordered-field
  assumes y > 0
  shows k * y + 4 ≤ x + 2 * y ⟷ k ≤ (x − 4)/y+ 2
  unfolding le-diff-eq[symmetric]
  unfolding diff-conv-add-uminus
  unfolding add.assoc add.commute[of 2*y]
  unfolding add.assoc[symmetric]
  unfolding diff-le-eq[of - 2*y x + − 4,symmetric] left-diff-distrib'[symmetric]
  unfolding pos-le-divide-eq[OF ‹y > 0›, symmetric]
  unfolding diff-le-eq..


end


theory Binary-Code-Morphisms
  imports CoWBasic Submonoids Morphisms

begin
```

Chapter 7

Binary alphabet and binary morphisms

7.1 Datatype of a binary alphabet

Basic elements for construction of binary words.

type-notation *Enum.finite-2* (*binA*)
notation *finite-2.*a_1 (*bina*)
notation *finite-2.*a_2 (*binb*)

lemmas *bin-distinct* = *Enum.finite-2.distinct*
lemmas *bin-exhaust* = *Enum.finite-2.exhaust*
lemmas *bin-induct* = *Enum.finite-2.induct*
lemmas *bin-UNIV* = *Enum.UNIV-finite-2*
lemmas *bin-eq-neq-iff* = *Enum.neq-finite-2-*a_2*-iff*
lemmas *bin-eq-neq-iff'* = *Enum.neq-finite-2-*a_1*-iff*

abbreviation *bin-word-a* :: *binA list* (𝔞) **where**
bin-word-a ≡ [*bina*]

abbreviation *bin-word-b* :: *binA list* (𝔟) **where**
bin-word-b ≡ [*binb*]

abbreviation *binUNIV* :: *binA set* **where** *binUNIV* ≡ *UNIV*

lemma *binUNIV-I* [*simp*, *intro*]: *bina* ∈ *A* ⟹ *binb* ∈ *A* ⟹ *A* = *UNIV*
unfolding *UNIV-finite-2* **by** *auto*

lemma *bin-basis-code*: *code* {𝔞,𝔟}
by (*rule bin-code-code*) *blast*

lemma *bin-num*: *bina* = *0 binb* = *1*
by *simp-all*

lemma *binA-simps* [*simp*]: *bina − binb = binb binb − bina = binb 1 − bina = binb 1 − binb = bina a − a = bina 1 − (1 − a) = a*
by *simp-all*

definition *bin-swap* :: *binA ⇒ binA* **where** *bin-swap x ≡ 1 − x*

lemma *bin-swap-if-then*: *1−x = (if x = bina then binb else bina)*
by *fastforce*

definition *bin-swap-morph* **where** *bin-swap-morph ≡ map bin-swap*

lemma *alphabet-or*[*simp*]: *a = bina ∨ a = binb*
by *auto*

lemma *bin-im-or*: *f* [*a*] = *f* 𝔞 ∨ *f* [*a*] = *f* 𝔟
by (*rule bin-exhaust*[*of a*], *simp-all*)

thm *triv-forall-equality*

lemma *binUNIV-card*: *card binUNIV = 2*
unfolding *bin-UNIV card-2-iff* **by** *auto*

lemma *other-letter*: **obtains** *b* **where** *b ≠* (*a* :: *binA*)
using *finite-2.distinct*(*1*) **by** *metis*

lemma *alphabet-or-neq*: *x ≠ y ⟹ x =* (*a* :: *binA*) *∨ y = a*
using *alphabet-or*[*of x*] *alphabet-or*[*of y*] *alphabet-or*[*of a*] **by** *argo*

lemma *binA-neq-cases*: **assumes** *neq*: *a ≠ b*
obtains *a = bina* **and** *b = binb* | *a = binb* **and** *b = bina*
using *alphabet-or-neq assms* **by** *auto*

lemma *bin-neq-sym-pred*: **assumes** *a ≠ b* **and** *P bina binb* **and** *P binb bina* **shows** *P a b*
using *assms*(*2−3*) *binA-neq-cases*[*OF* ‹*a ≠ b*›, *of P a b*] **by** *blast*

lemma *no-third*: (*c* :: *binA*) *≠ a ⟹ b ≠ a ⟹ b = c*
using *alphabet-or*[*of a*] **by** *fastforce*

lemma *two-in-bin-UNIV*: **assumes** *a ≠ b* **and** *a ∈ S* **and** *b ∈ S* **shows** *S = binUNIV*
using ‹*a ∈ S*› ‹*b ∈ S*› *alphabet-or-neq*[*OF* ‹*a ≠ b*›] **by** *fast*

lemmas *two-in-bin-set = two-in-bin-UNIV*[*unfolded bin-UNIV*]

lemma *bin-not-comp-set-UNIV*: **assumes** *¬ u ⋈ v* **shows** *set* (*u · v*) = *binUNIV*
proof−
have *uv*: *u · v = ((u ∧*$_p$ *v*) *·* ([*hd* ((*u ∧*$_p$ *v*)$^{-1>}$*u*)] *· tl* ((*u ∧*$_p$ *v*)$^{-1>}$*u*))) *·* (*u ∧*$_p$ *v*) *·* ([*hd* ((*u ∧*$_p$ *v*)$^{-1>}$*v*)] *· tl* ((*u ∧*$_p$ *v*)$^{-1>}$*v*))

unfolding *hd-tl*[*OF lcp-mismatch-lq*(*1*)[*OF assms*]] *hd-tl*[*OF lcp-mismatch-lq*(*2*)[*OF assms*]] *lcp-lq*..
from *this*[*unfolded rassoc*]
have *hd* ((*u* $\wedge_p$ *v*)$^{-1>}$*u*) $\in$ *set* (*u* · *v*) **and** *hd* ((*u* $\wedge_p$ *v*)$^{-1>}$*v*) $\in$ *set* (*u* · *v*)
unfolding *uv* **by** *simp-all*
with *lcp-mismatch-lq*(*3*)[*OF assms*]
show *?thesis*
using *two-in-bin-UNIV* **by** *blast*
qed

lemma *bin-basis-singletons*: {[*q*] |*q*. *q* $\in$ {*bina*, *binb*}} = {𝔞,𝔟}
by *blast*

lemma *bin-basis-generates*: ⟨{𝔞,𝔟}⟩ = *UNIV*
using *sings-gen-lists*[*of binUNIV*, *unfolded lists-UNIV bin-UNIV bin-basis-singletons*, *folded bin-UNIV*, *unfolded lists-UNIV*].

lemma *a-in-bin-basis*: [*a*] $\in$ {𝔞,𝔟}
using *Set.UNIV-I* **by** *auto*

lemma *lcp-zero-one-emp*: 𝔞 $\wedge_p$ 𝔟 = ε **and** *lcp-one-zero-emp*: 𝔟 $\wedge_p$ 𝔞 = ε
by *simp*+

lemma *bin-neq-induct*: (*a*::*binA*) $\neq$ *b* $\Longrightarrow$ *P a* $\Longrightarrow$ *P b* $\Longrightarrow$ *P c*
proof (*elim binA-neq-cases*)
show *P a* $\Longrightarrow$ *P b* $\Longrightarrow$ *a* = *bina* $\Longrightarrow$ *b* = *binb* $\Longrightarrow$ *P c*
using *finite-2.induct* **by** *blast*
show *P a* $\Longrightarrow$ *P b* $\Longrightarrow$ *a* = *binb* $\Longrightarrow$ *b* = *bina* $\Longrightarrow$ *P c*
using *finite-2.induct* **by** *blast*
qed

lemma *bin-neq-induct'*: **assumes**(*a*::*binA*) $\neq$ *b* **and** *P a* **and** *P b* **shows** $\bigwedge$ *c*. *P c*
using *bin-neq-induct*[*OF assms*] **by** *blast*

lemma *neq-exhaust*: **assumes** (*a*::*binA*) $\neq$ *b* **obtains** *c* = *a* | *c* = *b*
using *assms* **by** (*elim binA-neq-cases*) (*hypsubst*, *elim finite-2.exhaust*, *assumption*)+

lemma *bin-swap-neq* [*simp*]: *1*−(*a* :: *binA*) $\neq$ *a*
by *simp*
lemmas *bin-swap-neq'*[*simp*] = *bin-swap-neq*[*symmetric*]

lemmas *bin-swap-induct* = *bin-neq-induct*[*OF bin-swap-neq'*]
and *bin-swap-exhaust* = *neq-exhaust*[*OF bin-swap-neq'*]

lemma *bin-swap-induct'*: *P* (*a* :: *binA*) $\Longrightarrow$ *P* (*1*−*a*) $\Longrightarrow$ ($\bigwedge$ *c*. *P c*)
using *bin-swap-induct* **by** *auto*

lemma *swap-UNIV*: {*a*, *1*−*a*} = *binUNIV* (**is** *?P a*)

using *bin-swap-induct*[*of ?P bina*, *unfolded binA-simps*] **by** *fastforce*

lemma *bin-neq-swap'*[*intro*]: $a \neq b \Longrightarrow 1 - b = (a :: binA)$
by (*rule bin-swap-exhaust*[*of a b*]) *blast+*

lemma *bin-neq-swap*[*intro*]: $a \neq b \Longrightarrow 1 - a = (b :: binA)$
using *bin-neq-swap'* **by** *auto*

lemma *bin-neq-swap''*[*intro*]: $a \neq b \Longrightarrow b = 1-(a:: binA)$
using *bin-neq-swap* **by** *blast*

lemma *bin-neq-swap'''*[*intro*]: $a \neq b \Longrightarrow a = 1-(b:: binA)$
using *bin-neq-swap* **by** *blast*

lemma *bin-neq-iff*: $c \neq d \longleftrightarrow 1 - d = (c :: binA)$
using *bin-neq-swap*[*of d c*] *bin-swap-neq*[*of d*] **by** *argo*

lemma *bin-neq-iff'*: $c \neq d \longleftrightarrow 1 - c = (d :: binA)$
unfolding *bin-neq-iff* **by** *force*

lemma *binA-neq-cases-swap*: **assumes** *neq*: $a \neq (b :: binA)$
obtains $a = c$ **and** $b = 1 - c \mid a = 1 - c$ **and** $b = c$
using *assms bin-neq-swap bin-swap-exhaust*[*of a c*] **by** *metis*

lemma *im-swap-neq*: $f\ a = f\ b \Longrightarrow f\ bina \neq f\ binb \Longrightarrow a = b$
using *binA-neq-cases-swap*[*of a b bina False*, *unfolded binA-simps*] **by** *fastforce*

lemma *bin-without-letter*: **assumes** $(a :: binA) \notin set\ w$
obtains k **where** $w = [1-a]^@k$
proof–
have $set\ w \subseteq \{1-a\}$
using *assms bin-swap-exhaust* **by** *blast*
from *that sing-set-wordE*[*OF this*]
show *thesis* **by** *blast*
qed

lemma *bin-empty-iff*: $S = \{\} \longleftrightarrow (a :: binA) \notin S \wedge 1-a \notin S$
using *bin-swap-induct*[*of* $\lambda a.\ a \notin S$] **by** *blast*

lemma *bin-UNIV-iff*: $S = binUNIV \longleftrightarrow a \in S \wedge 1-a \in S$
using *two-in-bin-UNIV*[*OF bin-swap-neq'*] **by** *blast*

lemma *bin-UNIV-I*: $a \in S \Longrightarrow 1-a \in S \Longrightarrow S = binUNIV$
using *bin-UNIV-iff* **by** *blast*

lemma *bin-sing-iff*: $A = \{a :: binA\} \longleftrightarrow a \in A \wedge 1-a \notin A$
proof (*rule sym*, *intro iffI conjI*, *elim conjE*)
assume $a \in A$ **and** $1-a \notin A$
have $b \in A \longleftrightarrow b = a$ **for** b

```
    using ‹a ∈ A› ‹1−a ∉ A› bin-swap-neq
    by (intro bin-swap-induct[of λc. (c ∈ A) = (c = a) a b]) blast+
  then show A = {a} by blast
qed simp-all

lemma set-binUNIV-iff: ¬ set w ⊆ {hd w} ⟷ set w = binUNIV
proof
 show set w = binUNIV ⟹ ¬ set w ⊆ {hd w}
   using bin-sing-iff by blast
next
  assume ¬ set w ⊆ {hd w}
  hence w ≠ ε and set w ≠ {hd w}
    by force+
  hence hd w ∈ set w
    by force
  with ‹set w ≠ {hd w}›[unfolded bin-sing-iff]
  have 1 − hd w ∈ set w
    by blast
  show set w = binUNIV
    using two-in-bin-UNIV[OF - ‹hd w ∈ set w› ‹1−hd w ∈ set w›]
    by force
qed

lemma bin-set-cases: obtains S = {} | S = {a} | S = {1−a} | S = binUNIV
  unfolding bin-empty-iff[of - a] bin-UNIV-iff[of - 1−a] bin-sing-iff
  by fastforce

lemma not-UNIV-E: assumes A ≠ binUNIV obtains a where A ⊆ {a}
  using assms by (cases rule: bin-set-cases[of A]) auto

lemma not-UNIV-nempE: assumes A ≠ binUNIV and A ≠ {} obtains a where
A = {a}
  using assms by (cases rule: bin-set-cases[of A]) auto

lemma bin-sing-gen-iff: x ∈ ⟨{[a]}⟩ ⟷ 1−(a :: binA) ∉ set x
 unfolding sing-gen-lists[symmetric] in-lists-conv-set using bin-empty-iff bin-sing-iff
by metis

lemma set-hd-pow-conv: set w ⊆ {hd w} ⟷ set w ≠ binUNIV
proof
  assume set w ⊆ {hd w}
  thus set w ≠ binUNIV
    unfolding bin-UNIV  using bin-distinct(1) by force
next
  assume set w ≠ binUNIV
  thus set w ⊆ {hd w}
  proof (cases w = ε)
    assume set w ≠ binUNIV and w ≠ ε
    from hd-tl[OF this(2)] this(2)
```

have *hd w* ∈ *set w* **by** *simp*
hence *1−hd w* ∉ *set w*
using ‹*set w* ≠ *binUNIV*› **unfolding** *bin-UNIV-iff*[*of set w hd w*] **by** *blast*
thus *set w* ⊆ {*hd w*}
using *bin-sing-iff* **by** *auto*
qed *simp*
qed

lemma *not-swap-eq*: *P a b* ⟹ (⋀ (*c* :: *binA*). ¬ *P c* (*1−c*)) ⟹ *a* = *b*
using *bin-neq-iff* **by** *metis*

lemma *bin-distinct-letter*: **assumes** *set w* = *binUNIV*
obtains *k w′* **where** [*hd w*]@*Suc k* · [*1−hd w*] · *w′* = *w*
proof−
from *distinct-letter-in-hd′*[*of w*, *unfolded set-hd-pow-conv*[*of w*] *bool-simps*(*1*), *OF assms*]
obtain *m b q* **where** [*hd w*] @ *Suc m* · [*b*] · *q* = *w b* ≠ *hd w*.
with *that bin-neq-swap′*[*OF this*(*2*)]
show *thesis*
by *blast*
qed

lemma *P a* ⟷ *P* (*1−a*) ⟹ *P a* ⟹ (⋀ (*b* :: *binA*). *P b*)
using *bin-swap-induct′* **by** *blast*

lemma *bin-sym-all*: *P* (*a* :: *binA*) ⟷ *P* (*1−a*) ⟹ *P a* ⟹ *P x*
using *bin-swap-induct*[*of* λ *a*. *P a a*, *unfolded binA-simps*] **by** *blast*

lemma *bin-sym-all-comm*:
f [*a*] · *f* [*1−a*] ≠ *f* [*1−a*] · *f* [*a*] ⟹ *f* [*b*] · *f* [*1−b*] ≠ *f* [*1−b*] · *f* [(*b* :: *binA*)] (**is** *?P a* ⟹ *?P b*)
using *bin-sym-all*[*of ?P a*, *unfolded binA-simps*, *OF neq-commute*].

lemma *bin-sym-all-neq*:
f [(*a* :: *binA*)] ≠ *f* [*1−a*] ⟹ *f* [*b*] ≠ *f* [*1−b*] (**is** *?P a* ⟹ *?P b*)
using *bin-sym-all*[*of ?P a*, *unfolded binA-simps*, *OF neq-commute*].

lemma *bin-len-count*:
fixes *w* :: *binA list*
shows |*w*| = *count-list w a* + *count-list w* (*1−a*)
using *sum-count-set*[*of w* {*a*,*1−a*}] *swap-UNIV* **by** *force*

lemma *bin-len-count′*:
fixes *w* :: *binA list*
shows |*w*| = *count-list w bina* + *count-list w binb*
using *bin-len-count*[*of w bina*] **by** *force*

7.2 Binary morphisms

lemma *bin-map-core-lists*: $(map\ f^{\mathcal{C}}\ w) \in lists\ \{f\ \mathfrak{a}, f\ \mathfrak{b}\}$
unfolding *core-def* **by** (*induct w*, *simp*, *unfold map-hd*)
(*rule append-in-lists*, *simp-all add*: *bin-im-or*)

lemma *bin-range*: $range\ f = \{f\ bina, f\ binb\}$
unfolding *bin-UNIV* **by** *simp*

lemma *bin-core-range*: $range\ f^{\mathcal{C}} = \{f\ \mathfrak{a}, f\ \mathfrak{b}\}$
unfolding *core-def bin-range*..

lemma *bin-core-range-swap*: $range\ f^{\mathcal{C}} = \{f\ [(a :: binA)], f\ [1-a]\}$ (**is** *?P a*)
by (*rule bin-induct*[*of ?P*, *unfolded binA-simps*], *unfold bin-core-range*, *simp*, *force*)

lemma *bin-map-core-lists-swap*: $(map\ f^{\mathcal{C}}\ w) \in lists\ \{f\ [(a :: binA)], f\ [1-a]\}$
using *map-core-lists*[*of f*, *unfolded bin-core-range-swap*[*of f a*]].

locale *binary-morphism* = *morphism f*
for $f :: binA\ list \Rightarrow 'a\ list$
begin

lemma *bin-len-count-im*:
fixes $a :: binA$
shows $|f\ w| = count\text{-}list\ w\ a * |f\ [a]| + count\text{-}list\ w\ (1-a) * |f\ [1-a]|$
proof (*induct w*)
case (*Cons b w*)
show *?case*
unfolding *hd-word*[*of b w*] *morph lenmorph Cons.hyps count-list-append*
by (*induct a*) *simp-all*
qed *simp*

lemma *bin-len-count-im'*:
shows $|f\ w| = count\text{-}list\ w\ bina * |f\ \mathfrak{a}| + count\text{-}list\ w\ binb * |f\ \mathfrak{b}|$
proof (*induct w*)
case (*Cons a w*)
show *?case*
unfolding *hd-word*[*of a w*] *morph lenmorph Cons.hyps count-list-append*
by (*induct a*) *simp-all*
qed *simp*

lemma *bin-neq-inj-core*: **assumes** $f\ [a] \neq f\ [1-a]$ **shows** $inj\ f^{\mathcal{C}}$
proof
show $f^{\mathcal{C}}\ x = f^{\mathcal{C}}\ y \Longrightarrow x = y$ **for** $x\ y$
proof (*rule ccontr*)
assume $x \neq y$
from *bin-sym-all-neq*[*OF assms*]

```
    have f^C x ≠ f^C y
      unfolding core-def bin-neq-swap''[OF ‹x ≠ y›].
    thus f^C x = f^C y ⟹ False
      by blast
  qed
qed

lemma bin-code-morphism-inj: assumes  f [a] · f [1−a] ≠ f [1−a] · f [a]
  shows inj f
  unfolding inj-on-def
proof (rule ballI, rule ballI, rule impI)
  have f [b]  ≠ f [1−b] for b
    using bin-sym-all-comm[OF assms, of b] by force
  from bin-neq-inj-core[OF this]
  have inj f^C.
  fix xs ys assume f xs = f ys
  hence concat (map f^C xs) = concat (map f^C ys)
    unfolding morph-concat-map.
  from bin-code-code[unfolded code-def, rule-format,
    OF ‹f [a] · f [1−a] ≠ f [1−a] · f [a]› bin-map-core-lists-swap bin-map-core-lists-swap
this]
  show xs = ys
    using ‹inj f^C› by simp
qed

lemma bin-code-morphismI: f [a] · f [1−a] ≠ f [1−a] · f [a] ⟹ code-morphism f
  using code-morphismI[OF bin-code-morphism-inj].

end
```

7.2.1 Binary periodic morphisms

```
locale binary-periodic-morphism = periodic-morphism f
  for f :: binA list ⇒ 'a list
begin

sublocale binary-morphism
  by unfold-locales

definition fn0 where fn0 ≡ (SOME n. f 𝔞 = mroot^@ n)
definition fn1 where fn1 ≡ (SOME n. f 𝔟 = mroot^@ n)

lemma bin0-im: f 𝔞 = mroot^@ fn0
  using per-morph-rootI[rule-format, of 𝔞] someI[of λ n. f 𝔞 = mroot^@ n] unfold-
ing fn0-def by blast

