Combinatorics on Words formalized
Basics

Štěpán Holub
Martin Raška
Štěpán Starosta

March 17, 2025

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

Contents

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*: $i \leq j \Longrightarrow (0::nat) = j - i \Longrightarrow j = i$
  ⟨*proof*⟩

lemma *zero-less-diff'*: $i < j \Longrightarrow j - i \neq (0::nat)$
  ⟨*proof*⟩

lemma *nat-prod-le*: $m \neq (0 :: nat) \Longrightarrow m*n \leq k \Longrightarrow n \leq k$
  ⟨*proof*⟩

lemma *get-div*: $(p :: nat) < a \Longrightarrow m = (m * a + p)\ div\ a$
  ⟨*proof*⟩

lemma *get-mod*: $(p :: nat) < a \Longrightarrow p = (m * a + p)\ mod\ a$
  ⟨*proof*⟩

lemma *plus-one-between*: $(a :: nat) < b \Longrightarrow \neg\ b < a + 1$
  ⟨*proof*⟩

lemma *quotient-smaller*: $k \neq (0 :: nat) \Longrightarrow b \leq k * b$
  ⟨*proof*⟩

lemma *mult-cancel-le*: $b \neq 0 \Longrightarrow a*b \leq c*b \Longrightarrow a \leq (c::nat)$
  ⟨*proof*⟩

lemma *add-lessD2*: $k + m < (n::nat) \Longrightarrow m < n$
⟨*proof*⟩

lemma *mod-offset*: **assumes** $M \neq (0 :: nat)$
obtains k **where** $n\ mod\ M = (l + k)\ mod\ M$
⟨*proof*⟩

lemma assumes $q \neq (0::nat)$ **shows** $p \leq p + q - gcd\ p\ q$
⟨*proof*⟩

lemma *less-mult-one*: **assumes** $(m-1)*k < k$ **obtains** $m = 0 \mid m = (1::nat)$
⟨*proof*⟩

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*: $1 \leq k \longleftrightarrow 0 < (k::nat)$
⟨*proof*⟩

lemma *per-lemma-len-le*: **assumes** *le*: $p + q - gcd\ p\ q \leq (n :: nat)$ **and** $0 < q$ **shows** $p \leq n$
⟨*proof*⟩

lemma *Suc-less-iff-Suc-le*: $Suc\ n < k \longleftrightarrow Suc\ n \leq k - 1$
⟨*proof*⟩

lemma *nat-induct-pair*: $P\ 0\ 0 \Longrightarrow (\bigwedge m\ n.\ P\ m\ n \Longrightarrow P\ m\ (Suc\ n)) \Longrightarrow (\bigwedge m\ n.\ P\ m\ n \Longrightarrow P\ (Suc\ m)\ n) \Longrightarrow P\ m\ n$
⟨*proof*⟩

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

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

lemma *at-least3-Suc*: **assumes** $3 \leq k$
obtains k' **where** $k = Suc(Suc(Suc\ k'))$
⟨*proof*⟩

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

lemma *le1-SucE*: **assumes** $1 \leq n$
obtains k **where** $n = Suc\ k$ ⟨*proof*⟩

lemma *Suc-minus*: $k \neq 0 \Longrightarrow Suc\ (k - 1) = k$
⟨*proof*⟩

lemma *Suc-minus'*: $1 \leq k \Longrightarrow Suc(k - 1) = k$
⟨*proof*⟩

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

lemma *Suc-minus2*: *2* ≤ *k* ⟹ *Suc* (*Suc*(*k* − *2*)) = *k*
⟨*proof*⟩

lemma *Suc-leE*: **assumes** *Suc k* ≤ *n* **obtains** *m* **where** *n* = *Suc m* **and** *k* ≤ *m*
⟨*proof*⟩

lemma *two-three-add-le-mult*: *2* ≤ (*l*::*nat*) ⟹ *3* ≤ *k* ⟹ *k* + *l* + *1* ≤ *k*∗*l*
⟨*proof*⟩

lemma *almost-equal-equal*: **assumes** (*a*:: *nat*) ≠ *0* **and** *b* ≠ *0* **and** *eq*: *k*∗(*a*+*b*) + *a* = *m*∗(*a*+*b*) + *b*
shows *k* = *m* **and** *a* = *b*
⟨*proof*⟩

lemma *crossproduct-le*: **assumes** (*a*::*nat*) ≤ *b* **and** *c* ≤ *d*
shows *a*∗*d* + *b*∗*c* ≤ *a*∗*c* + *b*∗*d*
⟨*proof*⟩

lemma (**in** *linorder*) *le-less-cases*: (*a* ≤ *b* ⟹ *P*) ⟹ (*b* < *a* ⟹ *P*) ⟹ *P*
⟨*proof*⟩

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 ∀, ∃ and ∃! 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 ∃ 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*⟩

lemma *All-surj-conv*: **assumes** *surj f* **shows** $(\forall\, x.\ P\ (f\ x)) \longleftrightarrow (\forall\, y.\ P\ y)$
⟨*proof*⟩

lemma *Ex-surj-conv*: **assumes** *surj f* **shows** $(\exists x.\ P\ (f\ x)) \longleftrightarrow (\exists y.\ P\ y)$
⟨*proof*⟩

lemma *Ex1-bij-conv*: **assumes** *bij f* **shows** $(\exists!x.\ P\ (f\ x)) \longleftrightarrow (\exists!y.\ P\ y)$
⟨*proof*⟩

lemma *Ball-inj-conv*: **assumes** *inj f* **shows** $(\forall y{\in}f\ `\ A.\ P\ (inv\ f\ y)) \longleftrightarrow (\forall x{\in}A.\ P\ x)$
⟨*proof*⟩

lemma *Bex-inj-conv*: **assumes** *inj f* **shows** $(\exists y{\in}f\ `\ A.\ P\ (inv\ f\ y)) \longleftrightarrow (\exists x{\in}A.\ P\ x)$
⟨*proof*⟩

1.1.1 Quantifications and reverse

lemma *rev-involution'*: $rev \circ rev = id$
⟨*proof*⟩

lemma *rev-inv*: $inv\ rev = rev$
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *snocs-Nil*: *snocs* [] *xs* = *xs*
⟨*proof*⟩

lemma *snocs-is-append*: *snocs xs ys* = *xs* @ *ys*
⟨*proof*⟩ **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*⟩

1.2.4 Tracing attribute

⟨*ML*⟩

1.2.5 Rule attribute *reversed*

private lemma *rev-Nil*: *rev* [] ≡ []
⟨*proof*⟩ **lemma** *map-Nil*: *map f* [] ≡ []
⟨*proof*⟩ **lemma** *image-empty*: *f* ‘ *Set.empty* ≡ *Set.empty*
⟨*proof*⟩

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

lemma *COMP-assoc*: *F oo* (*f o g*) ≡ (*F oo f*) *oo g*
⟨*proof*⟩ **lemma** *image-comp-image*: (‘) *f* ∘ (‘) *g* ≡ (‘) (*f* ∘ *g*)
⟨*proof*⟩ **lemma** *rev-involution*: *rev* ∘ *rev* ≡ *id*
⟨*proof*⟩ **lemma** *map-involution*: **assumes** *f* ∘ *f* ≡ *id* **shows** (*map f*) ∘ (*map f*) ≡ *id*
⟨*proof*⟩ **lemma** *image-involution*: **assumes** *f* ∘ *f* ≡ *id* **shows** (*image f*) ∘ (*image f*) ≡ *id*
⟨*proof*⟩ **lemma** *rev-map-comm*: *rev* ∘ *map f* ≡ *map f* ∘ *rev*
⟨*proof*⟩ **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*
⟨*proof*⟩ **lemma** *rev-map-involution*: **assumes** *g o g* ≡ *id*
shows (*rev* ∘ *map g*) ∘ (*rev* ∘ *map g*) ≡ *id*
⟨*proof*⟩ **lemma** *prop-abs-subst*: **assumes** *f o f* ≡ *id* **shows** (λ*x*. *F* (*f x*)) *oo f* ≡ (λ*x*. *F x*)
⟨*proof*⟩ **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*)
⟨*proof*⟩ **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*)
⟨*proof*⟩ **lemma** *obj-abs-subst*: **assumes** *f o f* ≡ *id* **shows** (λ*x*. *F* (*f x*)) *o f* ≡ (λ*x*. *F x*)
⟨*proof*⟩ **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*)
⟨*proof*⟩ **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*)
⟨*proof*⟩

⟨*ML*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩
lemma *Ex-surj-conv′* [*reversal-rule*]: **assumes** *surj f* **shows** *Ex* (*P o f*) ⟷ *Ex P*
⟨*proof*⟩
lemma *Ex1-bij-conv′* [*reversal-rule*]: **assumes** *bij f* **shows** *Ex1* (*P o f*) ⟷ *Ex1 P*
⟨*proof*⟩
lemma *if-image* [*reversal-rule*]: (*if P then f u else f v*) = *f* (*if P then u else v*)
⟨*proof*⟩

1.3.3 *HOL.Set*

lemma *collect-image*: *Collect* (*P* ∘ *f*) = *f* −‘ (*Collect P*)
⟨*proof*⟩

lemma *collect-image′* [*reversal-rule*]: **assumes** *f* ∘ *f* = *id* **shows** *Collect* (*P* ∘ *f*) = *f* ‘ (*Collect P*)
⟨*proof*⟩
lemma *Ball-image* [*reversal-rule*]: **assumes** (*g* ∘ *f*) ‘ *A* = *A* **shows** *Ball* (*f* ‘ *A*) (*P* ∘ *g*) = *Ball A P*
⟨*proof*⟩
lemma *Bex-image-comp*: *Bex* (*f* ‘ *A*) *g* = *Bex A* (*g* ∘ *f*)
⟨*proof*⟩

lemma *Bex-image* [*reversal-rule*]: **assumes** (*g* ∘ *f*) ‘ *A* = *A* **shows** *Bex* (*f* ‘ *A*) (*P* ∘ *g*) = *Bex A P*
⟨*proof*⟩
lemma *insert-image* [*reversal-rule*]: *insert* (*f x*) (*f* ‘ *X*) = *f* ‘ (*insert x X*)
⟨*proof*⟩
lemmas [*reversal-rule*] = *inj-image-mem-iff*

— (⊆)
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*])
⟨*proof*⟩

lemma *Cons-map*: (*f x*) # (*map f xs*) = *map f* (*x* # *xs*)
⟨*proof*⟩

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*)
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *coset-map*: **assumes** *bij f* **shows** *List.coset* (*map f xs*) = *f* ‘ *List.coset xs*
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *concat-rev-map-rev′*: *concat* (*rev* (*map* (*rev* ∘ *f*) *xs*)) = *rev* (*concat* (*map f* *xs*))
⟨*proof*⟩

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*)
⟨*proof*⟩
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*⟩
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*
⟨*proof*⟩

lemma *map-in-lists*: *inj f* ⟹ *map f u* ∈ *lists* (*f* ' *A*) ⟷ *u* ∈ *lists A*
⟨*proof*⟩

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*])
⟨*proof*⟩

1.4

lemma *bij-rev*: *bij rev*
⟨*proof*⟩

lemma *bij-map*: *bij f* ⟹ *bij* (*map f*)
⟨*proof*⟩

lemma *surj-map*: *surj f* ⟹ *surj* (*map f*)

⟨*proof*⟩

lemma *bij-image*: *bij f* $\Longrightarrow$ *bij* (*image f*)
⟨*proof*⟩

lemma *inj-image*: *inj f* $\Longrightarrow$ *inj* (*image f*)
⟨*proof*⟩

lemma *surj-image*: *surj f* $\Longrightarrow$ *surj* (*image f*)
⟨*proof*⟩

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*
⟨*proof*⟩

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*: $xs \neq [] \Longrightarrow rotate\ 1\ xs = tl\ xs\ @\ [hd\ xs]$
⟨*proof*⟩

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

end

end

theory *CoWBasic*
imports *HOL−Library.Sublist Arithmetical-Hints Reverse-Symmetry HOL−Eisbach.Eisbach-Tools*
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 (‹|-|›) — 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*
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** ‹≤p› *50*)
notation (*latex* **output**) *prefix* (‹$\leq_p$›)

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

lemma *prefI*[*intro*]: $p \cdot s = w \Longrightarrow p \leq p\ w$
⟨*proof*⟩

lemma *prefD*: $u \leq p\ v \Longrightarrow \exists\ z.\ v = u \cdot z$
⟨*proof*⟩

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

lemma *pref-compI1*: $u \leq p\ v \Longrightarrow u \bowtie v$
⟨*proof*⟩

lemma *pref-compI2*: $v \leq p\ u \Longrightarrow u \bowtie v$
⟨*proof*⟩

lemma *pref-compE* [*elim*]: **assumes** $u \bowtie v$ **obtains** $u \leq p\ v \mid v \leq p\ u$
⟨*proof*⟩

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

⟨*proof*⟩

definition *nonempty-prefix* (**infixl** ‹≤np› *50*) **where** *nonempty-prefix-def*[*simp*]:
$u \le np\ v \equiv u \neq \varepsilon \wedge u \le p\ v$
notation (*latex* **output**) *nonempty-prefix* (‹$\le_{np}$› *50*)

lemma *npI*[*intro*]: $u \neq \varepsilon \Longrightarrow u \le p\ v \Longrightarrow u \le np\ v$
⟨*proof*⟩

lemma *npI'*[*intro*]: $u \neq \varepsilon \Longrightarrow (\exists\ z.\ u \cdot z = v) \Longrightarrow u \le np\ v$
⟨*proof*⟩

lemma *npD*: $u \le np\ v \Longrightarrow u \le p\ v$
⟨*proof*⟩

lemma *npD'*: $u \le np\ v \Longrightarrow u \neq \varepsilon$
⟨*proof*⟩

notation *strict-prefix* (**infixl** ‹<p› *50*)
notation (*latex* **output**) *strict-prefix* (‹$<_p$›)
lemmas [*simp*] = *strict-prefix-def*

interpretation *lcp*: *semilattice-order* ($\wedge_p$) *prefix strict-prefix*
⟨*proof*⟩

lemmas *sprefI* = *strict-prefixI*

lemma *sprefI1*[*intro*]: $v = u \cdot z \Longrightarrow z \neq \varepsilon \Longrightarrow u <p\ v$
⟨*proof*⟩

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

lemma *sprefI2*[*intro*]: $u \le p\ v \Longrightarrow |u| < |v| \Longrightarrow u <p\ v$
⟨*proof*⟩

lemma *sprefD*: $u <p\ v \Longrightarrow u \le p\ v \wedge u \neq v$
⟨*proof*⟩

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** $z \neq \varepsilon$
⟨*proof*⟩

2.1.3 Suffix

notation *suffix* (**infixl** ‹≤s› *50*)
notation (*latex* **output**) *suffix* (‹$\leq_s$›)

lemma *sufI*[*intro*]: $p \cdot s = w \Longrightarrow s \leq s\ w$
⟨*proof*⟩

lemma *sufD*[*elim*]: $u \leq s\ v \Longrightarrow \exists\ z.\ z \cdot u = v$
⟨*proof*⟩

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*]: $u \cdot v = w \Longrightarrow u \neq \varepsilon \Longrightarrow v <s\ w$
⟨*proof*⟩

lemma *ssufI2*[*intro*]: $u \leq s\ v \Longrightarrow$ *length* $u <$ *length* $v \Longrightarrow u <s\ v$
⟨*proof*⟩

lemma *ssufE*: $u <s\ v \Longrightarrow (u \leq s\ v \Longrightarrow u \neq v \Longrightarrow$ *thesis*$) \Longrightarrow$ *thesis*
⟨*proof*⟩

lemma *ssufI3*[*intro*]: $u \cdot v = w \Longrightarrow u \leq np\ w \Longrightarrow v <s\ w$
⟨*proof*⟩

lemma *ssufD*[*elim*]: $u <s\ v \Longrightarrow u \leq s\ v \wedge u \neq v$
⟨*proof*⟩

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*]: $u \leq s\ v \Longrightarrow u \bowtie_s v$
⟨*proof*⟩

lemma *suf-compI2*[*intro*]: $v \leq s\ u \Longrightarrow u \bowtie_s v$
⟨*proof*⟩

definition *nonempty-suffix* (**infixl** ‹≤ns› *60*) **where** *nonempty-suffix-def*[*simp*]:
$u \leq ns\ v \equiv u \neq \varepsilon \wedge u \leq s\ v$
notation (*latex* **output**) *nonempty-suffix* (‹$\leq_{ns}$› *50*)

lemma *nsI*[*intro*]: $u \neq \varepsilon \Longrightarrow u \leq s\ v \Longrightarrow u \leq ns\ v$
⟨*proof*⟩

lemma *nsI'*[*intro*]: $u \neq \varepsilon \Longrightarrow (\exists\ z.\ z \cdot u = v) \Longrightarrow u \leq ns\ v$
⟨*proof*⟩

lemma *nsD*: $u \leq ns\ v \Longrightarrow u \leq s\ v$
⟨*proof*⟩

lemma *nsD'*: $u \leq ns\ v \Longrightarrow u \neq \varepsilon$
⟨*proof*⟩

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** ‹≤*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** ‹≤*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'*: $a \cdot u \cdot b = w \Longrightarrow u \leq f\ w$
⟨*proof*⟩

lemma *facE*[*elim*]: **assumes** $u \leq f\ v$
obtains *p s* **where** $v = p \cdot u \cdot s$
⟨*proof*⟩

lemma *facE'*[*elim*]: **assumes** $u \leq f\ v$
obtains *p s* **where** $p \cdot u \cdot s = v$
⟨*proof*⟩