lemma  bin1-im: f 𝔟 = mroot^@ fn1
  using per-morph-rootI[rule-format, of 𝔟] someI[of λ n. f 𝔟 = mroot^@ n] unfold-
ing fn1-def by blast
```

lemma *sorted-image* : $f\ w = (f\ [a])^{@}(count\text{-}list\ w\ a) \cdot (f\ [1{-}a])^{@}(count\text{-}list\ w\ (1{-}a))$
proof−
obtain k **where** $f\ w = mroot^{@}k$
using *per-morph-rootI* **by** *blast*
have *len*: $|f\ w| = |(f\ [a])^{@}(count\text{-}list\ w\ a) \cdot (f\ [1{-}a])^{@}(count\text{-}list\ w\ (1{-}a))|$
using *bin-len-count-im* **unfolding** *lenmorph pow-len***.**
have $*$: $(f\ [a])^{@}(count\text{-}list\ w\ a) \cdot (f\ [1{-}a])^{@}(count\text{-}list\ w\ (1{-}a)) = mroot^{@}(fn0 * (count\text{-}list\ w\ bina) +\ fn1 * (count\text{-}list\ w\ binb))$
by (*induct a*) (*unfold binA-simps bin0-im bin1-im*, *unfold pow-mult*[*symmetric*] *pow-add*[*symmetric*], *simp-all add*: *add.commute*)
show *?thesis*
using *len nemp-len*[*OF prim-nemp*[*OF per-morph-root-prim*]]
unfolding $*$ ‹$f\ w = mroot^{@}k$› *pow-len* **by** *force*
qed

lemma *bin-per-morph-expI*: $f\ u = mroot^{@}((im\text{-}exp\ \mathfrak{a}) * (count\text{-}list\ u\ bina) + im\text{-}exp\ \mathfrak{b} * (count\text{-}list\ u\ binb))$
using *sorted-image*[*of u bina*, *unfolded binA-simps*]
by (*simp add*: *pow-add per-morph-im-exp pow-mult*)

end

7.3 From two words to a binary morphism

definition *bin-morph-of'* :: *'a list* ⇒ *'a list* ⇒ *binA list* ⇒ *'a list* **where** *bin-morph-of' x y u* = *concat* (*map* (λ *a*. (*case a of bina* ⇒ *x* | *binb* ⇒ *y*)) *u*)

definition *bin-morph-of* :: *'a list* ⇒ *'a list* ⇒ *binA list* ⇒ *'a list* **where** *bin-morph-of x y u* = *concat* (*map* (λ *a*. *if a* = *bina then x else y*) *u*)

lemma *case-finite-2-if-else*: *case-finite-2 x y* = (λ *a*. *if a* = *bina then x else y*)
by (*standard*, *simp split*: *finite-2.split*)

lemma *bin-morph-of-case-def*: *bin-morph-of x y u* = *concat* (*map* (λ *a*. (*case a of bina* ⇒ *x* | *binb* ⇒ *y*)) *u*)
unfolding *bin-morph-of-def case-finite-2-if-else***..**

lemma *case-finiteD*: *case-finite-2* $(f\ \mathfrak{a})\ (f\ \mathfrak{b}) = f^{\mathcal{C}}$
proof
show (*case x of bina* ⇒ $f\ \mathfrak{a}$ | *binb* ⇒ $f\ \mathfrak{b}$) = $f^{\mathcal{C}}\ x$ **for** *x*
unfolding *core-def* **by** (*cases rule*: *finite-2.exhaust*[*of x*]) *auto*
qed

lemma *case-finiteD'*: *case-finite-2* $(f\ \mathfrak{a})\ (f\ \mathfrak{b})\ u = f^{\mathcal{C}}\ u$
using *case-finiteD* **by** *metis*

lemma *bin-morph-of-maps*: *bin-morph-of x y* = *List.maps* (*case-finite-2 x y*)

unfolding *bin-morph-of-def List.maps-eq* **unfolding** *case-finite-2-if-else* **by** *simp*

lemma *bin-morph-ofD*: (*bin-morph-of x y*) 𝔞 = *x* (*bin-morph-of x y*) 𝔟 = *y*
unfolding *bin-morph-of-def* **by** *simp-all*

lemma *bin-range-swap*: *range f* = {*f* (*a*::*binA*), *f* (*1*−*a*)} (**is** *?P a*)
using *bin-swap-induct*[*of ?P bina*] **unfolding** *binA-simps bin-UNIV* **by** *auto*

lemma *bin-morph-of-core-range*: *range* (*bin-morph-of x y*)$^{\mathcal{C}}$ = {*x*,*y*}
unfolding *bin-core-range bin-morph-ofD*..

lemma *bin-morph-of-morph*: *morphism* (*bin-morph-of x y*)
unfolding *bin-morph-of-def* **by** (*simp add*: *morphism.intro*)

lemma *bin-morph-of-bin-morph*: *binary-morphism* (*bin-morph-of x y*)
unfolding *binary-morphism-def* **using** *bin-morph-of-morph*.

lemma *bin-morph-of-range*: *range* (*bin-morph-of x y*) = ⟨{*x*,*y*}⟩
using *morphism.range-hull*[*of bin-morph-of x y*, *unfolded bin-morph-of-core-range*, *OF bin-morph-of-morph*].

context *binary-code*
begin

lemma *code-morph-of*: *code-morphism* (*bin-morph-of* u_0 u_1)
using *binary-morphism.bin-code-morphismI*[*OF bin-morph-of-bin-morph*, *of* u_0 u_1 *bina*]
unfolding *binA-simps bin-morph-ofD* **using** *non-comm*.

lemma *inj-morph-of*: *inj* (*bin-morph-of* u_0 u_1)
using *code-morphism.code-morph*[*OF code-morph-of*].

end

7.4 Two binary morphism

locale *two-binary-morphisms* = *two-morphisms g h*
for *g h* :: *binA list* ⇒ *'a list*

begin

lemma *eq-on-letters-eq*: *g* 𝔞 = *h* 𝔞 ⟹ *g* 𝔟 = *h* 𝔟 ⟹ *g* = *h*
by (*rule def-on-sings-eq*, *rule bin-induct*) *blast*+

sublocale *g*: *binary-morphism g*
by *unfold-locales*
sublocale *h*: *binary-morphism h*
by *unfold-locales*

lemma *rev-morphs*: *two-binary-morphisms* (*rev-map g*) (*rev-map h*)
using *rev-maps* **by** (*intro two-binary-morphisms.intro*)

lemma *solution-UNIV*:
assumes $s \neq \varepsilon$ **and** $g\ s = h\ s$ **and** $\bigwedge a.\ g\ [a] \neq h\ [a]$
shows *set s* = *UNIV*
proof (*rule ccontr*, *elim not-UNIV-E sing-set-wordE*)
fix *a k* **assume** $*$: $s = [a] \ ^{@}\ k$
then have $0 < k$ **using** ‹$s \neq \varepsilon$› **by** *blast*
have $g\ [a] = h\ [a]$ **using** ‹$g\ s = h\ s$› **unfolding** $*$ *g.pow-morph h.pow-morph*
by (*fact pow-eq-eq*[*OF* - ‹$0 < k$›])
then show *False* **using** ‹$g\ [a] \neq h\ [a]$› **by** *contradiction*
qed

lemma *solution-len-im-sing-less*:
assumes *sol*: $g\ s = h\ s$ **and** *set*: $a \in$ *set s* **and** *less*: $|g\ [a]| < |h\ [a]|$
shows $|h\ [1-a]| < |g\ [1-a]|$
proof (*intro not-le-imp-less notI*)
assume $|g\ [1-a]| \leq |h\ [1-a]|$
with *less-imp-le*[*OF less*] **have** $|g\ [b]| \leq |h\ [b]|$ **for** *b*
by (*fact bin-swap-induct*)
from *this set less*
have $|g\ s| < |h\ s|$ **by** (*rule len-im-less*[*of s*])
then show *False* **using** *lenarg*[*OF sol*] **by** *simp*
qed

lemma *solution-len-im-sing-le*:
assumes *sol*: $g\ s = h\ s$ **and** *set*: *set s* = *UNIV* **and** *less*: $|g\ [a]| \leq |h\ [a]|$
shows $|h\ [1-a]| \leq |g\ [1-a]|$
proof (*intro leI notI*)
assume $|g\ [1-a]| < |h\ [1-a]|$
from *solution-len-im-sing-less*[*OF sol* - *this*]
have $|h\ [a]| < |g\ [a]|$ **unfolding** *set binA-simps* **by** *blast*
then show *False* **using** ‹$|g\ [a]| \leq |h\ [a]|$› **by** *simp*
qed

lemma *solution-sing-len-cases*:
assumes *set*: *set s* = *UNIV* **and** *sol*: $g\ s = h\ s$ **and** $g \neq h$
obtains *a* **where** $|g\ [a]| < |h\ [a]|$ **and** $|h\ [1-a]| < |g\ [1-a]|$
proof (*cases rule*: *linorder-cases*)
show $|g\ [hd\ s]| < |h\ [hd\ s]| \Longrightarrow$ *thesis*
using *solution-len-im-sing-less*[*OF sol*] *that* **unfolding** *set* **by** *blast*
interpret *swap*: *two-binary-morphisms h g* **by** *unfold-locales*
show $|h\ [hd\ s]| < |g\ [hd\ s]| \Longrightarrow$ *thesis*
using *swap.solution-len-im-sing-less*[*OF sol*[*symmetric*]]
solution-len-im-sing-less[*OF sol*] *that* **unfolding** *set* **by** *blast*
have $s \neq \varepsilon$ **using** *set* **by** *auto*
assume $*$: $|g\ [hd\ s]| = |h\ [hd\ s]|$
moreover have $|g\ [1 - (hd\ s)]| = |h\ [1 - (hd\ s)]|$

proof (*rule ccontr*, *elim linorder-neqE*)
show $|g\ [1 - (hd\ s)]| < |h\ [1 - (hd\ s)]| \Longrightarrow False$
using *solution-len-im-sing-less*[*OF sol*, *of* $1 - (hd\ s)$]
unfolding *set binA-simps* $*$ **by** *blast*
next
show $|h\ [1-(hd\ s)]| < |g\ [1-(hd\ s)]| \Longrightarrow False$
using *swap.solution-len-im-sing-less*[*OF sol*[*symmetric*], *of* $1 - (hd\ s)$]
unfolding *set binA-simps* $*$ **by** *blast*
qed
ultimately have $|g\ [a]| = |h\ [a]|$ **for** a **by** (*fact bin-swap-induct*)
from *def-on-sings*[*OF solution-eq-len-eq*[*OF sol this*]]
show *thesis* **unfolding** *set* **using** $\langle g \neq h \rangle$ **by** *blast*
qed

lemma *len-ims-sing-neq*:
assumes $g\ s = h\ s\ g \neq h\ set\ s = binUNIV$
shows $|g\ [c]| \neq |h\ [c]|$
proof(*rule solution-sing-len-cases*[*OF* $\langle set\ s = binUNIV \rangle$ $\langle g\ s\ = h\ s \rangle$ $\langle g \neq h \rangle$])
fix a **assume** *less*: $|g\ [a]| < |h\ [a]|\ |h\ [1 - a]| < |g\ [1 - a]|$
show $|g\ [c]| \neq |h\ [c]|$
by (*rule bin-swap-exhaust*[*of c a*]) (*use less* **in** *force*)+
qed

end

lemma *two-binary-morphismsI*: *binary-morphism g* $\Longrightarrow$ *binary-morphism h* $\Longrightarrow$ *two-binary-morphisms g h*
unfolding *binary-morphism-def two-binary-morphisms-def* **using** *two-morphisms.intro*.

7.5 Binary code morphism

7.5.1 Locale - binary code morphism

locale *binary-code-morphism* = *code-morphism f* :: *binA list* $\Rightarrow$ *'a list* **for** *f*

begin

lemma *morph-bin-morph-of*: $f = bin\text{-}morph\text{-}of\ (f\ \mathfrak{a})\ (f\ \mathfrak{b})$
using *morph-concat-map* **unfolding** *bin-morph-of-def case-finiteD*
case-finite-2-if-else[*symmetric*] **by** *simp*

lemma *non-comm-morph* [*simp*]: $f\ [a] \cdot f\ [1-a] \neq f\ [1-a] \cdot f\ [a]$
unfolding *morph*[*symmetric*] **using** *code-morph-code bin-swap-neq* **by** *blast*

lemma *non-comp-morph*: $\neg\ f\ [a] \cdot f\ [1-a] \bowtie f\ [1-a] \cdot f\ [a]$
using *comm-comp-eq non-comm-morph* **by** *blast*

lemma *swap-non-comm-morph* [*simp*, *intro*]: $a \neq b \Longrightarrow f\ [a] \cdot f\ [b] \neq f\ [b] \cdot f\ [a]$
using *bin-neq-swap non-comm-morph* **by** *blast*

thm *bin-core-range*[*of f*]

lemma *bin-code-morph-rev-map*: *binary-code-morphism* (*rev-map f*)
unfolding *binary-code-morphism-def* **using** *code-morphism-rev-map*.

sublocale *swap*: *binary-code f* 𝔟 *f* 𝔞
using *non-comm-morph*[*of binb*] **unfolding** *binA-simps* **by** *unfold-locales*

sublocale *binary-code f* 𝔞 *f* 𝔟
using *swap.bin-code-swap*.

notation *bin-code-lcp* (α) **and**
bin-code-lcs (β) **and**
bin-code-mismatch-fst (c_0) **and**
bin-code-mismatch-snd (c_1)
term *bin-lcp* (*f* 𝔞) (*f* 𝔟)
abbreviation *bin-morph-mismatch* (𝔠)
where *bin-morph-mismatch a* ≡ *bin-mismatch* (*f*[*a*]) (*f*[*1*−*a*])
abbreviation *bin-morph-mismatch-suf* (𝔡)
where *bin-morph-mismatch-suf a* ≡ *bin-mismatch-suf* (*f*[*1*−*a*]) (*f*[*a*])

lemma *bin-lcp-def'*: $\alpha = f([a] \cdot [1-a]) \wedge_p f([1-a] \cdot [a])$
by (*rule bin-exhaust*[*of a* $\alpha = f([a] \cdot [1-a]) \wedge_p f([1-a] \cdot [a])$],
unfold morph, *use binA-simps*(*3*−*4*) *bin-lcp-def* **in** *force*)
(*unfold bin-lcp-def lcp-sym*[*of f*[*a*] · *f*[*1*−*a*] *f*[*1*−*a*] · *f*[*a*]],
use binA-simps(*3*−*4*) **in** *auto*)

lemma *bin-lcp-neq*: $a \neq b \Longrightarrow \alpha = f([a] \cdot [b]) \wedge_p f([b] \cdot [a])$
using *bin-neq-swap*[*of a b*] **unfolding** *bin-lcp-def'*[*of a*] **by** *blast*

lemma *sing-im*: *f* [*a*] ∈ {*f* 𝔞, *f* 𝔟}
using *finite-2.exhaust*[*of a ?thesis*] **by** *fastforce*

lemma *bin-mismatch-inj*: *inj* 𝔠
unfolding *inj-on-def*
using *non-comm-morph*[*folded bin-mismatch-comm*] *bin-neq-swap* **by** *force*

lemma *map-in-lists*: *map* (λ*x*. *f* [*x*]) *w* ∈ *lists* {*f* 𝔞, *f* 𝔟}
proof (*induct w*)
case (*Cons a w*)
then show *?case*
unfolding *list.map*(*2*) **using** *sing-im* **by** *simp*
qed *simp*

lemma *bin-morph-lcp-short*: |α| < |*f* [*a*]| + |*f*[*1*−*a*]|
using *finite-2.exhaust*[*of a ?thesis*] *bin-lcp-short* **by** *force*

lemma *swap-not-pref-bin-lcp*: ¬ *f*([*a*] · [*1*−*a*]) ≤*p* α

using *pref-len*[*of f* [*a*] · *f* [*1*−*a*] *α*] **unfolding** *morph lenmorph* **using** *bin-morph-lcp-short*[*of a*] **by** *force*

thm *local.bin-mismatch-inj*

lemma *bin-mismatch-suf-inj*: *inj* 𝔡
using *binary-code-morphism.bin-mismatch-inj*[*OF bin-code-morph-rev-map*, *reversed*]**.**

lemma *bin-lcp-sing*: *bin-lcp* (*f* [*a*]) (*f* [*1*−*a*]) = *α*
unfolding *bin-lcp-def*
by (*rule finite-2.exhaust*[*of a*], *simp-all add*: *lcp-sym*)

lemma *bin-lcs-sing*: *bin-lcs* (*f* [*a*]) (*f* [*1*−*a*]) = *β*
unfolding *bin-lcs-def*
by (*rule finite-2.exhaust*[*of a*], *simp-all add*: *lcs-sym*)

lemma *bin-code-morph-sing*: *binary-code* (*f* [*a*]) (*f* [*1*−*a*])
unfolding *binary-code-def*
by (*cases rule*: *binA-neq-cases*[*OF bin-swap-neq′*, *of a*]) *simp-all*

lemma *bin-mismatch-swap-neq*: 𝔠 *a* ≠ 𝔠 (*1*−*a*)
using *bin-code-morph-sing binary-code.bin-mismatch-neq* **by** *auto*

lemma *set w* ⊆ {*hd w*} ⟷ (∃ *k*. *w* = [*hd w*]^@*k*)
using *sing-pow-set′ sing-set-pow* **by** *metis*

lemma *long-bin-lcp-hd*: **assumes** |*f w*| ≤ |*α*|
shows *set w* ⊆ {*hd w*}
proof (*rule ccontr*)
assume ¬ *set w* ⊆ {*hd w*}
from *distinct-letter-in-hd*[*OF this*]
obtain *m b suf* **where** *w*: [*hd w*]^@*m* · [*b*]· *suf* = *w* **and** *b* ≠ *hd w* **and** *m* ≠ *0***.**
have *ineq*: |*f* [*b*]| + |*f* [*hd w*]| ≤ |*f w*|
using *quotient-smaller*[*OF* ‹*m* ≠ *0*›, *of* |*f* [*hd w*]|]
unfolding *arg-cong*[*OF w*, *of* λ *x*. |*f*(*x*)|, *unfolded morph lenmorph pow-morph pow-len*, *symmetric*]
by *linarith*
hence |*f* 𝔞| + |*f* 𝔟| ≤ |*f w*|
using ‹*b* ≠ *hd w*› *alphabet-or*[*of b*] *alphabet-or*[*of hd w*] *add.commute* **by** *fastforce*
thus *False*
using *bin-lcp-short* ‹|*f w*| ≤ |*α*|› **by** *linarith*
qed

thm *nonerasing*
nonerasing-morphism.nonerasing
lemmas *nonerasing* = *nonerasing*
thm *nonerasing-morphism.nonerasing*

binary-code-morphism.nonerasing

lemma *bin-morph-lcp-mismatch-pref*:
$\alpha \cdot [\mathfrak{c}\ a] \leq p\ f\ [a] \cdot \alpha$
using *binary-code.bin-fst-mismatch*[*OF bin-code-morph-sing*] **unfolding** *bin-lcp-sing*.

lemma $[\mathfrak{d}\ a] \cdot \beta \leq s\ \beta \cdot f\ [a]$ **using** *binary-code-morphism.bin-morph-lcp-mismatch-pref*[*OF bin-code-morph-rev-map*, *reversed*].

lemma *bin-lcp-pref-all*: $\alpha \leq p\ f\ w \cdot \alpha$
proof(*induct w rule*: *rev-induct*)
case (*snoc x xs*)
from *pref-prolong*[*OF this*, *of* $f[x] \cdot \alpha$, *unfolded lassoc*]
show *?case*
unfolding *morph*[*of xs* [*x*]] **using** *bin-lcp-fst-lcp bin-lcp-snd-lcp alphabet-or*[*of x*] **by** *blast*
qed *simp*

lemma *bin-lcp-spref-all*: $w \neq \varepsilon \Longrightarrow \alpha <p\ f\ w \cdot \alpha$
using *per-rootI*[*OF bin-lcp-pref-all*] *nemp-to-nemp* **by** *presburger*

lemma *pref-mono-lcp*: **assumes** $w \leq p\ w'$ **shows** $f\ w \cdot \alpha \leq p\ f\ w' \cdot \alpha$
proof–
from ‹$w \leq p\ w'$›
obtain *z* **where** $w' = w \cdot z$
unfolding *prefix-def* **by** *fast*
show *?thesis*
unfolding ‹$w' = w \cdot z$› *morph rassoc pref-cancel-conv* **using** *bin-lcp-pref-all*.
qed

lemma *long-bin-lcp*: **assumes** $|f\ w| \leq |\alpha|$
shows *set* $w \subseteq \{hd\ w\}$
proof(*rule ccontr*)
assume ¬ *set* $w \subseteq \{hd\ w\}$
obtain *m b q* **where** $[hd\ w]^{@}m \cdot [b] \cdot q = w$ **and** $b \neq hd\ w$ **and** $m \neq 0$
using *distinct-letter-in-hd*[*OF* ‹¬ *set* $w \subseteq \{hd\ w\}$›].
have *ineq*: $|f\ ([hd\ w]^{@}m \cdot [b])| \leq |f\ w|$
using *arg-cong*[*OF* ‹$[hd\ w]^{@}\ m \cdot [b] \cdot q = w$›, *of* $\lambda\ x.\ |f\ x|$]
unfolding *morph lenmorph* **by** *force*
have *eq*: $m*|f\ [hd\ w]| + |f\ [b]| = |f\ ([hd\ w]^{@}m \cdot [b])|$
by (*simp add*: *morph pow-len pow-morph*)
have $|f\ [hd\ w]| + |f\ [b]| \leq m*|f\ [hd\ w]| + |f\ [b]|$
using *ineq* ‹$m \neq 0$› **by** *simp*
hence $|f\ [hd\ w]| + |f\ [b]| \leq |f\ w|$
using *eq ineq* **by** *linarith*
hence $|f\ \mathfrak{a}| + |f\ \mathfrak{b}| \leq |f\ w|$
using *binA-neq-cases* [*OF* ‹$b \neq hd\ w$›] **by** *fastforce*
thus *False*
using *bin-lcp-short* ‹$|f\ w| \leq |\alpha|$› **by** *linarith*

qed

thm *sing-to-nemp*
nonerasing

lemma *bin-mismatch-code-morph*: $c_0 = \mathfrak{c}\ 0\ c_1 = \mathfrak{c}\ 1$
unfolding *bin-mismatch-def bin-lcp-def* **by** *simp-all*

lemma *bin-lcp-mismatch-pref-all*: $\alpha \cdot [\mathfrak{c}\ a] \leq p\ f\ [a] \cdot f\ w \cdot \alpha$
using *pref-prolong*[*OF bin-fst-mismatch bin-lcp-pref-all*[*of w*]]
pref-prolong[*OF bin-snd-mismatch bin-lcp-pref-all*[*of w*]]
unfolding *bin-mismatch-code-morph*
by (*cases rule*: *finite-2.exhaust*[*of a*]) *simp-all*

lemma *bin-fst-mismatch-all*: $\alpha \cdot [c_0] \leq p\ f\ \mathfrak{a} \cdot f\ w \cdot \alpha$
using *pref-prolong*[*OF bin-fst-mismatch bin-lcp-pref-all*[*of w*]]**.**

lemma *bin-snd-mismatch-all*: $\alpha \cdot [c_1] \leq p\ f\ \mathfrak{b} \cdot f\ w \cdot \alpha$
using *pref-prolong*[*OF bin-snd-mismatch bin-lcp-pref-all*[*of w*]] **by** *simp*

lemma *bin-long-mismatch*: **assumes** $|\alpha| < |f\ w|$ **shows** $\alpha \cdot [\mathfrak{c}\ (hd\ w)] \leq p\ f\ w$
proof–
have $w \neq \varepsilon$
using *assms emp-to-emp emp-len* **by** *force*
have $f\ w = f[hd\ w] \cdot f\ (tl\ w)$
unfolding *pop-hd*[*symmetric*] **unfolding** *hd-word*[*of hd w tl w*]
hd-tl[*OF* ‹$w \neq \varepsilon$›]**..**
have $\alpha \cdot [\mathfrak{c}\ (hd\ w)] \leq p\ f\ w \cdot \alpha$
using *bin-lcp-mismatch-pref-all*[*of hd w tl w*]
unfolding *lassoc* ‹$f\ w = f[hd\ w] \cdot f\ (tl\ w)$›[*symmetric*]**.**
moreover have $|\alpha \cdot [\mathfrak{c}\ (hd\ w)]| \leq |f\ w|$
unfolding *lenmorph sing-len* **using** *assms* **by** *linarith*
ultimately show *?thesis* **by** *blast*
qed

lemma *sing-pow-mismatch*: **assumes** $f\ [a] = [b]^{@} Suc\ n$ **shows** $\mathfrak{c}\ a = b$
proof–
— auxiliary
have *aritm*: $Suc\ n * Suc\ |\alpha| = Suc\ (n{*}|\alpha| + n + |\alpha|)$
by *auto*
have *set*: $set\ ([b]\ ^{@}\ (Suc\ n * Suc\ |\alpha|)) = \{b\}$
unfolding *aritm* **using** *sing-pow-set-Suc***.**
have *elem*: $\mathfrak{c}\ a \in set\ (\alpha \cdot [\mathfrak{c}\ a])$
by *simp*
have *hd*: $hd\ ([a]\ ^{@}\ Suc\ |\alpha|) = a$
by *fastforce*
— proof
let *?w* = $[a]^{@} Suc\ |\alpha|$
have *fw*: $f\ ?w = [b]^{@}(Suc\ n{*}Suc\ |\alpha|)$

unfolding *pow-mult assms*[*symmetric*] *pow-morph*..
have $|\alpha| < |f\ ?w|$
unfolding *fw pow-len sing-len* **by** *force*
from *set-mono-prefix*[*OF bin-long-mismatch*[*OF this*, *unfolded fw*]]
show $\mathfrak{c}\ a = b$
unfolding *hd set* **using** *elem* **by** *blast*
qed

lemma *sing-pow-mismatch-suf*: $f\ [a] = [b] \hat{@} Suc\ n \Longrightarrow \mathfrak{d}\ a = b$
using *binary-code-morphism.sing-pow-mismatch*[*OF bin-code-morph-rev-map*, *reversed*].

lemma *bin-lcp-swap-hd*: $f\ [a] \cdot f\ w \cdot \alpha\ \wedge_p f\ [1-a] \cdot f\ w' \cdot \alpha = \alpha$
using *lcp-first-mismatch*[*OF bin-mismatch-swap-neq*, *of* α *a*]
bin-lcp-mismatch-pref-all[*of a w*] *bin-lcp-mismatch-pref-all*[*of* $1-a$ w']
unfolding *prefix-def rassoc* **by** *force*

lemma *bin-lcp-neq-hd*: $a \neq b \Longrightarrow f\ [a] \cdot f\ w \cdot \alpha\ \wedge_p f\ [b] \cdot f\ w' \cdot \alpha = \alpha$
using *bin-lcp-swap-hd bin-neq-swap* **by** *blast*

lemma *bin-mismatch-swap-not-comp*: $\neg\ f\ [a] \cdot f\ w \cdot \alpha\ \bowtie f\ [1-a] \cdot f\ w' \cdot \alpha$
unfolding *prefix-comparable-def lcp-pref-conv*[*symmetric*] *bin-lcp-swap-hd*
bin-lcp-swap-hd[*of* $1-a$, *unfolded binA-simps*] **using** *sing-to-nemp* **by** *auto*

lemma *bin-lcp-root*: $\alpha <p f\ [a] \cdot \alpha$
using *alphabet-or*[*of a*] *per-rootI*[*OF bin-lcp-pref-all*[*of* $\mathfrak{b}$] *bin-snd-nemp*] *per-rootI*[*OF bin-lcp-pref-all*[*of* $\mathfrak{a}$] *bin-fst-nemp*] **by** *blast*

lemma *bin-lcp-pref*: **assumes** *set w* = *binUNIV*
shows $\alpha \leq p\ (f\ w)$
using *pref-prod-le*[*OF bin-lcp-pref-all -*, *of w*] *long-bin-lcp*[*of w*]
assms[*folded set-binUNIV-iff*] **by** *linarith*

lemma *bin-lcp-pref''*: $[a] \leq f\ w \Longrightarrow [1-a] \leq f\ w\ \Longrightarrow \alpha \leq p\ (f\ w)$
using *bin-lcp-pref*[*of w*] *sing-pow-fac*[*OF bin-distinct*(*1*), *of w*] *sing-pow-fac*[*OF bin-distinct*(*2*), *of w*]
apply (*cases rule*: *finite-2.exhaust*[*of a*])
apply (*simp add*: *sing-fac-set*)
by (*meson bin-UNIV-I sing-fac-set*)
lemma *bin-lcp-pref'*: $\mathfrak{a} \leq f\ w \Longrightarrow \mathfrak{b} \leq f\ w\ \Longrightarrow \alpha \leq p\ (f\ w)$
using *bin-lcp-pref''*[*of bina*, *unfolded binA-simps*].

lemma *bin-lcp-mismatch-pref-all-set*: **assumes** $1-a \in$ *set w*
shows $\alpha \cdot [\mathfrak{c}\ a] \leq p\ f\ [a] \cdot f\ w$
proof–
have $|f[1-a]| \leq |f\ w|$

```
    using fac-len' morph split-list'[OF assms] by metis
  hence |α · [𝔠 a]| ≤ |f [a] · f w|
    using bin-lcp-short unfolding lenmorph sing-len
    by (cases rule: finite-2.exhaust[of a]) fastforce+
  from bin-lcp-mismatch-pref-all[unfolded lassoc, THEN pref-prod-le, OF this]
  show ?thesis.
qed

lemma bin-lcp-comp-hd: α ⋈ f (𝔞 · w0) ∧p f (𝔟 · w1)
  using ruler[OF bin-lcp-pref-all[of 𝔞 · w0]
      pref-trans[OF lcp-pref[of f (𝔞 · w0) f (𝔟 · w1)], of f (𝔞 · w0) · α, OF triv-pref]]
  unfolding prefix-comparable-def.

lemma sing-mismatch: assumes set (f 𝔞) ⊆ {a} shows c0 = a
proof−
  have set α ⊆ {a}
    using per-one'[OF assms bin-lcp-root[of bina]] by blast
  hence set (f 𝔞 · α) ⊆ {a}
    using ‹set (f 𝔞) ⊆ {a}› by simp
  from sing-pow-fac[OF - this, of c0]
  show c0 = a
    using facI'[OF lq-pref[OF bin-fst-mismatch, unfolded rassoc]] by blast
qed

lemma sing-mismatch': assumes set (f 𝔟) ⊆ {a} shows c1 = a
proof−
  have set α ⊆ {a}
    using per-one'[OF assms bin-lcp-root[of binb]] by blast
  hence set (f 𝔟 · α) ⊆ {a}
    using ‹set (f 𝔟) ⊆ {a}› by simp
  from sing-pow-fac[OF - this, of c1]
  show c1 = a
    using facI'[OF lq-pref[OF bin-snd-mismatch, unfolded rassoc]] by blast
qed

lemma bin-lcp-comp-all: α ⋈ (f w)
  unfolding prefix-comparable-def using ruler[OF bin-lcp-pref-all triv-pref].

lemma not-comp-bin-swap: ¬ f [a] · α ⋈ f [1−a] · α
  using not-comp-bin-fst-snd bin-exhaust[of a ?thesis]
  unfolding prefix-comparable-def
  by simp

lemma mismatch-pref:
  assumes α ≤p f ([a] · w0) and α ≤p f ([1−a] · w1)
  shows α = f ([a] · w0) ∧p f ([1−a] · w1)
proof−
  have f ([a] · w0) · α ∧p f ([1−a] · w1) · α = α
    unfolding morph using bin-lcp-swap-hd[unfolded lassoc].
```

hence $f\ ([a] \cdot w0) \wedge_p f\ ([1-a] \cdot w1) \leq p\ \alpha$
using *lcp.mono*[*OF triv-pref*[*of f* $([a] \cdot w0)\ \alpha$] *triv-pref*[*of f* $([1-a] \cdot w1)\ \alpha$]]
by *presburger*
moreover have $\alpha \leq p\ f\ ([a] \cdot w0) \wedge_p f\ ([1-a] \cdot w1)$
using *assms pref-pref-lcp* **by** *blast*
ultimately show *?thesis*
using *pref-antisym* **by** *blast*
qed

lemma *bin-set-UNIV-length*: **assumes** *set w* = *UNIV* **shows** $|f\ [a]| + |f\ [1-a]| \leq |f\ w|$
proof−
have $w \neq \varepsilon$
using ‹*set w* = *UNIV*› **by** *force*
from *set-ConsD*[*of 1*− *hd w hd w tl w*, *unfolded list.collapse*[*OF this*] *assms*[*folded swap-UNIV*[*of hd w*]]]
have $1 - (hd\ w) \in set\ (tl\ w)$
using *bin-swap-neq*[*of hd w*] **by** *blast*
from *in-set-morph-len*[*OF this*]
have $|f\ [1-hd\ w]| \leq |f\ (tl\ w)|$.
with *lenarg*[*OF arg-cong*[*of - - f*, *OF hd-tl*[*OF* ‹$w \neq \varepsilon$›]]]
have $|f\ [hd\ w]| + |f\ [1-hd\ w]| \leq |f\ w|$
unfolding *morph lenmorph* **by** *linarith*
thus *?thesis*
using *bin-swap-exhaust*[*of a hd w ?thesis*] **by** *force*
qed

lemma *set-UNIV-bin-lcp-pref*: **assumes** *set w* = *UNIV* **shows** $\alpha \cdot [\mathfrak{c}\ (hd\ w)] \leq p\ f\ w$
using *bin-long-mismatch*[*OF less-le-trans*[*OF bin-morph-lcp-short bin-set-UNIV-length*[*OF assms*]]]**.**

lemmas *not-comp-bin-lcp-pref* = *bin-not-comp-set-UNIV*[*THEN set-UNIV-bin-lcp-pref*]

lemma *marked-lcp-conv*: *marked-morphism f* ⟷ $\alpha = \varepsilon$
proof
assume *marked-morphism f*
then interpret *marked-morphism f* **by** *blast*
from *marked-core*[*unfolded core-def*] *core-nemp*[*unfolded core-def*]
have $hd\ (f\ \mathfrak{a} \cdot f\ \mathfrak{b}) \neq hd\ (f\ \mathfrak{b} \cdot f\ \mathfrak{a})$
using *hd-append finite-2.distinct* **by** *auto*
thus $\alpha = \varepsilon$
unfolding *bin-lcp-def* **using** *lcp-distinct-hd* **by** *blast*
next
assume $\alpha = \varepsilon$
have $hd\ (f\ \mathfrak{a}) \neq hd\ (f\ \mathfrak{b})$
by (*rule nemp-lcp-distinct-hd*[*OF sing-to-nemp sing-to-nemp*])
(*use lcp-append-monotone*[*of f* $\mathfrak{a}$ *f* $\mathfrak{b}$ *f* $\mathfrak{b}$ *f* $\mathfrak{a}$, *unfolded* ‹$\alpha = \varepsilon$›[*unfolded bin-lcp-def*]]

```
    in simp)
  show marked-morphism f
  proof
    fix a b :: binA assume hd (f^C a) = hd (f^C b)
    from im-swap-neq[OF this[unfolded core-def] ‹hd (f 𝔞) ≠ hd (f 𝔟)›]
    show a = b.
  qed
qed

lemma im-comm-lcp: f w · α = α · f w ⟹ (∀ a. a ∈ set w ⟶ f [a] · α = α · f
[a])
proof (induct w)
  case (Cons a w)
  then show ?case
  proof (cases w = ε)
    assume w = ε
    show ?thesis
      using Cons.prems(1) unfolding ‹w = ε› by force
  next
    assume w ≠ ε
    have eq: f [a] · f w · α = α · f [a] · f w
      unfolding morph[symmetric]
      unfolding lassoc morph[symmetric] hd-tl[OF ‹w ≠ ε›]
      using ‹f (a # w) · α = α · f (a # w)› by force
    have f [a] · α ≤p f [a] · f w · α
      unfolding pref-cancel-conv using bin-lcp-pref-all.
    hence f [a] · α = α · f [a]
      using eqd-eq[of α · f [a], OF -swap-len] unfolding prefix-def eq rassoc by
metis
    from eq[unfolded lassoc, folded this, unfolded rassoc cancel]
    have f w · α = α · f w.
    from Cons.hyps[OF this] ‹f [a] · α = α · f [a]›
    show ?thesis by fastforce
  qed
qed simp

lemma im-comm-lcp-nemp: assumes f w · α = α · f w and w ≠ ε and α ≠ ε
  obtains k where w = [hd w]^@k and 0 < k
proof–
  have set w = {hd w}
  proof–
    have hd w ∈ set w using ‹w ≠ ε› by force
    have a = hd w if a ∈ set w for a
    proof–
      have f [a] · α = α · f [a] and f [hd w] · α = α · f [hd w]
     using that im-comm-lcp[OF ‹f w · α = α · f w›] ‹hd w ∈ set w› by presburger+
    from comm-trans[OF this ‹α ≠ ε›]
    show a = hd w
      using swap-non-comm-morph by blast
```

```
    qed
    thus set w = {hd w}
      using ‹hd w ∈ set w› by blast
  qed
  from sing-set-wordE'[OF this]
  show thesis
    using that by blast
qed

lemma bin-lcp-ims-im-lcp: f w · α ∧p f w' · α = f (w ∧p w') · α
proof (cases w ⋈ w')
  assume w ⋈ w'
  from disjE[OF this[unfolded prefix-comparable-def]]
  consider w ≤p w' | w' ≤p w by blast
  thus ?thesis
  proof (cases)
    assume w ≤p w'
    hence f w · α ≤p f w' · α
      using pref-mono-lcp by blast
    from this[folded lcp-pref-conv]
    show ?thesis
      unfolding ‹w ≤p w'›[folded lcp-pref-conv].
  next
    assume w' ≤p w
    hence f w' · α ≤p f w · α
      using pref-mono-lcp by blast
    from this[folded lcp-pref-conv]
    show ?thesis
    unfolding lcp-sym[of f w · α] ‹w' ≤p w›[folded lcp-pref-conv, unfolded lcp-sym[of
w']].
  qed
next
  assume ¬ w ⋈ w'
  from lcp-mismatchE[OF this]
  obtain ws ws' where (w ∧p w') · ws = w (w ∧p w') · ws' = w'
        ws ≠ ε ws' ≠ ε hd ws ≠ hd ws'.
  note hd-tl[OF ‹ws ≠ ε›] hd-tl[OF ‹ws' ≠ ε›]
  have w: (w ∧p w') · [hd ws] · tl ws = w
    using ‹(w ∧p w') · ws = w› ‹[hd ws] · tl ws = ws› by auto
  have w': (w ∧p w') · [hd ws'] · tl ws' = w'
    using ‹(w ∧p w') · ws' = w'› ‹[hd ws'] · tl ws' = ws'› by auto
  have f((w ∧p w') · [hd ws] · tl ws) · α ∧p f((w ∧p w') · [hd ws'] · tl ws') · α =
       f(w ∧p w') · (f ([hd ws] · tl ws) · α ∧p f([hd ws'] · tl ws') · α)
    unfolding morph using lcp-ext-left by auto
  thus ?thesis
    unfolding w w' bin-lcp-neq-hd[OF ‹hd ws ≠ hd ws'›, folded rassoc morph].
qed