2.2 Various elementary lemmas

lemmas *drop-all-iff* = *drop-eq-Nil* — backward compatibility with Isabelle 2021

lemma *exE2*: $\exists\ x\ y.\ P\ x\ y \Longrightarrow (\bigwedge x\ y.\ P\ x\ y \Longrightarrow thesis) \Longrightarrow thesis$
⟨*proof*⟩

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*: $rev\ u \in A \longleftrightarrow u \in rev\ `\ A$
⟨*proof*⟩

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

lemma *map-rev-lists-rev*: *map rev ‘ (lists (rev ‘ A)) = lists A*
⟨*proof*⟩

lemma *inj-on-map-lists*: **assumes** *inj-on f A*
shows *inj-on (map f) (lists A)*
⟨*proof*⟩

lemma *bij-lists*: *bij-betw f X Y* $\Longrightarrow$ *bij-betw (map f) (lists X) (lists Y)*
⟨*proof*⟩

lemma *concat-sing′*: *concat* $[r] = r$
⟨*proof*⟩

lemma *concat-sing*: **assumes** $s = [a]$ **shows** *concat* $s = a$
⟨*proof*⟩

lemma *rev-sing*: *rev* $[x] = [x]$
⟨*proof*⟩

lemma *hd-word*: $a\#ws = [a] \cdot ws$
⟨*proof*⟩

lemma *pref-singE*: **assumes** $p \le p\ [a]$ **obtains** $p = \varepsilon \mid p = [a]$
⟨*proof*⟩

lemma *map-hd*: $map\ f\ (a\#v) = [f\ a] \cdot (map\ f\ v)$
⟨*proof*⟩

lemma *hd-tl*: $w \neq \varepsilon \Longrightarrow [hd\ w] \cdot tl\ w = w$
⟨*proof*⟩

lemma *hd-tlE*: **assumes** $w \neq \varepsilon$
obtains $a\ w'$ **where** $w = a\#w'$

⟨*proof*⟩

lemma *hd-tl-lenE*: **assumes** $0 < |w|$
obtains $a\ w'$ **where** $w = a\#w'$
⟨*proof*⟩

lemma *hd-tl-longE*: **assumes** $Suc\ 0 < |w|$
obtains $a\ w'$ **where** $w = a\#w'$ **and** $w' \neq \varepsilon$ **and** $hd\ w = a$ **and** $tl\ w = w'$
⟨*proof*⟩

lemma *hd-pref*: $w \neq \varepsilon \Longrightarrow [hd\ w] \leq p\ w$
⟨*proof*⟩

lemma *add-nth*: **assumes** $n < |w|$ **shows** $(take\ n\ w) \cdot [w!n] \leq p\ w$
⟨*proof*⟩

lemma *hd-pref'*: **assumes** $w \neq \varepsilon$ **shows** $[w!0] \leq p\ w$
⟨*proof*⟩

lemma *sub-lists-mono*: $A \subseteq B \Longrightarrow x \in lists\ A \Longrightarrow x \in lists\ B$
⟨*proof*⟩

lemma *lists-hd-in-set*[*simp*]: $us \neq \varepsilon \Longrightarrow us \in lists\ Q \Longrightarrow hd\ us \in Q$
⟨*proof*⟩

lemma *lists-last-in-set*[*simp*]: $us \neq \varepsilon \Longrightarrow us \in lists\ Q \Longrightarrow last\ us \in Q$
⟨*proof*⟩

lemma *replicate-in-lists*: $replicate\ k\ z \in lists\ \{z\}$
⟨*proof*⟩

lemma *tl-in-lists*: **assumes** $us \in lists\ A$ **shows** $tl\ us \in lists\ A$
⟨*proof*⟩

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

lemma *long-list-tl*: **assumes** $1 < |us|$ **shows** $tl\ us \neq \varepsilon$
⟨*proof*⟩

lemma *tl-set*: $x \in set\ (tl\ a) \Longrightarrow x \in set\ a$
⟨*proof*⟩

lemma *drop-take-inv*: $n \leq |u| \Longrightarrow drop\ n\ (take\ n\ u \cdot w) = w$
⟨*proof*⟩

lemma *split-list-long*: **assumes** $1 < |us|$ **and** $u \in set\ us$
obtains $xs\ ys$ **where** $us = xs \cdot [u] \cdot ys$ **and** $xs{\cdot}ys{\neq}\varepsilon$
⟨*proof*⟩

lemma *flatten-lists*: $G \subseteq lists\ B \Longrightarrow xs \in lists\ G \Longrightarrow concat\ xs \in lists\ B$
⟨*proof*⟩

lemma *concat-map-sing-ident*: $concat\ (map\ (\lambda\ x.\ [x])\ xs) = xs$
⟨*proof*⟩

lemma *hd-concat-tl*: **assumes** $ws \neq \varepsilon$ **shows** $hd\ ws \cdot concat\ (tl\ ws) = concat\ ws$
⟨*proof*⟩

lemma *concat-butlast-last*: **assumes** $ws \neq \varepsilon$ **shows** $concat\ (butlast\ ws) \cdot last\ ws = concat\ ws$
⟨*proof*⟩

lemma *spref-butlast-pref*: **assumes** $u \leq p\ v$ **and** $u \neq v$ **shows** $u \leq p\ butlast\ v$
⟨*proof*⟩

lemma *last-concat*: $xs \neq \varepsilon \Longrightarrow last\ xs \neq \varepsilon \Longrightarrow last\ (concat\ xs) = last\ (last\ xs)$
⟨*proof*⟩

lemma *concat-last-suf*: $ws \neq \varepsilon \Longrightarrow last\ ws \leq s\ concat\ ws$
⟨*proof*⟩

lemma *concat-hd-pref*: $ws \neq \varepsilon \Longrightarrow hd\ ws \leq p\ concat\ ws$
⟨*proof*⟩

lemma *set-nemp-concat-nemp*: **assumes** $ws \neq \varepsilon$ **and** $\varepsilon \notin set\ ws$ **shows** $concat\ ws \neq \varepsilon$
⟨*proof*⟩

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

lemma *suf-drop-conv*: $u \leq s\ w \longleftrightarrow drop\ (|w| - |u|)\ w = u$
⟨*proof*⟩

lemma *comm-rev-iff*: $rev\ u \cdot rev\ v = rev\ v \cdot rev\ u \longleftrightarrow u \cdot v = v \cdot u$
⟨*proof*⟩

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*⟩

lemma *fin-bin*: $finite\ \{x,y\}$
⟨*proof*⟩

lemma *rev-rev-image-eq* [*reversal-rule*]: $rev\ `\ rev\ `\ X = X$

⟨*proof*⟩

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

lemma *inj-map-inv*: $inj\text{-}on\ f\ A \Longrightarrow u \in lists\ A \Longrightarrow u = map\ (the\text{-}inv\text{-}into\ A\ f)\ (map\ f\ u)$
⟨*proof*⟩

lemma *last-sing*[*simp*]: $last\ [c] = c$
⟨*proof*⟩

lemma *hd-hdE*: $(u = \varepsilon \Longrightarrow thesis) \Longrightarrow (u = [hd\ u] \Longrightarrow thesis) \Longrightarrow (u = [hd\ u,\ hd\ (tl\ u)] \cdot tl\ (tl\ u) \Longrightarrow thesis) \Longrightarrow thesis$
⟨*proof*⟩

lemma *same-sing-pref*: $u \cdot [a] \leq p\ v \Longrightarrow u \cdot [b] \leq p\ v \Longrightarrow a = b$
⟨*proof*⟩

lemma *compow-Suc*: $(f\hat{}\hat{}(Suc\ k))\ w = f\ ((f\hat{}\hat{}k)\ w)$
⟨*proof*⟩

lemma *compow-Suc'*: $(f\hat{}\hat{}(Suc\ k))\ w = (f\hat{}\hat{}k)\ (f\ w)$
⟨*proof*⟩

2.2.1 General facts

lemma *two-elem-sub*: $x \in A \Longrightarrow y \in A \Longrightarrow \{x,y\} \subseteq A$
⟨*proof*⟩

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

lemma *inj-comp*: **assumes** $inj\ (f :: 'a\ list \Rightarrow 'b\ list)$ **shows** $(g\ \ w = h\ w \longleftrightarrow (f \circ g)\ w = (f \circ h)\ w)$
⟨*proof*⟩

lemma *inj-comp-eq*: **assumes** $inj\ (f :: 'a\ list \Rightarrow 'b\ list)$ **shows** $(g = h \longleftrightarrow f \circ g = f \circ h)$
⟨*proof*⟩

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

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

lemma *two-elemP*: $u \in \{x,\ y\} \Longrightarrow P\ x \Longrightarrow P\ y \Longrightarrow P\ u$
⟨*proof*⟩

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

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

lemma *doubleton-subset-cases*: **assumes** $A \subseteq \{x,y\}$
obtains $A = \{\}\ |\ A = \{x\}\ |\ A = \{y\}\ |\ A = \{x,y\}$
⟨*proof*⟩

2.2.2 Map injective function

lemma *map-pref-conv* [*reversal-rule*]: **assumes** *inj f* **shows** $map\ f\ u \le p\ map\ f\ v \longleftrightarrow u \le p\ v$
⟨*proof*⟩

lemma *map-suf-conv* [*reversal-rule*]: **assumes** *inj f* **shows** $map\ f\ u \le s\ map\ f\ v \longleftrightarrow u \le s\ v$
⟨*proof*⟩

lemma *map-fac-conv* [*reversal-rule*]: **assumes** *inj f* **shows** $map\ f\ u \le f\ map\ f\ v \longleftrightarrow u \le f\ v$
⟨*proof*⟩

lemma *map-lcp-conv*: **assumes** *inj f* **shows** $(map\ f\ xs) \wedge_p (map\ f\ ys) = map\ f\ (xs \wedge_p ys)$
⟨*proof*⟩

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*]: $u \in S \Longrightarrow u \neq \varepsilon \Longrightarrow u \in S - \{\varepsilon\}$
⟨*proof*⟩

lemma *emp-concat-emp*: $us \in lists\ (S - \{\varepsilon\}) \Longrightarrow concat\ us = \varepsilon \Longrightarrow us = \varepsilon$
⟨*proof*⟩

lemma *take-nemp*: $w \neq \varepsilon \Longrightarrow 0 < n \Longrightarrow take\ n\ w \neq \varepsilon$
⟨*proof*⟩

lemma *pref-nemp* [*intro*]: $u \neq \varepsilon \Longrightarrow u \cdot v \neq \varepsilon$
⟨*proof*⟩

lemma *suf-nemp* [*intro*]: $v \neq \varepsilon \Longrightarrow u \cdot v \neq \varepsilon$
⟨*proof*⟩

lemma *pref-of-emp*: $u \cdot v = \varepsilon \Longrightarrow u = \varepsilon$
⟨*proof*⟩

lemma *suf-of-emp*: $u \cdot v = \varepsilon \Longrightarrow v = \varepsilon$
⟨*proof*⟩

lemma *nemp-comm*: $(u \neq \varepsilon \Longrightarrow v \neq \varepsilon \Longrightarrow u \neq v \Longrightarrow u \cdot v = v \cdot u) \Longrightarrow u \cdot v = v \cdot u$
⟨*proof*⟩

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

lemma *split-list'*: $a \in set\ ws \Longrightarrow \exists\, p\ s.\ ws = p \cdot [a] \cdot s$
⟨*proof*⟩

lemma *split-listE*: **assumes** $a \in set\ w$
obtains $p\ s$ **where** $w = p \cdot [a] \cdot s$
⟨*proof*⟩

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*
⟨*proof*⟩

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 $u \in \{x,y\} \Longrightarrow P\ u$
⟨*proof*⟩

lemma $\bigwedge u.\ u \in \{x,y\} \Longrightarrow u = x \vee u = y$
⟨*proof*⟩

2.3 Length and its properties

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

lemma *lenarg-not*: $|u| \neq |v| \Longrightarrow u \neq v$
⟨*proof*⟩

lemma *len-less-neq*: $|u| < |v| \Longrightarrow u \neq v$
⟨*proof*⟩

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

lemma *npos-len*: $|u| \leq 0 \Longrightarrow u = \varepsilon$
⟨*proof*⟩

lemma *nemp-pos-len*: $w \neq \varepsilon \Longrightarrow 0 < |w|$
⟨*proof*⟩

lemma *nemp-le-len*: $r \neq \varepsilon \Longrightarrow 1 \leq |r|$
⟨*proof*⟩

lemma *swap-len*: $|u \cdot v| = |v \cdot u|$
⟨*proof*⟩

lemma *len-after-drop*: $p + q \leq |w| \Longrightarrow q \leq |drop\ p\ w|$
⟨*proof*⟩

lemma *short-take-append*: $n \leq |u| \Longrightarrow take\ n\ (u \cdot v) = take\ n\ u$
⟨*proof*⟩

lemma *sing-word*: $|us| = 1 \Longrightarrow [hd\ us] = us$
⟨*proof*⟩

lemma *sing-word-concat*: **assumes** $|us| = 1$ **shows** $[concat\ us] = us$
⟨*proof*⟩

lemma *len-one-concat-in*: $ws \in lists\ A \Longrightarrow |ws| = 1 \Longrightarrow concat\ ws \in A$
⟨*proof*⟩

lemma *concat-nemp*: $concat\ us \neq \varepsilon \Longrightarrow us \neq \varepsilon$
⟨*proof*⟩

lemma *sing-len*: $|[a]| = 1$
⟨*proof*⟩

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'*: $|u| \leq |u \cdot z|$
⟨*proof*⟩

lemma *suf-len'*: $|u| \leq |z \cdot u|$
⟨*proof*⟩

lemma *fac-len*: $u \leq f\ v \Longrightarrow |u| \leq |v|$
⟨*proof*⟩

lemma *fac-len'*: $|w| \leq |u \cdot w \cdot v|$
⟨*proof*⟩

lemma *fac-len-eq*: $u \leq f\ v \Longrightarrow |u| = |v| \Longrightarrow u = v$
⟨*proof*⟩

thm *length-take*

lemma *len-take1*: $|take\ p\ w| \leq p$
⟨*proof*⟩

lemma *len-take2*: $|take\ p\ w| \leq |w|$
⟨*proof*⟩

lemma *drop-len*: $|u \cdot w| \leq |u \cdot v \cdot w|$
⟨*proof*⟩

lemma *drop-pref*: $drop\ |u|\ (u \cdot w) = w$
⟨*proof*⟩

lemma *take-len*: $p \leq |w| \Longrightarrow |take\ p\ w| = p$
⟨*proof*⟩

lemma *conj-len*: $p \cdot x = x \cdot s \Longrightarrow |p| = |s|$
⟨*proof*⟩

lemma *take-nemp-len*: $u \neq \varepsilon \Longrightarrow r \neq \varepsilon \Longrightarrow take\ |r|\ u \neq \varepsilon$
⟨*proof*⟩

lemma *nemp-len*: $u \neq \varepsilon \Longrightarrow |u| \neq 0$

⟨*proof*⟩

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

lemma *take-self*: $take\ |w|\ w = w$
⟨*proof*⟩

lemma *len-le-concat*: $\varepsilon \notin\ set\ ws \Longrightarrow |ws| \leq |concat\ ws|$
⟨*proof*⟩

lemma *eq-len-iff*: **assumes** *eq*: $x \cdot y = u \cdot v$ **shows** $|x| \leq |u| \longleftrightarrow |v| \leq |y|$
⟨*proof*⟩

lemma *eq-len-iff-less*: **assumes** *eq*: $x \cdot y = u \cdot v$ **shows** $|x| < |u| \longleftrightarrow |v| < |y|$
⟨*proof*⟩

lemma *Suc-len-nemp*: $|w| = Suc\ n \Longrightarrow w \neq \varepsilon$
⟨*proof*⟩

lemma *same-sufix-nil*: **assumes** $z \cdot u \leq p\ u$ **shows** $z = \varepsilon$
⟨*proof*⟩

lemma *count-list-gr-0-iff*: $0 < count\text{-}list\ u\ a \longleftrightarrow a \in set\ u$
⟨*proof*⟩

lemma *mid-fac-ex*: **assumes** $2 \leq |w|$
shows $w = [hd\ w] \cdot (butlast\ (tl\ w)) \cdot [last\ w]$
⟨*proof*⟩