lemma per-comp:
```

assumes $r <p\ f\ w \cdot r$
shows $r \bowtie f\ w \cdot \alpha$
using *assms*
proof
have $r \leq p\ f\ w \cdot r\ f\ w \neq \varepsilon$
using *assms* **by** *force+*
from *per-root-powE*[*OF this*]
obtain n **where** $r <p\ f\ w^{@}n\ 0 < n$**.**
have $f\ w \cdot \alpha \leq p\ f\ w \cdot f\ w\ ^{@}\ (n - 1) \cdot \alpha$
using *bin-lcp-pref-all*[*of* $w^{@}(n-1)$]
unfolding *pref-cancel-conv pow-morph***.**
with *ruler*[*OF pref-ext*[*OF sprefD1*[*OF* ‹$r <p\ f\ w^{@}n$›], *of* α], *of* $f\ w \cdot \alpha$]
show *?thesis*
unfolding *prefix-comparable-def pow-pos*[*OF* ‹$0 < n$›] *rassoc***.**
qed

end

7.5.2 More translations

lemma *bin-code-morph-iff'*: *binary-code-morphism* $f \longleftrightarrow$ *morphism* $f \wedge f\ [a] \cdot f\ [1-a] \neq f\ [1-a] \cdot f\ [a]$
proof
assume *binary-code-morphism f*
hence *morphism f*
by (*simp add*: *binary-code-morphism-def code-morphism-def*)
have $f\ [a] \cdot f\ [1-a] \neq f\ [1-a] \cdot f\ [a]$
using ‹*binary-code-morphism f*› *binary-code-morphism.non-comm-morph* **by** *auto*
thus *morphism* $f \wedge f\ [a] \cdot f\ [1-a] \neq f\ [1-a] \cdot f\ [a]$
using ‹*morphism f*› **by** *blast*
next
assume *morphism* $f \wedge f\ [a] \cdot f\ [1-a] \neq f\ [1-a] \cdot f\ [a]$
hence *morphism f* **and** $f\ [a] \cdot f\ [1-a] \neq f\ [1-a] \cdot f\ [a]$ **by** *force+*
from *binary-code-morphism.intro*[*OF binary-morphism.bin-code-morphismI*[*OF binary-morphism.intro*], *OF this*]
show *binary-code-morphism f***.**
qed

lemma *bin-code-morph-iff*: *binary-code-morphism* (*bin-morph-of x y*) $\longleftrightarrow x \cdot y \neq y \cdot x$
unfolding *bin-code-morph-iff'*[*of bin-morph-of x y bina*, *unfolded binA-simps bin-morph-ofD*]
using *bin-morph-of-morph* **by** *blast*

lemma *bin-noner-morph-iff*: *nonerasing-morphism* (*bin-morph-of x y*) $\longleftrightarrow x \neq \varepsilon \wedge y \neq \varepsilon$
proof
show $x \neq \varepsilon \wedge y \neq \varepsilon \Longrightarrow$ *nonerasing-morphism* (*bin-morph-of x y*)

by (*rule morphism.nonerI*[*OF bin-morph-of-morph*, *of x y*], *unfold core-def bin-morph-of-def*)
(*simp split*: *finite-2.split*)
show *nonerasing-morphism* (*bin-morph-of x y*) $\Longrightarrow x \neq \varepsilon \wedge y \neq \varepsilon$
using *nonerasing-morphism.nemp-to-nemp*[*of bin-morph-of x y*, *of* [*bina*]]
nonerasing-morphism.nemp-to-nemp[*of bin-morph-of x y*, *of* [*binb*]]
unfolding *bin-morph-of-def* **by** *simp-all*
qed

lemma *morph-bin-morph-of*: *morphism f* $\longleftrightarrow$ *bin-morph-of* (*f* $\mathfrak{a}$) (*f* $\mathfrak{b}$) = *f*
proof
show *morphism f* $\Longrightarrow$ *bin-morph-of* (*f* $\mathfrak{a}$) (*f* $\mathfrak{b}$) = *f*
using *morphism.morph-concat-map'*[*of f*]
unfolding *bin-morph-of-def case-finiteD*[*symmetric*, *of f*] *case-finite-2-if-else* **by** *blast*
qed (*use bin-morph-of-morph* **in** *metis*)

lemma *two-bin-code-morphs-nonerasing-morphs*: *binary-code-morphism g* $\Longrightarrow$ *binary-code-morphism h* $\Longrightarrow$ *two-nonerasing-morphisms g h*
by (*simp add*: *binary-code-morphism.nonerasing binary-code-morphism-def code-morphism.axioms*(*1*) *nonerasing-morphism.intro nonerasing-morphism-axioms.intro two-morphisms-def two-nonerasing-morphisms.intro*)

7.6 Marked binary morphism

lemma *marked-binary-morphI*: **assumes** *morphism f* **and** *f* [*a* :: *binA*] $\neq \varepsilon$ **and** *f* [*1*−*a*] $\neq \varepsilon$ **and** *hd* (*f* [*a*]) $\neq$ *hd* (*f* [*1*−*a*])
shows *marked-morphism f*
proof (*unfold-locales*)
have *f* [*b*] $\neq \varepsilon$ **for** *b*
by (*rule bin-swap-exhaust*[*of b a*]) (*use assms* **in** *force*)+
thus *w* = ε **if** *f w* = ε **for** *w*
using *morphism.noner-sings-conv*[*OF* ‹*morphism f*›] *that* **by** *blast*

show $c = b$ **if** $hd\ (f^{\mathcal{C}}\ c) = hd\ (f^{\mathcal{C}}\ b)$ **for** $c\ b$
proof (*rule ccontr*)
assume $c \neq b$
have $hd\ (f\ [c]) \neq hd\ (f\ [b])$
by (*rule binA-neq-cases-swap*[*OF* ‹$c \neq b$›, *of a*])
(*use* ‹$hd\ (f\ [a]) \neq hd\ (f\ [1-a])$› **in** *fastforce*)+
thus *False*
using *that*[*unfolded core-def*] **by** *contradiction*
qed
qed (*simp add*: *morphism.morph*[*OF assms*(*1*)])

locale *marked-binary-morphism* = *marked-morphism f* :: *binA list* ⇒ *'a list* **for** *f*

begin

lemma *bin-marked*: $hd\ (f\ \mathfrak{a}) \neq hd\ (f\ \mathfrak{b})$
using *marked-morph*[*of bina binb*] **by** *blast*

lemma *bin-marked-sing*: $hd\ (f\ [a]) \neq hd\ (f\ [1-a])$
by (*cases rule*: *finite-2.exhaust*[*of a*]) (*simp-all add*: *bin-marked bin-marked*[*symmetric*])

sublocale *binary-code-morphism*
using *binary-code-morphism-def code-morphism-axioms* **by** *blast*

lemma *marked-lcp-emp*: $\alpha = \varepsilon$
unfolding *bin-lcp-def*
proof (*rule lcp-distinct-hd*)
show $hd\ (f\ \mathfrak{a} \cdot f\ \mathfrak{b}) \neq hd\ (f\ \mathfrak{b} \cdot f\ \mathfrak{a})$
unfolding *hd-append if-not-P*[*OF sing-to-nemp*]
using *bin-marked*.
qed

lemma *bin-marked'*: $(f\ \mathfrak{a})!0 \neq (f\ \mathfrak{b})!0$
using *bin-marked* **unfolding** *hd-conv-nth*[*OF bin-snd-nemp*] *hd-conv-nth*[*OF bin-fst-nemp*].

lemma *marked-bin-morph-prefix-code*: $r \bowtie s \vee f\ (r \cdot z1) \wedge_p f\ (s \cdot z2) = f\ (r \wedge_p s)$
using *lcp-ext-right marked-morph-lcp*[*of r* · *z1 s* · *z2*] **by** *metis*

end

lemma *bin-marked-preimg-hd*:
assumes *marked-binary-morphism* (*f* :: *binA list* ⇒ *binA list*)

obtains c **where** $hd\ (f\ [c]) = a$
proof−
interpret *marked-binary-morphism f*
using *assms*.
from *that alphabet-or-neq*[*OF bin-marked*]
show *thesis*
by *blast*
qed

7.7 Marked version

context *binary-code-morphism*

begin

definition *marked-version* (f_m) **where** $f_m = (\lambda\ w.\ \alpha^{-1>}(f\ w \cdot \alpha))$

lemma *marked-version-conjugates*: $\alpha \cdot f_m\ w = f\ w \cdot \alpha$
unfolding *marked-version-def* **using** *lq-pref*[*OF bin-lcp-pref-all*, *of w*].

lemma *marked-eq-conv*: $f\ w = f\ w' \longleftrightarrow f_m\ w = f_m\ w'$
using *cancel*[*of* $\alpha\ f_m\ w\ f_m\ w'$] **unfolding** *marked-version-conjugates cancel-right*.

lemma *marked-marked*: **assumes** *marked-morphism f* **shows** $f_m = f$
using *marked-version-conjugates*[*unfolded emp-simps* ‹*marked-morphism f*›[*unfolded marked-lcp-conv*]]
by *blast*

lemma *marked-version-all-nemp*: $w \neq \varepsilon \Longrightarrow f_m\ w \neq \varepsilon$
unfolding *marked-version-def*
using *conjug-emp-emp*[*OF bin-lcp-pref-all*, *THEN nonerasing*] **by** *blast*

lemma *marked-version-binary-code-morph*: *binary-code-morphism* f_m
unfolding *bin-code-morph-iff′ morphism-def*
proof (*unfold-locales*)
have $f\ (u{\cdot}v) \cdot \alpha = (f\ u \cdot \alpha) \cdot \alpha^{-1>}(f\ v \cdot \alpha)$ **for** $u\ v$
unfolding *rassoc morph cancel lq-pref*[*OF bin-lcp-pref-all*[*of v*]]..
thus $\bigwedge u\ v.\ f_m\ (u \cdot v) = f_m\ u \cdot f_m\ v$
unfolding *marked-version-def lq-reassoc*[*OF bin-lcp-pref-all*] **by** *presburger*
from *code-morph*
show *inj* f_m
unfolding *inj-def marked-eq-conv*.
qed

interpretation *mv-bcm*: *binary-code-morphism* f_m
using *marked-version-binary-code-morph* .

lemma *marked-lcs*: *bin-lcs* $(f_m\ \mathfrak{a})\ (f_m\ \mathfrak{b}) = \beta \cdot \alpha$
unfolding *bin-lcs-def morph*[*symmetric*] *lcs-ext-right*[*symmetric*] *marked-version-conjugates*[*symmetric*]

mv-bcm.morph[*symmetric*]
by (*rule lcs-ext-left*[*of* f_m (𝔞 · 𝔟) f_m (𝔟 · 𝔞) f_m (𝔞 · 𝔟) $\wedge_s$ f_m (𝔟 · 𝔞) = α · f_m (𝔞 · 𝔟) $\wedge_s$ α · f_m (𝔟 · 𝔞) α α], *unfold mv-bcm.morph*)
(*use mv-bcm.bin-not-comp-suf* **in** *argo*, *simp*)

lemma *bin-lcp-shift*: **assumes** |α| < |*f w*| **shows** (*f w*)!|α| = *hd* (f_m *w*)
proof−
have *w* ≠ ε
using *assms emp-to-emp* **by** *fastforce*
hence f_m *w* ≠ ε
using *marked-version-all-nemp* **by** *blast*
show *?thesis*
using *pref-index*[*of f w* α· f_m *w* |α|, *OF prefI*[*of f w* α α · f_m *w*, *OF marked-version-conjugates*[*of w*, *symmetric*]], *OF assms*]
unfolding *nth-append-length-plus*[*of* α f_m *w 0*, *unfolded add-0-right*] *hd-conv-nth*[*of* f_m *w*, *symmetric*, *OF* ‹f_m *w* ≠ ε›].
qed

lemma *mismatch-fst*: *hd* (f_m 𝔞) = c_0
proof−
have (*f* [*bina*,*binb*])!|α| = *hd* (f_m [*bina*,*binb*])
using *bin-lcp-shift*[*of* [*bina*,*binb*], *unfolded pop-hd*[*of bina* 𝔟] *lenmorph*, *OF bin-lcp-short*]
unfolding *pop-hd*[*of bina* 𝔟].
from *this*[*unfolded mv-bcm.pop-hd*[*of bina* 𝔟, *unfolded not-Cons-self2*[*of bina* ε]] *hd-append2*[*OF mv-bcm.bin-fst-nemp*, *of* f_m 𝔟], *symmetric*]
show *?thesis*
unfolding *bin-mismatch-def hd-word*[*of bina* 𝔟] *morph*.
qed

lemma *mismatch-snd*: *hd* (f_m 𝔟) = c_1
proof−
have (*f* [*binb*,*bina*])!|α| = *hd* (f_m [*binb*,*bina*])
using *bin-lcp-shift*[*of* [*binb*,*bina*], *unfolded pop-hd*[*of binb* 𝔞] *lenmorph*, *OF bin-lcp-short*[*unfolded add.commute*[*of* |*f* 𝔞| |*f* 𝔟|]]]
unfolding *pop-hd*[*of binb* 𝔞].
from *this*[*unfolded mv-bcm.pop-hd*[*of binb* 𝔞, *unfolded not-Cons-self2*[*of binb* ε]] *hd-append2*[*OF mv-bcm.bin-snd-nemp*, *of* f_m 𝔞],*symmetric*]
show *?thesis*
unfolding *bin-mismatch-def hd-word*[*of binb* 𝔞] *morph bin-lcp-sym*[*of f* 𝔞].
qed

lemma *marked-hd-neq*: *hd* (f_m [*a*]) ≠ *hd* (f_m [*1*−*a*]) (**is** *?P* (*a* :: *binA*))
by (*rule bin-induct*[*of ?P*, *unfolded binA-simps*])
(*use mismatch-fst mismatch-snd bin-mismatch-neq* **in** *presburger*)+

lemma *marked-version-marked-morph*: *marked-morphism* f_m
by (*standard*, *unfold core-def*)
(*use not-swap-eq*[*of* λ *a b*. *hd* (f_m [*a*]) = *hd* (f_m [*b*]), *OF - marked-hd-neq*] **in**

force)

interpretation *mv-mbm*: *marked-binary-morphism* f_m
using *marked-version-marked-morph*
by (*simp add*: *marked-binary-morphism-def*)

lemma *bin-code-pref-morph*: $f\ u \cdot \alpha \leq p\ f\ w \cdot \alpha \Longrightarrow u \leq p\ w$
unfolding *marked-version-conjugates*[*symmetric*] *pref-cancel-conv*
using *mv-mbm.pref-free-morph*.

lemma *mismatch-pref0*: $[c_0] \leq p\ f_m\ \mathfrak{a}$
using *mv-bcm.sing-to-nemp*[*THEN hd-pref*, *of bina*] **unfolding** *mismatch-fst*.

lemma *mismatch-pref1*: $[c_1] \leq p\ f_m\ \mathfrak{b}$
using *mv-bcm.bin-snd-nemp*[*THEN hd-pref*] **unfolding** *mismatch-snd*.

lemma *marked-version-len*: $|f_m\ w| = |f\ w|$
using *add-left-imp-eq*[*OF*
lenmorph[*of* $\alpha\ f_m\ w$, *unfolded lenmorph*[*of f w* α, *folded marked-version-conjugates*[*of w*]],*symmetric*,
unfolded add.commute[*of* $|f\ w|\ |\alpha|$]]].

lemma *bin-code-lcp*: $(f\ r \cdot \alpha) \wedge_p (f\ s \cdot \alpha) = f\ (r \wedge_p s) \cdot \alpha$
by (*metis lcp-ext-left marked-version-conjugates mv-mbm.marked-morph-lcp*)

lemma *not-comp-lcp*: **assumes** $\neg\ r \bowtie s$
shows $f\ (r \wedge_p s) \cdot \alpha = f\ r \cdot f\ (r \cdot s) \wedge_p f\ s \cdot f\ (r \cdot s)$
proof–
let $?r' = (r \wedge_p s)^{-1>} r$
let $?s' = (r \wedge_p s)^{-1>} s$
from *lcp-mismatch-lq*[*OF* ‹$\neg\ r \bowtie s$›]
have $?r' \neq \varepsilon$ **and** $?s' \neq \varepsilon$ **and** $hd\ ?r' \neq hd\ ?s'$.
have $\neg\ f\ ((r \wedge_p s) \cdot [hd\ ?r'] \cdot tl\ ?r') \cdot \alpha \bowtie f\ ((r \wedge_p s) \cdot [hd\ ?s'] \cdot tl\ ?s') \cdot \alpha$
using *bin-mismatch-swap-not-comp*
unfolding *morph prefix-comparable-def rassoc pref-cancel-conv*
‹$hd\ ?r' \neq hd\ ?s'$›[*symmetric*, *unfolded bin-neq-iff*, *symmetric*].
hence $\neg\ f\ r \cdot \alpha \bowtie f\ s \cdot \alpha$
unfolding *hd-tl*[*OF* ‹$?r' \neq \varepsilon$›] *hd-tl*[*OF* ‹$?s' \neq \varepsilon$›] *lcp-lq*.
have *pref*: $f\ w \cdot \alpha \leq p\ \ f\ w \cdot f\ (r \cdot s)$ **for** w
unfolding *pref-cancel-conv*
using *append-prefixD*[*OF not-comp-bin-lcp-pref*, *OF* ‹$\neg\ r \bowtie s$›] **by** *blast*
from *prefE*[*OF pref*[*of r*], *unfolded rassoc*]
obtain gr' **where** gr': $f\ r \cdot f\ (r \cdot s) = f\ r \cdot \alpha \cdot gr'$.
from *prefE*[*OF pref*[*of s*], *unfolded rassoc*]
obtain gs' **where** gs': $f\ s \cdot f\ (r \cdot s) = f\ s \cdot \alpha \cdot gs'$.
thus $f\ (r \wedge_p s) \cdot \alpha = f\ r \cdot f\ (r \cdot s) \wedge_p f\ s \cdot f\ (r \cdot s)$
unfolding *bin-code-lcp*[*symmetric*, *of r s*] *prefix-def* **using** ‹$\neg\ f\ r \cdot \alpha \bowtie f\ s \cdot \alpha$›
lcp-ext-right[*of f r* · α *f s* · α - gr' gs', *unfolded rassoc*, *folded gr' gs'*] **by** *argo*
qed

lemma *bin-morph-pref-conv*: $f\ u \cdot \alpha \leq p\ f\ v \cdot \alpha \longleftrightarrow u \leq p\ v$
proof
assume $u \leq p\ v$
from *this*[*unfolded prefix-def*]
obtain z **where** $v = u \cdot z$ **by** *blast*
show $f\ u \cdot \alpha \leq p\ f\ v \cdot \alpha$
unfolding *arg-cong*[*OF* ‹$v = u \cdot z$›, *of f*, *unfolded morph*] *rassoc pref-cancel-conv*
using *bin-lcp-pref-all*.
next
assume $f\ u \cdot \alpha \leq p\ f\ v \cdot \alpha$
then show $u \leq p\ v$
unfolding *marked-version-conjugates*[*symmetric*] *prefix-comparable-def pref-cancel-conv*
using *mv-mbm.pref-free-morph* **by** *meson*
qed

lemma *bin-morph-compare-conv*: $f\ u \cdot \alpha \bowtie f\ v \cdot \alpha \longleftrightarrow u \bowtie v$
using *bin-morph-pref-conv* **unfolding** *prefix-comparable-def* **by** *auto*

lemma *code-lcp'*: $\neg\ r \bowtie s \Longrightarrow \alpha \leq p\ f\ z \Longrightarrow \alpha \leq p\ f\ z' \Longrightarrow f\ (r \cdot z) \wedge_p f\ (s \cdot z')$
$= f\ (r \wedge_p s) \cdot \alpha$
proof−
assume $\alpha \leq p\ f\ z\ \alpha \leq p\ f\ z' \neg\ r \bowtie s$
hence *eqs*: $f\ (r \cdot z) = (f\ r \cdot \alpha) \cdot (\alpha^{-1>} f\ z)\ \ f\ (s \cdot z') = (f\ s \cdot \alpha) \cdot (\alpha^{-1>} f\ z')$
unfolding *rassoc* **by** (*metis lq-pref morph*)+
show *?thesis*
using *bin-morph-compare-conv* ‹$\neg\ r \bowtie s$› *bin-code-lcp lcp-ext-right* **unfolding** *eqs*
by *metis*
qed

lemma *non-comm-im-lcp*: **assumes** $u \cdot v \neq v \cdot u$
shows $f\ (u \cdot v) \wedge_p f\ (v \cdot u) = f\ (u \cdot v \wedge_p v \cdot u) \cdot \alpha$
proof−
have $\neg\ f\ (u \cdot v) \bowtie f\ (v \cdot u)$
using *assms comm-comp-eq*[*of f u f v*, *folded morph*, *THEN code-morph-code*]
by *blast*
from *lcp-ext-right-conv*[*OF this*, *of* α α, *unfolded bin-code-lcp*, *symmetric*]
show *?thesis*.
qed

end

— Obtaining one morphism marked from two general morphisms by shift (conjugation)

locale *binary-code-morphism-shift* = *binary-code-morphism* +
fixes α'
assumes *shift-pref*: $\alpha' \leq p\ \alpha$

begin

definition *shifted-f* **where** *shifted-f* = (λ *w*. $\alpha'^{-1>}$(*f w* · α'))

lemma *shift-pref-all*: $\alpha' \leq_p f\ w \cdot \alpha'$
proof−
  have $\alpha' \cdot \alpha'^{-1>}\alpha \leq_p f\ w \cdot \alpha' \cdot \alpha'^{-1>}\alpha$
    unfolding *lq-pref*[*OF shift-pref*] *rassoc* **using** *bin-lcp-pref-all*.
  thus *?thesis*
    using *pref-keeps-per-root* **by** *blast*
qed

sublocale *shifted*: *binary-code-morphism shifted-f*
proof−
  have *morph*: *f* (*u*·*v*) · α' = (*f u* · α') · $\alpha'^{-1>}$(*f v* · α') **for** *u v*
    unfolding *rassoc morph cancel lq-pref*[*OF shift-pref-all*]..
  then interpret *morphism shifted-f*
    unfolding *shifted-f-def morphism-def*
     using *lq-reassoc*[*OF shift-pref-all*] **by** *presburger*
  have *inj shifted-f*
    unfolding *inj-on-def shifted-f-def* **using** *lq-pref*[*OF shift-pref-all*]
    using *cancel-right code-morph-code* **by** *metis*
  then show *binary-code-morphism shifted-f*
    by *unfold-locales*
qed

lemma *shifted-lcp*: $\alpha' \cdot$ *shifted.bin-code-lcp* = α
  unfolding *bin-lcp-def shifted-f-def lcp-ext-left*[*symmetric*]
  unfolding *lassoc lq-pref*[*OF shift-pref-all*]
  unfolding *rassoc lq-pref*[*OF shift-pref-all*]
  using *lcp-ext-right-conv*[*OF bin-not-comp*, *unfolded rassoc*].

lemma $\alpha' = \alpha \Longrightarrow$ *shifted-f* = f_m
  unfolding *shifted-f-def marked-version-def* **by** *fast*

end

7.8 Two binary code morphisms

locale *two-binary-code-morphisms* =
  g: *binary-code-morphism g* +
  h: *binary-code-morphism h*
  for *g h* :: *binA list* ⇒ *'a list*

begin

notation *h.bin-code-lcp* (α_h)
notation *g.bin-code-lcp* (α_g)

notation *g.marked-version* (g_m)
notation *h.marked-version* (h_m)

sublocale *gm*: *marked-binary-morphism* g_m
  by (*simp add*: *g.marked-version-marked-morph marked-binary-morphism.intro*)

sublocale *hm*: *marked-binary-morphism* h_m
  by (*simp add*: *h.marked-version-marked-morph marked-binary-morphism.intro*)

sublocale *two-binary-morphisms g h*..

sublocale *marked*: *two-marked-morphisms* g_m h_m..

sublocale *code*: *two-code-morphisms g h*
  by *unfold-locales*

lemma *marked-two-binary-code-morphisms*: *two-binary-code-morphisms* g_m h_m
    using *g.marked-version-binary-code-morph h.marked-version-binary-code-morph*
    by *unfold-locales*

lemma *revs-two-binary-code-morphisms*: *two-binary-code-morphisms* (*rev-map g*) (*rev-map h*)
  using *code.revs-two-code-morphisms rev-morphs*
  by (*simp add*: *g.bin-code-morph-rev-map h.bin-code-morph-rev-map rev-morphs two-binary-code-morphisms-def*)

lemma *swap-two-binary-code-morphisms*: *two-binary-code-morphisms h g*
    by *unfold-locales*

Each successful overflow has a unique minimal successful continuation

lemma *min-completionE*:
  assumes $z \cdot g_m\ r = z' \cdot h_m\ s$
  obtains $p\ q$ **where** $z \cdot g_m\ p = z' \cdot h_m\ q$ **and**
    $\bigwedge r\ s.\ z \cdot g_m\ r = z' \cdot h_m\ s \implies p \leq p\ r \wedge q \leq p\ s$
proof−
  interpret *swap*: *two-binary-code-morphisms h g*
    by *unfold-locales*
  define P **where** $P = (\lambda\ m.\ \exists\ p\ q.\ z \cdot g_m\ p = z' \cdot h_m\ q \wedge |p| = m)$
  have $P\ |r|$ **using** *assms P-def*
    by *blast*
  obtain n **where** *ndef*: $n = (LEAST\ m.\ P\ m)$
    by *simp*
  then obtain $p\ q$ **where** $z \cdot g_m\ p = z' \cdot h_m\ q\ |p| = n$ **using** ‹$P\ |r|$›
    using *LeastI P-def* **by** *metis*
  have $p \leq p\ r' \wedge q \leq p\ s'$ **if** $z \cdot g_m\ r' = z' \cdot h_m\ s'$ **for** $r'\ s'$
  proof
    have $z \cdot g_m\ (p \wedge_p r') = z' \cdot h_m\ (q \wedge_p s')$

using ‹$z \cdot g_m\ p = z' \cdot h_m\ q$› ‹$z \cdot g_m\ r' = z' \cdot h_m\ s'$›
marked.unique-continuation **by** *blast*
thus $p \leq_p r'$
using *P-def le-antisym* ‹$|p| = n$› *lcp-len' ndef not-less-Least*
by *metis*
from *this*[*folded lcp-pref-conv*]
have $h_m\ q = h_m\ (q \wedge_p s')$
using ‹$z \cdot g_m\ (p \wedge_p r') = z' \cdot h_m\ (q \wedge_p s')$› ‹$z \cdot g_m\ p = z' \cdot h_m\ q$›
by *force*
thus $q \leq_p s'$
using *hm.code-morph-code lcp-pref-conv* **by** *metis*
qed
thus *thesis*
using ‹$z \cdot g_m\ p = z' \cdot h_m\ q$› *that* **by** *blast*
qed

lemma *two-equals*:
assumes $g\ r = h\ r$ **and** $g\ s = h\ s$ **and** $\neg\ r \bowtie s$
shows $g\ (r \wedge_p s) \cdot \alpha_g = h\ (r \wedge_p s) \cdot \alpha_h$
unfolding *g.not-comp-lcp*[*OF* ‹$\neg\ r \bowtie s$›] *h.not-comp-lcp*[*OF* ‹$\neg\ r \bowtie s$›] *g.morph h.morph assms*..

lemma *solution-sing-len-diff*: **assumes** $g \neq h$ **and** $g\ s = h\ s$ **and** *set s* = *binUNIV*
shows $|g\ [c]| \neq |h\ [c]|$
proof (*rule solution-sing-len-cases*[*OF* ‹*set s* = *binUNIV*› ‹$g\ s = h\ s$› ‹$g \neq h$›])
fix *a* **assume** *less*: $|g\ [a]| < |h\ [a]|\ |h\ [1 - a]| < |g\ [1 - a]|$
show $|g\ [c]| \neq |h\ [c]|$
by (*rule bin-swap-exhaust*[*of c a*]) (*use less* **in** *force*)+
qed

lemma *alphas-pref*: **assumes** $|\alpha_h| \leq |\alpha_g|$ **and** $g\ r =_m h\ s$ **shows** $\alpha_h \leq_p \alpha_g$
proof−
have $s \neq \varepsilon$ **and** $r \neq \varepsilon$
using *min-coinD'*[*OF* ‹$g\ r =_m h\ s$›] **by** *force*+
have $\alpha_h \leq_p h\ s \cdot \alpha_h\ h\ s \neq \varepsilon$
using *h.bin-lcp-spref-all*[*OF* ‹$s \neq \varepsilon$›] **by** *blast*+
have $\alpha_g \leq_p h\ s \cdot \alpha_g$
using *g.bin-lcp-spref-all*[*OF* ‹$r \neq \varepsilon$›, *unfolded min-coinD*[*OF* ‹$g\ r =_m h\ s$›]] **by** *force*
from
root-ruler[*OF* ‹$\alpha_h \leq_p h\ s \cdot \alpha_h$› ‹$\alpha_g \leq_p h\ s \cdot \alpha_g$› ‹$h\ s \neq \varepsilon$›]
show $\alpha_h \leq_p \alpha_g$
unfolding *prefix-comparable-def* **using** *ruler-le*[*OF self-pref - assms*(*1*)] **by** *blast*
qed

end

locale *binary-codes-coincidence* = *two-binary-code-morphisms* +
assumes *alphas-len*: $|\alpha_h| \leq |\alpha_g|$ **and**

coin-ex: $\exists\ r\ s.\ g\ r =_m h\ s$
begin

lemma *alphas-pref*: $\alpha_h \leq p\ \alpha_g$
using *alphas-pref*[*OF alphas-len*] *coin-ex* **by** *force*

definition α **where** $\alpha \equiv {\alpha_h}^{-1>}\alpha_g$
definition *critical-overflow* (c) **where** *critical-overflow* $\equiv {\alpha_g}^{<-1}\alpha_h$

lemma *lcp-diff*: $\alpha_h \cdot \alpha = \alpha_g$
unfolding α-*def lq-pref* **using** *lq-pref*[*OF alphas-pref*]**.**

lemma *solution-marked-version-conv*: $g\ r = h\ s \longleftrightarrow \alpha \cdot\ g_m\ r = h_m\ s \cdot \alpha$
unfolding *cancel*[*of* $\alpha_h\ \alpha \cdot g_m\ r\ h_m\ s \cdot \alpha$, *symmetric*]
unfolding *lassoc lcp-diff h.marked-version-conjugates g.marked-version-conjugates*
unfolding *rassoc lcp-diff cancel-right***..**

end

locale *binary-code-coincidence-sym* = *two-binary-code-morphisms*
+ **assumes**
coin-ex: $\exists\ r\ s.\ g\ r =_m h\ s$
begin

lemma *coinE*: **obtains** $u\ v$ **where** $g\ u =_m h\ v$ **and** $h\ v =_m g\ u$
using *coin-ex code.min-coin-prefE*[*OF min-solD*[*of - g h*]] *min-coin-sym* **by** *metis*

definition α' **where** $\alpha' = (\mathit{if}\ |\alpha_g| \leq |\alpha_h|\ \mathit{then}\ \alpha_g\ \mathit{else}\ \alpha_h)$
definition g' **where** $g' = (\mathit{if}\ |\alpha_g| \leq |\alpha_h|\ \mathit{then}\ (\lambda\ w.\ {\alpha'}^{-1>}(g\ w \cdot \alpha'))\ \mathit{else}\ (\lambda\ w.\ {\alpha'}^{-1>}(h\ w \cdot \alpha')))$
definition h' **where** $h' = (\mathit{if}\ |\alpha_g| \leq |\alpha_h|\ \mathit{then}\ (\lambda\ w.\ {\alpha'}^{-1>}(h\ w \cdot \alpha'))\ \mathit{else}\ (\lambda\ w.\ {\alpha'}^{-1>}(g\ w \cdot \alpha')))$

lemma *shift-pref-fst*: $\alpha' \leq p\ \alpha_g$
proof (*cases* $|\alpha_g| \leq |\alpha_h|$)
assume *a*: $\neg\ |\alpha_g| \leq |\alpha_h|$
hence *a'*: $|\alpha_h| \leq |\alpha_g|$
by *fastforce*
show $\alpha' \leq p\ \alpha_g$
unfolding α'-*def*[*unfolded if-not-P*[*OF a*]] **using** *coinE*[*OF alphas-pref*[*OF a'*]]
by *auto*
qed (*simp add*: α'-*def*)

interpretation *gshift*: *binary-code-morphism-shift g* α'
using *shift-pref-fst* **by** *unfold-locales*

interpretation *swap*: *two-binary-code-morphisms h g*
by (*simp add*: *swap-two-binary-code-morphisms*)

lemma *shift-pref-snd*: $\alpha' \leq_p \alpha_h$
unfolding α'*-def*
proof (*cases* $\neg\ |\alpha_g| \leq |\alpha_h|$, *simp-all*)
show $|\alpha_g| \leq |\alpha_h| \Longrightarrow \alpha_g \leq_p \alpha_h$
using *swap.alphas-pref*[*OF - coinE*] **by** *blast*
qed

interpretation *hshift*: *binary-code-morphism-shift* $h\ \alpha'$
using *shift-pref-snd* **by** *unfold-locales*

lemma *shifted-eq-conv*:$g\ r = h\ s \longleftrightarrow g'\ r = h'\ s$
oops

lemma *shifted-eq-conv*:$g\ r = h\ r \longleftrightarrow g'\ r = h'\ r$
proof−
have $g\ r = h\ r \longleftrightarrow \alpha'^{-1>}(g\ r \cdot \alpha') = \alpha'^{-1>}(h\ r \cdot \alpha')$
using *cancel-right lq-pref gshift.shift-pref-all hshift.shift-pref-all* **by** *metis*
thus $g\ r = h\ r \longleftrightarrow g'\ r = h'\ r$
unfolding g'*-def* h'*-def*
by (*cases* $|\alpha_g| \leq |\alpha_h|$, *presburger*)
fastforce
qed

lemma *shifted-eq-conv'*:$g = h \longleftrightarrow g' = h'$
using *shifted-eq-conv* **by** *fastforce*

interpretation *shifted-g*: *binary-code-morphism* $(\lambda\ w.\ \alpha'^{-1>}(g\ w \cdot \alpha'))$
using *gshift.shifted.binary-code-morphism-axioms*[*unfolded gshift.shifted-f-def*]**.**

interpretation *shifted-h*: *binary-code-morphism* $(\lambda\ w.\ \alpha'^{-1>}(h\ w \cdot \alpha'))$
using *hshift.shifted.binary-code-morphism-axioms*[*unfolded hshift.shifted-f-def*]**.**

lemma *shifted-min-sol-conv*: $r \in g =_M h \longleftrightarrow r \in g' =_M h'$
unfolding *min-sol-def* **using** *shifted-eq-conv* **by** *blast*

lemma *shifted-not-triv*: $g = h \longleftrightarrow g' = h'$
using *shifted-eq-conv* **by** *fastforce*

sublocale *shifted*: *two-binary-code-morphisms* $g'\ h'$
proof−
interpret g': *binary-code-morphism* g'
unfolding g'*-def* **using** *shifted-g.binary-code-morphism-axioms shifted-h.binary-code-morphism-axioms*
by *presburger*
interpret h': *binary-code-morphism* h'
unfolding h'*-def* **using** *shifted-g.binary-code-morphism-axioms shifted-h.binary-code-morphism-axioms*
by *presburger*
show *two-binary-code-morphisms* $g'\ h'$**..**
qed

lemma *shifted-fst-lcp-emp*: *shifted.g.bin-code-lcp* = ε
unfolding *bin-lcp-def*
proof (*cases* $|\alpha_g| \leq |\alpha_h|$)
assume $|\alpha_g| \leq |\alpha_h|$
hence $*$: $\alpha' = \alpha_g\ g' = (\lambda\ w.\ \alpha'^{-1>}(g\ w \cdot \alpha'))$
unfolding α'-*def* g'-*def* **by** *simp-all*
have $g_m\ \mathfrak{a} \cdot g_m\ \mathfrak{b} \wedge_p g_m\ \mathfrak{b} \cdot g_m\ \mathfrak{a} = \varepsilon$
using *gm.marked-lcp-emp* **unfolding** *bin-lcp-def*.
thus $g'\ \mathfrak{a} \cdot g'\ \mathfrak{b} \wedge_p g'\ \mathfrak{b} \cdot g'\ \mathfrak{a} = \varepsilon$
unfolding $*$ *g.marked-version-def*.
next
assume $\neg\ |\alpha_g| \leq |\alpha_h|$
hence *c*: $\alpha' = \alpha_h\ g' = (\lambda\ w.\ \alpha'^{-1>}(h\ w \cdot \alpha'))$
unfolding α'-*def* g'-*def* **by** *simp-all*
have $h_m\ \mathfrak{a} \cdot h_m\ \mathfrak{b} \wedge_p h_m\ \mathfrak{b} \cdot h_m\ \mathfrak{a} = \varepsilon$
using *hm.marked-lcp-emp* **unfolding** *bin-lcp-def*.
thus $g'\ \mathfrak{a} \cdot g'\ \mathfrak{b} \wedge_p g'\ \mathfrak{b} \cdot g'\ \mathfrak{a} = \varepsilon$
unfolding *c h.marked-version-def*.
qed

lemma *shifted-alphas*: **assumes** *le*: $|\alpha_g| \leq |\alpha_h|$
shows $\alpha' \cdot$ *shifted.g.bin-code-lcp* = α_g **and** $\alpha' \cdot$ *shifted.h.bin-code-lcp* = α_h
proof−
have *c*: $\alpha' = \alpha_g\ g' = (\lambda\ w.\ \alpha'^{-1>}(g\ w \cdot \alpha'))\ h' = (\lambda\ w.\ \alpha'^{-1>}(h\ w \cdot \alpha'))$
using *le* **unfolding** α'-*def* g'-*def* h'-*def* **by** *simp-all*
interpret *binary-codes-coincidence h g*
proof
show $\exists\ r\ s.\ h\ r =_m g\ s$
using *coin-ex*[*unfolded min-coin-sym-iff*[*of g*]] **by** *blast*
qed *fact*
show $\alpha' \cdot$ *shifted.g.bin-code-lcp* = α_g
unfolding *c*
unfolding *bin-lcp-def*[*of* $g'\ \mathfrak{a}$, *unfolded c*] *lcp-ext-left*[*symmetric*]
unfolding *lassoc lq-pref*[*OF g.bin-lcp-pref-all*]
unfolding *rassoc lq-pref*[*OF g.bin-lcp-pref-all*]
unfolding *lcp-ext-right-conv*[*OF g.non-comp-morph*[*of bina*], *unfolded binA-simps rassoc*]
unfolding *bin-lcp-def*..
from *pref-prod-pref*[*OF pref-trans*[*OF alphas-pref h.bin-lcp-pref-all*] *alphas-pref*]
have *pref-all*: $\alpha_g \leq p\ h\ w \cdot \alpha_g$ **for** *w*.
show $\alpha' \cdot$ *shifted.h.bin-code-lcp* = α_h
unfolding *c*
unfolding *bin-lcp-def*[*of* $h'\ \mathfrak{a}$, *unfolded c*] *lcp-ext-left*[*symmetric*]
unfolding *lassoc lq-pref*[*OF pref-all*]
unfolding *rassoc lq-pref*[*OF pref-all*]
unfolding *lcp-ext-right-conv*[*OF h.non-comp-morph*[*of bina*], *unfolded binA-simps rassoc*]
unfolding *bin-lcp-def*..
qed

interpretation *swapped*: *binary-code-coincidence-sym h g*
proof
  show $\exists\, r\ s.\ h\ r =_m g\ s$
  using *coin-ex*[*unfolded min-coin-sym-iff*[*of g*]] **by** *blast*
qed

lemma *eq-len-eq-conv*: $\alpha_g = \alpha_h \longleftrightarrow |\alpha_g| = |\alpha_h|$
proof
  show $\alpha_g = \alpha_h$ **if** $|\alpha_g| = |\alpha_h|$
    using *swap.alphas-pref*[*OF eq-imp-le*[*OF that*]] *alphas-pref*[*OF eq-imp-le*[*OF that*[*symmetric*]]]
    using *coinE*[*of* $\alpha_g = \alpha_h$] **by** *fastforce*
qed *simp*

lemma *shift-swapped*: $swapped.\alpha' = \alpha'$
  unfolding α'*-def swapped.*α'*-def* **using** *eq-len-eq-conv* **by** *fastforce*

lemma *morphs-swapped*: **assumes** $|\alpha_g| \neq |\alpha_h|$ **shows** $swapped.g' = g'$ $swapped.h' = h'$
  unfolding g'*-def swapped.*g'*-def* h'*-def swapped.*h'*-def shift-swapped* **using** *assms* **by** *fastforce+*

lemma *morphs-swapped'*: **assumes** $|\alpha_g| = |\alpha_h|$ **shows** $swapped.g' = h'$ $swapped.h' = g'$
  unfolding g'*-def swapped.*g'*-def* h'*-def swapped.*h'*-def shift-swapped* **using** *assms* **by** *fastforce+*

lemma *shifted-lcp-len-eq*: $|shifted.g.bin\text{-}code\text{-}lcp| = |shifted.h.bin\text{-}code\text{-}lcp| \longleftrightarrow |\alpha_g| = |\alpha_h|$ **and**
  shifted-lcp-len-le: $|shifted.g.bin\text{-}code\text{-}lcp| \leq |shifted.h.bin\text{-}code\text{-}lcp|$
  unfolding *atomize-conj*
proof (*cases*)
  assume *le*: $|\alpha_g| \leq |\alpha_h|$
  note *shifted-alphas*[*OF this*]
  from *lenarg*[*OF this*(*1*)] *lenarg*[*OF this*(*2*)]
   show $(|shifted.g.bin\text{-}code\text{-}lcp| = |shifted.h.bin\text{-}code\text{-}lcp| \longleftrightarrow |\alpha_g| = |\alpha_h|) \wedge |shifted.g.bin\text{-}code\text{-}lcp| \leq |shifted.h.bin\text{-}code\text{-}lcp|$
    unfolding *lenmorph* **using** *le* **by** *fastforce+*
next
  assume $\neg\ |\alpha_g| \leq |\alpha_h|$
  hence *le*: $|\alpha_h| \leq |\alpha_g|$ **by** *fastforce*
  note *swapped.shifted-alphas*[*OF this*]
  note *lens* = *lenarg*[*OF this*(*1*)] *lenarg*[*OF this*(*2*)]
   show $(|shifted.g.bin\text{-}code\text{-}lcp| = |shifted.h.bin\text{-}code\text{-}lcp| \longleftrightarrow |\alpha_g| = |\alpha_h|) \wedge |shifted.g.bin\text{-}code\text{-}lcp| \leq |shifted.h.bin\text{-}code\text{-}lcp|$
  proof (*cases*)
    assume *eq*: $|\alpha_g| = |\alpha_h|$
     show $(|shifted.g.bin\text{-}code\text{-}lcp| = |shifted.h.bin\text{-}code\text{-}lcp| \longleftrightarrow |\alpha_g| = |\alpha_h|) \wedge$

$|shifted.g.bin\text{-}code\text{-}lcp| \leq |shifted.h.bin\text{-}code\text{-}lcp|$
using *lens eq* **unfolding** *shift-swapped lenmorph bin-lcp-def morphs-swapped'*[*OF eq*] **by** *linarith+*
next
assume *neq*: $|\alpha_g| \neq |\alpha_h|$
from *lens*
show ($|shifted.g.bin\text{-}code\text{-}lcp| = |shifted.h.bin\text{-}code\text{-}lcp| \longleftrightarrow |\alpha_g| = |\alpha_h|$) $\wedge$ $|shifted.g.bin\text{-}code\text{-}lcp| \leq |shifted.h.bin\text{-}code\text{-}lcp|$
using *le* **unfolding** *shift-swapped lenmorph bin-lcp-def morphs-swapped*[*OF neq*] **by** *fastforce+*
qed
qed

end

locale *two-marked-binary-morphisms* = *two-marked-morphisms g h*
for *g h* :: *binA list* $\Rightarrow$ *'a list*
begin

sublocale *two-binary-code-morphisms g h* **..**

lemma *not-comm-im*: **assumes** $g \neq h$ **and** $g\ s = h\ s$ **and** $s \neq \varepsilon$
and $hd\ s = a$ **and** $set\ s = binUNIV$
shows $g[a] \cdot h\ [a] \neq h[a] \cdot g[a]$
proof (*rule notI*)
assume *comm*: $g[a] \cdot h\ [a] = h[a] \cdot g[a]$
from *hd-im-comm-eq*[*OF* ‹$g\ s = h\ s$› ‹$s \neq \varepsilon$›] *comm*
have $g\ [a] = h\ [a]$
unfolding *core-def* ‹$hd\ s\ = a$› **by** *blast*
thus *False*
using *len-ims-sing-neq*[*OF* ‹$g\ s = h\ s$› ‹$g \neq h$› ‹$set\ s = binUNIV$›] **by** *metis*
qed

lemma *sol-set-not-com-hd*:
assumes
morphs-neq: $g \neq h$ **and**
sol: $g\ s = h\ s$ **and**
sol-set: $set\ s = binUNIV$
shows $g\ ([hd\ s]) \cdot\ h\ ([hd\ s]) \neq h\ ([hd\ s]) \cdot\ g\ ([hd\ s])$
proof
assume *comm*: $g\ [hd\ s] \cdot h\ [hd\ s] = h\ [hd\ s] \cdot g\ [hd\ s]$

obtain *n s'* **where** *s*: $[hd\ s]^{@} Suc\ n \cdot [1 - hd\ s] \cdot s' = s$
using *bin-distinct-letter*[*OF sol-set*].
have $[hd\ s]\ ^{@}\ Suc\ n \cdot [1 - hd\ s] \cdot s' \neq \varepsilon$ **by** *blast*
from *hd-im-comm-eq*[*OF - this*]
have *g* $[hd\ s] = h\ [hd\ s]$
unfolding *core-def s comm* **using** *sol* **by** *blast*
thus *False*
using *len-ims-sing-neq*[*OF sol* ‹$g \neq h$› *sol-set*, *of hd s*] **by** *argo*
qed

sublocale *g*: *marked-binary-morphism g*
using *g.marked-marked g.marked-morphism-axioms gm.marked-binary-morphism-axioms*
by *auto*

sublocale *h*: *marked-binary-morphism h*
using *h.marked-marked h.marked-morphism-axioms hm.marked-binary-morphism-axioms*
by *auto*

sublocale *revs*: *two-binary-code-morphisms rev-map g rev-map h*
using *revs-two-binary-code-morphisms*.

end

7.9 Two marked binary morphisms with blocks

locale *two-binary-marked-blocks* = *two-marked-binary-morphisms* +
assumes *both-blocks*: $\bigwedge$ *a*. *blockP a*

begin

sublocale *sucs*: *two-marked-binary-morphisms suc-fst suc-snd*
using *sucs-marked-morphs*[*OF both-blocks*, *folded two-marked-binary-morphisms-def*].

sublocale *sucs-enc*: *two-marked-binary-morphisms suc-fst' suc-snd'*
using *encoded-sucs*[*OF both-blocks*, *folded two-marked-binary-morphisms-def*].

lemma *bin-blocks-swap*: *two-binary-marked-blocks h g*
proof (*unfold-locales*)
fix *a*
obtain *c* **where** *hd* (*suc-snd* [*c*]) = *a*
using *bin-marked-preimg-hd*[*of suc-snd*]
marked-binary-morphism-def sucs.h.marked-morphism-axioms **by** *blast*
show *two-marked-morphisms.blockP h g a*
proof (*rule two-marked-morphisms.blockI*, *unfold-locales*)
show *hd* (*suc-snd* [*c*]) = *a* **by** *fact*
show *h* (*suc-snd* [*c*]) =$_m$ *g* (*suc-fst* [*c*])
using *min-coin-sym*[*OF blockP-D*[*OF both-blocks*]].
qed
qed

lemma *blocks-all-letters-fst*: $[b] \leq_f$ *suc-fst* ($[a] \cdot [1-a]$)
proof–
have $*$: *suc-fst* ($[a] \cdot [1 - a]$) = $[a]$ · *tl* (*suc-fst* $[a]$) · $[1-a]$ · *tl* (*suc-fst* $[1 - a]$)
unfolding *sucs.g.morph lassoc hd-tl*[*OF sucs.g.sing-to-nemp*, *unfolded blockP-D-hd*[*OF both-blocks*]]..
show *?thesis*
by (*cases rule*: *neq-exhaust*[*OF bin-swap-neq*, *of b a*], *unfold* $*$)
(*blast*+)
qed

lemma *blocks-all-letters-snd*: $[b] \leq_f$ *suc-snd* ($[a] \cdot [1-a]$)
proof–
have $*$: *suc-snd* ($[a] \cdot [1 - a]$) = [*hd* (*suc-snd* $[a]$)] · *tl* (*suc-snd* $[a]$) · [*hd* (*suc-snd* $[1-a]$)] · *tl* (*suc-snd* $[1-a]$)
unfolding *sucs.h.morph rassoc hd-tl*[*OF sucs.h.sing-to-nemp*, *unfolded blockP-D-hd*[*OF both-blocks*]]
unfolding *lassoc hd-tl*[*OF sucs.h.sing-to-nemp*, *unfolded blockP-D-hd*[*OF both-blocks*]]..
show *?thesis*
proof (*cases rule*: *neq-exhaust*[*OF sucs.h.bin-marked-sing*, *of b a*])
assume *b*: b = *hd* (*suc-snd* $[a]$)
show *?thesis*
unfolding b $*$ **by** *blast*
next
assume *b*: b = *hd* (*suc-snd* $[1-a]$)
show *?thesis*
unfolding b $*$ **by** *blast*
qed
qed

lemma *lcs-suf-blocks-fst*: *g.bin-code-lcs* $\leq_s$ *g* (*suc-fst* ($[a] \cdot [1-a]$))
using *revs.g.bin-lcp-pref''*[*reversed*] *g.bin-lcp-pref''* *blocks-all-letters-fst* **by** *simp*

lemma *lcs-suf-blocks-snd*: *h.bin-code-lcs* $\leq_s$ *h* (*suc-snd* ($[a] \cdot [1-a]$))
using *revs.h.bin-lcp-pref''*[*reversed*] *h.bin-lcp-pref''* *blocks-all-letters-snd* **by** *simp*

lemma *lcs-fst-suf-snd*: *g.bin-code-lcs* $\leq_s$ *h.bin-code-lcs* · *h* *sucs.h.bin-code-lcs*
proof–
have *g.bin-code-lcs* $\leq_s$ *g* (*suc-fst* $[a]$ · *suc-fst* $[1-a]$) **for** *a*
using *lcs-suf-blocks-fst*[*of a*]
unfolding *binA-simps sucs.g.morph*.
have *g.bin-code-lcs* $\leq_s$ *g* (*suc-fst* $\mathfrak{a}$ · *suc-fst* $\mathfrak{b}$) **and** *g.bin-code-lcs* $\leq_s$ *g* (*suc-fst* $\mathfrak{b}$ · *suc-fst* $\mathfrak{a}$)
using *lcs-suf-blocks-fst*[*of bina*] *lcs-suf-blocks-fst*[*of binb*]
unfolding *binA-simps sucs.g.morph*.
hence *g.bin-code-lcs* $\leq_s$ *h* (*suc-snd* $\mathfrak{a}$ · *suc-snd* $\mathfrak{b}$) **and** *g.bin-code-lcs* $\leq_s$ *h* (*suc-snd* $\mathfrak{b}$ · *suc-snd* $\mathfrak{a}$)
unfolding *g.morph h.morph* *block-eq*[*OF both-blocks*].

from *suf-ext*[*OF this*(*1*)] *suf-ext*[*OF this*(*2*)]
have *g.bin-code-lcs* $\leq_s$ *h.bin-code-lcs* $\cdot$ *h* (*suc-snd* $\mathfrak{a} \cdot$ *suc-snd* $\mathfrak{b}$) **and** *g.bin-code-lcs* $\leq_s$ *h.bin-code-lcs* $\cdot$ *h* (*suc-snd* $\mathfrak{b} \cdot$ *suc-snd* $\mathfrak{a}$)**.**
hence *g.bin-code-lcs* $\leq_s$ *h.bin-code-lcs* $\cdot$ *h* (*suc-snd* $\mathfrak{a} \cdot$ *suc-snd* $\mathfrak{b}$) $\wedge_s$ *h.bin-code-lcs* $\cdot$ *h* (*suc-snd* $\mathfrak{b} \cdot$ *suc-snd* $\mathfrak{a}$)
using *suf-lcs-iff* **by** *blast*
thus *g.bin-code-lcs* $\leq_s$ *h.bin-code-lcs* $\cdot$ *h sucs.h.bin-code-lcs*
unfolding *revs.h.bin-code-lcp*[*reversed*] *bin-lcs-def*[*symmetric*]**.**
qed

lemma *suf-comp-lcs*: *g.bin-code-lcs* $\bowtie_s$ *h.bin-code-lcs*
using *lcs-suf-blocks-fst lcs-suf-blocks-snd*
unfolding *g.morph h.morph sucs.g.morph sucs.h.morph block-eq*[*OF both-blocks*] *suf-comp-or* **using** *ruler*[*reversed*] **by** *blast*

end

7.10 Binary primitivity preserving morphism given by a pair of words

definition *bin-prim* :: *'a list* $\Rightarrow$ *'a list* $\Rightarrow$ *bool*
where *bin-prim x y* $\longleftrightarrow$ *primitivity-preserving-morphism* (*bin-morph-of x y*)

lemma *bin-prim-code*:
assumes *bin-prim x y*
shows $x \cdot y \neq y \cdot x$
proof −
have *inj* (*bin-morph-of x y*)
using ‹*bin-prim x y*› *primitivity-preserving-morphism.code-morph*
by (*simp add*: *bin-prim-def*)
then have (*bin-morph-of x y*) ($\mathfrak{a} \cdot \mathfrak{b}$) $\neq$ (*bin-morph-of x y*) ($\mathfrak{b} \cdot \mathfrak{a}$)
by (*blast dest*: *injD*)
then show $x \cdot y \neq y \cdot x$
by (*simp add*: *bin-morph-of-def*)
qed

7.10.1 Translating to to list concatenation

lemma *bin-concat-prim-pres-noner1*:
assumes $x \neq y$
and *prim-pres*: $\bigwedge$ *ws*. *ws* $\in$ *lists* {*x*,*y*} $\Longrightarrow$ *2* $\leq$ |*ws*| $\Longrightarrow$ *primitive ws* $\Longrightarrow$ *primitive* (*concat ws*)
shows $x \neq \varepsilon$
proof
assume $x = \varepsilon$
with ‹$x \neq y$› **have** $y \neq \varepsilon$
by *blast*
have *primitive* [*x*, *y*, *y*]

using *prim-abk*[*OF* ‹$x \neq y$›, *of 2*] **by** *simp*
with ‹$x \neq y$› **have** *primitive* (*concat* [x, y, y])
by (*intro prim-pres*) *simp-all*
then show *False*
by (*simp add*: ‹$x = \varepsilon$› *eq-append-not-prim*)
qed

lemma *bin-concat-prim-pres-noner*:
assumes $x \neq y$
and *prim-pres*: $\bigwedge$ *ws*. *ws* ∈ *lists* {*x*,*y*} ⟹ *2* ≤ |*ws*| ⟹ *primitive ws* ⟹ *primitive* (*concat ws*)
shows *nonerasing-morphism* (*bin-morph-of x y*)
proof (*intro morphism.nonerI bin-morph-of-morph*)
fix *a*
have $x \neq \varepsilon$ **and** $y \neq \varepsilon$
using ‹$x \neq y$› *prim-pres*
by (*fact bin-concat-prim-pres-noner1*, *intro bin-concat-prim-pres-noner1*)
(*unfold insert-commute*[*of x y*] *eq-commute*[*of x y*])
then show (*bin-morph-of x y*)$^{\mathcal{C}}$ $a \neq \varepsilon$
by (*simp add*: *bin-morph-of-def core-def*)
qed

lemma *bin-prim-concat-prim-pres-conv*:
assumes $x \neq y$
shows *bin-prim x y* ⟷ (∀ *ws* ∈ *lists* {*x*,*y*}. *2* ≤ |*ws*| ⟶ *primitive ws* ⟶ *primitive* (*concat ws*))
(**is** - ⟷ *?condition*)
proof −
define *f* **where** *f* = (λ*a*. (*if a* = *bina then x else y*))
have *inj f*
using ‹$x \neq y$›
by (*intro linorder-injI*) (*simp add*: *less-finite-2-def f-def*)
moreover have *f* ‘ *UNIV* = {x, y}
by (*simp add*: *f-def insert-commute*)
ultimately have *bij-betw f UNIV* {x, y}
unfolding *bij-betw-def*..
then have *bij*: *bij-betw* (*map f*) (*lists UNIV*) (*lists* {x, y})
by (*fact bij-lists*)
have *concat-map-f*: *concat* (*map f w*) = *bin-morph-of x y w* **for** *w*
by (*simp add*: *bin-morph-of-def f-def*)
have *?condition* ⟹ *nonerasing-morphism* (*bin-morph-of x y*)
by (*simp add*: ‹$x \neq y$› *bin-concat-prim-pres-noner*)
then show *bin-prim x y* ⟷ *?condition*
unfolding *bin-prim-def primitivity-preserving-morphism-def primitivity-preserving-morphism-axioms-def*
unfolding *bij-betw-ball*[*OF bij*] *prim-map-iff*[*OF* ‹*inj f*›] *concat-map-f*
by *auto*
qed

lemma *bin-prim-concat-prim-pres*:

assumes *bin-prim x y*
shows *ws* ∈ *lists* {*x*, *y*} ⟹ 2 ≤ |*ws*| ⟹ *primitive ws* ⟹ *primitive* (*concat ws*)
using *bin-prim-code*[*OF* ‹*bin-prim x y*›] ‹*bin-prim x y*› *bin-prim-concat-prim-pres-conv*[*of x y*]
by (*cases x* = *y*) *blast*+

lemma *bin-prim-altdef1*:
bin-prim x y ⟷
(*x* ≠ *y*) ∧ (∀ *ws* ∈ *lists* {*x*,*y*}. 2 ≤ |*ws*| ⟶ *primitive ws* ⟶ *primitive* (*concat ws*))
using *bin-prim-code*[*of x y*] *bin-prim-concat-prim-pres-conv*[*of x y*]
by *blast*

lemma *bin-prim-altdef2*:
bin-prim x y ⟷
(*x* · *y* ≠ *y* · *x*) ∧ (∀ *ws* ∈ *lists* {*x*,*y*}. 2 ≤ |*ws*| ⟶ *primitive ws* ⟶ *primitive* (*concat ws*))
using *bin-prim-code*[*of x y*] *bin-prim-concat-prim-pres-conv*[*of x y*]
by *blast*

7.10.2 Basic properties of *bin-prim*

lemma *bin-prim-irrefl*: ¬ *bin-prim x x*
using *bin-prim-code* **by** *blast*

lemma *bin-prim-symm* [*sym*]: *bin-prim x y* ⟹ *bin-prim y x*
using *bin-prim-concat-prim-pres-conv*[*of x y*] *bin-prim-concat-prim-pres-conv*[*of y x*]
unfolding *eq-commute*[*of y x*] *insert-commute*[*of y x*]
by *blast*

lemma *bin-prim-commutes*: *bin-prim x y* ⟷ *bin-prim y x*
by (*blast intro*: *bin-prim-symm*)

end

theory *Equations-Basic*
imports
Periodicity-Lemma
Lyndon-Schutzenberger
Submonoids
Binary-Code-Morphisms
begin

Chapter 8

Equations on words - basics

Contains various nontrivial auxiliary or rudimentary facts related to equations. Often moderately advanced or even fairly advanced. May change significantly in the future.

8.1 Miscellanea

8.1.1 Mismatch additions

lemma *mismatch-pref-comm-len*: **assumes** *w1* ∈ ⟨{*u*,*v*}⟩ **and** *w2* ∈ ⟨{*u*,*v*}⟩ **and**
p ≤*p* *w1*
  u · *p* ≤*p* *v* · *w2* **and** |*v*| ≤ |*p*|
shows *u* · *v* = *v* · *u*
proof (*rule ccontr*)
  assume *u* · *v* ≠ *v* · *u*
  then interpret *binary-code u v*
    by *unfold-locales*
  show *False*
    using *bin-code-prefs*[*OF* ‹*w1* ∈ ⟨{*u*,*v*}⟩› ‹*p* ≤*p* *w1*› ‹*w2* ∈ ⟨{*u*,*v*}⟩› ‹|*v*| ≤ |*p*|›]
      ‹*u* · *p* ≤*p* *v* · *w2*›
    by *blast*
qed

lemma *mismatch-pref-comm*: **assumes** *w1* ∈ ⟨{*u*,*v*}⟩ **and** *w2* ∈ ⟨{*u*,*v*}⟩ **and**
  u · *w1* · *v* ≤*p* *v* · *w2* · *u*
shows *u* · *v* = *v* · *u*
  using *assms* **by** *mismatch*

lemma *mismatch-eq-comm*: **assumes** *w1* ∈ ⟨{*u*,*v*}⟩ **and** *w2* ∈ ⟨{*u*,*v*}⟩ **and**
  u · *w1* = *v* · *w2*
shows *u* · *v* = *v* · *u*
  using *assms* **by** *mismatch*

lemmas *mismatch-suf-comm* = *mismatch-pref-comm*[*reversed*] **and**

mismatch-suf-comm-len = mismatch-pref-comm-len[*reversed*, *unfolded rassoc*]

8.1.2 Conjugate words with conjugate periods

lemma *conj-pers-conj-comm-aux*:
assumes $(u \cdot v)^{@}k \cdot u = r \cdot s$ **and** $(v \cdot u)^{@}l \cdot v = (s \cdot r)^{@}m$ **and** $0 < k$ $0 < l$ **and** $2 \leq m$
shows $u \cdot v = v \cdot u$
proof (*rule nemp-comm*)
assume $u \neq \varepsilon$ **and** $v \neq \varepsilon$ **hence** $u \cdot v \neq \varepsilon$ **and** $v \cdot u \neq \varepsilon$ **by** *blast+*
have $l \neq 1$ — impossible by a length argument
proof (*rule notI*)
assume $l = 1$
hence $v \cdot u \cdot v = (s \cdot r)^{@}m$
using *assms*(*2*) **by** *simp*
have $|v \cdot u| + |u| \leq |r \cdot s|$
unfolding *lenmorph add.commute*[*of* $|u|$]
lenarg[*OF assms*(*1*), *unfolded lenmorph pow-len*, *symmetric*]
using ‹$0 < k$› **by** *simp*
hence $|v \cdot u \cdot v \cdot u| \leq 2{*}|r \cdot s|$
unfolding *lenmorph pow-len* **by** *simp*
hence $|v \cdot u \cdot v| < 2{*}|r \cdot s|$
unfolding *lenmorph pow-len* **using** *nemp-len*[*OF* ‹$u \neq \varepsilon$›] **by** *linarith*
from *this*[*unfolded* ‹$v \cdot u \cdot v = (s \cdot r)^{@}m$›]
show *False*
using *mult-le-mono1*[*OF* ‹$2 \leq m$›, *of* $|r| + |s|$]
unfolding *lenmorph pow-len add.commute*[*of* $|s|$] **by** *force*
qed
— we can therefore use the periodicity lemma
hence $2 \leq l$
using ‹$0 < l$› **by** *force*
let *?w* $= (v \cdot u)^{@}l \cdot v$
have *per1*: *?w* $\leq p$ $(v \cdot u) \cdot$ *?w*
unfolding *lassoc pow-comm*[*symmetric*] **by** *force*
have *per2*: *?w* $\leq p$ $(s \cdot r) \cdot$ *?w*
unfolding *assms*(*2*) **using** *pref-pow-ext′* **by** *blast*
have *len*: $|v \cdot u| + |s \cdot r| \leq |$*?w*$|$
proof–
have *len1*: $2{*}|s \cdot r| \leq |$*?w*$|$
using *mult-le-mono1*[*OF* ‹$2 \leq m$›, *of* $|s| + |r|$]
unfolding ‹$(v \cdot u)^{@}l \cdot v = (s \cdot r)^{@}m$› *lenmorph pow-len*.
moreover have *len2*: $2{*}|v \cdot u| \leq |$*?w*$|$
using *mult-le-mono1*[*OF* ‹$2 \leq l$›, *of* $|v| + |u|$]
unfolding *lenmorph pow-len* **by** *linarith*
ultimately show *?thesis*
using *len1 len2* **by** *linarith*
qed
from *two-pers*[*OF per1 per2 len*]
have $(v \cdot u) \cdot (s \cdot r) = (s \cdot r) \cdot (v \cdot u)$.

hence $(v \cdot u)^{@} l \cdot (s \cdot r)^{@} m = (s \cdot r)^{@} m \cdot (v \cdot u)^{@} l$
using *comm-pows-comm* **by** *blast*
from *comm-drop-exp'*[*OF this*[*folded assms*(*2*), *unfolded rassoc cancel*] ‹*0* < *l*›]
show $u \cdot v = v \cdot u$
unfolding *rassoc cancel* **by** *blast*
qed

lemma *conj-pers-conj-comm*: **assumes** $\varrho\ (v \cdot (u \cdot v)^{@} k) \sim \varrho\ ((u \cdot v)^{@} m \cdot u)$ **and** $0 < k$ **and** $0 < m$
shows $u \cdot v = v \cdot u$
proof (*rule nemp-comm*)
let *?v* = $v \cdot (u \cdot v)^{@} k$ **and** *?u* = $(u \cdot v)^{@} m \cdot u$
assume $u \neq \varepsilon$ **and** $v \neq \varepsilon$
hence $u \cdot v \neq \varepsilon$ **and** *?v* $\neq \varepsilon$ **and** *?u* $\neq \varepsilon$ **by** *simp-all*
obtain *r s* **where** $r \cdot s = \varrho$ *?v* **and** $s \cdot r = \varrho$ *?u*
using *conjugE*[*OF assms*(*1*)]**.**
then obtain *k1 k2* **where** *?v* = $(r \cdot s)^{@} k1$ **and** *?u* = $(s \cdot r)^{@} k2$ **and** *0* < *k1* **and** *0* < *k2*
using *primroot-expE*[*of ?u*] *primroot-expE*[*of ?v*] **unfolding** *shift-pow* **by** *metis*
hence *eq*: $(s \cdot r)^{@} k2 \cdot (r \cdot s)^{@} k1 = (u \cdot v)^{@} (m + 1 + k)$
unfolding *pow-add pow-list-one rassoc* **by** *simp*
have *ineq*: *2* ≤ *m* + *1* + *k*
using ‹*0* < *k*› **by** *simp*
consider (*two-two*) *2* ≤ *k1* ∧ *2* ≤ *k2*|
(*one-one*) *k1* = *1* ∧ *k2* = *1* |
(*two-one*) *2* ≤ *k1* ∧ *k2* = *1* |
(*one-two*) *k1* = *1* ∧ *2* ≤ *k2*
using ‹*0* < *k1*› ‹*0* < *k2*› **by** *linarith*
then show $u \cdot v = v \cdot u$
proof (*cases*)
case (*two-two*)
with *Lyndon-Schutzenberger*(*1*)[*OF eq - - ineq*]
have $(s \cdot r) \cdot (r \cdot s) = (r \cdot s) \cdot (s \cdot r)$
using *eqd-eq*[*of s · r r · s r · s s · r*] **by** *fastforce*

from *comm-pows-comm*[*OF this, of k2 k1, folded* ‹*?v* = $(r \cdot s)^{@} k1$› ‹*?u* = $(s \cdot r)^{@} k2$›]
show $u \cdot v = v \cdot u$
by *mismatch*
next
case (*one-one*)
hence $(s \cdot r)\ ^{@}\ k2 \cdot (r \cdot s)\ ^{@}\ k1 = (s \cdot r) \cdot (r \cdot s)$
by *simp*
from *eq*[*unfolded conjunct1*[*OF one-one*] *conjunct2*[*OF one-one*] *pow-list-1*]
pow-nemp-imprim[*OF ineq, folded eq*[*unfolded this*]]
Lyndon-Schutzenberger-conjug[*of s · r r · s, OF conjugI'*]
have $(s \cdot r) \cdot (r \cdot s) = (r \cdot s) \cdot (s \cdot r)$ **by** *metis*

from *comm-pows-comm*[*OF this, of k2 k1, folded* ‹*?v* = $(r \cdot s)^{@} k1$› ‹*?u* = (*s* ·

$r)^{@}k2\rangle$, *folded shift-pow*]
show $u \cdot v = v \cdot u$
by *mismatch*
next
case (*two-one*)
hence *?u* $= s \cdot r$
using ‹*?u* $= (s \cdot r)^{@}k2\rangle$
by *simp*
from ‹*?v* $= (r \cdot s)^{@}k1\rangle$[*folded shift-pow*, *unfolded this*]
have $(v \cdot u)\ ^{@}\ k \cdot v = (r \cdot s)^{@}k1$**.**
from *conj-pers-conj-comm-aux*[*OF* ‹*?u* $= s \cdot r\rangle$ *this* ‹$0 < m\rangle$ ‹$0 < k\rangle$]
show $u \cdot v = v \cdot u$
using *two-one* **by** *auto*
next
case (*one-two*)
hence *?v* $= r \cdot s$
using ‹*?v* $= (r \cdot s)^{@}k1\rangle$ **by** *simp*
from ‹*?u* $= (s \cdot r)^{@}k2\rangle$[*unfolded this*]
have $(u \cdot v)\ ^{@}\ m \cdot u = (s \cdot r)\ ^{@}\ k2$**.**
from *conj-pers-conj-comm-aux*[*OF* ‹*?v* $= r \cdot s\rangle$[*folded shift-pow*] *this* ‹$0 < k\rangle$ ‹$0 < m\rangle$]
show $u \cdot v = v \cdot u$
using *one-two* **by** *argo*
qed
qed

hide-fact *conj-pers-conj-comm-aux*

8.1.3 Covering uvvu

lemma *uv-fac-uvv*: **assumes** $p \cdot u \cdot v \leq p\ u \cdot v \cdot v$ **and** $p \neq \varepsilon$ **and** $p \leq s\ w$ **and** $w \in \langle\{u,v\}\rangle$
shows $u \cdot v = v \cdot u$
proof (*rule nemp-comm*)
assume $u \neq \varepsilon$ **and** $v \neq \varepsilon$
obtain s **where** $u \cdot v \cdot v = p \cdot u \cdot v \cdot s$
using ‹$p \cdot u \cdot v \leq p\ u \cdot v \cdot v\rangle$ **by** (*auto simp add*: *prefix-def*)
obtain p' **where** $u \cdot p' = p \cdot u$ **and** $p' \cdot v \cdot s = v \cdot v$
using *eqdE*[*of* $u\ v \cdot v\ p \cdot u\ v \cdot s$, *unfolded rassoc*, *OF* ‹$u \cdot v \cdot v = p \cdot u \cdot v \cdot s\rangle$ *suf-len*']**.**
hence $p' \neq \varepsilon$
using ‹$p \neq \varepsilon\rangle$ **by** *force*
have $p' \cdot v \cdot s = v \cdot v$
using ‹$u \cdot v \cdot v = p \cdot u \cdot v \cdot s\rangle$ ‹$u \cdot p' = p \cdot u\rangle$ *cancel rassoc* **by** *metis*
from *mid-sq*[*OF this*]
have $v \cdot p' = p' \cdot v$ **by** *simp*
from *this comm-primroots*[*OF* ‹$v \neq \varepsilon\rangle$ ‹$p' \neq \varepsilon\rangle$]
have $\varrho\ v = \varrho\ p'$
by *simp*

obtain m **where** $p' = \varrho\ v^{@}m\ 0 < m$
using *primroot-expE* **unfolding** ‹$\varrho\ v = \varrho\ p'$› **by** *metis*
have $(u \cdot \varrho\ v^{@}(m-1)) \cdot \varrho\ v \leq s\ (\varrho\ v \cdot w) \cdot u$
using ‹$p \leq s\ w$›
unfolding *rassoc pow-pos2*[*OF* ‹$0 < m$›, *symmetric*] ‹$p' = \varrho\ v^{@}m$›[*symmetric*]
‹$u \cdot p' = p \cdot u$› *suffix-def* **by** *force*
thus $u \cdot v = v \cdot u$
using ‹$w \in \langle\{u, v\}\rangle$› **by** *mismatch*
qed

lemmas *uv-fac-uvv-suf* = *uv-fac-uvv*[*reversed*, *unfolded rassoc*]

lemma $u \leq p\ v \Longrightarrow u' \leq p\ v' \Longrightarrow u \wedge_p u' \neq u \Longrightarrow u \wedge_p u' \neq u' \Longrightarrow u \wedge_p u' = v \wedge_p v'$
using *lcp.absorb2 lcp.orderE lcp-rulers pref-compE* **by** *metis*

lemma *comm-puv-pvs-eq-uq*: **assumes** $p \cdot u \cdot v = u \cdot v \cdot p$ **and** $p \cdot v \cdot s = u \cdot q$ **and**
$p \leq p\ u\ q \leq p\ w$ **and** $s \leq p\ \ w'$ **and**
$w \in \langle\{u,v\}\rangle$ **and** $w' \in \langle\{u,v\}\rangle$ **and** $|u| \leq |s|$
shows $u \cdot v = v \cdot u$
proof (*rule ccontr*)
assume $u \cdot v \neq v \cdot u$
then interpret *binary-code u v*
by *unfold-locales*
write *bin-code-lcp* (α) **and**
bin-code-mismatch-fst (c_0) **and**
bin-code-mismatch-snd (c_1)
have $|\alpha| < |v \cdot s|$
using ‹$|u| \leq |s|$› *bin-lcp-short* **by** *force*
show *False*
proof (*cases*)
assume $p = \varepsilon$
hence $v \cdot s = u \cdot q$
using ‹$p \cdot v \cdot s = u \cdot q$› **by** *fastforce*
with ‹$|\alpha| < |v \cdot s|$› ‹$|\alpha| < |v \cdot s|$›[*unfolded this*]
have $\alpha \cdot [c_1] \leq p\ v \cdot s$ **and** $\alpha \cdot [c_0] \leq p\ u \cdot q$
using ‹$s \leq p\ \ w'$› ‹$w' \in \langle\{u,v\}\rangle$› ‹$q \leq p\ w$› ‹$w \in \langle\{u,v\}\rangle$›
bin-lcp-mismatch-pref-all-snd bin-lcp-mismatch-pref-all-fst ‹$v \cdot s = u \cdot q$›
by *blast+*
thus *False*
unfolding ‹$v \cdot s = u \cdot q$› **using** *bin-mismatch-neq*
by (*simp add*: *same-sing-pref*)
next
assume $p \neq \varepsilon$
show *False*
proof−

— Preliminaries
have $u \neq \varepsilon$ **and** $v \neq \varepsilon$ **and** $u \cdot v \neq v \cdot u$
by (*simp-all add*: *bin-fst-nemp bin-snd-nemp non-comm*)
have $w \cdot u \cdot v \in \langle\{u, v\}\rangle$
using ‹$w \in \langle\{u, v\}\rangle$› **by** *blast*
have $|w \cdot u \cdot v| \geq |\alpha|$
using *bin-lcp-short* **by** *auto*
— The main idea: compare maximum p-prefixes
— the maximum p-prefix of $p \cdot v \cdot s$
have *p-pref1*: $p \cdot v \cdot s \wedge_p p \cdot p \cdot v \cdot s = p \cdot \alpha$
using *bin-per-root-max-pref-short*[*of p s w'*, *OF* - ‹$s \leq_p w'$› ‹$w' \in \langle\{u, v\}\rangle$›]
‹$p \neq \varepsilon$› ‹$p \cdot u \cdot v = u \cdot v \cdot p$› **unfolding** *lcp-ext-left cancel take-all*[*OF less-imp-le*[*OF*‹$|\alpha| < |v \cdot s|$›]] **by** *force*
— the maximum p-prefix of $u \cdot w \cdot u \cdot v$ is at least $u \cdot \alpha$
have *p-pref2*: $u \cdot \alpha \leq_p u \cdot (w \cdot u \cdot v) \wedge_p p \cdot u \cdot (w \cdot u \cdot v)$
using *bin-root-max-pref-long*[*OF* ‹$p \cdot u \cdot v = u \cdot v \cdot p$› *self-pref* ‹$w \cdot u \cdot v \in \langle\{u, v\}\rangle$› ‹$|\alpha| \leq |w \cdot u \cdot v|$ ›].
— But those maximum p-prefixes are equal
have $u \cdot w \cdot u \cdot v \wedge_p p \cdot u \cdot w \cdot u \cdot v = p \cdot v \cdot s \wedge_p p \cdot p \cdot v \cdot s$
proof(*rule lcp-rulers'*)
show $\neg\ p \cdot v \cdot s \bowtie p \cdot p \cdot v \cdot s$
proof (*rule notI*)
assume $p \cdot v \cdot s \bowtie p \cdot p \cdot v \cdot s$
hence $p \cdot v \cdot s \wedge_p p \cdot p \cdot v \cdot s = p \cdot v \cdot s$
using ‹$p \neq \varepsilon$› *lcp.absorb1 pref-compE same-sufix-nil* **by** *meson*
from *this*[*unfolded p-pref1 cancel*]
show *False*
using *bin-lcp-short* ‹$|u| \leq |s|$› **by** *force*
qed
show $p \cdot v \cdot s \leq_p u \cdot (w \cdot u \cdot v)$ $p \cdot p \cdot v \cdot s \leq_p p \cdot u \cdot w \cdot u \cdot v$
by (*simp-all add*: *assms*(*2*) *assms*(*4*))
qed
from *p-pref2*[*unfolded rassoc this p-pref1*]
have $p = u$
using ‹$p \leq_p u$› *pref-cancel-right* **by** *force*
thus *False*
using ‹$p \cdot u \cdot v = u \cdot v \cdot p$› *non-comm* **by** *blast*
qed
qed
qed

lemma assumes $u \cdot v \cdot v \cdot u = p \cdot u \cdot v \cdot u \cdot q$ **and** $p \neq \varepsilon$ **and** $q \neq \varepsilon$
shows $u \cdot v = v \cdot u$
oops — counterexample: v = abaaba, u = a, p = aab, q = baa; aab.a.abaaba.a.baa = a.abaaba.abaaba.a

lemma *uvu-pref-uvv*: **assumes** $p \cdot u \cdot v \cdot v \cdot s = u \cdot v \cdot u \cdot q$ **and**

$p \leq_p u$ **and** $q \leq_p w$ **and** $s \leq_p w'$ **and**
$w \in \langle\{u,v\}\rangle$ **and** $w' \in \langle\{u,v\}\rangle$ **and** $|u| \leq |s|$
shows $u \cdot v = v \cdot u$
proof(*rule nemp-comm*)
have $|p \cdot u \cdot v| \leq |u \cdot v \cdot u|$
using ‹$p \leq_p u$› **unfolding** *lenmorph*
by (*simp add*: *prefix-length-le*)

— p commutes with $u \cdot v$
have $p \cdot (u \cdot v) = (u \cdot v) \cdot p$
by (*rule pref-marker*[*of* $u \cdot v \cdot u$], *force*)
(*rule eq-le-pref*, *use assms* **in** *force*, *fact*)

have $p \cdot v \cdot s = u \cdot q$
proof−
have $((u \cdot v) \cdot p) \cdot v \cdot s = (u \cdot v) \cdot u \cdot q$
unfolding ‹$p \cdot u \cdot v = (u \cdot v) \cdot p$›[*symmetric*] **unfolding** *rassoc* **by** *fact*
from *this*[*unfolded rassoc cancel*]
show *?thesis*.
qed

from *comm-puv-pvs-eq-uq*[*OF* ‹$p \cdot (u \cdot v) = (u \cdot v) \cdot p$›[*unfolded rassoc*] *this assms*(*2*−)]
show $u \cdot v = v \cdot u$.
qed

lemma *uvu-pref-uvvu*: **assumes** $p \cdot u \cdot v \cdot v \cdot u = u \cdot v \cdot u \cdot q$ **and**
$p \leq_p u$ **and** $q \leq_p w$ **and** $w \in \langle\{u,v\}\rangle$
shows $u \cdot v = v \cdot u$
using *uvu-pref-uvv*[*OF* ‹$p \cdot u \cdot v \cdot v \cdot u = u \cdot v \cdot u \cdot q$› ‹$p \leq_p u$› ‹$q \leq_p w$› - ‹$w \in \langle\{u,v\}\rangle$›, *of u*]
by *blast*

lemma *uvu-pref-uvvu-interp*: **assumes** *interp*: $p\ u \cdot v \cdot v \cdot u\ s \sim_{\mathcal{I}} ws$ **and**
$[u, v, u] \leq_p ws$ **and** $ws \in lists\ \{u,v\}$
shows $u \cdot v = v \cdot u$
proof−
note *fac-interpD*[*OF interp*]
obtain ws' **where** $[u,v,u] \cdot ws' = ws$ **and** $ws' \in lists\ \{u,v\}$
using ‹$[u, v, u] \leq_p ws$› ‹$ws \in lists\ \{u,v\}$› **by** (*force simp add*: *prefix-def*)
have $p \cdot u \cdot v \cdot v \cdot u \cdot s = u \cdot v \cdot u \cdot concat\ ws'$
using ‹$p \cdot (u \cdot v \cdot v \cdot u) \cdot s = concat\ ws$›[*folded* ‹$[u,v,u] \cdot ws' = ws$›, *unfolded concat-morph rassoc*]
by *simp*
from *lenarg*[*OF this*, *unfolded lenmorph*]
have $|s| \leq |concat\ ws'|$
by *auto*

hence $s \leq_s$ *concat ws'*
using *eqd*[*reversed*, *OF* ‹$p \cdot u \cdot v \cdot v \cdot u \cdot s = u \cdot v \cdot u \cdot$ *concat ws'*›[*unfolded lassoc*]]
by *blast*
note *local-rule* = *uvu-pref-uvv*[*of p u v u concat ws'*$^{<-1}$*s concat ws' u*]
show $u \cdot v = v \cdot u$
proof (*rule local-rule*)
show $p \leq_p u$
using ‹$p <_p$ *hd ws*› *pref-hd-eq*[*OF* ‹[*u*, *v*, *u*] $\leq_p$ *ws*› *list.distinct*(*1*)[*of u* [*v*,*u*], *symmetric*]]
by *force*
have $p \cdot u \cdot v \cdot v \cdot u \cdot s = u \cdot v \cdot u \cdot$ (*concat ws'*$^{<-1}$*s*) $\cdot$ *s*
using ‹$p \cdot u \cdot v \cdot v \cdot u \cdot s = u \cdot v \cdot u \cdot$ *concat ws'*› **unfolding** *rq-suf*[*OF* ‹*s* $\leq_s$ *concat ws'*›]**.**
thus $p \cdot u \cdot v \cdot v \cdot u = u \cdot v \cdot u \cdot$ *concat ws'*$^{<-1}$*s*
by *simp*
show *concat ws'* $\in \langle\{u,v\}\rangle$
using ‹*ws'* $\in$ *lists* $\{u,v\}$› **by** *blast*
show *concat ws'*$^{<-1}$*s* $\leq_p$ *concat ws'*
using *rq-suf*[*OF* ‹*s* $\leq_s$ *concat ws'*›] **by** *force*
qed *auto*
qed

lemmas *uvu-suf-uvvu* = *uvu-pref-uvvu*[*reversed*, *unfolded rassoc*] **and**
uvu-suf-uvv = *uvu-pref-uvv*[*reversed*, *unfolded rassoc*]

lemma *uvu-suf-uvvu-interp*: *p u* $\cdot$ *v* $\cdot$ *v* $\cdot$ *u s* $\sim_{\mathcal{I}}$ *ws* $\Longrightarrow$ [*u*, *v*, *u*] $\leq_s$ *ws*
$\Longrightarrow$ *ws* $\in$ *lists* $\{u,v\}$ $\Longrightarrow$ *u* $\cdot$ *v* = *v* $\cdot$ *u*
by (*rule uvu-pref-uvvu-interp*[*reversed*, *unfolded rassoc emp-simps*, *symmetric*, *of p u v s ws*],
simp, *force*, *simp add*: *image-iff rev-in-lists rev-map*)

8.1.4 Conjugate words

lemma *conjug-pref-suf-mismatch*: **assumes** *w1* $\in \langle\{r\cdot s,s\cdot r\}\rangle$ **and** *w2* $\in \langle\{r\cdot s,s\cdot r\}\rangle$ **and** *r* $\cdot$ *w1* = *w2* $\cdot$ *s*
shows $r = s \vee r = \varepsilon \vee s = \varepsilon$
proof (*rule ccontr*)
assume $\neg$ $(r = s \vee r = \varepsilon \vee s = \varepsilon)$
hence $r \neq s$ **and** $r \neq \varepsilon$ **and** $s \neq \varepsilon$ **by** *simp-all*
from *assms*
show *False*
proof (*induct* |*w1*| *arbitrary*: *w1 w2 rule*: *less-induct*)
case *less*
have *w1* $\in \langle\{r,s\}\rangle$
using ‹*w1* $\in \langle\{r\cdot s,s\cdot r\}\rangle$› **by** *force*
obtain *w1'* **where** (*w1* = ε $\vee$ *w1* = *r* $\cdot$ *s* $\cdot$ *w1'* $\vee$ *w1* = *s* $\cdot$ *r* $\cdot$ *w1'*) $\wedge$ *w1'* $\in \langle\{r\cdot s,s\cdot r\}\rangle$
using *hull.cases*[*OF* ‹*w1* $\in \langle\{r\cdot s,s\cdot r\}\rangle$›] *empty-iff insert-iff rassoc* ‹*w1* $\in \langle\{r$

$\cdot\ s,\ s \cdot r\}\rangle\rangle$ **by** *metis*
hence $w1' \in \langle\{r{\cdot}s,s{\cdot}r\}\rangle$ **and** *cases1*: $(w1 = \varepsilon \vee w1 = r \cdot s \cdot w1' \vee w1 = s \cdot r \cdot w1')$ **by** *blast+*
hence $w1' \in \langle\{r,s\}\rangle$ **by** *force*
obtain $w2'$ **where** $(w2 = \varepsilon \vee w2 = r \cdot s \cdot w2' \vee w2 = s \cdot r \cdot w2') \wedge w2' \in \langle\{r{\cdot}s,s{\cdot}r\}\rangle$
using *hull.cases*[*OF* ‹$w2 \in \langle\{r{\cdot}s,s{\cdot}r\}\rangle$›] *empty-iff insert-iff rassoc* ‹$w1 \in \langle\{r \cdot s,\ s \cdot r\}\rangle$› **by** *metis*
hence $w2' \in \langle\{r{\cdot}s,s{\cdot}r\}\rangle$ **and** *cases2*: $(w2 = \varepsilon \vee w2 = r \cdot s \cdot w2' \vee w2 = s \cdot r \cdot w2')$ **by** *blast+*
hence $w2' \in \langle\{r,s\}\rangle$ **by** *force*
consider (*empty2*) $w2 = \varepsilon \mid$ (*sr2*) $w2 = s \cdot r \cdot w2' \mid$ (*rs2*) $w2 = r \cdot s \cdot w2'$**using** *cases2* **by** *blast*
thus *False*
proof (*cases*)
case *empty2*
consider (*empty1*) $w1 = \varepsilon \mid$ (*sr1*) $w1 = s \cdot r \cdot w1' \mid$ (*rs1*) $w1 = r \cdot s \cdot w1'$
using *cases1* **by** *blast*
thus *False*
proof (*cases*)
case *empty1*
show *False*
using ‹$r \cdot w1 = w2 \cdot s$›[*unfolded empty1 empty2 rassoc*] ‹$r \neq s$› **by** *simp*
next
case *sr1*
show *False*
using ‹$r \cdot w1 = w2 \cdot s$›[*unfolded sr1 empty2 rassoc*] ‹$r \neq \varepsilon$› *fac-triv* **by** *auto*
next
case *rs1*
show *False*
using ‹$r \cdot w1 = w2 \cdot s$›[*unfolded rs1 empty2 rassoc emp-simps*] ‹$r \neq \varepsilon$› *fac-triv*[*of* $r \cdot r\ s\ w1'$, *unfolded rassoc*] **by** *force*
qed
next
case *sr2*
have $r \cdot s = s \cdot r$
using ‹$w2' \in \langle\{r,s\}\rangle$› ‹$w1 \in \langle\{r,s\}\rangle$› ‹$r \cdot w1 = w2 \cdot s$›[*unfolded sr2 rassoc*]
by (*mismatch*)
consider (*empty1*) $w1 = \varepsilon \mid$ (*sr1*) $w1 = s \cdot r \cdot w1' \mid$ (*rs1*) $w1 = r \cdot s \cdot w1'$ **using** *cases1* **by** *blast*
thus *False*
proof (*cases*)
case *empty1*
then show *False*
using ‹$r \cdot w1 = w2 \cdot s$›[*unfolded sr2 empty1 rassoc cancel emp-simps*, *symmetric*] ‹$s \neq \varepsilon$› *fac-triv* **by** *blast*
next
case *rs1*

then have $r \cdot s = s \cdot r$
using ‹$w2' \in \langle\{r,s\}\rangle$› ‹$w1' \in \langle\{r,s\}\rangle$› ‹$r \cdot w1 = w2 \cdot s$›[*unfolded rs1 sr2 rassoc cancel*]
by *mismatch*
hence $r \cdot w1' = w2' \cdot s$
using ‹$r \cdot w1 = w2 \cdot s$›[*unfolded rs1 sr2*] *rassoc cancel* **by** *metis*
from *less.hyps*[*OF* - ‹$w1' \in \langle\{r \cdot s,\ s \cdot r\}\rangle$› ‹$w2' \in \langle\{r \cdot s,\ s \cdot r\}\rangle$› *this*]
show *False*
using *lenarg*[*OF* ‹$w1 = r \cdot s \cdot w1'$›, *unfolded lenmorph*] *nemp-len*[*OF* ‹$s \neq \varepsilon$›] **by** *linarith*
next
case *sr1*
then have $r \cdot s = s \cdot r$
using ‹$w2' \in \langle\{r,s\}\rangle$› ‹$w1' \in \langle\{r,s\}\rangle$› ‹$r \cdot w1 = w2 \cdot s$›[*unfolded sr1 sr2 rassoc cancel*]
by *mismatch*
hence $r \cdot w1' = w2' \cdot s$
using ‹$r \cdot w1 = w2 \cdot s$›[*unfolded sr1 sr2*] *rassoc cancel* **by** *metis*
from *less.hyps*[*OF* - ‹$w1' \in \langle\{r \cdot s,\ s \cdot r\}\rangle$› ‹$w2' \in \langle\{r \cdot s,\ s \cdot r\}\rangle$› *this*]
show *False*
using *less.hyps*[*OF* - ‹$w1' \in \langle\{r \cdot s,\ s \cdot r\}\rangle$› ‹$w2' \in \langle\{r \cdot s,\ s \cdot r\}\rangle$› ‹$r \cdot w1' = w2' \cdot s$›]
lenarg[*OF* ‹$w1 = s \cdot r \cdot w1'$›, *unfolded lenmorph*] *nemp-len*[*OF* ‹$s \neq \varepsilon$›] **by** *linarith*
qed
next
case *rs2*
consider (*empty1*) $w1 = \varepsilon$ | (*sr1*) $w1 = s \cdot r \cdot w1'$ | (*rs1*) $w1 = r \cdot s \cdot w1'$ **using** *cases1* **by** *blast*
thus *False*
proof (*cases*)
case *empty1*
then show *False*
using ‹$r \cdot w1 = w2 \cdot s$›[*unfolded rs2 empty1 rassoc cancel*] ‹$s \neq \varepsilon$› **by** *blast*
next
case *rs1*
then have $r \cdot s = s \cdot r$
using ‹$w2' \in \langle\{r,s\}\rangle$› ‹$w1' \in \langle\{r,s\}\rangle$› ‹$r \cdot w1 = w2 \cdot s$›[*unfolded rs2 rs1 rassoc cancel*]
by *mismatch*
hence $r \cdot w1' = w2' \cdot s$
using ‹$r \cdot w1 = w2 \cdot s$›[*unfolded rs1 rs2*] *rassoc cancel* **by** *metis*
from *less.hyps*[*OF* - ‹$w1' \in \langle\{r \cdot s,\ s \cdot r\}\rangle$› ‹$w2' \in \langle\{r \cdot s,\ s \cdot r\}\rangle$› *this*]
show *False*
using *less.hyps*[*OF* - ‹$w1' \in \langle\{r \cdot s,\ s \cdot r\}\rangle$› ‹$w2' \in \langle\{r \cdot s,\ s \cdot r\}\rangle$› ‹$r \cdot w1' = w2' \cdot s$›]
lenarg[*OF* ‹$w1 = r \cdot s \cdot w1'$›, *unfolded lenmorph*] *nemp-len*[*OF* ‹$s \neq \varepsilon$›] **by** *linarith*
next

case *sr1*
then show *False*
using *less.hyps*[*OF* - ‹$w1' \in \langle\{r \cdot s, s \cdot r\}\rangle$› ‹$w2' \in \langle\{r \cdot s, s \cdot r\}\rangle$› ‹$r \cdot w1 = w2 \cdot s$›[*unfolded rs2 sr1 rassoc cancel*]]
lenarg[*OF* ‹$w1 = s \cdot r \cdot w1'$›, *unfolded lenmorph*] *nemp-len*[*OF* ‹$s \neq \varepsilon$›]
by *linarith*
qed
qed
qed
qed

lemma *conjug-conjug-primroots*: **assumes** $u \neq \varepsilon$ **and** $r \neq \varepsilon$ **and** $\varrho\ (u \cdot v) = r \cdot s$ **and** $\varrho\ (v \cdot u) = s \cdot r$
obtains $k\ m$ **where** $(r \cdot s)^{@}k \cdot r = u$ **and** $(s \cdot r)^{@}m \cdot s = v$
proof−
have $v \cdot u \neq \varepsilon$ **and** $u \cdot v \neq \varepsilon$
using ‹$u \neq \varepsilon$› **by** *blast*+
have $\varrho\ (s \cdot r) = s \cdot r$
using *primroot-idemp*[*of* $v \cdot u$, *unfolded* ‹$\varrho\ (v \cdot u) = s \cdot r$›].
obtain n **where** $(r \cdot s)^{@}n = u \cdot v\ 0 < n$
using *primroot-expE*[*unfolded* ‹$\varrho\ (u \cdot v) = r \cdot s$›]
using *assms*(*3*) **by** *metis*
obtain n' **where** $(s \cdot r)^{@}\ n' = v \cdot u\ 0 < n'$
using *primroot-expE*[*of* $v \cdot u$,*unfolded* ‹$\varrho\ (v \cdot u) = s \cdot r$›].
have $(s \cdot u) \cdot (s \cdot r) = (s \cdot r) \cdot (s \cdot u)$
proof (*rule pref-marker*)
show $(s \cdot u) \cdot s \cdot r \leq p\ s \cdot (r \cdot s)^{@}(n+ n)$
unfolding *rassoc pow-add* ‹$(r \cdot s)^{@}n = u \cdot v$›
unfolding *lassoc* ‹$(s \cdot r)^{@}n' = v \cdot u$›[*symmetric*] **using** ‹$0 < n'$› **by** *comparison*
show $s \cdot (r \cdot s)\ ^{@}\ (n + n) \leq p\ (s \cdot r) \cdot s \cdot (r \cdot s)\ ^{@}\ (n + n)$
by *comparison*
qed
from *comm-primroot-exp*[*OF primroot-nemp*[*OF* ‹$v \cdot u \neq \varepsilon$›, *unfolded* ‹$\varrho\ (v \cdot u) = s \cdot r$›] *this*]
obtain k **where** $(s \cdot r)^{@}k = s \cdot u$
unfolding ‹$\varrho\ (s \cdot r) = s \cdot r$›.
from *suf-nemp*[*OF* ‹$u \neq \varepsilon$›, *of s*, *folded this*]
have $0 < k$
by *blast*
have *u*: $(r \cdot s)^{@}(k-1) \cdot r = u$
using ‹$(s \cdot r)^{@}k = s \cdot u$› **unfolding** *pow-pos*[*OF* ‹$0 < k$›] *rassoc cancel shift-pow*
by *fast*
from *exp-pref-cancel*[*OF* ‹$(r \cdot s)^{@}n = u \cdot v$›[*folded this*, *unfolded rassoc*, *symmetric*]]
have $r \cdot v = (r \cdot s)\ ^{@}\ (n + 1 - k)$
using ‹$0 < k$› **by** *fastforce*
from *pref-nemp*[*OF* ‹$r \neq \varepsilon$›, *of v*, *unfolded this*]
have $0 < n + 1 - k$
by *blast*

from ‹$r \cdot v = (r \cdot s)\ ^{@}\ (n + 1 - k)$›[*unfolded pow-pos*[*OF* ‹$0 < n + 1 - k$›] *rassoc cancel shift-pow*[*symmetric*], *symmetric*]
have *v*: $(s \cdot r)^{@}(n + 1 - k - 1) \cdot s = v$**.**
show *thesis*
using *that*[*OF u v*]**.**
qed

8.1.5 Predicate "commutes"

definition *commutes* :: *'a list set* ⇒ *bool*
where *commutes A* = $(\forall\, x\ y.\ x \in A \longrightarrow y \in A \longrightarrow x{\cdot}y = y{\cdot}x)$

lemma *commutesE*: *commutes A* ⟹ $x \in A$ ⟹ $y \in A$ ⟹ $x{\cdot}y = y{\cdot}x$
using *commutes-def* **by** *blast*

lemma *commutes-root*: **assumes** *commutes A*
obtains *r* **where** $\bigwedge x.\ x \in A \Longrightarrow x \in \langle\{r\}\rangle$
using *assms comm-primroots emp-in sing-gen-primroot*
unfolding *commutes-def*
by *metis*

lemma *commutes-primroot*: **assumes** *commutes A*
obtains *r* **where** $\bigwedge x.\ x \in A \Longrightarrow x \in \langle\{r\}\rangle$ **and** *primitive r*
by (*cases* $\forall\ x \in A.\ x = \varepsilon$, *fast*)
(*use assms*[*unfolded commutes-def*] *comm-primroots emp-in sing-gen-primroot primroot-prim* **in** *metis*)

lemma *commutesI* [*intro*]: $(\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow x{\cdot}y = y{\cdot}x) \Longrightarrow$ *commutes A*
unfolding *commutes-def*
by *blast*

lemma *commutesI'*: **assumes** $x \neq \varepsilon$ **and** $\bigwedge y.\ y \in A \Longrightarrow x{\cdot}y = y{\cdot}x$
shows *commutes A*
proof–
have $\bigwedge x'\ y'.\ x' \in A \Longrightarrow y' \in A \Longrightarrow x'{\cdot}y' = y'{\cdot}x'$
proof–
fix $x'\ y'$
assume $x' \in A\ y' \in A$
hence $x'{\cdot}x = x{\cdot}x'$ **and** $y'{\cdot}x = x{\cdot}y'$
using *assms*(*2*) **by** *auto*+
from *comm-trans*[*OF this assms*(*1*)]
show $x'{\cdot}y' = y'{\cdot}x'$**.**
qed
thus *?thesis*
by (*simp add*: *commutesI*)
qed

lemma *commutesI-root*[*intro*]: $(\bigwedge x.\ x \in A \Longrightarrow x \in \langle\{t\}\rangle) \Longrightarrow$ *commutes* A
by (*meson comm-rootI commutesI*)

lemma *commutesI-ex-root*: $\exists\ t.\ \forall\ x \in A.\ x \in \langle\{t\}\rangle \Longrightarrow$ *commutes* A
by *fast*

lemma *commutes-sub*: *commutes* $A \Longrightarrow B \subseteq A \Longrightarrow$ *commutes* B
by (*simp add*: *commutes-def subsetD*)

lemma *commutes-insert*: *commutes* $A \Longrightarrow x \in A \Longrightarrow x \neq \varepsilon \Longrightarrow\ x{\cdot}y = y{\cdot}x \Longrightarrow$ *commutes* (*insert* y A)
using *commutesE*[*of* A x] *commutesI'*[*of* x *insert* y A] *insertE*
by *blast*

lemma *commutes-emp* [*simp*]: *commutes* $\{\varepsilon, w\}$
by (*simp add*: *commutes-def*)

lemma *commutes-emp'*[*simp*]: *commutes* $\{w, \varepsilon\}$
by (*simp add*: *commutes-def*)

lemma *commutes-cancel*: **assumes** $y \in A$ **and** $x \cdot y \in A$ **and** *commutes* A
shows *commutes* (*insert* x A)
proof–
from *commutes-root*[*OF* ‹*commutes* A›]
obtain r **where** $(\bigwedge x.\ x \in A \Longrightarrow x \in \langle\{r\}\rangle)$
by *metis*
hence $y \in \langle\{r\}\rangle$ **and** $(x \cdot y) \in \langle\{r\}\rangle$
using ‹$y \in A$› ‹$x \cdot y \in A$› **by** *blast*+
hence $x \in \langle\{r\}\rangle$
by *force*
thus *commutes* (*insert* x A)
using ‹$\bigwedge x.\ x \in A \Longrightarrow x \in \langle\{r\}\rangle$› **by** *blast*
qed

lemma *commutes-cancel'*: **assumes** $x \in A$ **and** $x \cdot y \in A$ **and** *commutes* A
shows *commutes* (*insert* y A)
proof–
from *commutes-root*[*OF* ‹*commutes* A›]
obtain r **where** $(\bigwedge x.\ x \in A \Longrightarrow x \in \langle\{r\}\rangle)$
by *metis*
hence $x \in \langle\{r\}\rangle$ **and** $x \cdot y \in \langle\{r\}\rangle$
using ‹$x \in A$› ‹$x \cdot y \in A$› **by** *blast*+
hence $y \in \langle\{r\}\rangle$
using *sing-gen-pref-cancel* **by** *auto*
thus *commutes* (*insert* y A)
using ‹$\bigwedge x.\ x \in A \Longrightarrow\ x \in \langle\{r\}\rangle$› **by** *blast*
qed

8.1.6 Strong elementary lemmas

Discovered by smt

lemma *xyx-per-comm*: **assumes** $x{\cdot}y{\cdot}x \leq_p q{\cdot}x{\cdot}y{\cdot}x$
and $q \neq \varepsilon$ **and** $q \leq_p y \cdot q$
shows $x{\cdot}y = y{\cdot}x$
proof(*cases*)
assume $x \cdot y \leq_p q$
from *pref-marker*[*OF* ‹$q \leq_p y \cdot q$› *this*]
show $x \cdot y = y \cdot x$**.**
next
have $(x \cdot y) \cdot x \leq_p q \cdot x \cdot y \cdot x$ **unfolding** *rassoc* **by** *fact*
assume $\neg\ x \cdot y \leq_p q$
hence $q \leq_p\ x \cdot y$
using *ruler-prefE*[*OF* ‹$(x \cdot y) \cdot x \leq_p q \cdot x \cdot y \cdot x$›] **by** *argo*
from *pref-prolong*[*OF* ‹$q \leq_p y \cdot q$› *this*, *unfolded lassoc*]
have$q \leq_p (y \cdot x) \cdot y$**.**
from *ruler-pref'*[*OF this*, *THEN disjE*] ‹$q \leq_p x \cdot y$›
have $q \leq_p y \cdot x$
using *pref-trans*[*OF* - ‹$q \leq_p x \cdot y$›, *of* $y \cdot x$, *THEN pref-comm-eq*] **by** *metis*
from *pref-cancel'*[*OF this*, *of* x]
have $x \cdot q = q \cdot x$
using *pref-marker*[*OF* ‹$x \cdot y \cdot x \leq_p q \cdot x \cdot y \cdot x$›, *of* x] **by** *blast*
hence $x \cdot y \cdot x \leq_p x \cdot x \cdot y \cdot x$
using *root-comm-root*[*OF* - - ‹$q \neq \varepsilon$›, *of* $x \cdot y \cdot x\ x$] ‹$x \cdot y \cdot x \leq_p q \cdot x \cdot y \cdot x$›
by *fast*
thus $x{\cdot}y = y{\cdot}x$
by *mismatch*
qed

lemma *two-elem-root-suf-comm*: **assumes** $u \leq_p v \cdot u$ **and** $v \leq_s p \cdot u$ **and** $p \in \langle\{u,v\}\rangle$
shows $u \cdot v = v \cdot u$
using *root-suf-comm*[*OF* ‹$u \leq_p v \cdot u$› *two-elem-suf*[*OF* ‹$v \leq_s p \cdot u$› ‹$p \in \langle\{u,v\}\rangle$›], *symmetric*]**.**

8.1.7 Binary words without a letter square

lemma *no-repetition-list*:
assumes *set ws* $\subseteq \{a,b\}$
and *not-per*: $\neg\ ws \leq_p [a,b] \cdot ws\ \neg\ ws \leq_p [b,a] \cdot ws$
and *not-square*: $\neg\ [a,a] \leq_f ws$ **and** $\neg\ [b,b] \leq_f ws$
shows *False*
using *assms*
proof (*induction ws*)
case (*Cons d ws*)
show *?case*
proof (*rule Cons.IH*)
show *set ws* $\subseteq \{a,b\}$

```
    using ‹set (d # ws) ⊆ {a, b}› by auto
  have ws ≠ ε
    using Cons.IH Cons.prems by force
  from hd-tl[OF this]
  have hd ws ≠ d
    using Cons.prems(1,4−5) hd-pref[OF ‹ws ≠ ε›] by force
  thus ¬ [a, a] ≤f ws and ¬ [b, b] ≤f ws
    using Cons.prems(4−5) unfolding sublist-code(3) by blast+
  show ¬ ws ≤p [a, b] · ws
  proof (rule notI)
    assume ws ≤p [a, b] · ws
    from pref-hd-eq[OF this ‹ws ≠ ε›]
    have hd ws = a by simp
    hence  d = b
      using ‹set (d # ws) ⊆ {a, b}› ‹hd ws ≠ d› by auto
    show False
      using ‹ws ≤p [a, b] · ws›  ‹¬ d # ws ≤p [b, a] · d # ws›[unfolded ‹d = b›]
by force
  qed
  show ¬ ws ≤p [b, a] · ws
  proof (rule notI)
    assume ws ≤p [b, a] · ws
    from pref-hd-eq[OF this ‹ws ≠ ε›]
    have hd ws = b by simp
    hence  d = a
      using ‹set (d # ws) ⊆ {a, b}› ‹hd ws ≠ d› by auto
    show False
      using ‹ws ≤p [b, a] · ws›  ‹¬ d # ws ≤p [a, b] · d # ws›[unfolded ‹d = a›]
by force
  qed
 qed
qed simp