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*⟩

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-inspection* = (
((*match* **premises in** *len*[*thin*]: $|w| \le k$ **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. $k \le (2::'a)$
(*unfold conj-ac*(*1*)[*of* $w \in$ *lists A* $|w| \le k$ **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*: *empty-iff lists.Nil bool-simps*)*?* — solve simple cases
)

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* $\{x,y,z\} \Longrightarrow P\ w$
⟨*proof*⟩

The required premise as 2 separate assumptions.

lemma $|w| \le 2 \Longrightarrow w \in$ *lists* $\{x,y,z\} \Longrightarrow P\ w$
⟨*proof*⟩

lemma $w \le p\ w \Longrightarrow |w| \le 2 \Longrightarrow w \in$ *lists* $\{a,b\} \Longrightarrow hd\ w = a \Longrightarrow w \ne \varepsilon \Longrightarrow w = [a,\ b] \vee w = [a,\ a] \vee w = [a]$
⟨*proof*⟩

lemma $w \le p\ w \Longrightarrow |w| = 2 \Longrightarrow w \in$ *lists* $\{a,b\} \Longrightarrow hd\ w = a \Longrightarrow w = [a,\ b] \vee w = [a,\ a]$
⟨*proof*⟩

lemma $w \le p\ w \Longrightarrow |w| = 2 \wedge w \in$ *lists* $\{a,b\} \Longrightarrow hd\ w = a \Longrightarrow w = [a,\ b] \vee w = [a,\ a]$

⟨*proof*⟩

lemma $w \le p\ w \Longrightarrow w \in lists\ \{a,b\} \wedge |w| = 2 \Longrightarrow hd\ w = a \Longrightarrow\ \ w = [a,\ b] \vee w = [a,\ a]$
⟨*proof*⟩

end

2.5 Prefix and prefix comparability properties

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

lemma *triv-pref*: $r \le p\ r \cdot s$
⟨*proof*⟩

lemma *triv-spref*: $s \neq \varepsilon \Longrightarrow r < p\ r \cdot s$
⟨*proof*⟩

lemma *pref-cancel*: $z \cdot u \le p\ z \cdot v \Longrightarrow u \le p\ v$
⟨*proof*⟩

lemma *pref-cancel′*: $u \le p\ v \Longrightarrow z \cdot u \le p\ z \cdot v$
⟨*proof*⟩

lemma *spref-cancel*: $z \cdot u < p\ z \cdot v \Longrightarrow u < p\ v$
⟨*proof*⟩

lemma *spref-cancel′*: $u < p\ v \Longrightarrow z \cdot u < p\ z \cdot v$
⟨*proof*⟩

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

lemma *pref-cancel-hd-conv*: $a\ \#\ u \le p\ a\ \#\ v \longleftrightarrow u \le p\ v$
⟨*proof*⟩

lemma *spref-cancel-conv*: $z \cdot x < p\ z \cdot y \longleftrightarrow x < p\ y$
⟨*proof*⟩

lemma *spref-snoc-iff* [*simp*]: $u < p\ v \cdot [a] \longleftrightarrow u \le p\ v$
⟨*proof*⟩

lemma *spref-two-lettersE*: **assumes** $p < p\ [a,b]$ **obtains** $p = \varepsilon \mid p = [a]$
⟨*proof*⟩

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

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

lemma *spref-extD*: $x \cdot y <p\ z \Longrightarrow x <p\ z$
⟨*proof*⟩

lemma *spref-ext*: $r <p\ u \Longrightarrow r <p\ u \cdot v$
⟨*proof*⟩

lemma *pref-ext-nemp*: $r \leq p\ u \Longrightarrow v \neq \varepsilon \Longrightarrow r <p\ u \cdot v$
⟨*proof*⟩

lemma *pref-take*: $p \leq p\ w \Longrightarrow take\ |p|\ w = p$
⟨*proof*⟩

lemma *pref-take-conv*: $take\ (|r|)\ w = r \longleftrightarrow r \leq p\ w$
⟨*proof*⟩

lemma *le-suf-drop*: **assumes** $i \leq j$ **shows** $drop\ j\ w \leq s\ drop\ i\ w$
⟨*proof*⟩

lemma *spref-take*: $p <p\ w \Longrightarrow take\ |p|\ w = p$
⟨*proof*⟩

lemma *pref-same-len*: $u \leq p\ v \Longrightarrow |u| = |v| \Longrightarrow u = v$
⟨*proof*⟩

lemma *pref-same-len'*: $u \cdot z \leq p\ v \cdot w \Longrightarrow |u| = |v| \Longrightarrow u = v$
⟨*proof*⟩

lemma *pref-comp-eq*: $u \bowtie v \Longrightarrow |u| = |v| \Longrightarrow u = v$
⟨*proof*⟩

lemma *ruler-eq-len*: $u \leq p\ w \Longrightarrow v \leq p\ w \Longrightarrow |u| = |v| \Longrightarrow u = v$
⟨*proof*⟩

lemma *pref-prod-eq*: $u \leq p\ v \cdot z \Longrightarrow |u| = |v| \Longrightarrow u = v$
⟨*proof*⟩

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*: $u \cdot v \leq p\ v \cdot u \longleftrightarrow u \cdot v = v \cdot u$
⟨*proof*⟩

lemma *add-nth-pref*: **assumes** $u <p\ w$ **shows** $u \cdot [w!|u|] \leq p\ w$
⟨*proof*⟩

lemma *index-pref*: $|u| \leq |w| \Longrightarrow (\forall\ i < |u|.\ \ u!i = w!i) \Longrightarrow u \leq p\ w$
⟨*proof*⟩

lemma *pref-index*: **assumes** $u \leq p\ w\ i < |u|$ **shows** $u!i = w!i$
⟨*proof*⟩

lemma *pref-drop*: $u \leq p\ v \Longrightarrow drop\ p\ u \leq p\ drop\ p\ v$
⟨*proof*⟩

2.5.1 Prefix comparability

lemma *pref-comp-sym*[*sym*]: $u \bowtie v \Longrightarrow v \bowtie u$
⟨*proof*⟩

lemma *not-pref-comp-sym*[*sym*]: $\neg\ u \bowtie v \Longrightarrow \neg\ v \bowtie u$
⟨*proof*⟩

lemma *pref-comp-sym-iff*: $u \bowtie v \longleftrightarrow v \bowtie u$
⟨*proof*⟩

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

lemma *ruler-eq*: $u \cdot x = v \cdot y \Longrightarrow u \leq p\ v \lor v \leq p\ u$
⟨*proof*⟩

lemma *ruler-eq′*: $u \cdot x = v \cdot y \Longrightarrow u \leq p\ v \lor v <p\ u$
⟨*proof*⟩

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*: $u \leq p\ v \Longrightarrow u' \leq p\ v' \Longrightarrow v \bowtie v' \Longrightarrow u \bowtie u'$
⟨*proof*⟩

lemma *ruler-pref′*: $w \leq p\ v \cdot z \Longrightarrow w \leq p\ v \lor v \leq p\ w$
⟨*proof*⟩

lemma *ruler-pref′′*: $w \leq p\ v \cdot z \Longrightarrow w \bowtie v$
⟨*proof*⟩

lemma *pref-cancel-right*: **assumes** $u \cdot z \leq p\ v \cdot z$ **shows** $u \leq p\ v$
⟨*proof*⟩

lemma *pref-prod-pref-short*: $u \leq p\ z \cdot w \Longrightarrow v \leq p\ w \Longrightarrow |u| \leq |z \cdot v| \Longrightarrow u \leq p\ z \cdot v$
⟨*proof*⟩

lemma *pref-prod-pref*: $u \leq p\ z \cdot w \Longrightarrow u \leq p\ w \Longrightarrow\ u \leq p\ z \cdot u$
⟨*proof*⟩

lemma *pref-prod-pref′*: **assumes** $u \leq p\ z{\cdot}u{\cdot}w$ **shows** $u \leq p\ z{\cdot}u$
⟨*proof*⟩

lemma *pref-prod-long*: $u \leq p\ v \cdot w \Longrightarrow |v| \leq |u| \Longrightarrow v \leq p\ u$
⟨*proof*⟩

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

lemma *pref-prod-long-less*: **assumes** $u \leq p\ v \cdot w$ **and** $|v| < |u|$ **shows** $v <p\ u$
⟨*proof*⟩

lemma *pref-keeps-per-root*: $u \leq p\ r \cdot u \Longrightarrow v \leq p\ u \Longrightarrow v \leq p\ r \cdot v$
⟨*proof*⟩

lemma *pref-keeps-per-root′*: $u <p\ r \cdot u \Longrightarrow v \leq p\ u \Longrightarrow v <p\ r \cdot v$
⟨*proof*⟩

lemma *per-root-pref-sing*: $w <p\ r \cdot w \Longrightarrow u \cdot [a] \leq p\ w \Longrightarrow u \cdot [a] \leq p\ r \cdot u$
⟨*proof*⟩

lemma *pref-prolong*: $w \leq p\ z \cdot r \Longrightarrow r \leq p\ s \Longrightarrow w \leq p\ z \cdot s$
⟨*proof*⟩

lemma *spref--pref-prolong*: $w <p\ z \cdot r \Longrightarrow r \leq p\ s \Longrightarrow w <p\ z \cdot s$
⟨*proof*⟩

lemma *pref-spref-prolong*: $w \leq p\ z \cdot r \Longrightarrow r <p\ s \Longrightarrow w <p\ z \cdot s$
⟨*proof*⟩

lemma *spref-spref-prolong*: $w <p\ z \cdot r \Longrightarrow r <p\ s \Longrightarrow w <p\ z \cdot s$
⟨*proof*⟩

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

lemma *pref-prolong′*: $u \leq p\ w \cdot z \Longrightarrow v \cdot u \leq p\ z \Longrightarrow u \leq p\ w \cdot v \cdot u$
⟨*proof*⟩

lemma *pref-prolong-per-root*: $u \leq p\ r \cdot s \Longrightarrow s \leq p\ r \cdot s \Longrightarrow u \leq p\ r \cdot u$
⟨*proof*⟩

thm *pref-compE*
lemma *pref-prolong-comp*: $u \leq p\ w \cdot z \Longrightarrow v \cdot u \bowtie z \Longrightarrow u \leq p\ w \cdot v \cdot u$
⟨*proof*⟩

lemma *pref-prod-le*[*intro*]: $u \leq p\ v \cdot w \Longrightarrow |u| \leq |v| \Longrightarrow u \leq p\ v$

⟨*proof*⟩

lemma *prod-pref-prod-le*: $u \cdot v \leq p\ x \cdot y \Longrightarrow |u| \leq |x| \Longrightarrow u \leq p\ x$
⟨*proof*⟩

lemma *pref-prod-less*: $u \leq p\ v \cdot w \Longrightarrow |u| < |v| \Longrightarrow u <p\ v$
⟨*proof*⟩

lemma *eq-le-pref*[*elim*]: $x \cdot y = u \cdot v \Longrightarrow |x| \leq |u| \Longrightarrow x \leq p\ u$
⟨*proof*⟩

lemma *eq-less-pref*: $x \cdot y = u \cdot v \Longrightarrow |x| < |u| \Longrightarrow x <p\ u$
⟨*proof*⟩

lemma *eq-less-suf*: **assumes** $x \cdot y = u \cdot v$ **shows** $|x| < |u| \Longrightarrow v <s\ y$
⟨*proof*⟩

lemma *pref-prod-cancel*: **assumes** $u \leq p\ p \cdot w \cdot q$ **and** $|p| \leq |u|$ **and** $|u| \leq |p \cdot w|$
obtains r **where** $p \cdot r = u$ **and** $r \leq p\ w$
⟨*proof*⟩

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*⟩

lemma *non-comp-parallel*: $\neg\ u \bowtie v \longleftrightarrow u \parallel v$
⟨*proof*⟩

lemma *comp-refl*: $u \bowtie u$
⟨*proof*⟩

lemma *incomp-cancel*: $\neg\ p \cdot u \bowtie p \cdot v \Longrightarrow \neg\ u \bowtie v$
⟨*proof*⟩

lemma *comm-ruler*: $r \cdot s \leq p\ w1 \Longrightarrow s \cdot r \leq p\ w2 \Longrightarrow w1 \bowtie w2 \Longrightarrow r \cdot s = s \cdot r$
⟨*proof*⟩

lemma *comm-comp-eq*: $r \cdot s \bowtie s \cdot r \Longrightarrow r \cdot s = s \cdot r$
⟨*proof*⟩

lemma *pref-share-take*: $u \leq p\ v \Longrightarrow q \leq |u| \Longrightarrow take\ q\ u = take\ q\ v$
⟨*proof*⟩

lemma *pref-prod-longer*: $u \leq p\ z \cdot w \Longrightarrow v \leq p\ w \Longrightarrow |z \cdot v| \leq |u| \Longrightarrow z \cdot v \leq p\ u$
⟨*proof*⟩

lemma *pref-comp-not-pref*: $u \bowtie v \Longrightarrow \neg\ v \leq p\ u \Longrightarrow u <p\ v$
⟨*proof*⟩

lemma *pref-comp-not-spref*: $u \bowtie v \Longrightarrow \neg\ u <p\ v \Longrightarrow v \leq p\ u$
⟨*proof*⟩

lemma *hd-prod*: $u \neq \varepsilon \Longrightarrow (u \cdot v)!0 = u!0$
⟨*proof*⟩

lemma *distinct-first*: **assumes** $w \neq \varepsilon$ $z \neq \varepsilon$ $w!0 \neq z!0$ **shows** $w \cdot w' \neq z \cdot z'$
⟨*proof*⟩

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

lemma *last-no-split'*: $u <p\ w \Longrightarrow w \leq p\ u \cdot [a] \Longrightarrow w = u \cdot [a]$
⟨*proof*⟩

lemma *comp-shorter*: $v \bowtie w \Longrightarrow |v| \leq |w| \Longrightarrow v \leq p\ w$
⟨*proof*⟩

lemma *comp-shorter-conv*: $|u| \leq |v| \Longrightarrow u \bowtie v \longleftrightarrow u \leq p\ v$
⟨*proof*⟩

lemma *pref-comp-len-trans*: $w \leq p\ v \Longrightarrow u \bowtie v \Longrightarrow |w| \leq |u| \Longrightarrow w \leq p\ u$
⟨*proof*⟩

lemma *comp-cancel*: $z \cdot w1 \bowtie z \cdot w2 \longleftrightarrow w1 \bowtie w2$
⟨*proof*⟩

lemma *emp-pref*: $\varepsilon \leq p\ u$
⟨*proof*⟩

lemma *emp-spref*: $u \neq \varepsilon \Longrightarrow \varepsilon <p\ u$
⟨*proof*⟩

lemma *long-pref*: $u \leq p\ v \Longrightarrow |v| \leq |u| \Longrightarrow u = v$
⟨*proof*⟩

lemma *not-comp-ext*: $\neg\ w1 \bowtie w2 \Longrightarrow \neg\ w1 \cdot z \bowtie w2 \cdot z'$
⟨*proof*⟩

lemma *mismatch-incopm*: $|u| = |v| \Longrightarrow x \neq y \Longrightarrow \neg\ u \cdot [x] \bowtie v \cdot [y]$
⟨*proof*⟩

lemma *comp-prefs-comp*: $u \cdot z \bowtie v \cdot w \Longrightarrow u \bowtie v$
⟨*proof*⟩

lemma *comp-hd-eq*: $u \bowtie v \Longrightarrow u \neq \varepsilon \Longrightarrow v \neq \varepsilon \Longrightarrow hd\ u = hd\ v$
⟨*proof*⟩

lemma *pref-hd-eq'*: $p \leq p\ u \Longrightarrow p \leq p\ v \Longrightarrow p \neq \varepsilon \Longrightarrow hd\ u = hd\ v$
⟨*proof*⟩

lemma *pref-hd-eq*: $u \leq p\ v \Longrightarrow u \neq \varepsilon \Longrightarrow\ hd\ u = hd\ v$
⟨*proof*⟩

lemma *sing-pref-hd*: $[a] \leq p\ v \Longrightarrow hd\ v = a$
⟨*proof*⟩

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

lemma *comp-hd-eq'*: $u \cdot r \bowtie v \cdot s \Longrightarrow u \neq \varepsilon \Longrightarrow v \neq \varepsilon \Longrightarrow hd\ u = hd\ v$
⟨*proof*⟩

2.5.2 Minimal and maximal prefix with a given property

lemma *le-take-pref*: **assumes** $k \leq n$ **shows** $take\ k\ ws \leq p\ take\ n\ ws$
⟨*proof*⟩

lemma *min-pref*: **assumes** $u \leq p\ w$ **and** $P\ u$
obtains v **where** $v \leq p\ w$ **and** $P\ v$ **and** $\bigwedge y.\ y \leq p\ w \Longrightarrow P\ y \Longrightarrow v \leq p\ y$
⟨*proof*⟩

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*⟩

lemma *max-pref*: **assumes** $u \leq p\ w$ **and** $P\ u$
obtains v **where** $v \leq p\ w$ **and** $P\ v$ **and** $\bigwedge y.\ y \leq p\ w \Longrightarrow P\ y \Longrightarrow y \leq p\ v$
⟨*proof*⟩

2.6 Suffix and suffix comparability properties

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

lemma *triv-suf*: $u \leq s\ v \cdot u$
⟨*proof*⟩

lemma *emp-ssuf*: $u \neq \varepsilon \Longrightarrow \varepsilon < s\ u$
⟨*proof*⟩

lemma *suf-cancel*: $u{\cdot}v \leq s\ w{\cdot}v \Longrightarrow u \leq s\ w$
⟨*proof*⟩

lemma *suf-cancel'*: $u \leq s\ w \Longrightarrow u{\cdot}v \leq s\ w{\cdot}v$
⟨*proof*⟩

lemma *ssuf-cancel-conv*: $x \cdot z < s\ y \cdot z \longleftrightarrow x < s\ y$
⟨*proof*⟩

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*: $u \leq p\ v \longleftrightarrow rev\ u \leq s\ rev\ v$
⟨*proof*⟩

lemma *spref-rev-suf-iff*: $s <p\ w \longleftrightarrow rev\ s <s\ rev\ w$
⟨*proof*⟩

lemma *nsuf-rev-pref-iff*: $s \leq ns\ w \longleftrightarrow rev\ s \leq np\ rev\ w$
⟨*proof*⟩

lemma *npref-rev-suf-iff*: $s \leq np\ w \longleftrightarrow rev\ s \leq ns\ rev\ w$
⟨*proof*⟩

lemmas [*reversal-rule*] =
suf-rev-pref-iff[*symmetric*]
pref-rev-suf-iff[*symmetric*]
nsuf-rev-pref-iff[*symmetric*]
npref-rev-suf-iff[*symmetric*]
ssuf-rev-pref-iff[*symmetric*]
spref-rev-suf-iff[*symmetric*]

lemmas *sufE* = *prefixE*[*reversed*] **and**
prefE = *prefixE* **and**
ssuf-exE = *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*]: $u <s\ a\ \#\ v \longleftrightarrow u \leq s\ v$
⟨*proof*⟩

lemma *ssuf-induct* [*case-names ssuf*]:
assumes $\bigwedge u.\ (\bigwedge v.\ v <s\ u \Longrightarrow P\ v) \Longrightarrow P\ u$
shows $P\ u$
⟨*proof*⟩

2.6.1 Suffix comparability

lemma *eq-le-suf*[*elim*]: **assumes** $x \cdot y = u \cdot v\ |x| \leq |u|$ **shows** $v \leq s\ y$
⟨*proof*⟩

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** $x \leq s\ u$
⟨*proof*⟩

lemma *pref-comp-rev-suf-comp*[*reversal-rule*]: $(rev\ w) \bowtie_s (rev\ v) \longleftrightarrow w \bowtie v$
⟨*proof*⟩

lemma *suf-comp-rev-pref-comp*[*reversal-rule*]: $(rev\ w) \bowtie (rev\ v) \longleftrightarrow w \bowtie_s v$
⟨*proof*⟩

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-less = *pref-prod-less*[*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-prolong-comp = *pref-prolong-comp*[*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*]

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*: $u \bowtie_s v \longleftrightarrow u \le s\ v \lor v \le s\ u$
⟨*proof*⟩

lemma *comm-comp-eq-conv*: $r \cdot s \bowtie s \cdot r \longleftrightarrow r \cdot s = s \cdot r$
⟨*proof*⟩

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

lemma *suf-comp-last-eq*: **assumes** $u \bowtie_s v\ u \neq \varepsilon\ v \neq \varepsilon$
shows *last* u = *last* v
⟨*proof*⟩

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
⟨*proof*⟩

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* (‹- $^{-1}$ · - ›)

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 $u^{<-1}v^{<-1}w = (u^{<-1}v)^{<-1}w$
⟨*proof*⟩

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

⟨*proof*⟩

lemma $u^{-1>}v^{<-1}w = u^{-1>}(v^{<-1}w)$
⟨*proof*⟩

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

lemma *rq-rev-lq*[*reversal-rule*]: $(rev\ v)^{<-1}(rev\ u) = rev\ (u^{-1>}v)$
⟨*proof*⟩

lemma *lq-rev-rq*[*reversal-rule*]: $(rev\ v)^{-1>}rev\ u = rev\ (u^{<-1}v)$
⟨*proof*⟩

2.7.1 Left Quotient

lemma *lqI*: $u \cdot z = v \Longrightarrow u^{-1>}v = z$
⟨*proof*⟩

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

lemma *lq-triv′*[*simp*]: $u \cdot u^{-1>}(u \cdot z) = u \cdot z$
⟨*proof*⟩

lemma *append-lq*: **assumes** $u \cdot v \le p\ w$ **shows** $(u \cdot v)^{-1>}w = v^{-1>}(u^{-1>}w)$
⟨*proof*⟩

lemma *lq-self*[*simp*]: $u^{-1>}u = \varepsilon$
⟨*proof*⟩

lemma *lq-emp*[*simp*]: $\varepsilon^{-1>}u = u$
⟨*proof*⟩

lemma *lq-pref*[*simp*]: $u \le p\ v \Longrightarrow u \cdot (u^{-1>}v) = v$
⟨*proof*⟩

lemma *lq-pref-conv*: $|u| \le |v| \Longrightarrow u \le p\ v \longleftrightarrow u \cdot u^{-1>}v = v$
⟨*proof*⟩

lemma *lq-len*: $|u^{-1>}v| = |v| - |u|$
⟨*proof*⟩

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

lemma *lq-pref-cancel*: $u \le p\ v \Longrightarrow v \cdot r = u \cdot s \Longrightarrow (u^{-1>}v) \cdot r = s$
⟨*proof*⟩

lemma *lq-the*: **assumes** $u \le p\ v$

shows (*THE* $z.\ u \cdot z = v$) = $(u^{-1>}v)$
⟨*proof*⟩

lemma *lq-same-len*: $|u| = |v| \Longrightarrow u^{-1>}v = \varepsilon$
⟨*proof*⟩

lemma *lq-assoc*: $|u| \le |v| \Longrightarrow (u^{-1>}v)^{-1>}w = v^{-1>}(u \cdot w)$
⟨*proof*⟩

lemma *lq-assoc'*: $(u \cdot w)^{-1>}v = w^{-1>}(u^{-1>}v)$
⟨*proof*⟩

lemma *lq-reassoc*: $u \le p\ v \Longrightarrow (u^{-1>}v){\cdot}w = u^{-1>}(v{\cdot}w)$
⟨*proof*⟩

lemma *lq-trans*: $u \le p\ v \Longrightarrow v \le p\ w \Longrightarrow (u^{-1>}v) \cdot (v^{-1>}w) = u^{-1>}w$
⟨*proof*⟩

lemma *lq-rq-reassoc-suf*: **assumes** $u \le p\ z$ $u \le s\ w$ **shows** $w{\cdot}u^{-1>}z = w^{<-1}u \cdot z$
⟨*proof*⟩

lemma *lq-ne*: $p \le p\ u{\cdot}p \Longrightarrow u \ne \varepsilon \Longrightarrow p^{-1>}(u{\cdot}p) \ne \varepsilon$
⟨*proof*⟩

lemma *lq-spref*: $u < p\ v \Longrightarrow u^{-1>}v \ne \varepsilon$
⟨*proof*⟩

lemma *lq-suf-suf*: $r \le p\ s \Longrightarrow (r^{-1>}s) \le s\ s$
⟨*proof*⟩

lemma *lq-short-len*: $r \le p\ s \Longrightarrow |r| + |r^{-1>}s| = |s|$
⟨*proof*⟩

lemma *pref-lq*: $v \le p\ w \Longrightarrow u^{-1>}v \le p\ u^{-1>}w$
⟨*proof*⟩

lemma *spref-lq*: $u \le p\ v \Longrightarrow v < p\ w \Longrightarrow u^{-1>}v < p\ u^{-1>}w$
⟨*proof*⟩

lemma *pref-gcd-lq*: **assumes** $u \le p\ v$ **shows** $(gcd\ |u|\ |u^{-1>}v|) = gcd\ |u|\ |v|$
⟨*proof*⟩

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

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

lemma *hd-lq-conv-nth*: **assumes** $u <p\ v$ **shows** $hd(u^{-1>}v) = v!|u|$
⟨*proof*⟩

lemma *concat-morph-lq*: $us \leq p\ ws \Longrightarrow concat\ (us^{-1>}ws) = (concat\ us)^{-1>}(concat\ ws)$
⟨*proof*⟩

lemma *pref-cancel-lq*: **assumes** $u \leq p\ x \cdot y$
shows $x^{-1>}u \leq p\ y$
⟨*proof*⟩

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*⟩

lemma *pref-cancel-lq-ext'*: **assumes** $u \cdot v \leq p\ x \cdot y$ **and** $|u| \leq |x|$ **shows** $v \leq p\ u^{-1>}x \cdot y$
⟨*proof*⟩

lemma *empty-lq-eq*: $r \leq p\ z \Longrightarrow r^{-1>}z = \varepsilon \Longrightarrow r = z$
⟨*proof*⟩

lemma *le-if-then-lq*: $|u| \leq |v| \Longrightarrow (if\ |v| \leq |u|\ then\ v^{-1>}u\ \ else\ u^{-1>}v) = u^{-1>}v$
⟨*proof*⟩

lemma *append-comp-lq*: $u \cdot v \bowtie w \Longrightarrow v \bowtie u^{-1>}w$
⟨*proof*⟩

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 = *lq-spref*[*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-suf-suf = *lq-suf-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*: $p \leq p\ w \Longrightarrow w^{<-1}(p^{-1>}w) = p$
⟨*proof*⟩

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

lemma *rev-lq′*: $r \leq p\ s \Longrightarrow rev\ (r^{-1>}s) = (rev\ s)^{<-1}(rev\ r)$
⟨*proof*⟩

lemma *pref-rq-suf-lq*: $s \leq s\ u \Longrightarrow r \leq p\ (u^{<-1}s) \Longrightarrow s \leq s\ (r^{-1>}u)$
⟨*proof*⟩

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

lemma $w{\cdot}s = v \Longrightarrow v^{<-1}s = w$ ⟨*proof*⟩

lemma *lq-rq-assoc*: $s \leq s\ u \Longrightarrow r \leq p\ (u^{<-1}s) \Longrightarrow (r^{-1>}u)^{<-1}s = r^{-1>}(u^{<-1}s)$
⟨*proof*⟩

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

lemma *lq-prod*: $u \leq p\ v{\cdot}u \Longrightarrow u \leq p\ w \Longrightarrow u^{-1>}(v{\cdot}u){\cdot}u^{-1>}w = u^{-1>}(v{\cdot}w)$
⟨*proof*⟩

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*⟩

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*: $x \cdot y = u \cdot v \Longrightarrow |x| \leq |u| \Longrightarrow \exists\ t.\ x \cdot t = u \wedge t \cdot v = y$
⟨*proof*⟩

lemma *eqdE*: **assumes** $x \cdot y = u \cdot v$ **and** $|x| \leq |u|$
obtains t **where** $x \cdot t = u$ **and** $t \cdot v = y$
⟨*proof*⟩

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** $t \neq \varepsilon$
⟨*proof*⟩

lemma *eqdE'*: **assumes** $x \cdot y = u \cdot v$ **and** $|v| \leq |y|$
obtains t **where** $x \cdot t = u$ **and** $t \cdot v = y$
⟨*proof*⟩

thm *long-pref*

lemma *eqd-pref-suf-iff*: **assumes** $x \cdot y = u \cdot v$ **shows** $x \leq p\ u \longleftrightarrow v \leq s\ y$
⟨*proof*⟩

lemma *eqd-spref-ssuf-iff*: **assumes** $x \cdot y = u \cdot v$ **shows** $x <p\ u \longleftrightarrow v <s\ y$
⟨*proof*⟩

lemma *eqd-pref*: $x \cdot y = u \cdot v \Longrightarrow |x| \leq |u| \Longrightarrow x \cdot (x^{-1>}u) = u \wedge (x^{-1>}u) \cdot v = y$
⟨*proof*⟩

lemma *eqd-pref1*: $x \cdot y = u \cdot v \Longrightarrow |x| \leq |u| \Longrightarrow x \cdot (x^{-1>}u) = u$
⟨*proof*⟩

lemma *eqd-pref2*: $x \cdot y = u \cdot v \Longrightarrow |x| \leq |u| \Longrightarrow (x^{-1>}u) \cdot v = y$
⟨*proof*⟩

lemma *eqd-eq*: **assumes** $x \cdot y = u \cdot v$ $|x| = |u|$ **shows** $x = u$ $y = v$
⟨*proof*⟩

lemma *eqd-eq-suf*: $x \cdot y = u \cdot v \Longrightarrow |y| = |v| \Longrightarrow x = u \wedge y = v$
⟨*proof*⟩

lemma *eqd-comp*: **assumes** $x \cdot y = u \cdot v$ **shows** $x \bowtie u$
⟨*proof*⟩
lemma *eqd-suf1*: $x \cdot y = u \cdot v \Longrightarrow |x| \leq |u| \Longrightarrow (y^{<-1}v) \cdot v = y$
⟨*proof*⟩
lemma *eqd-suf2*: **assumes** $x \cdot y = u \cdot v$ $|x| \leq |u|$ **shows** $x \cdot (y^{<-1}v) = u$
⟨*proof*⟩
lemma *eqd-suf*: **assumes** $x \cdot y = u \cdot v$ **and** $|x| \leq |u|$
shows $(y^{<-1}v) \cdot v = y \wedge x \cdot (y^{<-1}v) = u$

⟨*proof*⟩

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 $u' \cdot v' = x' \cdot y'$
⟨*proof*⟩

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 $u' \cdot v' = x' \cdot y'$
⟨*proof*⟩

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*: $u \leq p\ v \Longrightarrow u \leq p\ (u \cdot w) \wedge_p (v \cdot z)$
⟨*proof*⟩

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*⟩

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

lemma *lcp-take-eq*: $take\ (|u \wedge_p v|)\ u = take\ (|u \wedge_p v|)\ v$
⟨*proof*⟩

lemma *lcp-pref-conv*: $u \wedge_p v = u \longleftrightarrow u \leq p\ v$
⟨*proof*⟩

lemma *lcp-pref-conv'*: $u \wedge_p v = v \longleftrightarrow v \leq p\ u$
⟨*proof*⟩

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*: $r \cdot s \wedge_p s \cdot r \leq p\ r \cdot (r \cdot s \wedge_p s \cdot r)$
⟨*proof*⟩

lemma *lcp-per-root'*: $r \cdot s \wedge_p s \cdot r \leq p\ s \cdot (r \cdot s \wedge_p s \cdot r)$
⟨*proof*⟩

lemma *pref-lcp-pref*: $w \leq p\ u \wedge_p v \Longrightarrow w \leq p\ u$
⟨*proof*⟩

lemma *pref-lcp-pref'*: $w \leq p\ u \wedge_p v \Longrightarrow w \leq p\ v$
⟨*proof*⟩

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

lemma *lcp-eq-len*: $|u| = |u \wedge_p v| \Longrightarrow u = u \wedge_p v$
⟨*proof*⟩

lemma *lcp-len*: $|u| \leq |u \wedge_p v| \Longrightarrow u \leq p\ v$
⟨*proof*⟩

lemma *lcp-len'*: $\neg\ u \leq p\ v \Longrightarrow |u \wedge_p v| < |u|$
⟨*proof*⟩

lemma *incomp-lcp-len*: $\neg\ u \bowtie v \Longrightarrow |u \wedge_p v| < min\ |u|\ |v|$
⟨*proof*⟩

lemma *lcp-ext-right-conv*: $\neg\ r \bowtie r' \Longrightarrow (r \cdot u) \wedge_p (r' \cdot v) = r \wedge_p r'$
⟨*proof*⟩

lemma *lcp-ext-right* [*case-names comp non-comp*]: **obtains** $r \bowtie r' \mid (r \cdot u) \wedge_p (r' \cdot v) = r \wedge_p r'$
⟨*proof*⟩

lemma *lcp-same-len*: $|u| = |v| \Longrightarrow u \neq v \Longrightarrow u \cdot w \wedge_p v \cdot w' = u \wedge_p v$
⟨*proof*⟩

lemma *lcp-mismatch*: $|u \wedge_p v| < |u| \Longrightarrow |u \wedge_p v| < |v| \Longrightarrow u!\ |u \wedge_p v| \neq v!\ |u \wedge_p v|$
⟨*proof*⟩

lemma *lcp-mismatch'*: $\neg\ u \bowtie v \Longrightarrow u!\ |u \wedge_p v| \neq v!\ |u \wedge_p v|$
⟨*proof*⟩