lemma no-repetition-list-bin:
 fixes ws :: binA list
 assumes not-square:  ⋀ c. ¬ [c,c] ≤f ws
 shows ws ≤p [hd ws, 1−(hd ws)] · ws
proof (cases ws = ε)
 assume ws ≠ ε
 have set: set ws ⊆ {hd ws, 1−hd ws}
   using swap-UNIV by auto
 have ¬ ws ≤p [1 − hd ws, hd ws] · ws
   using pref-hd-eq[OF - ‹ws ≠ ε›, of [1 − hd ws, hd ws] · ws] bin-swap-neq′
   unfolding hd-Cons-append by blast
 from no-repetition-list[OF set - this not-square not-square]
 show ws ≤p [hd ws, 1−(hd ws)] · ws by blast
qed simp

lemma per-root-hd-last-root: assumes ws ≤p [a,b] · ws and hd ws ≠ last ws
```

shows $ws \in \langle\{[a,b]\}\rangle$
using *assms*
proof (*induction* $|ws|$ *arbitrary*: *ws rule*: *less-induct*)
case *less*
then show *?case*
proof (*cases* $ws = \varepsilon$)
assume $ws \neq \varepsilon$
with ‹$hd\ ws \neq last\ ws$›
have $*$: $[hd\ ws, hd\ (tl\ ws)] \cdot tl\ (tl\ ws) = ws$
using *append-Cons last-ConsL*[*of tl ws hd ws*] *list.exhaust-sel*[*of ws*] *hd-tl* **by** *metis*
have *ind*: $|tl\ (tl\ ws)| < |ws|$ **using** ‹$ws \neq \varepsilon$› **by** *auto*
have $[hd\ ws, hd\ (tl\ ws)] \cdot tl\ (tl\ ws) \leq p\ [a,b] \cdot ws$
unfolding $*$ **using** ‹$ws \leq p\ [a, b] \cdot ws$›**.**
hence $[a,b] = [hd\ ws, hd\ (tl\ ws)]$ **by** *simp*
hence $hd\ ws = a$ **by** *simp*
have $tl\ (tl\ ws) \leq p\ [a,b] \cdot tl\ (tl\ ws)$
unfolding *pref-cancel-conv*[*of* $[a,b]$ $tl\ (tl\ ws)$, *symmetric*] ‹$[a,b] = [hd\ ws, hd\ (tl\ ws)]$› $*$
using ‹$ws \leq p\ [a, b] \cdot ws$›[*unfolded* ‹$[a,b] = [hd\ ws, hd\ (tl\ ws)]$›]**.**
have $(tl\ (tl\ ws)) \in \langle\{[a, b]\}\rangle$
proof (*cases* $tl\ (tl\ ws) = \varepsilon$)
assume $tl\ (tl\ ws) \neq \varepsilon$
from *pref-hd-eq*[*OF* ‹$tl\ (tl\ ws) \leq p\ [a, b] \cdot tl\ (tl\ ws)$› *this*]
have $hd\ (tl\ (tl\ ws)) = a$ **by** *simp*
have $last\ (tl\ (tl\ ws)) = last\ ws$
using ‹$tl\ (tl\ ws) \neq \varepsilon$› *last-tl tl-Nil* **by** *metis*
from ‹$hd\ ws \neq last\ ws$›[*unfolded* ‹$hd\ ws = a$›, *folded this* ‹$hd\ (tl\ (tl\ ws)) = a$›]
have $hd\ (tl\ (tl\ ws)) \neq last\ (tl\ (tl\ ws))$**.**
from *less.hyps*[*OF ind* ‹$tl\ (tl\ ws) \leq p\ [a,b] \cdot tl\ (tl\ ws)$› *this*]
show $(tl\ (tl\ ws)) \in \langle\{[a, b]\}\rangle$**.**
qed *simp*
from *prod-cl*[*OF singletonI this*]
show $ws \in \langle\{[a,b]\}\rangle$
unfolding $*$[*folded* ‹$[a,b] = [hd\ ws, hd\ (tl\ ws)]$›]**.**
qed *simp*
qed

lemma *no-cyclic-repetition-list*:
assumes *set ws* $\subseteq \{a,b\}$ $ws \notin \langle\{[a,b]\}\rangle$ $ws \notin \langle\{[b,a]\}\rangle$ $hd\ ws \neq last\ ws$
$\neg\ [a,a] \leq f\ ws$ $\neg\ [b,b] \leq f\ ws$
shows *False*
using *per-root-hd-last-root*[*OF* - ‹$hd\ ws \neq last\ ws$›] ‹$ws \notin \langle\{[a,b]\}\rangle$› ‹$ws \notin \langle\{[b,a]\}\rangle$›
no-repetition-list[*OF assms*(*1*) - - *assms*(*5*−*6*)] **by** *blast*

8.1.8 Three covers

lemma *three-covers-example*:
assumes

v: $v = \mathfrak{a}$ **and**
t: $t = (\mathfrak{b} \cdot \mathfrak{a}^{@}(j{+}1))^{@}(m + l + 1) \cdot \mathfrak{b} \cdot \mathfrak{a}$ **and**
r: $r = \mathfrak{a} \cdot \mathfrak{b} \cdot (\mathfrak{a}^{@}(j{+}1) \cdot \mathfrak{b})^{@}(m + l + 1)$ **and**
t': $t' = (\mathfrak{b} \cdot \mathfrak{a}^{@}(j + 1))^{@}m \cdot \mathfrak{b} \cdot \mathfrak{a}$ **and**
r': $r' = \mathfrak{a} \cdot \mathfrak{b} \cdot (\mathfrak{a}^{@}(j + 1) \cdot \mathfrak{b})^{@}l$ **and**
w: $w = \mathfrak{a} \cdot (\mathfrak{b} \cdot \mathfrak{a}^{@}(j + 1))^{@}(m + l + 1) \cdot \mathfrak{b} \cdot \mathfrak{a}$
shows $w = v \cdot t$ **and** $w = r \cdot v$ **and** $w = r' \cdot v^{@}(j + 1) \cdot t'$ **and** $t' <_p t$ **and** $r' <_s r$
proof–
show $w = v \cdot t$
unfolding $w\ v\ t$**..**
show $w = r \cdot v$
unfolding $w\ r\ v$ **by** *comparison*
find-theorems $?u \cdot ?u^{@}?j = ?u^{@}?j \cdot ?u$
show $t' <_p t$
unfolding $t\ t'$ **unfolding** *add.assoc* **unfolding** *add.commute*[*of l*]
unfolding *pow-add rassoc spref-cancel-conv* **unfolding** *pow-list-1*
unfolding *rassoc spref-cancel-conv*
unfolding *lassoc shifts*(*20*)
unfolding *rassoc* **by** *blast*
have $r = \mathfrak{a} \cdot \mathfrak{b} \cdot (\mathfrak{a}^{@}Suc\ j \cdot \mathfrak{b})^{@}\ m \cdot \mathfrak{a}^{@}j \cdot\ r'$
unfolding $r'\ r$ **by** *comparison*
thus $r' <_s r$
by *force*
show $w = r' \cdot v^{@}(j + 1) \cdot t'$
unfolding $w\ r'\ v\ t'$
by *comparison*
qed

lemma *three-covers-pers*: — alias Old Good Lemma
assumes $w = v \cdot t$ **and** $w = r' \cdot v^{@}j \cdot t'$ **and** $w = r \cdot v$ **and** $0 < j$ **and**
$r' <_s r$ **and** $t' <_p t$
shows *period* w $(|t| - |t'|)$ **and** *period* w $(|r| - |r'|)$ **and**
$(|t| - |t'|) + (|r| - |r'|) = |w| + j{*}|v| - 2{*}|v|$
proof–
let *?per-r* $= |r| - |r'|$
let *?per-t* $= |t| - |t'|$
let *?gcd* $= gcd\ (|t| - |t'|)\ (|r| - |r'|)$
have $w \neq \varepsilon$
using $\langle w = v \cdot t \rangle\ \langle t' <_p t \rangle$ **by** *auto*
obtain r'' **where** $r'' \cdot r' = r$ **and** $r'' \neq \varepsilon$
using *ssufD*[*OF* $\langle r' <_s r \rangle$] *sufD* **by** *blast*
have $w <_p r'' \cdot w$
using *per-rootI*[*OF* - $\langle r'' \neq \varepsilon \rangle$, *of w*] $\langle w = r \cdot v \rangle\ \langle w = r' \cdot v\ ^{@}\ j \cdot t' \rangle\ \langle r'' \cdot r' = r \rangle$
unfolding *pow-pos*[*OF* $\langle 0 < j \rangle$] **using** *rassoc triv-pref* **by** *metis*
thus *period w ?per-r*
using *lenarg*[*OF* $\langle r'' \cdot r' = r \rangle$] *sprefD1*[*OF* $\langle w <_p r'' \cdot w \rangle$] *diff-add-inverse2 periodI-pref'*

```
    unfolding lenmorph by metis
  have |r'| < |r|
    using suffix-length-less[OF ‹r' <s r›].
  have |t'| < |t|
    using prefix-length-less[OF ‹t' <p t›].
  obtain t'' where t' · t'' = t and t'' ≠ ε
    using ‹t' <p t› by blast
  have w <s w · t''
    using per-rootI[reversed, OF - ‹t'' ≠ ε›, of w]
     ‹w = v · t› ‹w = r' · v @ j · t'› ‹t' · t'' = t›
      unfolding pow-pos2[OF ‹0 < j›] using rassoc triv-suf by metis
   show period w ?per-t
   using lenarg[OF ‹t' · t'' = t›] ssufD1[OF ‹w <s w · t''›] periodI-suf' diff-add-inverse
    unfolding lenmorph  by metis
  show eq: ?per-t + ?per-r = |w| + j*|v| − 2*|v|
    using lenarg[OF ‹w = r' · v@ j · t'›]
      lenarg[OF ‹w = v · t›] lenarg[OF ‹w = r · v›] ‹|t'| < |t|› ‹|r'| < |r|›
    unfolding pow-len lenmorph by force
qed

lemma three-covers-per0: assumes w = v · t and w = r' · v@ j · t' and w = r ·
v and 0 < j
  r' <s r and t' <p t and |t'| ≤ |r'|
  and primitive v
shows period w (gcd (|t| − |t'|) (|r| − |r'|))
  using assms
proof (induct |w| arbitrary: w t r t' r' v rule: less-induct)
  case less
  then show ?case
  proof–
    let ?per-r = |r| − |r'|
    let ?per-t = |t| − |t'|
    let ?gcd = gcd (|t| − |t'|) (|r| − |r'|)
    have v ≠ ε using prim-nemp[OF ‹primitive v›].
    have w ≠ ε
      using ‹w = v · t› ‹v ≠ ε› by blast
    note prefix-length-less[OF ‹t' <p t›] prefix-length-less[reversed, OF ‹r' <s r›]
    have  ?gcd ≠ 0
      using gcd-eq-0-iff zero-less-diff'[OF ‹|t'| < |t|›] by simp
    have period w ?per-t and period w ?per-r
         and eq: ?per-t + ?per-r = |w| + j*|v| − 2*|v|
      using three-covers-pers[OF ‹w = v · t› ‹w = r' · v @ j · t'› ‹w = r · v› ‹0 <
j› ‹r' <s r› ‹t' <p t›].

    obtain r'' where r'' · r' = r and r'' ≠ ε
      using  ssufD[OF ‹r' <s r›] sufD by blast
    hence w ≤p r'' · w
      using less.prems unfolding pow-pos[OF ‹0 < j›] using rassoc triv-pref by
metis
```

obtain t'' **where** $t' \cdot t'' = t$ **and** $t'' \neq \varepsilon$
using *sprefD*[*OF* ‹$t' <p\ t$›] *prefD* **by** *blast*

show *period w ?gcd*
proof (*cases*)
have *local-rule*: $a - c \leq b \Longrightarrow k + a - c - b \leq k$ **for** $a\ b\ c\ k$ :: *nat*
by *simp*
assume $j*|v| - 2*|v| \leq$ *?gcd* — Condition allowing to use the periodicity lemma
from *local-rule*[*OF this*]
have *len*: *?per-t* + *?per-r* − *?gcd* ≤ $|w|$
unfolding *eq*.
show *period w ?gcd*
using *per-lemma*[*OF* ‹*period w ?per-t*› ‹*period w ?per-r*› *len*].
next
assume ¬ $j*|v| - 2*|v| \leq$ *?gcd* — Periods are too long for the periodicity lemma
hence *?gcd* ≤ $|v^{@}j| - 2*|v|$ — But then we have a potential for using the periodicity lemma on the power of v's
unfolding *pow-len* **by** *linarith*
hence *?gcd* + $|v|$ ≤ $|v\ ^{@}\ \ j|$
using ‹*?gcd* ≠ *0*› **by** *linarith*

show *period w ?gcd*
proof (*cases*)
assume $|r'| = |t'|$ — The trivial case
hence $|t| - |t'| = |r| - |r'|$
using *eq-conjug-len*[*OF* ‹$w = v \cdot t$›[*unfolded* ‹$w = r \cdot v$›]] **by** *force*
show *period w* (*gcd* ($|t| - |t'|$) ($|r| - |r'|$))
unfolding ‹$|t| - |t'| = |r| - |r'|$› *gcd-idem-nat* **using** ‹*period w* ($|r| - |r'|$)›.
next
assume $|r'| \neq |t'|$ — The nontrivial case
hence $|t'| < |r'|$
using ‹$|t'| \leq |r'|$› **by** *force*
have $r' \cdot v <p\ w$
using ‹$|r'| < |r|$› ‹$r'' \cdot r' = r$› ‹$w \leq p\ r'' \cdot w$› ‹$w = r \cdot v$› **by** *force*
obtain p **where** $r' \cdot v = v \cdot p$
using *ruler-le*[*OF triv-pref*[*of v t* , *folded* ‹$w = v \cdot t$›], *of* $r' \cdot v$]
unfolding *lenmorph* ‹$w = r' \cdot v^{@}j \cdot t'$›[*unfolded pow-pos*[*OF* ‹*0* < *j*›]] *rassoc*
by (*force simp add*: *prefix-def*)
from ‹$w = r' \cdot v^{@}j \cdot t'$›[*unfolded pow-pos*[*OF* ‹*0* < *j*›] *lassoc this* ‹$w = v \cdot t$›, *unfolded rassoc cancel*]
have $p \leq p\ t$
by *blast*
have $|v \cdot p| < |w|$
using *prefix-length-less*[*OF* ‹$r' \cdot v <p\ w$›, *unfolded* ‹$r' \cdot v = v \cdot p$›].

have $v \cdot p \leq_s w$ — r'v is a long border of w
using ‹$r' \cdot v = v \cdot p$› ‹$w = r \cdot v$› ‹$r' <_s r$› *same-suffix-suffix ssufD* **by** *metis*
have $|r'| = |p|$
using *eq-conjug-len*[*OF* ‹$r' \cdot v = v \cdot p$›].
note ‹$|t'| \leq |r'|$›[*unfolded* ‹$|r'| = |p|$›]
hence $t' <_p p$
using ‹$t = p \cdot v\ ^{@}\ (j - 1) \cdot t'$› ‹$t' \cdot t'' = t$› ‹$|r'| = |p|$› ‹$|t'| < |r'|$› ‹$p \leq_p t$› *pref-prod-long-less* **by** *metis*
hence $p \neq \varepsilon$
by *auto*
show *?thesis*
proof (*cases*)
assume $|v \cdot p| \leq |v^{@}j \cdot t'|$ — The border does not cover the whole power of v's. In this case, everything commutes
have $\varrho\ (rev\ v) = rev\ (\varrho\ v)$
using ‹$v \neq \varepsilon$› *primroot-rev* **by** *auto*
from *pref-marker-ext*[*reversed*, *OF* ‹$|t'| \leq |p|$› ‹$v \neq \varepsilon$›]
suf-prod-le[*OF* ‹$v \cdot p \leq_s w$›[*unfolded* ‹$w = r' \cdot v^{@}j \cdot t'$›] ‹$|v \cdot p| \leq |v^{@}j \cdot t'|$›]
obtain k **where** $p = v^{@}k \cdot t'$
unfolding *prim-primroot*[*OF* ‹*primitive v*›].
hence $p \leq_p v^{@}k \cdot p$
using ‹$t' <_p p$› **by** *simp*
from *root-comm-root*[*OF this pow-comm*[*symmetric*]]
have $p \leq_p v \cdot p$
using ‹$|r'| = |p|$› ‹$|r'| \neq |t'|$› ‹$p = v\ ^{@}\ k \cdot t'$› **by** *force*
hence $p = r'$
using ‹$|r'| = |p|$› ‹$r' \cdot v = v \cdot p$› *pref-prod-eq* **by** *metis*
note ‹$r' \cdot v = v \cdot p$›[*folded this*] ‹$r' \cdot v = v \cdot p$›[*unfolded this*]
then obtain er' **where** $r' = v^{@}er'$
using ‹*primitive v*› **by** *auto*
from ‹$p \cdot v = v \cdot p$›[*unfolded* ‹$p = v^{@}k \cdot t'$› *lassoc pow-comm*[*symmetric*], *unfolded rassoc cancel*]
have $t' \cdot v = v \cdot t'$.
then obtain et' **where** $t' = v^{@}et'$
using ‹*primitive v*› **by** *auto*
have $t \cdot v = v \cdot t$
by (*simp add*: *pow-comm* ‹$p = r'$› ‹$r' \cdot v = v \cdot r'$› ‹$t = p \cdot v\ ^{@}\ (j - 1) \cdot t'$› ‹$t' \cdot v = v \cdot t'$›)
then obtain et **where** $t = v^{@}et$
using ‹*primitive v*› **by** *auto*
have $r \cdot v = v \cdot r$
using ‹$t \cdot v = v \cdot t$› *cancel-right* ‹$w = v \cdot t$› ‹$w = r \cdot v$› **by** *metis*
then obtain er **where** $r = v^{@}er$
using ‹*primitive v*› **by** *auto*
have $w \cdot v = v \cdot w$
by (*simp add*: ‹$r \cdot v = v \cdot r$› ‹$w = r \cdot v$›)
then obtain ew **where** $w = v^{@}ew$

using ‹*primitive v*› **by** *auto*
hence *period w* |*v*|
using ‹*v* ≠ *ε*› ‹*w* · *v* = *v* · *w*› ‹*w* ≠ *ε*› **by** *blast*
have *dift*: |*t*| − |*t′*| = (*et* − *et′*)∗|*v*|
using *lenarg*[*OF* ‹*t* = *v*[@]*et*›] *lenarg*[*OF* ‹*t′* = *v*[@]*et′*›] **unfolding** *lenmorph pow-len*
by (*simp add*: *diff-mult-distrib*)
have *difr*: (|*r*| − |*r′*|) = (*er* − *er′*)∗|*v*|
using *lenarg*[*OF* ‹*r* = *v*[@]*er*›] *lenarg*[*OF* ‹*r′* = *v*[@]*er′*›] **unfolding** *lenmorph pow-len*
by (*simp add*: *diff-mult-distrib*)
obtain *g* **where** *g*: *g*∗|*v*| = *?gcd*
unfolding *dift difr mult.commute*[*of* - |*v*|]
gcd-mult-distrib-nat[*symmetric*] **by** *blast*
from *per-mult*[*OF* ‹*period w* |*v*|›, *of g*]
show *?thesis*
unfolding *g*.
next
assume ¬ |*v* · *p*| ≤ |*v* [@] *j* · *t′*| — The border covers the whole power. An induction is available.
then obtain *ri′* **where** *v* · *p* = *ri′*· *v*[@]*j* · *t′* **and** *ri′* ≤*s r′*
using ‹*v* · *p* ≤*s w*› **unfolding** ‹*w* = *r′* · *v*[@]*j* · *t′*›
using *suffix-append suffix-length-le* **by** *blast*
from *len-less-neq*[*OF* ‹|*v* · *p*| < |*w*|›, *unfolded this*(*1*) ‹*w* = *r′* · *v*[@]*j* · *t′*›] *this*(*2*)
have *ri′* <*s r′*
by *blast*

have *gcd-eq*: *gcd* (|*p*| − |*t′*|) (|*r′*| − |*ri′*|) = *?gcd* — The two gcd's are the same
proof−
have |*r′*| ≤ |*r*|
by (*simp add*: ‹|*r′*| < |*r*|› *dual-order.strict-implies-order*)
have |*t*| = |*r*|
using *lenarg*[*OF* ‹*w* = *v* · *t*›] **unfolding** *lenarg*[*OF* ‹*w* = *r* · *v*›] *lenmorph* **by** *auto*
have *e1*: |*r′*| − |*ri′*| = |*r*| − |*r′*|
using *lenarg*[*OF* ‹*v* · *p* = *ri′*·*v*[@] *j* · *t′*›[*folded* ‹*r′* · *v* = *v* · *p*›]]
lenarg[*OF* ‹*w* = *r* · *v*›[*unfolded* ‹*w* = *r′* · *v*[@] *j* · *t′*›]]
unfolding *lenmorph pow-len* **by** (*simp add*: *add.commute diff-add-inverse diff-diff-add*)
have |*t*| = |*p*| + |*r′*| − |*ri′*|
unfolding *add-diff-assoc*[*OF suffix-length-le*[*OF* ‹*ri′* ≤*s r′*›], *unfolded e1*, *symmetric*]
‹|*t*| = |*r*|› **unfolding** ‹|*r′*| = |*p*|›
using ‹|*r′*| < |*r*|›[*unfolded* ‹|*r′*| = |*p*|›] **by** *linarith*
hence *e2*: |*t*| − |*t′*| = (|*p*| − |*t′*|) + (|*r′*| − |*ri′*|)
unfolding *add-diff-assoc2*[*OF* ‹|*t′*| ≤ |*p*|›] ‹|*t*| = |*p*| + |*r′*| − |*ri′*|›
using *suf-len*[*OF* ‹*ri′* ≤*s r′*›] **by** *force*

show *?thesis*
unfolding *e2 e1 gcd-add1*..
qed

have *per-vp*: *period* $(v \cdot p)$ *?gcd*
proof (*cases*)
assume $|t'| \leq |ri'|$
— By induction.
from *less.hyps*[*OF* ‹$|v \cdot p| < |w|$› *refl* ‹$v \cdot p = ri' \cdot v^{@} j \cdot t'$› ‹$r' \cdot v = v \cdot p$›[*symmetric*] ‹$0 < j$›
‹$ri' <_s r'$› ‹$t' <_p p$› *this* ‹*primitive v*›]
show *period* $(v \cdot p)$ *?gcd*
unfolding *gcd-eq* **by** *blast*
next — ...(using symmetry)
assume $\neg\ |t'| \leq |ri'|$ **hence** $|ri'| \leq |t'|$ **by** *simp*
have *period* $(rev\ p \cdot rev\ v)$ $(gcd\ (|rev\ r'| - |rev\ ri'|)\ (|rev\ p| - |rev\ t'|))$
proof (*rule less.hyps*[*OF - - - refl*])
show $|rev\ p \cdot rev\ v| < |w|$
using ‹$|v \cdot p| < |w|$› **by** *simp*
show $rev\ p \cdot rev\ v = rev\ v \cdot rev\ r'$
using ‹$r' \cdot v = v \cdot p$› **unfolding** *rev-append*[*symmetric*] **by** *simp*
show $rev\ p \cdot rev\ v = rev\ t' \cdot rev\ v\ ^{@}\ j \cdot rev\ ri'$
using ‹$v \cdot p = ri' \cdot v^{@}\ j \cdot t'$› **unfolding** *rev-append*[*symmetric*] *rev-pow*[*symmetric*] *rassoc* **by** *simp*
show $rev\ t' <_s rev\ p$
using ‹$t' <_p p$› **by** (*auto simp add*: *prefix-def*)
show $rev\ ri' <_p rev\ r'$
using ‹$ri' <_s r'$› *strict-suffix-to-prefix* **by** *blast*
show $|rev\ ri'| \leq |rev\ t'|$
by (*simp add*: ‹$|ri'| \leq |t'|$›)
show *primitive* $(rev\ v)$
by (*simp add*: ‹*primitive v*› *prim-rev-iff*)
qed *fact*
thus *?thesis*
unfolding *length-rev rev-append*[*symmetric*] *period-rev-conv gcd.commute*[*of* $|r'| - |ri'|$] *gcd-eq*.
qed

have *period* $(v^{@}\ j)$ $(gcd\ |v|\ ?gcd)$
proof (*rule per-lemma*)
show $|v| + ?gcd - gcd\ |v|\ (gcd\ (|t| - |t'|)\ (|r| - |r'|)) \leq |v\ ^{@}\ j|$
using ‹$?gcd + |v| \leq |v\ ^{@}\ j|$› **by** *linarith*
show *period* $(v\ ^{@}\ j)\ |v|$
using ‹$v \neq \varepsilon$› ‹$0 < j$›
by *blast*
find-theorems $?v^{@}\ ?n\ ?w = \varepsilon$
have $v\ ^{@}\ j \neq \varepsilon$
using ‹$0 < j$› ‹$v \neq \varepsilon$› **by** *blast*
from *period-fac-period*[*OF per-vp*[*unfolded* ‹$v \cdot p = ri' \cdot v\ ^{@}\ j \cdot t'$›]]