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*⟩

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*⟩

lemma *lcp-ext-left*: $(z \cdot u) \wedge_p (z \cdot v) = z \cdot (u \wedge_p v)$
⟨*proof*⟩

lemma *lcp-first-letters*: $u!0 \neq v!0 \Longrightarrow u \wedge_p v = \varepsilon$
⟨*proof*⟩

lemma *lcp-first-mismatch*: $a \neq b \Longrightarrow w \cdot [a] \cdot u \wedge_p w \cdot [b] \cdot v = w$
⟨*proof*⟩

lemma *lcp-first-mismatch'*: $a \neq b \Longrightarrow u \cdot [a] \wedge_p u \cdot [b] = u$
⟨*proof*⟩

lemma *lcp-mismatch-eq-len*: **assumes** $|u| = |v|$ $x \neq y$ **shows** $u \cdot [x] \wedge_p v \cdot [y] = u \wedge_p v$
⟨*proof*⟩

lemma *lcp-first-mismatch-pref*: **assumes** $p \cdot [a] \leq p\ u$ **and** $p \cdot [b] \leq p\ v$ **and** $a \neq b$
shows $u \wedge_p v = p$
⟨*proof*⟩

lemma *lcp-append-monotone*: $u \wedge_p x \leq p\ (u \cdot v) \wedge_p (x \cdot y)$
⟨*proof*⟩

lemma *lcp-distinct-hd*: $hd\ u \neq hd\ v \Longrightarrow u \wedge_p v = \varepsilon$
⟨*proof*⟩

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*⟩

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*⟩

lemma *lcp-prefs*: $|u \cdot w \wedge_p v \cdot w'| < |u| \Longrightarrow |u \cdot w \wedge_p v \cdot w'| < |v| \Longrightarrow u \wedge_p v = u \cdot w \wedge_p v \cdot w'$
⟨*proof*⟩

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*⟩

lemma *long-lcp-same*: **assumes** $\neg$ ($u \wedge_p v \leq p\ w$) **shows** $u \wedge_p w = v \wedge_p w$
⟨*proof*⟩

lemma *long-lcp-sameE*: **obtains** $u \wedge_p v \leq p\ w$ | $u \wedge_p w = v \wedge_p w$
⟨*proof*⟩

lemma *ruler-spref-lcp*: **assumes** $u \wedge_p w <p\ v \wedge_p w$
shows $u \wedge_p v = u \wedge_p w$
⟨*proof*⟩

2.9.1 Longest common prefix and prefix comparability

find-theorems *name*:*ruler*
lemma *lexord-cancel-right*: $(u \cdot z,\ v \cdot w) \in$ *lexord* $r \Longrightarrow \neg\ u \bowtie v \Longrightarrow (u,v) \in$ *lexord* r
⟨*proof*⟩

lemma *lcp-rulersE*: **assumes** $r \leq p\ s$ **and** $r' \leq p\ s'$ **obtains** $r \bowtie r'$ | $s \wedge_p s' = r \wedge_p r'$
⟨*proof*⟩

lemma *lcp-rulers*: $r \leq p\ s \Longrightarrow r' \leq p\ s' \Longrightarrow (r \bowtie r' \vee\ s \wedge_p s' = r \wedge_p r')$
⟨*proof*⟩

lemma *lcp-rulers'*: $w \leq p\ r \Longrightarrow w' \leq p\ s \Longrightarrow \neg\ w \bowtie w' \Longrightarrow (r \wedge_p s) = w \wedge_p w'$
⟨*proof*⟩

lemma *lcp-ruler*: $r \bowtie w1 \Longrightarrow r \bowtie w2 \Longrightarrow \neg\ w1 \bowtie w2 \Longrightarrow r \leq p\ w1 \wedge_p w2$
⟨*proof*⟩

lemma *comp-monotone*: $w \bowtie r \Longrightarrow u \leq p\ w \Longrightarrow u \bowtie r$
⟨*proof*⟩

lemma *comp-monotone'*: $w \bowtie r \Longrightarrow w \wedge_p w' \bowtie r$
⟨*proof*⟩

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*⟩

lemma *double-ruler*: **assumes** $w \bowtie r$ **and** $w' \bowtie r'$ **and** $\neg\ r \bowtie r'$
shows $w \wedge_p w' =$ *take* (*min* $|w|$ $|w'|$) $(r \wedge_p r')$
⟨*proof*⟩

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 $w \leq p\ u \wedge w \leq p\ v$
⟨*proof*⟩

lemma *comp-per-partes*:
shows $(u \bowtie w \wedge v \bowtie u^{-1>} w) \longleftrightarrow u \cdot v \bowtie w$
⟨*proof*⟩

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*)
⟨*proof*⟩

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

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*))
⟨*proof*⟩
termination
⟨*proof*⟩

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

lemma *mismatch-pair-lcp*: *mismatch-pair u v* = ($u!|u\wedge_p v|, v!|u\wedge_p v|$)
⟨*proof*⟩

For incomparable words the pair is out of diagonal.

lemma *incomp-neq*: ¬ *u* ⋈ *v* ⟹ (*mismatch-pair u v*) ∉ *Id*
⟨*proof*⟩

lemma *mismatch-ext-left*: ¬ *u* ⋈ *v* ⟹ *mismatch-pair u v* = *mismatch-pair* (*p*·*u*) (*p*·*v*)
⟨*proof*⟩

lemma *mismatch-ext-right*: **assumes** ¬ *u* ⋈ *v*
shows *mismatch-pair u v* = *mismatch-pair* (*u*·*z*) (*v*·*w*)
⟨*proof*⟩

lemma *mismatchI*: ¬ *u* ⋈ *v* ⟹ *i* < *min* |*u*| |*v*| ⟹ *take i u* = *take i v* ⟹ *u*!*i* ≠ *v*!*i*

$\Longrightarrow$ *mismatch-pair* u v = (u!i,v!i)
⟨*proof*⟩

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

lemma *mismatch-lexord*: **assumes** ¬ $u \bowtie v$ **and** *mismatch-pair* u v ∈ r
shows (u,v) ∈ *lexord* r
⟨*proof*⟩

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

lemma *lexord-mismatch*: **assumes** ¬ $u \bowtie v$ **and** *irrefl* r
shows *mismatch-pair* u v ∈ r ⟷ (u,v) ∈ *lexord* r
⟨*proof*⟩

2.11 Factor properties

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

lemma *rev-fac*[*reversal-rule*]: *rev* u ≤*f* *rev* v ⟷ u ≤*f* v
⟨*proof*⟩

lemma *fac-pref*: u ≤*f* v ≡ ∃ p. p · u ≤*p* v
⟨*proof*⟩

lemma *fac-pref-suf*: u ≤*f* v ⟹ ∃ p. p ≤*p* v ∧ u ≤*s* p
⟨*proof*⟩

lemma *pref-suf-fac*: r ≤*p* v ⟹ u ≤*s* r ⟹ u ≤*f* v
⟨*proof*⟩

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
⟨*proof*⟩

lemma *fac-triv*: $p \cdot x \cdot q = x$ ⟹ $p = \varepsilon$
⟨*proof*⟩

lemma *fac-triv'*: $p \cdot x \cdot q = x$ ⟹ $q = \varepsilon$
⟨*proof*⟩

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

lemma *fac-ConsE*: **assumes** $u \leq f\ (a\#v)$
obtains $u \leq p\ (a\#v) \mid u \leq f\ v$
⟨*proof*⟩

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

lemma *fac-elim-suf*: **assumes** $f \leq f\ m{\cdot}s\ \neg\ f \leq f\ s$
shows $f \leq f\ m{\cdot}(take\ (|f|-1)\ s)$
⟨*proof*⟩

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

lemma *fac-elim*: **assumes** $f \leq f\ p{\cdot}m{\cdot}s$ **and** $\neg\ f \leq f\ p$ **and** $\neg\ f \leq f\ s$
shows $f \leq f\ (drop\ (|p| - (|f| - 1))\ p) \cdot m \cdot (take\ (|f|-1)\ s)$
⟨*proof*⟩

lemma *fac-ext-pref*: $u \leq f\ w \Longrightarrow u \leq f\ p \cdot w$
⟨*proof*⟩

lemma *fac-ext-suf*: $u \leq f\ w \Longrightarrow u \leq f\ w \cdot s$
⟨*proof*⟩

lemma *fac-ext*: $u \leq f\ w \Longrightarrow u \leq f\ p \cdot w \cdot s$
⟨*proof*⟩

lemma *fac-ext-hd*:$u \leq f\ w \Longrightarrow u \leq f\ a\#w$
⟨*proof*⟩

lemma *card-switch-fac*: **assumes** $2 \leq card\ (set\ ws)$
obtains $c\ d$ **where** $c \neq d$ **and** $[c,d] \leq f\ ws$
⟨*proof*⟩

lemma *fac-overlap-len*: **assumes** $u \leq f\ x \cdot y \cdot z$ **and** $|u| \leq |y|$
shows $u \leq f\ x \cdot y \lor u \leq f\ y \cdot z$
⟨*proof*⟩

2.12 Power and its properties

Word powers are often investigated in Combinatorics on Words. We thus interpret words as *monoid-mult* and adopt a notation for the word power.

primrec *list-power* :: *'a list* ⇒ *nat* ⇒ *'a list* (**infixr** ‹@› *80*)
where
pow-0: $u\ ^{@}\ 0 = \varepsilon$
| *pow-Suc*: $u\ ^{@}\ Suc\ n = u \cdot u\ ^{@}\ n$

term *power.power*

context
begin

interpretation *monoid-mult* ε *append*
  rewrites *power u n* = $u^{@}n$
⟨*proof*⟩

lemma *emp-pow-emp*[*simp*]: $r = \varepsilon \Longrightarrow r^{@}n = \varepsilon$
  ⟨*proof*⟩

lemma *pow-pos*:$0 < k \Longrightarrow a^{@}k = a \cdot a^{@}(k-1)$
  ⟨*proof*⟩

lemma *pow-pos′*:$0 < k \Longrightarrow a^{@}k = a^{@}(k-1) \cdot a$
  ⟨*proof*⟩

lemma *pow-diff*: $k < n \Longrightarrow a^{@}(n - k) = a \cdot a^{@}(n-k-1)$
  ⟨*proof*⟩

lemma *pow-diff′*: $k < n \Longrightarrow a^{@}(n - k) = a^{@}(n-k-1) \cdot a$
  ⟨*proof*⟩

lemmas *pow-zero* = *power.power-0* **and**
  pow-one = *power-Suc0-right* **and**
  pow-1 = *power-one-right* **and**
  emp-pow[*emp-simps*] = *power-one* **and**
  pow-two[*simp*] = *power2-eq-square* **and**
  pow-Suc = *power-Suc* **and**
  pow-Suc′ = *power-Suc2* **and**
  pow-comm = *power-commutes* **and**
  add-exps = *power-add* **and**
  pow-eq-if-list = *power-eq-if* **and**
  pow-mult = *power-mult* **and**
   comm-add-exp = *power-commuting-commutes*

lemma *pow-rev-emp-conv*[*reversal-rule*]: *power.power* (*rev* ε) $(\cdot) = (^{@})$
      ⟨*proof*⟩

lemma *pow-rev-map-rev-emp-conv* [*reversal-rule*]: *power.power* (*rev* (*map rev* ε))
$(\cdot) = (^{@})$
    ⟨*proof*⟩

end

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

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

— more power properties

lemma *sing-Cons-to-pow*: $[a,\ a] = [a]\ ^{@}\ Suc\ (Suc\ 0)\ \ a\ \#\ [a]\ ^{@}\ k = [a]\ ^{@}\ Suc\ k$
⟨*proof*⟩

lemma *zero-exp*: $n = 0 \Longrightarrow r^{@}n = \varepsilon$
⟨*proof*⟩

lemma *nemp-pow*: $t^{@}m \neq \varepsilon \Longrightarrow 0 < m$
⟨*proof*⟩

lemma *pow-nemp-pos*[*intro*]: **assumes** $u = t^{@}m\ \ u \neq \varepsilon$ **shows** $0 < m$
⟨*proof*⟩

lemma *nemp-exp-pos*[*intro*]: $w \neq \varepsilon \Longrightarrow r^{@}k = w \Longrightarrow 0 < k$
⟨*proof*⟩

lemma *nemp-exp-pos'*[*intro*]: $w \neq \varepsilon \Longrightarrow w = r^{@}k \Longrightarrow 0 < k$
⟨*proof*⟩

lemma *nemp-pow-nemp*[*intro*]: $t^{@}m \neq \varepsilon \Longrightarrow t \neq \varepsilon$
⟨*proof*⟩

lemma *sing-pow-nth*:$i < m \Longrightarrow ([a]^{@}m)\ !\ i = a$
⟨*proof*⟩

lemma *pow-is-concat-replicate*: $u^{@}n = concat\ (replicate\ n\ u)$
⟨*proof*⟩

lemma *pow-slide*: $u \cdot (v \cdot u)^{@}n\ \ \cdot v = (u \cdot v)^{@}(Suc\ n)$
⟨*proof*⟩

lemma *hd-pow*: **assumes** $0 < n$ **shows** $hd(u^{@}n) = hd\ u$
⟨*proof*⟩

lemma *pop-pow*: $m \leq k \Longrightarrow u^{@}m \cdot u^{@}(k-m) = u^{@}k$
⟨*proof*⟩

lemma *pop-pow-cancel*: $u^{@}k \cdot v = u^{@}m \cdot w \Longrightarrow m \leq k \Longrightarrow u^{@}(k-m) \cdot v = w$
⟨*proof*⟩

lemma *pows-comm*: $t^{@}k \cdot t^{@}m = t^{@}m \cdot t^{@}k$
⟨*proof*⟩

lemma *comm-add-exps*: **assumes** $r \cdot u = u \cdot r$ **shows** $r^{@}m \cdot u^{@}k = u^{@}k \cdot r^{@}m$
⟨*proof*⟩

lemma *rev-pow*: $rev\ (x^{@}m) = (rev\ x)^{@}m$
⟨*proof*⟩

lemma *pows-comp*: $x^{@}i \bowtie x^{@}j$
⟨*proof*⟩

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

lemma *pow-len*: $|u^{@}k| = k * |u|$
⟨*proof*⟩

lemma *pow-set*: $set\ (w^{@}Suc\ k) = set\ w$
⟨*proof*⟩

lemma *eq-pow-exp*[*simp*]: **assumes** $u \neq \varepsilon$ **shows** $u^{@}k = u^{@}m \longleftrightarrow k = m$
⟨*proof*⟩

lemma *emp-pow-pos-emp* [*intro*]: **assumes** $v^{@}j = \varepsilon$ $0 < j$ **shows** $v = \varepsilon$
⟨*proof*⟩

lemma *nemp-emp-pow*: **assumes** $u \neq \varepsilon$ **shows** $u^{@}m = \varepsilon \longleftrightarrow m = 0$
⟨*proof*⟩

lemma *nemp-pow-nemp-pos-conv*: **assumes** $u \neq \varepsilon$ **shows** $u^{@}m \neq \varepsilon \longleftrightarrow 0 < m$
⟨*proof*⟩

lemma *nemp-Suc-pow-nemp*: $u \neq \varepsilon \Longrightarrow u^{@}Suc\ k \neq \varepsilon$
⟨*proof*⟩

lemma *nonzero-pow-emp*: $0 < m \Longrightarrow u^{@}m = \varepsilon \longleftrightarrow u = \varepsilon$
⟨*proof*⟩

lemma *pow-eq-eq*:

assumes $u^{@}k = v^{@}k$ **and** $0 < k$
shows $u = v$
⟨*proof*⟩

lemma *Suc-pow-eq-eq*[*elim*]: $u^{@}Suc\ k = v^{@}Suc\ k \Longrightarrow u = v$
⟨*proof*⟩

lemma *map-pow*[*simp*]: $map\ f\ (r^{@}k) = (map\ f\ r)^{@}k$
⟨*proof*⟩

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

lemma *concat-pow*[*simp*]: $concat\ (r^{@}k) = (concat\ r)^{@}k$
⟨*proof*⟩

lemma *concat-sing-pow*[*simp*]: $concat\ ([a]^{@}k) = a^{@}k$
⟨*proof*⟩

lemma *sing-pow-empty*: $[a]^{@}n = \varepsilon \longleftrightarrow n = 0$
⟨*proof*⟩

lemma *sing-pow-lists*: $a \in A \Longrightarrow [a]^{@}n \in lists\ A$
⟨*proof*⟩

lemma *long-pow*: $r \neq \varepsilon \Longrightarrow m \leq |r^{@}m|$
⟨*proof*⟩

lemma *long-pow-exp'*: $r \neq \varepsilon \Longrightarrow m < |r^{@}(Suc\ m)|$
⟨*proof*⟩

lemma *long-pow-expE*: **assumes** $r \neq \varepsilon$ **obtains** n **where** $m \leq |r^{@}Suc\ n|$
⟨*proof*⟩

lemma *pref-pow-ext*: $x \leq p\ r^{@}k \Longrightarrow x \leq p\ r^{@}Suc\ k$
⟨*proof*⟩

lemma *pref-pow-ext'*: $u \leq p\ r^{@}k \Longrightarrow u \leq p\ r \cdot r^{@}k$
⟨*proof*⟩

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

lemma *pref-prod-root*: $u \leq p\ r^{@}k \Longrightarrow u \leq p\ r \cdot u$
⟨*proof*⟩

lemma *le-exps-pref*: $k \leq l \Longrightarrow r^{@}k \leq p\ r^{@}l$
⟨*proof*⟩

lemma *pref-exp-le*: **assumes** $u \neq \varepsilon$ $u^{@}m \leq p\ u^{@}n$ **shows** $m \leq n$

⟨*proof*⟩

lemma *sing-exp-pref-iff*: **assumes** $a \neq b$
shows $[a]^@i \le p\ [a]^@k \cdot [b] \cdot w \longleftrightarrow i \le k$
⟨*proof*⟩

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-prod-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-power*: **assumes** $r \cdot u = u \cdot r$ **shows** $r^@|u| = u^@|r|$
⟨*proof*⟩

lemma *one-generated-list-power*: $u \in lists\ \{x\} \Longrightarrow \exists k.\ concat\ u = x^@k$
⟨*proof*⟩

lemma *pow-lists*: **assumes** $0 < k$ **shows** $u^@k \in lists\ B \Longrightarrow u \in lists\ B$
⟨*proof*⟩

lemma *concat-morph-power*: $xs \in lists\ B \Longrightarrow xs = ts^@k \Longrightarrow concat\ ts^@k = concat\ xs$
⟨*proof*⟩

lemma *per-exp-pref*: $u \le p\ r \cdot u \Longrightarrow u \le p\ r^@k \cdot u$
⟨*proof*⟩

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

lemma *hd-sing-pow*: $k \neq 0 \Longrightarrow hd\ ([a]^@k) = a$
⟨*proof*⟩

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*⟩

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*⟩

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

lemma *exp-pref-cancel*: **assumes** $t^{@}m \cdot y = t^{@}k$ **shows** $y = t^{@}(k - m)$
⟨*proof*⟩

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

lemma *index-pow-mod*: $i < |r^{@}k| \Longrightarrow (r^{@}k)!i = r!(i \ mod \ |r|)$
⟨*proof*⟩

lemma *sing-pow-len* [*simp*]: $|[r]^{@}l| = l$
⟨*proof*⟩

lemma *take-sing-pow*: $k \leq l \Longrightarrow take\ k\ ([r]^{@}l) = [r]^{@}k$
⟨*proof*⟩

lemma *concat-take-sing*: **assumes** $k \leq l$ **shows** $concat\ (take\ k\ ([r]^{@}l)) = r^{@}k$
⟨*proof*⟩

lemma *unique-letter-word*: **assumes** $\bigwedge c.\ c \in set\ w \Longrightarrow c = a$ **shows** $w = [a]^{@}|w|$
⟨*proof*⟩

lemma *card-set-le-1-imp-hd-pow*: **assumes** $card\ (set\ u) \leq 1$ **shows** $[hd\ u]\ ^{@}\ |u| = u$
⟨*proof*⟩

lemma *unique-letter-wordE'*[*elim*]: **assumes** $(\forall\ c.\ c \in set\ w \longrightarrow c = a)$ **obtains** k **where** $w = [a]^{@}k$
⟨*proof*⟩

lemma *unique-letter-wordE''*[*elim*]: **assumes** $set\ w \subseteq \{a\}$ **obtains** k **where** $w = [a]\ ^{@}\ k$
⟨*proof*⟩

lemma *unique-letter-wordE*[*elim*]: **assumes** $set\ w = \{a\}$ **obtains** k **where** $w = [a]^{@}Suc\ k$
⟨*proof*⟩

lemma *conjug-pow*: $x \cdot z = z \cdot y \Longrightarrow x^{@}k \cdot z = z \cdot y^{@}k$
⟨*proof*⟩

lemma *lq-conjug-pow*: **assumes** $p \leq p\ x \cdot p$ **shows** $p^{-1>}(x^{@}k \cdot p) = (p^{-1>}(x \cdot p))^{@}k$
⟨*proof*⟩

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 $k = 1$
⟨*proof*⟩

lemma *count-list-pow*: *count-list* $(w^{@}k)$ $a = k * ($*count-list* w $a)$
⟨*proof*⟩

lemma *comp-pows-pref*: **assumes** $v \neq \varepsilon$ **and** $(u \cdot v)^{@}k \cdot u \leq p\ (u \cdot v)^{@}m$ **shows** $k \leq m$
⟨*proof*⟩

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*⟩

lemma *comp-pows-not-pref*: $\neg\ (u \cdot v)^{@}k \cdot u \leq p\ (u \cdot v)^{@}m \Longrightarrow m \leq k$
⟨*proof*⟩

lemma *comp-pows-spref*: $u^{@}k <p\ u^{@}m \Longrightarrow k < m$
⟨*proof*⟩

lemma *comp-pows-spref-ext*: $(u \cdot v)^{@}k \cdot u <p\ (u \cdot v)^{@}m \Longrightarrow k < m$
⟨*proof*⟩

lemma *comp-pows-pref-zero*:$(u \cdot v)^{@}k <p\ u \Longrightarrow k = 0$
⟨*proof*⟩

lemma *comp-pows-spref'*: $(u \cdot v)^{@}k <p\ (u \cdot v)^{@}m \cdot u \Longrightarrow k < Suc\ m$
⟨*proof*⟩

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*]: $(u \cdot v)^{@}k \cdot u = u \cdot (v \cdot u)^{@}k$
⟨*proof*⟩
lemma[*shifts*]: $(u \cdot v)^{@}k \cdot u \cdot z = u \cdot (v \cdot u)^{@}k \cdot z$
⟨*proof*⟩
lemma[*shifts*]: $u^{@}k \cdot u \cdot z = u \cdot u^{@}k \cdot z$
⟨*proof*⟩
lemma[*shifts*]: $r^{@}k \leq p\ r \cdot r^{@}k$
⟨*proof*⟩
lemma [*shifts*]: $r^{@}k \leq p\ r \cdot r^{@}k \cdot z$
⟨*proof*⟩
lemma [*shifts*]: $(r \cdot q)^{@}k \leq p\ r \cdot q \cdot\ (r \cdot q)^{@}k \cdot z$

⟨*proof*⟩
lemma [*shifts*]: $(r \cdot q)^{@}k \le p\ r \cdot q \cdot\ (r \cdot q)^{@}k$
⟨*proof*⟩
lemma[*shifts*]: $r^{@}k \cdot u \le p\ r \cdot r^{@}k \cdot v \longleftrightarrow u \le p\ r \cdot v$
⟨*proof*⟩
lemma[*shifts*]: $u \cdot u^{@}k \cdot z = u^{@}k \cdot w \longleftrightarrow u \cdot z = w$
⟨*proof*⟩
lemma[*shifts*]: $(r \cdot q)^{@}k \cdot u \le p\ r \cdot q\ \cdot (r \cdot q)^{@}k \cdot v \longleftrightarrow u \le p\ r \cdot q \cdot v$
⟨*proof*⟩
lemma[*shifts*]: $(r \cdot q)^{@}k \cdot u = r \cdot q\ \cdot (r \cdot q)^{@}k \cdot v \longleftrightarrow u = r \cdot q \cdot v$
⟨*proof*⟩
lemma[*shifts*]: $r \cdot q\ \cdot (r \cdot q)^{@}k \cdot v = (r \cdot q)^{@}k \cdot u \longleftrightarrow r \cdot q \cdot v = u$
⟨*proof*⟩
lemma *shifts-spec* [*shifts*]: $(u^{@}k \cdot v)^{@}l \cdot u \cdot u^{@}k \cdot z = u^{@}k \cdot (v \cdot u^{@}k)^{@}l \cdot u \cdot z$
⟨*proof*⟩
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*]: $(u \cdot (v \cdot u)^{@}k)^{@}j \cdot (u \cdot v)^{@}k = (u \cdot v)^{@}k \cdot (u \cdot (u \cdot v)^{@}k)^{@}j$
⟨*proof*⟩
lemma[*shifts*]: $(u \cdot (v \cdot u)^{@}k \cdot z)^{@}j \cdot (u \cdot v)^{@}k = (u \cdot v)^{@}k \cdot (u \cdot z \cdot (u \cdot v)^{@}k)^{@}j$
⟨*proof*⟩
lemmas[*shifts*] = *pow-comm cancel rassoc pow-Suc pref-cancel-conv suf-cancel-conv add-exps cancel-right numeral-nat pow-zero emp-simps*
lemmas[*shifts*] = *less-eq-Suc-le*
lemmas[*shifts*] = *neq0-conv*
lemma *shifts-hd-hd* [*shifts*]: $a\#b\#v = [a] \cdot b\#v$
⟨*proof*⟩
lemmas [*shifts*] = *shifts-hd-hd*[*of* - - ε]
lemma[*shifts*]: $n \le k \Longrightarrow x^{@}k = x^{@}(n + (k - n))$
⟨*proof*⟩
lemma[*shifts*]: $n < k \Longrightarrow x^{@}k = x^{@}(n + (k - n))$
⟨*proof*⟩
lemmas[*shifts*] = *cancel cancel-right pref-cancel-conv suf-cancel-conv triv-pref*
lemmas[*shifts*] = *pow-diff*

lemmas *shifts-rev* = *shifts*[*reversed*]

lemmas *shift-simps* = *shifts shifts*[*reversed*]

method *comparison* = ((*simp only*: *shifts*; *fail*) | (*simp only*: *shifts-rev*; *fail*))

2.13 Rotation

lemma *rotate-root-self*: *rotate* $|r|$ $(r^{@}k) = r^{@}k$
⟨*proof*⟩

lemma *rotate-pow-self*: $rotate\ (l*|u|)\ (u^{@}k) = u^{@}k$
⟨*proof*⟩

lemma *rotate-pow-mod*: $rotate\ n\ (u^{@}k) = rotate\ (n\ mod\ |u|)\ (u^{@}k)$
⟨*proof*⟩

lemma *rotate-conj-pow*: $rotate\ |u|\ ((u \cdot v)^{@}k) = (v \cdot u)^{@}k$
⟨*proof*⟩

lemma *rotate-pow-comm*: $rotate\ n\ (u^{@}k) = (rotate\ n\ u)^{@}k$
⟨*proof*⟩

lemmas *rotate-pow-comm-two* = *rotate-pow-comm*[*of - - 2*, *unfolded pow-two*]

lemma *rotate-back*: $rotate\ (|u| - n\ mod\ |u|)\ (rotate\ n\ u) = u$
⟨*proof*⟩

lemma *rotate-backE*: **obtains** *m* **where** $rotate\ m\ (rotate\ n\ u) = u$
⟨*proof*⟩

lemma *rotate-back'*: **assumes** $rotate\ m\ w = rotate\ n\ w$
shows $rotate\ (m-n)\ w = w$
⟨*proof*⟩

lemma *rotate-class-rotate'*: $(\exists\, n.\ rotate\ n\ w = u) \longleftrightarrow (\exists\, n.\ rotate\ n\ (rotate\ l\ w) = u)$
⟨*proof*⟩

lemma *rotate-class-rotate*: $\{u\ .\ \exists\, n.\ rotate\ n\ w = u\} = \{u\ .\ \exists\, n.\ rotate\ n\ (rotate\ l\ w) = u\}$
⟨*proof*⟩

lemma *rotate-comp-eq*: $w \bowtie rotate\ n\ w \Longrightarrow rotate\ n\ w = w$
⟨*proof*⟩

corollary *mismatch-iff-lexord*: **assumes** $rotate\ n\ w \neq w$ **and** *irrefl r*
shows $mismatch\text{-}pair\ w\ (rotate\ n\ w) \in r \longleftrightarrow (w, rotate\ n\ w) \in lexord\ r$
⟨*proof*⟩

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*: $concat(take\ n\ ws) \le p\ concat\ ws$
⟨*proof*⟩

lemma *concat-take-Suc*: **assumes** $j < |ws|$ **shows** $concat(take\ j\ ws) \cdot ws!j = concat(take\ (Suc\ j)\ ws)$
⟨*proof*⟩

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*⟩

thm *prefI*

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*⟩

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*⟩

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*⟩

lemma *del-emp-concat*: $concat\ us = concat\ (filter\ (\lambda x.\ x \neq \varepsilon)\ us)$
⟨*proof*⟩

lemma *lists-minus*: $us \in lists\ (C - A) \Longrightarrow us \in lists\ C$
⟨*proof*⟩

lemma *lists-minus'*: $us \in lists\ C \Longrightarrow (filter\ (\lambda x.\ x \neq \varepsilon)\ us) \in lists\ (C - \{\varepsilon\})$
⟨*proof*⟩

lemma *pref-concat-pref*: $us \leq p\ ws \Longrightarrow concat\ us \leq p\ concat\ ws$
⟨*proof*⟩

lemmas *suf-concat-suf* = *pref-concat-pref*[*reversed*]

lemma *concat-mono-fac*: $us \leq f\ ws \Longrightarrow concat\ us \leq f\ concat\ ws$
⟨*proof*⟩

lemma *ruler-concat-less*: **assumes** $us \le p\ ws$ **and** $vs \le p\ ws$ **and** $|concat\ us| < |concat\ vs|$
shows $us < p\ vs$
⟨*proof*⟩

lemma *concat-take-mono-strict*: **assumes** $concat\ (take\ i\ ws) < p\ concat\ (take\ j\ ws)$
shows $take\ i\ ws < p\ take\ j\ ws$
⟨*proof*⟩

lemma *take-pp-less*: **assumes** $take\ k\ ws < p\ take\ n\ ws$ **shows** $k < n$
⟨*proof*⟩

lemma *concat-pp-less*: **assumes** $concat\ (take\ k\ ws) < p\ concat\ (take\ n\ ws)$ **shows** $k < n$
⟨*proof*⟩

lemma *take-le-take*: $j \le k \Longrightarrow take\ j\ (take\ k\ xs) = take\ j\ xs$
⟨*proof*⟩

lemma *concat-interval*: **assumes** $concat\ (take\ k\ vs) = concat\ (take\ j\ vs) \cdot s$ **shows** $concat\ (drop\ j\ (take\ k\ vs)) = s$
⟨*proof*⟩

lemma *bin-lists-count-zero'*: **assumes** $ws \in lists\ \{x,y\}$ **and** $count\text{-}list\ ws\ y = 0$
shows $ws \in lists\ \{x\}$
⟨*proof*⟩

lemma *bin-lists-count-zero*: **assumes** $ws \in lists\ \{x,y\}$ **and** $count\text{-}list\ ws\ x = 0$
shows $ws \in lists\ \{y\}$
⟨*proof*⟩

lemma *count-in*: $count\text{-}list\ ws\ a \neq 0 \Longrightarrow a \in set\ ws$
⟨*proof*⟩

lemma *count-in-conv*: $count\text{-}list\ w\ a \neq 0 \longleftrightarrow a \in set\ w$
⟨*proof*⟩

lemma *two-in-set-concat-len*: **assumes** $u \neq v$ **and** $\{u,v\} \subseteq set\ ws$
shows $|u| + |v| \le |concat\ ws|$
⟨*proof*⟩

2.15 Root

definition *root* :: $'a\ list \Rightarrow 'a\ list \Rightarrow bool$ (‹- ∈ -∗› [51,51] 60)
where $u \in r* = (\exists\ k.\ r^{@}k = u)$
notation (*latex* **output**) *root* (‹- ∈ -*›)

abbreviation *not-root* :: $['a\ list, 'a\ list] \Rightarrow bool$ (‹- ∉ -∗› [51,51] 60)
where $u \notin r* \equiv \neg\ (u \in r*)$

Empty word has all roots, including the empty root.