```
      show period (v @ j) ?gcd.
    qed

    have per-vp': period (v · p) (gcd |v| ?gcd)
    proof (rule refine-per)
      show gcd |v| ?gcd dvd ?gcd by blast
      show ?gcd ≤ |v@j|
        using ‹?gcd + |v| ≤ |v @ j|› add-leE by blast
      show v @ j ≤f v · p
        using facI'[OF ‹v · p = ri' · v @ j · t'›[symmetric]].
    qed fact+

    have period w (gcd |v| ?gcd)
    proof (rule per-glue)
      show v · p ≤p w
        using ‹p ≤p t› ‹w = v · t› by auto
      have |v @ j| + |t'| ≤ |v| + |p|
        using ‹¬ |v · p| ≤ |v @ j · t'|› by auto
      moreover have |r'| + gcd |v| ?gcd ≤ |v| + |p|
        using lenarg[OF ‹r' · v = v · p›, unfolded lenmorph]
          ‹v ≠ ε› gcd-le1-nat length-0-conv nat-add-left-cancel-le by metis
      ultimately show |w| + gcd |v| ?gcd ≤ |v · p| + |v · p|
        unfolding lenarg[OF ‹w = r' · v @ j · t'›] lenmorph add.commute[of
|r'|] by linarith
    qed fact+

    obtain k where k: ?gcd = k*(gcd |v| ?gcd)
      using gcd-dvd2 unfolding dvd-def mult.commute[of - gcd |v| ?gcd] by
blast
    hence k ≠ 0
      using ‹?gcd ≠ 0› by algebra
    from per-mult[OF ‹period w (gcd |v| ?gcd)›, of k, folded k]
    show ?thesis.
  qed
 qed
qed
qed

lemma three-covers-per: assumes w = v · t and w = r' · v@j · t' and w = r · v
  r' <s r and t' <p t and 0 < j
shows period w (gcd (|t| − |t'|) (|r| − |r'|))
proof−
  let ?per-r = |r| − |r'|
  let ?per-t = |t| − |t'|
  let ?gcd = gcd (|t| − |t'|) (|r| − |r'|)
  have period w ?per-t and period w ?per-r and len: (|t| − |t'|) + (|r| − |r'|) =
|w| + j*|v| − 2*|v|
    using three-covers-pers[OF ‹w = v · t› ‹w = r' · v @ j · t'› ‹w = r · v› ‹0 <
```

$j\rangle$ $\langle r' <_s r\rangle$ $\langle t' <_p t\rangle$] **by** *blast+*

show *?thesis*
proof(*cases*)
assume $v = \varepsilon$
have $|t| - |t'| + (|r| - |r'|) = |w|$
using $\langle w = v \cdot t\rangle$ $\langle w = r' \cdot v^{@} j \cdot t'\rangle$ $\langle w = r \cdot v\rangle$ **unfolding** $\langle v = \varepsilon\rangle$ *emp-simps*
by *force*
from *per-lemma*[*OF* $\langle$*period w ?per-t*$\rangle$ $\langle$*period w ?per-r*$\rangle$, *unfolded this*]
show *period w ?gcd*
by *fastforce*
next
assume $v \neq \varepsilon$
show *?thesis*
proof (*cases*)
assume $j \leq 1$
hence $(j = 0 \Longrightarrow P) \Longrightarrow (j = 1 \Longrightarrow P) \Longrightarrow P$ **for** P **by** *force*
hence $|w| + j{*}|v| - 2{*}|v| - ?gcd \leq |w|$ — Condition allowing to use the periodicity lemma
by (*cases*, *simp-all*)
thus *period w ?gcd*
using *per-lemma*[*OF* $\langle$*period w ?per-t*$\rangle$ $\langle$*period w ?per-r*$\rangle$] **unfolding** *len* **by** *blast*
next
assume $\neg\, j \leq 1$ **hence** $2 \leq j$ **by** *simp*
obtain e **where** $v = \varrho\ v^{@} e\ 0 < e$
using *primroot-expE* **by** *metis*
have $w = \varrho\ v \cdot \varrho\ v^{@}(e - 1) \cdot t$
unfolding *lassoc pow-pos*[*OF* $\langle 0 < e\rangle$, *symmetric*] $\langle v = \varrho\ v^{@} e\rangle$[*symmetric*]
by *fact*
have $w = (r \cdot \varrho\ v^{@}(e - 1)) \cdot \varrho\ v$
unfolding *rassoc pow-pos2*[*OF* $\langle 0 < e\rangle$, *symmetric*] $\langle v = \varrho\ v^{@} e\rangle$[*symmetric*]
by *fact*
note *aux* = *add-less-mono*[*OF diff-less*[*OF zero-less-one* $\langle 0 < e\rangle$] *diff-less*[*OF zero-less-one* $\langle 0 < e\rangle$]]
have $(e-1) + (e-1) < j{*}e$
using *less-le-trans*[*OF aux mult-le-mono1*[*OF* $\langle 2 \leq j\rangle$, *unfolded mult-2*]].
then obtain e' **where** $(e-1) + (e-1) + e' = j{*}e\ 0 < e'$
using *less-imp-add-positive* **by** *blast*
hence *aux-sum*: $(e - 1) + e' + (e - 1) = j{*}e$
by *presburger*
have *cover3*: $w = (r' \cdot (\varrho\ v)^{@}(e-1)) \cdot (\varrho\ v)\ ^{@} e' \cdot ((\varrho\ v)^{@}(e-1) \cdot t')$
unfolding $\langle w = r' \cdot v^{@}\ j \cdot t'\rangle$ *rassoc cancel* **unfolding** *lassoc cancel-right*
unfolding *pow-add*[*symmetric*]
pow-mult **unfolding** *aux-sum* **unfolding** *mult.commute*[*of j*]
pow-mult $\langle v = \varrho\ v^{@} e\rangle$[*symmetric*]..
show *?thesis*
proof(*cases*)
assume $|t'| \leq |r'|$

have *dif1*: $|\varrho\ v\ ^{@}\ (e-1)\cdot t| - |\varrho\ v\ ^{@}\ (e-1)\cdot t'| = |t| - |t'|$
unfolding *lenmorph* **by** *simp*
have *dif2*: $|r\cdot\varrho\ v\ ^{@}\ (e-1)| - |r'\cdot\varrho\ v\ ^{@}\ (e-1)| = |r| - |r'|$
unfolding *lenmorph* **by** *simp*

show *?thesis*
proof (*rule three-covers-per0*[*OF* ‹$w = \varrho\ v\cdot(\varrho\ v^{@}(e-1)\cdot t)$›
cover3 ‹$w = (r\cdot\varrho\ v^{@}(e-1))\cdot\varrho\ v$› ‹$0 < e'$› - - - *primroot-prim*[*OF* ‹$v \neq \varepsilon$›],
unfolded dif1 dif2])
show $r'\cdot\varrho\ v\ ^{@}\ (e-1) <_s r\cdot\varrho\ v\ ^{@}\ (e-1)$
using ‹$r' <_s r$› **by** *auto*
show $\varrho\ v\ ^{@}\ (e-1)\cdot t' <_p \varrho\ v\ ^{@}\ (e-1)\cdot t$
using ‹$t' <_p t$› **by** *auto*
show $|\varrho\ v\ ^{@}\ (e-1)\cdot t'| \leq |r'\cdot\varrho\ v\ ^{@}\ (e-1)|$
unfolding *lenmorph* **using** ‹$|t'| \leq |r'|$› **by** *auto*
qed
next
let *?w* = *rev w* **and** *?r* = *rev t* **and** *?t* = *rev r* **and** *?ϱ* = *rev* (ϱ *v*) **and** *?r'* = *rev t'* **and** *?t'* = *rev r'*
assume ¬ $|t'| \leq |r'|$
hence $|?t'| \leq |?r'|$ **by** *auto*
have *?w* = (*?r* · *?ϱ*$^{@}$(*e*−1)) · *?ϱ*
unfolding *rev-pow*[*symmetric*] *rev-append*[*symmetric*] ‹$w = \varrho\ v\cdot(\varrho\ v^{@}(e-1)\cdot t)$› *rassoc*..
have *?w* = *?ϱ* · (*?ϱ*$^{@}$(*e*−1) · *?t*)
unfolding *rev-pow*[*symmetric*] *rev-append*[*symmetric*] ‹$w = (r\cdot\varrho\ v^{@}(e-1))\cdot\varrho\ v$›..
have *?w* = (*?r'* · *?ϱ*$^{@}$(*e*−1)) · *?ϱ*$^{@}$*e'* · (*?ϱ*$^{@}$(*e*−1) · *?t'*)
unfolding *rev-pow*[*symmetric*] *rev-append*[*symmetric*] ‹$w = (r'\cdot\varrho\ v^{@}(e-1))\cdot\varrho\ v\ ^{@}e'\cdot(\varrho\ v^{@}(e-1)\cdot t')$› *rassoc*..
have *dif1*: $|?\varrho\ ^{@}\ (e-1)\cdot ?t| - |?\varrho\ ^{@}\ (e-1)\cdot ?t'| = |r| - |r'|$
unfolding *lenmorph* **by** *simp*
have *dif2*: $|?r\cdot ?\varrho\ ^{@}\ (e-1)| - |?r'\cdot ?\varrho\ ^{@}\ (e-1)| = |t| - |t'|$
unfolding *lenmorph* **by** *simp*
show *?thesis*
proof(*rule three-covers-per0*[*OF* ‹*?w* = *?ϱ* · (*?ϱ*$^{@}$(*e*−1) · *?t*)›
‹*?w* = (*?r'* · *?ϱ*$^{@}$(*e*−1)) · *?ϱ*$^{@}$*e'* · (*?ϱ*$^{@}$(*e*−1) · *?t'*)› ‹*?w* = (*?r* · *?ϱ*$^{@}$(*e*−1)) · *?ϱ*› ‹0 < *e'*›,
unfolded dif1 dif2 period-rev-conv gcd.commute[*of* |*r*| − |*r'*|]])
show $?r'\cdot ?\varrho\ ^{@}\ (e-1) <_s ?r\cdot ?\varrho\ ^{@}\ (e-1)$
using ‹$t' <_p t$› **by** (*auto simp add*: *prefix-def*)
show $?\varrho\ ^{@}\ (e-1)\cdot ?t' <_p ?\varrho\ ^{@}\ (e-1)\cdot ?t$
using ‹$r' <_s r$› **by** (*auto simp add*: *suffix-def*)
show $|?\varrho\ ^{@}\ (e-1)\cdot ?t'| \leq |?r'\cdot ?\varrho\ ^{@}\ (e-1)|$
unfolding *lenmorph* **using** ‹$|?t'| \leq |?r'|$› **by** *auto*
show *primitive ?ϱ*
using *primroot-prim*[*OF* ‹$v \neq \varepsilon$›] **by** (*simp add*: *prim-rev-iff*)
qed

qed
qed
qed
qed

thm *per-root-modE'*

lemma assumes $w \leq p\ r \cdot w\ r \neq \varepsilon$
obtains $p\ q\ i$ **where** $w = (p \cdot q)^{@} i \cdot p\ p \cdot q = r$
using *assms* **by** *blast*

lemma *three-coversE*: **assumes** $w = v \cdot t$ **and** $w = r' \cdot v \cdot t'$ **and** $w = r \cdot v$ **and** $r' <s\ r$ **and** $t' <p\ t$
obtains $p\ q\ i\ k\ m$ **where** $t = (q \cdot p)^{@}(m+k)$ **and** $r = (p \cdot q)^{@}(m+k)$ **and** $t' = (q \cdot p)^{@} k$ **and** $r' = (p \cdot q)^{@} m$ **and** $v = (p \cdot q)^{@} i \cdot p$ **and** $w = (p \cdot q)^{@}(m + i + k) \cdot p$ **and** *primitive* $(p \cdot q)$ **and** $q \neq \varepsilon$ **and** $0 < m$ **and** $0 < k$
proof−
let *?d* = *gcd* $|r'|$ $|t'|$
have $r' \neq \varepsilon\ t' \neq \varepsilon$
using *assms* **by** *force*+
have $0 <$ *?d*
using *nemp-len*[*OF* ‹$r' \neq \varepsilon$›] **by** *simp*
have *diffs-eq*: $|t| - |t'| = |r'|\ |r| - |r'| = |t'|$
using *lenarg*[*OF* ‹$w = v \cdot t$›] *lenarg*[*OF* ‹$w = r \cdot v$›]
unfolding *lenarg*[*OF* ‹$w = r' \cdot v \cdot t'$›] *lenmorph* **by** *simp-all*
have *take* (*gcd* $|r'|$ $|t'|$) $w \neq \varepsilon$
by (*simp add*: ‹$r' \neq \varepsilon$› *assms*(*2*))
hence $w <p$ *take* (*gcd* $|r'|$ $|t'|$) $w \cdot w$
using *three-covers-per*[*of - - - -1*, *unfolded cow-simps*, *OF assms order.refl*, *unfolded diffs-eq period-def*] **by** *blast*
from *per-root-mod-primE*[*OF* ‹$w <p$ *take* (*gcd* $|r'|$ $|t'|$) $w \cdot w$›]
obtain $l\ p\ q$ **where** $p \cdot q = \varrho$ (*take ?d w*) $(p \cdot q)^{@} l \cdot p = w\ q \neq \varepsilon$.
hence *primitive* $(p \cdot q)$ **by** *auto*
define e **where** $e = e_{\varrho}$ (*take ?d w*)
have $e \neq 0$
unfolding *e-def*
using ‹$w <p$ *take* (*gcd* $|r'|$ $|t'|$) $w \cdot w$› *primroot-exp-nemp* **by** *blast*
have $(p \cdot q)^{@} e =$ *take ?d w*
unfolding *e-def* ‹$p \cdot q = \varrho$ (*take ?d w*)› **by** *force*
have $|(p \cdot q)^{@} e| \leq |w|$

unfolding ‹$(p \cdot q)^{@} e = take\ ?d\ w$›
using *len-take2* **by** *blast*
have *swap-e*: $|(p \cdot q)^{@} e| = |(q \cdot p)^{@} e|$
unfolding *pow-len swap-len*..
have $|(p \cdot q)^{@} e| = ?d$
unfolding ‹$(p \cdot q)^{@} e = take\ ?d\ w$›
by (*rule take-len*, *unfold lenarg*[*OF* ‹$w = r' \cdot v \cdot t'$›, *unfolded lenmorph*],
use gcd-le1-nat[*OF nemp-len-not0*[*OF* ‹$r' \neq \varepsilon$›]] *trans-le-add1* **in** *blast*)

hence $(p \cdot q)^{@} e \leq p\ r'$
unfolding *pref-take-conv*[*of* $(p \cdot q)^{@} e\ r'$, *symmetric*] **using** ‹$w = r' \cdot v \cdot t'$›
‹$(p \cdot q)^{@} e = take\ ?d\ w$›[*symmetric*] *gcd-le1-nat*[*OF nemp-len-not0*[*OF* ‹$r' \neq \varepsilon$›]] *short-take-append* **by** *metis*
hence $(p \cdot q)^{@} e = take\ ?d\ \ r'$
using *pref-take-conv* ‹$|(p \cdot q)^{@} e| = ?d$› **by** *metis*
have $r' \leq p\ (p \cdot q)^{@} e \cdot r'$
using *pref-keeps-per-root*[*OF sprefD1*[*OF* ‹$w <p\ take\ ?d\ w \cdot w$›]]
unfolding ‹$(p \cdot q)^{@} e = take\ (gcd\ |r'|\ |t'|)\ w$›
using ‹$w = r' \cdot v \cdot t'$› **by** *blast*
define m **where** $m = e*(|r'|\ div\ gcd\ |r'|\ |t'|)$
have $(p \cdot q)^{@} m = r'$
using *per-div*[*OF gcd-dvd1 periodI*[*OF* ‹$r' \leq p\ (p \cdot q)^{@} e \cdot r'$›[*unfolded* ‹$(p \cdot q)^{@} e = take\ ?d\ \ r'$›]]]
unfolding ‹$(p \cdot q)^{@} e = take\ ?d\ \ r'$›[*symmetric*] *pow-list-mult m-def*.
have $p \leq s\ (q \cdot p)\ ^{@}\ e$
unfolding *pow-Suc2*[*of e−1 q · p*, *unfolded Suc-minus*[*OF* ‹$e \neq 0$›] *lassoc*] **by** *blast*
note ‹$|(p \cdot q)^{@} e| \leq |w|$›[*unfolded swap-e*, *folded* ‹$(p \cdot q)^{@} l \cdot p = w$›, *unfolded shift-pow*]
have $(q \cdot p)^{@} e \leq s\ (r' \cdot v) \cdot t'$
unfolding *rassoc* ‹$w = r' \cdot v \cdot t'$›[*symmetric*, *folded* ‹$(p \cdot q)^{@} l \cdot p = w$›, *unfolded shift-pow*]
using *suf-prod-suf-short*[*OF* - ‹$p \leq s\ (q \cdot p)\ ^{@}\ e$› ‹ $|(q \cdot p)\ ^{@}\ e| \leq |p \cdot (q \cdot p)\ ^{@}\ l|$›]
unfolding *pows-comm*[*of e* $(q \cdot p)$ *l*] **by** *blast*
have $|(q \cdot p)\ ^{@}\ e| \leq |t'|$
using *gcd-le2-nat*[*OF nemp-len-not0*[*OF* ‹$t' \neq \varepsilon$›], *of* $|r'|$, *folded* ‹$|(p \cdot q)^{@} e| = ?d$›]
unfolding *swap-len*[*of p q*] *pow-len*.
have $(q \cdot p)^{@} e \leq s\ t'$
unfolding ‹$w = r' \cdot v \cdot t'$›[*unfolded lassoc*]
using *suf-prod-le*[*OF* ‹$(q \cdot p)^{@} e \leq s\ (r' \cdot v) \cdot t'$› ‹$|(q \cdot p)\ ^{@}\ e| \leq |t'|$›].
have $t' \leq s\ \ t' \cdot (q \cdot p)^{@} e$
proof (*rule pref-keeps-per-root*[*reversed*, *of w*])
show $w \leq s\ w \cdot (q \cdot p)^{@} e$
unfolding ‹$(p \cdot q)^{@} l \cdot p = w$›[*symmetric*, *unfolded shift-pow*] *rassoc pows-comm*
unfolding *lassoc shift-pow*[*symmetric*]
unfolding *rassoc* **unfolding** *shift-pow* **by** *blast*
show $t' \leq s\ w$

unfolding ‹$w = r' \cdot v \cdot t'$› *lassoc* **by** *blast*
qed
have *t-drop*: $(q \cdot p)^{@} e = drop\ (|t'| - ?d)\ t'$
using ‹$|(p \cdot q)^{@} e| = ?d$›[*unfolded swap-e*, *symmetric*] ‹$(q \cdot p)^{@} e \leq s\ t'$›[*unfolded suf-drop-conv*, *symmetric*]
by *argo*
define k **where** $k = (e * (|t'|\ div\ gcd\ |r'|\ |t'|))$
have $(q \cdot p)^{@} k = t'$
using *per-div*[*reversed*, *OF gcd-dvd2*, *OF periodI*[*reversed*], *OF* ‹$t' \leq s\ \ t' \cdot (q \cdot p)^{@} e$›[*unfolded t-drop*]]
unfolding *t-drop*[*symmetric*] *pow-mult*[*symmetric*] *k-def*.
have $m + k \leq l$
unfolding *linorder-not-less*[*symmetric*]
proof (*rule notI*)
assume $l < m + k$
hence $l + 1 \leq m + k$
by *force*
from *trans-le-add1*[*OF mult-le-mono1*[*OF this*]]
have $(l + 1) * |p \cdot q| \leq (m + k) * |p \cdot q| + |v|$.
with *lenarg*[*OF* ‹$w = r' \cdot v \cdot t'$›[*folded* ‹$(p \cdot q)^{@} l \cdot p = w$›, *folded* ‹$(q \cdot p)^{@} k = t'$› ‹$(p \cdot q)^{@} m = r'$›],
unfolded lenmorph, *unfolded pow-len add.assoc*[*symmetric*], *symmetric*]
show *False*
unfolding *distrib-right add.commute*[*of* - $|v|$] *lenmorph*
unfolding *distrib-left* **using** *nemp-len*[*OF* ‹$q \neq \varepsilon$›] **by** *linarith*
qed
then obtain i **where** $l = m + i + k$
by (*metis add.assoc add.commute le-Suc-ex*)

have $v = (p \cdot q)^{@} i \cdot p$
using ‹$w = r' \cdot v \cdot t'$›
unfolding ‹$(p \cdot q)^{@} l \cdot p = w$›[*symmetric*] ‹$(q \cdot p)^{@} k = t'$›[*symmetric*] ‹$(p \cdot q)^{@} m = r'$›[*symmetric*] ‹$l = m + i + k$› *pow-add*
rassoc cancel cancel-right
unfolding *lassoc shift-pow cancel-right* **by** *simp*

have $r = (p \cdot q)^{@} (m + k)$
using ‹$w = r \cdot v$› **unfolding** ‹$(p \cdot q)^{@} l \cdot p = w$›[*symmetric*] ‹$v = (p \cdot q)^{@} i \cdot p$› ‹$l = m + i + k$›
unfolding *lassoc cancel-right add.commute*[*of* - *k*] *add.assoc*[*symmetric*] *pow-add* **by** *simp*

have $t = (q \cdot p)^{@} (m + k)$
using ‹$w = v \cdot t$› **unfolding** ‹$(p \cdot q)^{@} l \cdot p = w$›[*symmetric*] ‹$v = (p \cdot q)^{@} i \cdot p$› ‹$l = m + i + k$›
unfolding *rassoc cancel add.commute*[*of m*] *add.assoc*[*symmetric*] *pow-add*
unfolding *shift-pow* **unfolding** *lassoc shift-pow* **unfolding** *rassoc cancel*
unfolding *pows-comm* **by** *simp*

have $0 < m$
using $\langle (p \cdot q)^{@} m = r' \rangle$ $\langle r' \neq \varepsilon \rangle$ **by** *blast*

have $0 < k$
using $\langle (q \cdot p)^{@} k = t' \rangle$ $\langle t' \neq \varepsilon \rangle$ **by** *blast*
thm *that*
from *that*[*OF* $\langle t = (q \cdot p)^{@}(m + k) \rangle$ $\langle r = (p \cdot q)^{@}(m + k) \rangle$ $\langle (q \cdot p)^{@} k = t' \rangle$[*symmetric*] $\langle (p \cdot q)^{@} m = r' \rangle$[*symmetric*] $\langle v = (p \cdot q)^{@} i \cdot p \rangle$
$\langle (p \cdot q)^{@} l \cdot p = w \rangle$[*symmetric*, *unfolded* $\langle l = m + i + k \rangle$] $\langle$*primitive* $(p \cdot q) \rangle$ $\langle q \neq \varepsilon \rangle$ $\langle 0 < m \rangle$ $\langle 0 < k \rangle$]
show *thesis*.
qed

lemma *three-covers-pref-suf-pow*: **assumes** $x \cdot y \leq p\ w$ **and** $y \cdot x \leq s\ w$ **and** $w \leq f\ y^{@} k$ **and** $|y| \leq |x|$
shows $x \cdot y = y \cdot x$
using *fac-marker-suf*[*OF fac-trans*[*OF pref-fac*[*OF* $\langle x \cdot y \leq p\ w \rangle$] $\langle w \leq f\ y^{@} k \rangle$]]
fac-marker-pref[*OF fac-trans*[*OF suf-fac*[*OF* $\langle y \cdot x \leq s\ w \rangle$] $\langle w \leq f\ y^{@} k \rangle$]]
root-suf-comm'[*OF - suf-prod-long*, *OF - -* $\langle |y| \leq |x| \rangle$, *of x*] **by** *presburger*

8.1.9 Binary Equality Words

definition *binary-equality-generator* :: *binA list* ⇒ *bool* **where**
binary-equality-generator w ≡ (*set w* = *UNIV*) ∧ (∃ (*g* :: *binA list* ⇒ *nat list*) *h*. *binary-code-morphism g* ∧ *binary-code-morphism h* ∧ *g* ≠ *h* ∧ *w* ∈ *g* =$_M$ *h*)

definition *canonical-binary-equality-generator* :: *binA list* ⇒ *bool* **where**
canonical-binary-equality-generator w = (∃ (*g* :: *binA list* ⇒ *nat list*) *h*. *binary-code-morphism g* ∧ *binary-code-morphism h* ∧ *w* ∈ *g* =$_M$ *h* ∧ |*h* 𝔞| < |*g* 𝔞| ∧ |*g* 𝔟| < |*h* 𝔟| ∧ |*g* 𝔞| ≤ |*h* 𝔟|)

lemma *begE*: **assumes** *binary-equality-generator w*
obtains *g h* **where** *binary-code-morphism* (*g* :: *binA list* ⇒ *nat list*) **and** *binary-code-morphism h* **and** *g* ≠ *h* **and** *w* ∈ *g* =$_M$ *h* **and** *set w* = *UNIV*
using *assms binary-equality-generator-def* **by** *auto*

lemma *cbegE*: **assumes** *canonical-binary-equality-generator w*
obtains *g h* **where** *binary-code-morphism* (*g* :: *binA list* ⇒ *nat list*) **and** *binary-code-morphism h* **and** *w* ∈ *g* =$_M$ *h* **and** |*h* 𝔞| < |*g* 𝔞| **and** |*g* 𝔟| < |*h* 𝔟| **and** |*g* 𝔞| ≤ |*h* 𝔟|
using *assms canonical-binary-equality-generator-def* **by** *auto*

lemma *cbeg-is-beg*: **assumes** *canonical-binary-equality-generator w* **shows** *binary-equality-generator w*
proof−
from *cbegE*[*OF assms*]
obtain *g h* **where** *gh*: *binary-code-morphism* (*g* :: *binA list* ⇒ *nat list*) *bi-*

nary-code-morphism h w ∈ *g* =$_M$ *h* **and** |*h* 𝔞| < |*g* 𝔞| |*g* 𝔟| < |*h* 𝔟| |*g* 𝔞| ≤ |*h* 𝔟|.
interpret *g*: *binary-code-morphism g*
by *fact*
interpret *h*: *binary-code-morphism h*
by *fact*
interpret *two-binary-morphisms g h*
using *two-binary-morphisms-def two-morphisms-def g.morphism-axioms h.morphism-axioms*
by *blast*
have *g* ≠ *h*
using ‹|*h* 𝔞| < |*g* 𝔞|› **by** *blast*
have *set w* = *UNIV*
using ‹|*h* 𝔞| < |*g* 𝔞|› ‹|*g* 𝔟| < |*h* 𝔟|› *solution-UNIV*[*OF min-solD'*[*OF* ‹*w* ∈ *g* =$_M$ *h*›] *min-solD*[*OF* ‹*w* ∈ *g* =$_M$ *h*›] *bin-induct*[*of* λ *c*. *g*[*c*] ≠ *h*[*c*]]] **by** *force*
then show *binary-equality-generator w*
unfolding *binary-equality-generator-def* **using** *gh* ‹*g* ≠ *h*› **by** *blast*
qed

lemma *beg-productE*: **assumes** *binary-equality-generator* (*u*·*v*) **and** *u* ≠ *ε* **and** *v* ≠ *ε*
obtains *g h* **where** *binary-code-morphism* (*g* :: *binA list* ⇒ *nat list*) **and** *binary-code-morphism h* **and** *g* ≠ *h* **and** (*u*·*v*) ∈ *g* =$_M$ *h* **and** |*g u*| < |*h u*| **and** |*h v*| < |*g v*|
proof−
from *begE*[*OF assms*(*1*)]
obtain *g h* **where** *gh*: *binary-code-morphism* (*g* :: *binA list* ⇒ *nat list*) *binary-code-morphism h g* ≠ *h* (*u* · *v*) ∈ *g* =$_M$ *h*.
interpret *g*: *binary-code-morphism g*
by *fact*
interpret *h*: *binary-code-morphism h*
by *fact*
have |*g u*| ≠ |*h u*|
using *min-solD-min*[*OF* ‹(*u* · *v*) ∈ *g* =$_M$ *h*› ‹*u* ≠ *ε*› *triv-pref*] ‹*v* ≠ *ε*› *eqd-eq*(*1*)[*OF min-solD*[*OF* ‹(*u* · *v*) ∈ *g* =$_M$ *h*›, *unfolded g.morph h.morph*]] **by** *blast*
let *?g* = *if* |*g u*| ≤ |*h u*| *then g else h*
let *?h* = *if* |*g u*| ≤ |*h u*| *then h else g*
have *?g* ≠ *?h binary-code-morphism ?g binary-code-morphism ?h* (*u*·*v*) ∈ *?g* =$_M$ *?h*
using *gh* **by** (*simp-all add*: *min-sol-sym*)
interpret *g'*: *binary-code-morphism ?g*
by *fact*
interpret *h'*: *binary-code-morphism ?h*
by *fact*
have |*?g u*| < |*?h u*|
using ‹|*g u*| ≠ |*h u*|› **by** *simp*
with *lenarg*[*OF min-solD*[*OF* ‹(*u* · *v*) ∈ *?g* =$_M$ *?h*›, *unfolded g'.morph h'.morph*]]
have |*?h v*| < |*?g v*|
unfolding *lenmorph* **by** *linarith*

show *thesis*
by (*rule that*[*of ?g ?h*]) *fact+*
qed

lemma *bew-baiba-eq′*: **assumes** $|y| < |v|$ **and** $x \leq s\ y$ **and** $u \leq s\ v$ **and**
$y \cdot x\ ^{@}\ k \cdot y = v \cdot u\ ^{@}\ k \cdot v$
shows *commutes* $\{x,y,u,v\}$
proof−
obtain p **where** $y{\cdot}p = v$
using *eqdE*[*OF* ‹$y \cdot x\ ^{@}\ k \cdot y = v \cdot u\ ^{@}\ k \cdot v$› *less-imp-le*[*OF* ‹$|y| < |v|$›]] **by** *blast*
have $|u\ ^{@}\ k \cdot v| \leq |x\ ^{@}\ k \cdot y|$
using *lenarg*[*OF* ‹$y \cdot x\ ^{@}\ k \cdot y = v \cdot u\ ^{@}\ k \cdot v$›] ‹$|y| < |v|$› **unfolding** *lenmorph*
by *linarith*
obtain s **where** $s{\cdot}y = v$
using *eqdE*[*reversed*, *OF* ‹$y \cdot x\ ^{@}\ k \cdot y = v \cdot u\ ^{@}\ k \cdot v$›[*unfolded lassoc*] *less-imp-le*[*OF* ‹$|y| < |v|$›]].

have $s \neq \varepsilon$
using ‹$|y| < |v|$› ‹$s \cdot y = v$› **by** *force*
have $p \neq \varepsilon$
using ‹$|y| < |v|$› ‹$y \cdot p = v$› **by** *force*

have $s \cdot y = y \cdot p$
by (*simp add*: ‹$s \cdot y = v$› ‹$y \cdot p = v$›)
obtain $w\ w'\ q\ t$ **where** *p-def*: $p = (w'{\cdot}w)^{@}q$ **and** *s-def*: $s = (w{\cdot}w')^{@}q$
and *y-def*: $y = (w{\cdot}w')^{@}t{\cdot}w$ **and** $w' \neq \varepsilon$ **and** *primitive* $(w{\cdot}w')$ **and** ‹$0 < q$›
using *conjug-eq-primrootE*[*OF* ‹$s \cdot y = y \cdot p$› ‹$s \neq \varepsilon$›, *of thesis*]
by *blast*
have *primitive* $(w'{\cdot}w)$
using ‹*primitive* $(w \cdot w')$› *prim-conjug* **by** *auto*

have $y \cdot x\ ^{@}\ k \cdot y = y{\cdot}\ p \cdot u\ ^{@}\ k \cdot s \cdot y$
using ‹$s \cdot y = v$› ‹$y \cdot p = v$› ‹$y \cdot x\ ^{@}\ k \cdot y = v \cdot u\ ^{@}\ k \cdot v$› **by** *auto*
hence $x^{@}k = p{\cdot}u^{@}k{\cdot}s$
by *auto*
hence $x \neq \varepsilon$
using ‹$p \neq \varepsilon$› **by** *force*

have $w{\cdot}w' \leq s\ x^{@}k$
using ‹$x\ ^{@}\ k = p \cdot u\ ^{@}\ k \cdot s$›[*unfolded s-def*]
unfolding *pow-pos2*[*OF* ‹$0 < q$›]
using *sufI*[*of* $p \cdot u\ ^{@}\ k \cdot (w \cdot w')\ ^{@}\ (q - 1)\ w \cdot w'\ x^{@}k$, *unfolded rassoc*]
by *argo*

have $|w{\cdot}w'| \leq |x|$
proof(*intro leI notI*)
assume $|x| < |w \cdot w'|$
have $x \leq s\ (w{\cdot}w){\cdot}y$

```
    using ‹x ≤s y› by (auto simp add: suffix-def)
  have (w'·w) ≤s (w·w')·y
    unfolding ‹y = (w·w')@t·w› lassoc pow-comm[symmetric] suf-cancel-conv
    by blast

  from ruler-le[reversed, OF ‹x ≤s (w·w')·y› this
      less-imp-le[OF ‹|x| < |w · w'|›][unfolded swap-len]]]
  have x ≤s w'· w.
  hence x ≤s p
    unfolding p-def pow-pos2[OF ‹0 < q›] suffix-append by blast
  from root-suf-comm[OF - suf-ext[OF this]]
  have x·p = p·x
   using pref-pow-root[OF prefI[OF ‹x @ k = p · u @ k · s›[symmetric]]] by blast
  from comm-drop-exp[OF - this[unfolded ‹p = (w' · w) @ q›]]
  have x · (w' · w) = (w' · w) · x
    using ‹0 < q› by force
  from prim-comm-short-emp[OF ‹primitive (w'·w)› this ‹|x| < |w·w'|›[unfolded
swap-len]]
  show False
    using ‹x ≠ ε› by blast
qed

hence w·w' ≤s x
  using suf-prod-le[OF suf-prod-root[OF ‹w · w' ≤s x @ k›]] by blast
from suffix-order.trans[OF this ‹x ≤s y›]
have w · w' ≤s y.
hence |w · w'| ≤ |y|
  using suffix-length-le by blast
then obtain t' where t = Suc t'
 unfolding y-def lenmorph pow-len ‹w' ≠ ε› add.commute[of - |w|] nat-add-left-cancel-le
  using ‹w' ≠ ε› mult-0[of |w| + |w'|] npos-len[of w'] not0-implies-Suc[of t] by
force
 from ruler-eq-len[reversed, OF ‹w · w' ≤s y› - swap-len, unfolded y-def this
pow-Suc2 rassoc]
have w · w' = w'· w
  unfolding lassoc suf-cancel-conv by blast
from comm-not-prim[OF - ‹w' ≠ ε› this]
have w = ε
  using ‹primitive (w · w')› by blast
hence primitive w'
  using ‹primitive (w' · w)› by auto

have 0 < k
  using ‹|y| < |v|› lenarg[OF ‹y · x @ k · y = v · u @ k · v›, unfolded lenmorph
pow-len]
  by (intro gr-zeroI) force

have y = w'@t
  using y-def ‹w = ε› by force
```

hence $y \in \langle\{w'\}\rangle$
by *blast*

have $s \in \langle\{w'\}\rangle$
using *s-def* ‹$w = \varepsilon$› **by** *blast*
hence $v \in \langle\{w'\}\rangle$
using ‹$s \cdot y = v$› ‹$y \in \langle\{w'\}\rangle$› **by** *blast*
have $w' \leq_p x$
using ‹$x^{@}k = p \cdot u^{@}k \cdot s$›[*symmetric*] *eq-le-pref*[*OF* - ‹$|w \cdot w'| \leq |x|$›, *of* $w'^{@}(q-1) \cdot u \cdot u^{@}(k-1) \cdot s\ x^{@}(k-1)$]
unfolding *p-def* ‹$w = \varepsilon$› *emp-simps pow-pos*[*OF* ‹$0 < k$›] *pow-pos*[*OF* ‹$0 < q$›] *pow-pos rassoc* **by** *argo*

have $x \cdot w' = w' \cdot x$
using ‹$x \leq_s y$› ‹$w' \leq_p x$› **unfolding** *y-def*[*unfolded* ‹$w = \varepsilon$› ‹$t = Suc\ t'$› *emp-simps*]
using *suf-prod-root*[*THEN suf-root-pref-comm*] **by** *meson*
from *prim-comm-exp*[*OF* ‹*primitive* w'› *this*]
have $x \in \langle\{w'\}\rangle$
unfolding *sing-gen-pow-ex-conv* **by** *blast*

have $p \in \langle\{w'\}\rangle$
using ‹$s \in \langle\{w'\}\rangle$› ‹$y \in \langle\{w'\}\rangle$› ‹$s \cdot y = v$›[*folded* ‹$y \cdot p = v$›] ‹$w \cdot w' = w' \cdot w$›
unfolding *p-def s-def* **by** *argo*

have $v \cdot u^{@}k \cdot v \in \langle\{w'\}\rangle$
unfolding ‹$y \cdot x^{@}k \cdot y = v \cdot u^{@}k \cdot v$›[*symmetric*]
using ‹$y \in \langle\{w'\}\rangle$› ‹$v \in \langle\{w'\}\rangle$› *hull-closed power-in*[*OF* ‹$x \in \langle\{w'\}\rangle$›] **by** *meson*
have $u^{@}k \in \langle\{w'\}\rangle$
using *sing-gen-pref-cancel*[*OF* ‹$v \cdot u^{@}k \cdot v \in \langle\{w'\}\rangle$› ‹$v \in \langle\{w'\}\rangle$›] *sing-gen-suf-cancel*[*OF* - ‹$v \in \langle\{w'\}\rangle$›] **by** *blast*

from *prim-root-drop-exp*[*OF this* ‹$0 < k$› ‹*primitive* w'›]
have $u \in \langle\{w'\}\rangle$**.**

have $\forall x \in \{x,y,u,v\}.\ x \in \langle\{w'\}\rangle$
using ‹$x \in \langle\{w'\}\rangle$› ‹$y \in \langle\{w'\}\rangle$› ‹$u \in \langle\{w'\}\rangle$› ‹$v \in \langle\{w'\}\rangle$› **by** *blast*
then show *commutes* $\{x,y,u,v\}$
using *commutesI-ex-root* **by** *auto*
qed

lemma *bew-baiba-eq*: **assumes** $y \cdot x \neq v \cdot u$ **and**
$y \cdot x^{@}(k+1) \cdot y \cdot x = v \cdot u^{@}(k+1) \cdot v \cdot u$
shows *commutes* $\{x,y,u,v\}$
proof−
have *eq'*:$(y \cdot x) \cdot x^{@}k \cdot y \cdot x = (v \cdot u) \cdot u^{@}k \cdot v \cdot u$ **and**
eq: $(y \cdot x^{@}(k+1)) \cdot y \cdot x = (v \cdot u^{@}(k+1)) \cdot v \cdot u$ **and**

perm: $\{u,\ v\cdot u,\ x,\ y\cdot x\} = \{x,\ y\cdot x,\ u,\ v\cdot u\}$
using *assms(2)* **unfolding** *rassoc* **by** (*simp-all add*: *insert-commute*)
from *eqd-eq(1)*[*reversed*,*OF eq*]
have $|y\cdot x| \neq |v\cdot u|$
using *assms(1)* **by** *blast*
hence *commutes* $\{x,\ y\cdot x,\ u,\ v\cdot u\}$
using *bew-baiba-eq'*[*OF - triv-suf triv-suf eq'*]
bew-baiba-eq'[*OF - triv-suf triv-suf eq'*[*symmetric*], *unfolded perm*] **by** *linarith*
from *commutes-root*[*OF this*]
obtain t **where** *roots*: $\bigwedge z.\ z \in \{x,\ y\cdot x,\ u,\ v\cdot u\} \Longrightarrow z \in \langle\{t\}\rangle$
by *blast*
have $y \in \langle\{t\}\rangle$
using *roots*[*of x*] *roots*[*of y*·*x*] **by** *force*
have $v \in \langle\{t\}\rangle$
using *roots*[*of u*] *roots*[*of v*·*u*] **by** *force*
have $\forall\ z \in \{x,\ y,\ u,\ v\}.\ z \in \langle\{t\}\rangle$
using *roots*[*of u*] *roots*[*of x*] ‹$v \in \langle\{t\}\rangle$› ‹$y \in \langle\{t\}\rangle$› **by** *blast*
thus *commutes* $\{x,y,u,v\}$
using *commutesI-ex-root*[*of* $\{x,\ y\cdot x,\ u,\ v\cdot u\}$] **by** *auto*
qed

lemmas *less-mult-le* [*intro*] = *mult-le-mono1*[*OF Suc-leI*, *unfolded mult-Suc*]
lemma *less-mult-le'* [*intro*]: $m < (k::nat) \Longrightarrow m*r + r \leq k*r$
by (*simp add*: *add.commute less-mult-le*)

lemma *bew-baibaib-eq-aux*: **assumes** $|x| < |u|$ **and** $1 < i$ **and**
eq: $x\cdot y^{@}i\cdot\ x\cdot y^{@}i\cdot x = u\cdot v^{@}i\cdot\ u\cdot v^{@}i\cdot u$
shows *commutes* $\{x,y,u,v\}$
proof−
from *lenarg*[*OF* ‹$x\cdot y^{@}i\cdot\ x\cdot y^{@}i\cdot x = u\cdot v^{@}i\cdot\ u\cdot v^{@}i\cdot u$›]
have $|u\cdot v^{@}i| < |x\cdot y^{@}i|$
using ‹$|x| < |u|$› **by** *fastforce*
have $0 < i$
using ‹$1 < i$› **by** *force*

have $(x{\cdot}y^{@}i)\cdot(u{\cdot}v^{@}i) = (u{\cdot}v^{@}i)\cdot(x{\cdot}y^{@}i)$
proof (*rule two-pers*)
show $x\cdot y^{@}i\cdot\ x\cdot y^{@}i\cdot x \leq p\ (u\cdot v\ ^{@}\ i)\cdot(x\cdot y^{@}i\cdot\ x\cdot y^{@}i\cdot x)$
unfolding *eq rassoc pref-cancel-conv* **by** *blast*
show $x\cdot y^{@}i\cdot\ x\cdot y^{@}i\cdot x \leq p\ (x\cdot y\ ^{@}\ i)\cdot x\cdot y^{@}i\cdot\ x\cdot y^{@}i\cdot x$
unfolding *rassoc pref-cancel-conv* **by** *blast*
show $|x\cdot y\ ^{@}\ i| + |u\cdot v\ ^{@}\ i| \leq |x\cdot y\ ^{@}\ i\cdot x\cdot y\ ^{@}\ i\cdot x|$
using ‹$|u\cdot v^{@}i| < |x\cdot y^{@}i|$› **unfolding** *lenmorph* **by** *auto*
qed

from *comm-primrootE'*[*OF this*]
obtain $r\ m\ k$ **where** $x{\cdot}y^{@}i = r^{@}k\ u{\cdot}v^{@}i = r^{@}m\ \ primitive\ r$**.**
note *prim-nemp*[*OF* ‹*primitive r*›]

have $|y| + |r| \leq |y \,^{@}\, i|$
proof−
have $|u \cdot v \,^{@}\, i \cdot u \cdot v \,^{@}\, i| \leq |x \cdot y \,^{@}\, i \cdot x \cdot y \,^{@}\, i|$ $|v \,^{@}\, i \cdot u| \leq |y \,^{@}\, i \cdot x|$
using ‹$|u \cdot v \,^{@}\, i| < |x \cdot y \,^{@}\, i|$› **unfolding** *lenmorph* **by** *linarith*+
from *eqdE*[*of* $u \cdot v^{@}i \cdot u \cdot v^{@}i\ u\ x \cdot y^{@}i \cdot x \cdot y^{@}i\ x$,
unfolded rassoc, *OF eq*[*symmetric*] *this*(*1*)]
obtain t' **where** t': $u \cdot v \,^{@}\, i \cdot u \cdot v \,^{@}\, i \cdot t' = x \cdot y \,^{@}\, i \cdot x \cdot y \,^{@}\, i\ t' \cdot x = u$**.**
from *eqdE*[*reversed*, *of* $u \cdot v^{@}i \cdot u\ v^{@}i \cdot u\ x \cdot y^{@}i \cdot x\ y^{@}i \cdot x$,
unfolded rassoc, *OF eq*[*symmetric*] ‹$|v \,^{@}\, i \cdot u| \leq |y \,^{@}\, i \cdot x|$›]
obtain t **where** t: $t \cdot v \,^{@}\, i \cdot u = y \,^{@}\, i \cdot x\ x \cdot y \,^{@}\, i \cdot x \cdot t = u \cdot v \,^{@}\, i \cdot u$**.**
have $(x \cdot y \,^{@}\, i \cdot x \cdot t) \cdot (v \,^{@}\, i \cdot t' \cdot x) = x \cdot y \,^{@}\, i \cdot x \cdot y \,^{@}\, i \cdot x$
unfolding t' t *rassoc eq*..
hence $y^{@}i = t \cdot v^{@}i \cdot t'$
by *force*

have $m < k$
using ‹$|u \cdot v \,^{@}\, i| < |x \cdot y \,^{@}\, i|$›
unfolding ‹$u \cdot v \,^{@}\, i = r \,^{@}\, m$› ‹$x \cdot y \,^{@}\, i = r \,^{@}\, k$› *pow-len*
by *simp*
hence $|u \cdot v \,^{@}\, i| + |r| \leq |x \cdot y \,^{@}\, i|$
unfolding ‹$u \cdot v \,^{@}\, i = r \,^{@}\, m$› ‹$x \cdot y \,^{@}\, i = r \,^{@}\, k$› *pow-len* **by** *blast*
hence $2{*}|r| \leq |y^{@}i|$
using ‹$|u \cdot v \,^{@}\, i| + |r| \leq |x \cdot y \,^{@}\, i|$› *lenarg*[*OF* t'(*1*)]
unfolding *lenarg*[*OF* ‹$y^{@}i = t \cdot v^{@}i \cdot t'$›] *lenmorph* **by** *force*
thus *?thesis*
using *mult-le-mono1*[*OF Suc-leI*[*OF* ‹*1* < *i*›], *of* $|y|$]
unfolding *pow-len mult-Suc* **by** *force*
qed

have $r \cdot y = y \cdot r$
proof (*rule two-pers*[*reversed*])
show $y^{@}i \leq_s y^{@}i \cdot r$
using *triv-suf*[*of* $y \,^{@}\, i\ x$, *unfolded* ‹$x \cdot y \,^{@}\, i = r \,^{@}\, k$›, *THEN suf-prod-root*]**.**
show $y^{@}i \leq_s y^{@}i \cdot y$
by (*simp add*: *suf-pow-ext*′)
qed *fact*
note *comm-pow-comm*[*OF this*[*symmetric*], *of i*, *symmetric*]

have $r \cdot x = x \cdot r$
proof (*rule comm-cancel-suf*)
show $r \cdot x \cdot y^{@}i = x \cdot y^{@}i \cdot r$
using ‹$x \cdot y^{@}i = r^{@}k$›
by (*simp add*: *lassoc pow-comm*)
show $r \cdot y \,^{@}\, i = y \,^{@}\, i \cdot r$
by *fact*
qed

have $r \cdot u = u \cdot r$
proof (*rule comm-cancel-pref*)

show $r \cdot ((u \cdot v^{@} i) \cdot (u \cdot v^{@} i)) = ((u \cdot v^{@} i) \cdot (u \cdot v^{@} i)) \cdot r$
unfolding $\langle u{\cdot}v^{@} i = r^{@} m\rangle$ **by** *comparison*
have *comm-aux*: $r \cdot (u \cdot v\ ^{@}\ i \cdot u \cdot v\ ^{@}\ i \cdot u) = (u \cdot v\ ^{@}\ i \cdot u \cdot v\ ^{@}\ i \cdot u) \cdot r$
unfolding *eq*[*symmetric*]
using $\langle x{\cdot}y^{@} i = r^{@} k\rangle$ $\langle r \cdot x = x \cdot r\rangle$ $\langle r \cdot y\ ^{@}\ i = y\ ^{@}\ i \cdot r\rangle$ *rassoc* **by** *metis*
show $r \cdot ((u \cdot v\ ^{@}\ i) \cdot u \cdot v\ ^{@}\ i) \cdot u = ((u \cdot v\ ^{@}\ i) \cdot u \cdot v\ ^{@}\ i) \cdot u \cdot r$
using *comm-aux* **unfolding** *rassoc*.
qed

have $r \cdot v = v \cdot r$
proof(*rule comm-drop-exp*[*OF* $\langle 0 < i\rangle$, *symmetric*])
show $r \cdot v^{@} i = v^{@} i \cdot r$
proof (*rule comm-cancel-pref*)
show $r \cdot u \cdot v^{@} i = u \cdot v^{@} i \cdot r$
using $\langle u{\cdot}v^{@} i = r^{@} m\rangle$ *pow-comm rassoc* **by** *metis*
qed *fact*
qed

show *commutes* {*x*,*y*,*u*,*v*}
by (*rule commutesI'*[*OF* $\langle r \neq \varepsilon\rangle$])
(*auto simp*: $\langle r \cdot u = u \cdot r\rangle$ $\langle r \cdot v = v \cdot r\rangle$ $\langle r \cdot x = x \cdot r\rangle$ $\langle r \cdot y = y \cdot r\rangle$)
qed

lemma *bew-baibaib-eq*: **assumes** $1 < i$ **and** $x \neq u$ **and**
eq: $x \cdot y^{@} i \cdot x \cdot y^{@} i \cdot x = u \cdot v^{@} i \cdot u \cdot v^{@} i \cdot u$
shows *commutes* {*x*,*y*,*u*,*v*}
proof (*rule linorder-cases*[*of* $|x|$ $|u|$])
assume $|x| < |u|$
from *bew-baibaib-eq-aux*[*OF this assms*(*1*,*3*)]
show *commutes* {*x*,*y*,*u*,*v*}.
next
assume $|u| < |x|$
from *bew-baibaib-eq-aux*[*OF this* $\langle 1 < i\rangle$ *eq*[*symmetric*]]
show *commutes* {*x*,*y*,*u*,*v*}
by (*simp add*: *insert-commute*)
qed (*use eqd-eq*(*1*)[*OF eq*] $\langle x \neq u\rangle$ **in** *blast*)

theorem *not-beg-abiab*: ¬ *binary-equality-generator* $(\mathfrak{a} \cdot \mathfrak{b}^{@}(i{+}1) \cdot \mathfrak{a} \cdot \mathfrak{b})$
proof
have *nemp*: $\mathfrak{a} \cdot \mathfrak{b}\ ^{@}\ (i + 1) \neq \varepsilon\ \mathfrak{a} \cdot \mathfrak{b} \neq \varepsilon$
by *simp-all*
assume *binary-equality-generator* $(\mathfrak{a} \cdot \mathfrak{b}\ ^{@}\ (i{+}1) \cdot \mathfrak{a} \cdot \mathfrak{b})$
from *beg-productE*[*of* $\mathfrak{a} \cdot \mathfrak{b}^{@}(i{+}1)\ \mathfrak{a} \cdot \mathfrak{b}$, *unfolded rassoc*, *OF this nemp*]
obtain *g h* **where** *binary-code-morphism* (*g* :: *binA list* ⇒ *nat list*) *binary-code-morphism*
h g ≠ *h* $(\mathfrak{a} \cdot \mathfrak{b}\ ^{@}\ (i + 1) \cdot \mathfrak{a} \cdot \mathfrak{b}) \in g =_M h\ |g\ (\mathfrak{a} \cdot \mathfrak{b}\ ^{@}\ (i + 1))| < |h\ (\mathfrak{a} \cdot \mathfrak{b}\ ^{@}\ (i +$
1))| $|h\ (\mathfrak{a} \cdot \mathfrak{b})| < |g\ (\mathfrak{a} \cdot \mathfrak{b})|$.
interpret *g*: *binary-code-morphism g*
by *fact*
interpret *h*: *binary-code-morphism h*

by *fact*
have $g\ \mathfrak{a} \cdot g\ \mathfrak{b} \neq h\ \mathfrak{a} \cdot h\ \mathfrak{b}$
using $\langle |h\ (\mathfrak{a} \cdot \mathfrak{b})| < |g\ (\mathfrak{a} \cdot \mathfrak{b})| \rangle$ **unfolding** *g.morph h.morph* **by** *fastforce*
from *bew-baiba-eq*[*OF this*] *min-solD*[*OF* $\langle (\mathfrak{a} \cdot \mathfrak{b}\ ^{@}\ (i + 1) \cdot \mathfrak{a} \cdot \mathfrak{b}) \in g =_M h \rangle$]
have *commutes* $\{g\ \mathfrak{b},\ g\ \mathfrak{a},\ h\ \mathfrak{b},\ h\ \mathfrak{a}\}$
unfolding *g.morph h.morph g.pow-morph h.pow-morph* **by** *blast*
from *commutesE*[*OF this*, *of g* $\mathfrak{a}$ *g* $\mathfrak{b}$]
show *False*
using *g.non-comm-morph*[*of bina*] **by** *simp*
qed

theorem *not-beg-baibaib*: **assumes** $1 < i$
shows ¬ *binary-equality-generator* $(\mathfrak{b} \cdot \mathfrak{a}^{@}i \cdot \mathfrak{b} \cdot \mathfrak{a}^{@}i \cdot \mathfrak{b})$
proof
have *nemp*: $\mathfrak{b} \cdot \mathfrak{a}^{@}i \cdot \mathfrak{b} \cdot \mathfrak{a}^{@}i \neq \varepsilon\ \mathfrak{b} \neq \varepsilon$
by *simp-all*
assume *binary-equality-generator* $(\mathfrak{b} \cdot \mathfrak{a}^{@}i \cdot \mathfrak{b} \cdot \mathfrak{a}^{@}i \cdot \mathfrak{b})$
from *beg-productE*[*of* $\mathfrak{b} \cdot \mathfrak{a}^{@}i \cdot \mathfrak{b} \cdot \mathfrak{a}^{@}i\ \mathfrak{b}$, *unfolded rassoc*, *OF this nemp*]
obtain *g h* **where** *binary-code-morphism* (*g* :: *binA list* ⇒ *nat list*) *binary-code-morphism h g* $\neq h\ (\mathfrak{b} \cdot \mathfrak{a}\ ^{@}\ i \cdot \mathfrak{b} \cdot \mathfrak{a}\ ^{@}\ i \cdot \mathfrak{b}) \in g =_M h\ |g\ (\mathfrak{b} \cdot \mathfrak{a}\ ^{@}\ i \cdot \mathfrak{b} \cdot \mathfrak{a}\ ^{@}\ i)| < |h\ (\mathfrak{b} \cdot \mathfrak{a}\ ^{@}\ i \cdot \mathfrak{b} \cdot \mathfrak{a}\ ^{@}\ i)|\ |h\ \mathfrak{b}| < |g\ \mathfrak{b}|$.
interpret *g*: *binary-code-morphism g*
by *fact*
interpret *h*: *binary-code-morphism h*
by *fact*
have $g\ \mathfrak{b} \neq h\ \mathfrak{b}$
using $\langle |h\ \mathfrak{b}| < |g\ \mathfrak{b}| \rangle$ **by** *fastforce*
from *bew-baibaib-eq*[*OF assms this*] *min-solD*[*OF* $\langle (\mathfrak{b} \cdot \mathfrak{a}\ ^{@}\ i \cdot \mathfrak{b} \cdot \mathfrak{a}\ ^{@}\ i \cdot \mathfrak{b}) \in g =_M h \rangle$]
have *commutes* $\{g\ \mathfrak{b},\ g\ \mathfrak{a},\ h\ \mathfrak{b},\ h\ \mathfrak{a}\}$
unfolding *g.morph h.morph g.pow-morph h.pow-morph* **by** *blast*
from *commutesE*[*OF this*, *of g* $\mathfrak{a}$ *g* $\mathfrak{b}$]
show *False*
using *g.non-comm-morph*[*of bina*] **by** *simp*
qed

end

References

[1] C. Choffrut and J. Karhumäki. *Combinatorics of Words*, page 329–438. Springer-Verlag, Berlin, Heidelberg, 1997.

[2] P. Dömösi and G. Horváth. Alternative proof of the Lyndon–Schützenberger theorem. *Theoret. Comput. Sci.*, 366(3):194–198, Nov. 2006.

[3] N. J. Fine and H. S. Wilf. Uniqueness theorems for periodic functions. *Proc. Am. Math. Soc.*, 16(1):109–114, 1965.

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

[5] M. Lothaire. *Algebraic Combinatorics on Words*. Number 90 in Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2002.

[6] M. Lothaire. *Applied Combinatorics on Words*. Number 105 in Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2005.

[7] R. C. Lyndon and P. Schützenberger. The equation $a^m = b^n c^p$ in a free group. *Michigan Math. J.*, 9:289–298, 1962.