lemma *emp-all-roots* [*simp*]: $\varepsilon \in r*$
⟨*proof*⟩

lemma *emp-all-roots'*[*elim*]: $u = \varepsilon \Longrightarrow u \in r*$
⟨*proof*⟩

lemma *rootI*: $r^{@}k \in r*$
⟨*proof*⟩

lemma *self-root*: $u \in u*$
⟨*proof*⟩

lemma *rootE*[*elim*]: **assumes** $u \in r*$ **obtains** k **where** $r^{@}k = u$
⟨*proof*⟩

lemma *root-exp*: $x \in r* \longleftrightarrow x = r^{@}(|x|\ div\ |r|)$
⟨*proof*⟩

lemma *root-nemp-expE*: **assumes** $w \in r*$ **and** $w \neq \varepsilon$
obtains k **where** $r^{@}k = w$ $0 < k$
⟨*proof*⟩

lemma *root-rev-iff*[*reversal-rule*]: $rev\ u \in rev\ t* \longleftrightarrow u \in t*$
⟨*proof*⟩

lemma *per-root-pref*: $w \neq \varepsilon \Longrightarrow w \in r* \Longrightarrow r \leq p\ w$
⟨*proof*⟩

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

lemma *per-exp-eq*: $u \leq p\ r \cdot u \Longrightarrow |u| = k*|r| \Longrightarrow u \in r*$
⟨*proof*⟩

lemma *take-root*: **assumes** $0 < k$ **shows** $r = take\ |r|\ (r^{@}k)$
⟨*proof*⟩

lemma *root-nemp*: $u \neq \varepsilon \Longrightarrow u \in r* \Longrightarrow r \neq \varepsilon$
⟨*proof*⟩

lemma *root-shorter*: **assumes** $u \neq \varepsilon$ $u \in r*$ $u \neq r$ **shows** $|r| < |u|$
⟨*proof*⟩

lemma *root-shorter-eq*: $u \neq \varepsilon \Longrightarrow u \in r* \Longrightarrow |r| \leq |u|$
⟨*proof*⟩

lemma *root-trans*[*trans*]: $[\![v \in u*;\ u \in t*]\!] \Longrightarrow v \in t*$
⟨*proof*⟩

lemma *root-pow-root*[*intro*]: $v \in u* \Longrightarrow v^{@}n \in u*$
⟨*proof*⟩

lemma *root-len*: $u \in q* \Longrightarrow \exists\, k.\ |u| = k*|q|$
⟨*proof*⟩

lemma *root-len-dvd*: $u \in t* \Longrightarrow |t|\ dvd\ |u|$
⟨*proof*⟩

lemma *common-root-len-gcd*: $u \in t* \Longrightarrow v \in t* \Longrightarrow |t|\ dvd\ (gcd\ |u|\ |v|)$
⟨*proof*⟩

lemma *add-root*[*simp*]: $z \cdot w \in z* \longleftrightarrow w \in z*$
⟨*proof*⟩

lemma *add-roots*[*intro*]: $w \in z* \Longrightarrow w' \in z* \Longrightarrow w \cdot w' \in z*$
⟨*proof*⟩

lemma *concat-sing-list-pow*: $ws \in lists\ \{u\} \Longrightarrow |ws| = k \Longrightarrow concat\ ws = u^{@}k$
⟨*proof*⟩

lemma *concat-sing-list-pow'*: $ws \in lists\{u\} \Longrightarrow concat\ ws = u^{@}|ws|$
⟨*proof*⟩

lemma *root-pref-cancel*[*elim*]: **assumes** $x{\cdot}y \in t*$ **and** $x \in t*$ **shows** $y \in t*$
⟨*proof*⟩

lemma *root-suf-cancel* [*elim*]: $u \cdot v \in r* \Longrightarrow v \in r* \Longrightarrow u \in r*$
⟨*proof*⟩

2.16 Commutation

The solution of the easiest nontrivial word equation, $x \cdot y = y \cdot x$, is in fact already contained in List.thy as the fact $xs \cdot ys = ys \cdot xs \Longrightarrow \exists\, m\ n\ zs.\ concat\ (replicate\ m\ zs) = xs \land concat\ (replicate\ n\ zs) = ys$.

theorem *comm*: $x \cdot y = y \cdot x \longleftrightarrow (\exists\ t\ k\ m.\ x = t^{@}k \land y = t^{@}m)$
⟨*proof*⟩

corollary *comm-root*: $x \cdot y = y \cdot x \longleftrightarrow (\exists\ t.\ x \in t* \land y \in t*)$
⟨*proof*⟩

lemma *comm-rootI*: $x \in t* \Longrightarrow y \in t* \Longrightarrow x \cdot y = y \cdot x$
⟨*proof*⟩

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*⟩

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*⟩

lemma *comm-prod*[*intro*]: **assumes** $r{\cdot}u = u{\cdot}r$ **and** $r{\cdot}v = v{\cdot}r$
shows $r{\cdot}(u{\cdot}v) = (u{\cdot}v){\cdot}r$
⟨*proof*⟩

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*⟩

2.17 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.

2.17.1 Periodic root

lemma $u <p\ r \cdot u \longleftrightarrow u \leq p\ r \cdot u \wedge r \neq \varepsilon$
⟨*proof*⟩

lemma *per-rootI*[*intro*]: $u \leq p\ r \cdot u \Longrightarrow r \neq \varepsilon \Longrightarrow u <p\ r \cdot u$
⟨*proof*⟩

lemma *per-rootI'*[*intro*]: **assumes** $u \leq p\ r^{@}k$ **and** $r \neq \varepsilon$ **shows** $u <p\ r \cdot u$
⟨*proof*⟩

lemma *per-root-nemp*[*dest*]: $u <p\ r \cdot u \Longrightarrow r \neq \varepsilon$
⟨*proof*⟩

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 $w <p\ r \cdot u \Longrightarrow |r| \leq |w| \Longrightarrow r \leq p\ w$
⟨*proof*⟩

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*⟩

lemma *pref-pow-take*: **assumes** $u \leq p\ r^{@}k$ **shows** $u \leq p\ take\ |r|\ u \cdot u$
⟨*proof*⟩

lemma *per-root-powE*: **assumes** $u <p\ r \cdot u$
obtains k **where** $u <p\ r^{@}k$ **and** $0 < k$
⟨*proof*⟩

thm *per-rootI per-rootI′*

lemma *per-root-powE′*: **assumes** $x <p\ r \cdot x$
obtains k **where** $x \leq p\ r^{@}k$ **and** $0 < k$
⟨*proof*⟩

lemma *per-root-modE′* [*elim*]: **assumes** $u <p\ r \cdot u$
obtains p **where** $p <p\ r$ **and** $r^{@}(|u|\ div\ |r|) \cdot p = u$
⟨*proof*⟩

lemma *per-root-modE* [*elim*]: **assumes** $u <p\ r \cdot u$
obtains $n\ p\ s$ **where** $p \cdot s = r$ **and** $r^{@}n \cdot p = u$ **and** $s \neq \varepsilon$
⟨*proof*⟩

lemma *nemp-per-root-conv*: $r \neq \varepsilon \Longrightarrow u <p\ r \cdot u \longleftrightarrow u \leq p\ r \cdot u$
⟨*proof*⟩

lemma *root-ruler*: **assumes** $w <p\ u{\cdot}w\ v <p\ u{\cdot}v$
shows $w \bowtie v$
⟨*proof*⟩

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** $u <p\ r^{@}m \cdot u$
⟨*proof*⟩

theorem *per-root-pow-conv*: $x <p\ r \cdot x \longleftrightarrow (\exists\ k.\ x \leq p\ r^{@}k) \wedge r \neq \varepsilon$
⟨*proof*⟩

lemma *per-root-exp′*: **assumes** $x \leq p\ r^{@}k$ **shows** $x \leq p\ r^{@}|x|$

⟨proof⟩

lemma *per-root-exp*: **assumes** $x <p\ r \cdot x$ **shows** $x \leq p\ r^{@}|x|$
⟨proof⟩

lemma *per-root-drop-exp*: $u <p\ (r^{@}m) \cdot u \implies u <p\ r \cdot u$
⟨proof⟩

lemma *per-root-exp-conv*: $u <p\ (r^{@}Suc\ m) \cdot u \longleftrightarrow u <p\ r \cdot u$
⟨proof⟩

lemma *pref-drop-exp*: **assumes** $x \leq p\ z \cdot x^{@}m$ **shows** $x \leq p\ z \cdot x$
⟨proof⟩

lemma *per-root-drop-exp'*: $x \leq p\ r^{@}(Suc\ k) \cdot x^{@}m \implies x \leq p\ r \cdot x$
⟨proof⟩

lemma *per-drop-exp'*: $0 < k \implies x \leq p\ r^{@}k \cdot x \implies x \leq p\ r \cdot x$
⟨proof⟩

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⟩

lemma *comm-drop-exp'*: **assumes** $u^{@}k \cdot v = v \cdot u^{@}k'$ $0 < k'$ **shows** $u \cdot v = v \cdot u$
⟨proof⟩

lemma *comm-drop-exps*[*elim*]: **assumes** $u^{@}m \cdot v^{@}k = v^{@}k \cdot u^{@}m$ **and** $0 < m$ **and** $0 < k$ **shows** $u \cdot v = v \cdot u$
⟨proof⟩

lemma *comm-pow-roots*:
assumes $0 < m$ **and** $0 < k$
shows $u^{@}m \cdot v^{@}k = v^{@}k \cdot u^{@}m \longleftrightarrow u \cdot v = v \cdot u$
⟨proof⟩

corollary *pow-comm-comm*: **assumes** $x^{@}j = y^{@}k$ **and** $0 < j$ **shows** $x{\cdot}y = y{\cdot}x$
⟨proof⟩

lemma *pow-comm-comm'*: **assumes** *comm*: $u^{@}(Suc\ k) = v^{@}(Suc\ l)$ **shows** $u \cdot v = v \cdot u$
⟨proof⟩

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⟩

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*⟩

lemma *drop-per-pref*: **assumes** $w <p\ u \cdot w$ **shows** $drop\ |u|\ w \leq p\ w$
⟨*proof*⟩

lemma *per-root-trans*[*intro*]: **assumes** $w <p\ u \cdot w$ **and** $u \in t*$ **shows** $w <p\ t \cdot w$
⟨*proof*⟩

lemma *per-root-trans'*[*intro*]: $w \leq p\ u \cdot w \Longrightarrow u \in r* \Longrightarrow u \neq \varepsilon \Longrightarrow w \leq p\ r \cdot w$
⟨*proof*⟩

lemmas *per-root-trans-suf'*[*intro*] = *per-root-trans'*[*reversed*]

Note that $[\![<_p\ w\ (u \cdot w);\ <_p\ u\ (t \cdot u)]\!] \Longrightarrow <_p\ w\ (t \cdot w)$ does not hold.

lemma *per-root-same-prefix*:$w <p\ r \cdot w \Longrightarrow w' \leq p\ r \cdot w' \Longrightarrow\ w \bowtie w'$
⟨*proof*⟩

lemma *take-after-drop*: $|u| + q \leq |w| \Longrightarrow w <p\ u \cdot w \Longrightarrow take\ q\ (drop\ |u|\ w) = take\ q\ w$
⟨*proof*⟩

The following lemmas are a weak version of the Periodicity lemma

lemma *two-pers*:
assumes *pu*: $w \leq p\ u \cdot w$ **and** *pv*: $w \leq p\ v \cdot w$ **and** *len*: $|u| + |v| \leq |w|$
shows $u \cdot v = v \cdot u$
⟨*proof*⟩

lemma *two-pers-root*: **assumes** $w <p\ u \cdot w$ **and** $w <p\ v \cdot w$ **and** $|u| + |v| \leq |w|$
shows $u{\cdot}v = v{\cdot}u$
⟨*proof*⟩

2.17.2 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*⟩

lemma *max-pref-per-root*: $u \wedge_p r \cdot u \leq p\ r \cdot (u \wedge_p r \cdot u)$
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

find-theorems $?u \wedge_p ?r \cdot ?u$

2.17.3 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 <p (take n w) · w*

lemma *period-I'*: $w \neq \varepsilon \Longrightarrow 0 < n \Longrightarrow w \leq p\ (take\ n\ w) \cdot w \Longrightarrow period\ w\ n$
⟨*proof*⟩

lemma *periodI*[*intro*]: $w \neq \varepsilon \Longrightarrow w <p\ r \cdot w \Longrightarrow period\ w\ |r|$
⟨*proof*⟩

The numeric definition respects the following convention about empty words and empty periods.

lemma *emp-no-period*: ¬ *period* ε *n*
⟨*proof*⟩

lemma ¬ *period w 0*
⟨*proof*⟩

lemma *per-nemp*: $period\ w\ n \Longrightarrow\ w \neq \varepsilon$
⟨*proof*⟩

lemma *per-not-zero*: $period\ w\ n \Longrightarrow\ 0 < n$
⟨*proof*⟩

lemma *per-pref*: $period\ w\ n \Longrightarrow\ w \leq p\ take\ n\ w \cdot w$
⟨*proof*⟩

A nonempty word has all ”long” periods

lemma *all-long-pers*: $\llbracket$ $w \neq \varepsilon$; $|w| \leq n$ $\rrbracket \Longrightarrow$ *period* w n
⟨*proof*⟩

lemma *len-is-per*: $w \neq \varepsilon \Longrightarrow$ *period* w $|w|$
⟨*proof*⟩

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*⟩

lemma *indeces-period*:
assumes $w \neq \varepsilon$ **and** $0 < n$ **and** *forall*: $\bigwedge i.\ i + n < |w| \Longrightarrow w!i = w!(i+n)$
shows *period* w n
⟨*proof*⟩

In some cases, the numeric definition is more useful than the definition using the period root.

lemma *period-rev*: **assumes** *period* w p **shows** *period* (*rev* w) p
⟨*proof*⟩

lemma *period-rev-conv* [*reversal-rule*]: *period* (*rev* w) $n \longleftrightarrow$ *period* w n
⟨*proof*⟩

lemma *period-fac*: **assumes** *period* $(u{\cdot}w{\cdot}v)$ p **and** $w \neq \varepsilon$
shows *period* w p
⟨*proof*⟩

lemma *period-fac'*: *period* v $p \Longrightarrow u \leq f\ v \Longrightarrow u \neq \varepsilon \Longrightarrow$ *period* u p
⟨*proof*⟩

lemma *pow-per*[*intro*]: **assumes** $y \neq \varepsilon$ **and** $0 < k$ **shows** *period* $(y^{@}k)$ $|y|$
⟨*proof*⟩

lemma *per-fac*: **assumes** $w \neq \varepsilon$ **and** $w \leq f\ y^{@}k$ **shows** *period* w $|y|$
⟨*proof*⟩

The numeric definition is equivalent to being prefix of a power.

theorem *period-pref*: *period* w $n \longleftrightarrow (\exists\, k\ r.\ w \leq np\ r^{@}k \wedge |r| = n)$ (**is** - $\longleftrightarrow$ *?R*)
⟨*proof*⟩

Two more characterizations of a period

theorem *per-shift*: **assumes** $w \neq \varepsilon$ $0 < n$
shows *period* w $n \longleftrightarrow$ *drop* n $w \leq p\ w$
⟨*proof*⟩

lemma *rotate-per-root*: **assumes** $w \neq \varepsilon$ **and** $0 < n$ **and** $w =$ *rotate* n w
shows *period* w n
⟨*proof*⟩

Various lemmas on periods

lemma *period-drop*: **assumes** *period w p* **and** $p < |w|$
shows *period* (*drop p w*) *p*
⟨*proof*⟩

lemma *ext-per-left*: **assumes** *period w p* **and** $p \leq |w|$
shows *period* (*take p w* · *w*) *p*
⟨*proof*⟩

lemma *ext-per-left-power*: *period w p* $\Longrightarrow p \leq |w| \Longrightarrow$ *period* ((*take p w*)$^{@}k$ · *w*) *p*
⟨*proof*⟩

lemma *take-several-pers*: **assumes** *period w n* **and** $m*n \leq |w|$
shows (*take n w*)$^{@}m$ = *take* ($m*n$) *w*
⟨*proof*⟩

lemma *per-div*: **assumes** *n dvd* $|w|$ **and** *period w n*
shows (*take n w*)$^{@}$($|w|$ *div n*) = *w*
⟨*proof*⟩

lemma *per-mult*: **assumes** *period w n* **and** $0 < m$ **shows** *period w* ($m*n$)
⟨*proof*⟩

theorem *two-periods*:
assumes *period w p period w q* $p + q \leq |w|$
shows *period w* (*gcd p q*)
⟨*proof*⟩

lemma *index-mod-per-root*: **assumes** $r \neq \varepsilon$ **and** *i*: $\forall\ i < |w|.\ w!i = r!(i\ mod\ |r|)$
shows *w* <*p r* · *w*
⟨*proof*⟩

lemma *index-pref-pow-mod*: *w* ≤*p* $r^{@}k \Longrightarrow i < |w| \Longrightarrow w!i = r!(i\ mod\ |r|\ )$
⟨*proof*⟩

lemma *index-per-root-mod*: *w* <*p r* · *w* $\Longrightarrow i < |w| \Longrightarrow w!i = r!(i\ mod\ |r|)$
⟨*proof*⟩

lemma *root-divisor*: **assumes** *period w k* **and** *k dvd* $|w|$ **shows** $w \in$ (*take k w*)∗
⟨*proof*⟩

lemma *per-pref′*: **assumes** $u \neq \varepsilon$ **and** *period v k* **and** *u* ≤*p v* **shows** *period u k*
⟨*proof*⟩

2.17.4 Period: overview

notepad
begin

⟨*proof*⟩
end

2.17.5 Singleton and its power

primrec *letter-pref-exp* :: *'a list* ⇒ *'a* ⇒ *nat* **where**
letter-pref-exp ε *a* = *0* |
letter-pref-exp (*b* # *xs*) *a* = (*if b* ≠ *a then 0 else Suc* (*letter-pref-exp xs a*))

definition *letter-suf-exp* :: *'a list* ⇒ *'a* ⇒ *nat* **where**
letter-suf-exp w a = *letter-pref-exp* (*rev w*) *a*

lemma *concat-len-one*: **assumes** |*us*| = *1* **shows** *concat us* = *hd us*
⟨*proof*⟩

lemma *sing-pow-hd-tl*: *c* # *w* ∈ [*a*]∗ ⟷ *c* = *a* ∧ *w* ∈ [*a*]∗
⟨*proof*⟩

lemma *pref-sing-pow*: **assumes** *w* ≤*p* $[a]^{@}m$ **shows** *w* = $[a]^{@}(|w|)$
⟨*proof*⟩

lemma *sing-pow-palindrom*: **assumes** *w* = $[a]^{@}k$ **shows** *rev w* = *w*
⟨*proof*⟩

lemma *suf-sing-power*: **assumes** *w* ≤*s* $[a]^{@}m$ **shows** *w* ∈ [*a*]∗
⟨*proof*⟩

lemma *sing-fac-pow*: **assumes** *w* ∈ [*a*]∗ **and** *v* ≤*f* *w* **shows** *v* ∈ [*a*]∗
⟨*proof*⟩

lemma *sing-pow-fac'*: **assumes** *a* ≠ *b* **and** *w* ∈ [*a*]∗ **shows** ¬ ([*b*] ≤*f* *w*)
⟨*proof*⟩

lemma *all-set-sing-pow*: (∀ *b*. *b* ∈ *set w* ⟶ *b* = *a*) ⟷ *w* ∈ [*a*]∗ (**is** *?All* ⟷ -)
⟨*proof*⟩

lemma *sing-fac-set*: [*a*] ≤*f* *x* ⟹ *a* ∈ *set x*
⟨*proof*⟩

lemma *set-sing-pow-hd* [*simp*]: **assumes** *0* < *k* **shows** *a* ∈ *set* ($[a]^{@}k$)
⟨*proof*⟩

lemma *neq-set-not-root*: *a* ≠ *b* ⟹ *b* ∈ *set x* ⟹ *x* ∉ [*a*]∗
⟨*proof*⟩

lemma *sing-pow-set-Suc*[*simp*]: *set* ($[a]^{@}$*Suc k*) = {*a*}
⟨*proof*⟩

lemma *sing-pow-set*[*simp*]: **assumes** *0* < *k* **shows** *set* ($[a]^{@}k$) = {*a*}

⟨*proof*⟩

lemma *sing-pow-set-sub*: $set\ ([a]^{@}k) \subseteq \{a\}$
⟨*proof*⟩

lemma *unique-letter-fac-expE*: **assumes** $w \leq f\ [a]^{@}k$
obtains m **where** $w = [a]^{@}m$
⟨*proof*⟩

lemma *neq-in-set-not-pow*: $a \neq b \Longrightarrow b \in set\ x \Longrightarrow x \neq [a]^{@}k$
⟨*proof*⟩

lemma *sing-pow-card-set-Suc*: **assumes** $c = [a]^{@}Suc\ k$ **shows** $card(set\ c) = 1$
⟨*proof*⟩

lemma *sing-pow-card-set*: **assumes** $k \neq 0$ **and** $c = [a]^{@}k$ **shows** $card(set\ c) = 1$
⟨*proof*⟩

lemma *sing-pow-set'*: $u \in [a]* \Longrightarrow u \neq \varepsilon \Longrightarrow set\ u = \{a\}$
⟨*proof*⟩

lemma *root-sing-set-iff*: $u \in [a]* \longleftrightarrow set\ u \subseteq \{a\}$
⟨*proof*⟩

lemma *letter-pref-exp-hd*: $u \neq \varepsilon \Longrightarrow hd\ u = a \Longrightarrow letter\text{-}pref\text{-}exp\ u\ a \neq 0$
⟨*proof*⟩

lemma *letter-pref-exp-pref*: $[a]^{@}(letter\text{-}pref\text{-}exp\ w\ a) \leq p\ w$
⟨*proof*⟩

lemma *letter-pref-exp-Suc*: $\neg\ [a]^{@}(Suc\ (letter\text{-}pref\text{-}exp\ w\ a)) \leq p\ w$
⟨*proof*⟩

lemma *takeWhile-letter-pref-exp*: $takeWhile\ (\lambda x.\ x = a)\ w =[a]^{@}(letter\text{-}pref\text{-}exp\ w\ a)$
⟨*proof*⟩

lemma *concat-takeWhile-sing*: $concat\ (takeWhile\ (\lambda\ x.\ x = u)\ ws) = u^{@}|takeWhile\ (\lambda\ x.\ x = u)\ ws|$
⟨*proof*⟩

lemma *dropWhile-distinct*: **assumes** $w \neq [a]^{@}(letter\text{-}pref\text{-}exp\ w\ a)$
shows $[a]^{@}(letter\text{-}pref\text{-}exp\ w\ a)\cdot[hd\ (dropWhile\ (\lambda x.\ x = a)\ w)] \leq p\ w$
⟨*proof*⟩

lemma *letter-pref-exp-mismatch*: $u = [a]^{@}letter\text{-}pref\text{-}exp\ u\ a \cdot v \Longrightarrow v \neq \varepsilon \Longrightarrow hd\ v \neq a$

⟨*proof*⟩

lemma *takeWhile-sing-root*: $takeWhile\ (\lambda\ x.\ x = a)\ w \in [a]*$
⟨*proof*⟩

lemma *takeWhile-sing-pow*: $takeWhile\ (\lambda\ x.\ x = a)\ w = w \longleftrightarrow w = [a]^{@}|w|$
⟨*proof*⟩

lemma *dropWhile-sing-pow*: $dropWhile\ (\lambda\ x.\ x = a)\ w = \varepsilon \longleftrightarrow w = [a]^{@}|w|$
⟨*proof*⟩

lemma *nemp-takeWhile-hd*: $us \neq \varepsilon \Longrightarrow hd\ (takeWhile\ (\lambda\ a.\ a = hd\ us)\ us) = hd\ us$
⟨*proof*⟩

lemma *nemp-takeWhile-last*: $us \neq \varepsilon \Longrightarrow last\ (takeWhile\ (\lambda\ a.\ a = hd\ us)\ us) = hd\ us$
⟨*proof*⟩

lemma *card-set-decompose*: **assumes** $1 < card\ (set\ us)$
shows $takeWhile\ (\lambda\ a.\ a = hd\ us)\ us \neq \varepsilon$ **and** $dropWhile\ (\lambda\ a.\ a = hd\ us)\ us \neq \varepsilon$ **and**
$set\ (takeWhile\ (\lambda\ a.\ a = hd\ us)\ us) = \{hd\ us\}$ **and**
$last\ (takeWhile\ (\lambda\ a.\ a = hd\ us)\ us) \neq hd(dropWhile\ (\lambda\ a.\ a = hd\ us)\ us)$
⟨*proof*⟩

lemma *distinct-letter-in*: **assumes** $w \notin [a]*$
obtains $m\ b\ q$ **where** $[a]^{@}m \cdot [b] \cdot q = w$ **and** $b \neq a$
⟨*proof*⟩

lemma *distinct-letter-in-hd*: **assumes** $w \notin [hd\ w]*$
obtains $m\ b\ q$ **where** $[hd\ w]^{@}m \cdot [b] \cdot q = w$ **and** $b \neq hd\ w$ **and** $m \neq 0$
⟨*proof*⟩

lemma *distinct-letter-in-hd′*: **assumes** $w \notin [hd\ w]*$
obtains $m\ b\ q$ **where** $[hd\ w]^{@}Suc\ m \cdot [b] \cdot q = w$ **and** $b \neq hd\ w$
⟨*proof*⟩

lemma *distinct-letter-in-suf*: **assumes** $w \notin [a]*$
obtains $m\ b$ **where** $[b] \cdot [a]^{@}m \leq s\ w$ **and** $b \neq a$
⟨*proof*⟩

lemma *sing-pow-exp*: **assumes** $w \in [a]*$ **shows** $w = [a]^{@}|w|$
⟨*proof*⟩

lemma *sing-power′*: **assumes** $w \in [a]*$ **and** $i < |w|$ **shows** $w\ !\ i = a$
⟨*proof*⟩

lemma *rev-sing-power*: $x \in [a]* \Longrightarrow rev\ x = x$

⟨*proof*⟩

lemma *lcp-letter-power*:
assumes $w \neq \varepsilon$ **and** $w \in [a]*$ **and** $[a]^{@}m \cdot [b] \leq p\ z$ **and** $a \neq b$
shows $w \cdot z \wedge_p z \cdot w = [a]^{@}m$
⟨*proof*⟩

lemma *per-one*: **assumes** $w <p\ [a] \cdot w$ **shows** $w \in [a]*$
⟨*proof*⟩

lemma *per-one'*: $w \in [a]* \Longrightarrow w <p\ [a] \cdot w$
⟨*proof*⟩

lemma *per-sing-one*: **assumes** $w \neq \varepsilon$ $w <p\ [a] \cdot w$ **shows** *period w 1*
⟨*proof*⟩

2.18 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*]: $border\ x\ w = (x \leq p\ w \wedge x \leq s\ w \wedge x \neq w \wedge x \neq \varepsilon)$

definition *bordered* :: *'a list* ⇒ *bool*
where [*simp*]: $bordered\ w = (\exists\, b.\ b \leq b\ w)$

lemma *borderI*[*intro*]: $x \leq p\ w \Longrightarrow x \leq s\ w \Longrightarrow x \neq w \Longrightarrow x \neq \varepsilon \Longrightarrow x \leq b\ w$
⟨*proof*⟩

lemma *borderD-pref*: $x \leq b\ w \Longrightarrow x \leq p\ w$
⟨*proof*⟩

lemma *borderD-spref*: $x \leq b\ w \Longrightarrow x <p\ w$
⟨*proof*⟩

lemma *borderD-suf*: $x \leq b\ w \Longrightarrow x \leq s\ w$
⟨*proof*⟩

lemma *borderD-ssuf*: $x \leq b\ w \Longrightarrow x <s\ w$
⟨*proof*⟩

lemma *borderD-nemp*: $x \leq b\ w \Longrightarrow x \neq \varepsilon$
⟨*proof*⟩

lemma *borderD-neq*: $x \leq b\ w \Longrightarrow x \neq w$
⟨*proof*⟩

lemma *borderedI*: $u \leq b\ w \Longrightarrow$ *bordered* w
⟨*proof*⟩

lemma *border-lq-nemp*: **assumes** $x \leq b\ w$ **shows** $x^{-1>}w \neq \varepsilon$
⟨*proof*⟩

lemma *border-rq-nemp*: **assumes** $x \leq b\ w$ **shows** $w^{<-1}x \neq \varepsilon$
⟨*proof*⟩

lemma *border-trans*[*trans*]: **assumes** $t \leq b\ x\ x \leq b\ w$
shows $t \leq b\ w$
⟨*proof*⟩

lemma *border-rev-conv*[*reversal-rule*]: *rev* $x \leq b$ *rev* $w \longleftrightarrow x \leq b\ w$
⟨*proof*⟩

lemma *border-lq-comp*: $x \leq b\ w \Longrightarrow (w^{<-1}x) \bowtie x$
⟨*proof*⟩

lemmas *border-lq-suf-comp* = *border-lq-comp*[*reversed*]

2.18.1 The shortest border

lemma *border-len*: **assumes** $x \leq b\ w$
shows $1 < |w|$ **and** $0 < |x|$ **and** $|x| < |w|$
⟨*proof*⟩

lemma *borders-compare*: **assumes** $x \leq b\ w$ **and** $x' \leq b\ w$ **and** $|x'| < |x|$
shows $x' \leq b\ x$
⟨*proof*⟩

lemma *unbordered-border*:
bordered $w \Longrightarrow \exists\ x.\ x \leq b\ w \land \neg$ *bordered* x
⟨*proof*⟩

lemma *unbordered-border-shortest*: $x \leq b\ w \Longrightarrow \neg$ *bordered* $x \Longrightarrow\ y \leq b\ w \Longrightarrow |x|$
$\leq |y|$
⟨*proof*⟩

lemma *long-border-bordered*: **assumes** *long*: $|w| < |x| + |x|$ **and** *border*: $x \leq b\ w$
shows $(w^{<-1}x)^{-1>}x \leq b\ x$
⟨*proof*⟩

thm *long-border-bordered*[*reversed*]

lemma *border-short-dec*: **assumes** *border*: $x \leq b\ w$ **and** *short*: $|x| + |x| \leq |w|$
shows $x \cdot x^{-1>}(w^{<-1}x) \cdot x = w$
⟨*proof*⟩

lemma *bordered-dec*: **assumes** *bordered w*
obtains *u v* **where** $u{\cdot}v{\cdot}u = w$ **and** $u \neq \varepsilon$
⟨*proof*⟩

lemma *emp-not-bordered*: $\neg$ *bordered* ε
⟨*proof*⟩

lemma *bordered-nemp*: *bordered* $w \Longrightarrow w \neq \varepsilon$
⟨*proof*⟩

lemma *sing-not-bordered*: $\neg$ *bordered* $[a]$
⟨*proof*⟩

2.18.2 Relation to period and conjugation

lemma *border-conjug-eq*: $x \leq b\ w \Longrightarrow (w^{<-1}x) \cdot w = w \cdot (x^{-1>}w)$
⟨*proof*⟩

lemma *border-per-root*: $x \leq b\ w \Longrightarrow w \leq p\ (w^{<-1}x) \cdot w$
⟨*proof*⟩

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*⟩

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*⟩

lemma *border-per*: $x \leq b\ w \Longrightarrow$ *period* $w\ (|w|-|x|)$
⟨*proof*⟩

lemma *per-border*: **assumes** $n < |w|$ **and** *period* $w\ n$
shows *take* $(|w| - n)\ w\ \leq b\ w$
⟨*proof*⟩

2.19 The longest border and the shortest period

2.19.1 The longest border

definition *max-borderP* :: *'a list* $\Rightarrow$ *'a list* $\Rightarrow$ *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* $\varepsilon\ \varepsilon$
⟨*proof*⟩

lemma *max-borderP-exE*: **obtains** *u* **where** *max-borderP* $u\ w$

⟨*proof*⟩

lemma *max-borderP-of-nemp*: *max-borderP u* $\varepsilon \Longrightarrow u = \varepsilon$
⟨*proof*⟩

lemma *max-borderP-D-neq*: $w \neq \varepsilon \Longrightarrow$ *max-borderP u w* $\Longrightarrow u \neq w$
⟨*proof*⟩

lemma *max-borderP-D-pref*: *max-borderP u w* $\Longrightarrow u \le p\ w$
⟨*proof*⟩

lemma *max-borderP-D-suf*: *max-borderP u w* $\Longrightarrow u \le s\ w$
⟨*proof*⟩

lemma *max-borderP-D-max*: *max-borderP u w* $\Longrightarrow v \le b\ w \Longrightarrow v \le p\ u$
⟨*proof*⟩

lemma *max-borderP-D-max'*: *max-borderP u w* $\Longrightarrow v \le b\ w \Longrightarrow v \le s\ u$
⟨*proof*⟩

lemma *unbordered-max-border-emp*: ¬ *bordered w* $\Longrightarrow$ *max-borderP u w* $\Longrightarrow u = \varepsilon$
⟨*proof*⟩

lemma *bordered-max-border-nemp*: *bordered w* $\Longrightarrow$ *max-borderP u w* $\Longrightarrow u \neq \varepsilon$
⟨*proof*⟩

lemma *max-borderP-border*: *max-borderP u w* $\Longrightarrow u \neq \varepsilon \Longrightarrow u \le b\ w$
⟨*proof*⟩

lemma *max-borderP-rev*: *max-borderP* (*rev u*) (*rev w*) $\Longrightarrow$ *max-borderP u w*
⟨*proof*⟩

lemma *max-borderP-rev-conv*: *max-borderP* (*rev u*) (*rev w*) $\longleftrightarrow$ *max-borderP u w*
⟨*proof*⟩

term *arg-max*
definition *max-border* :: *'a list* ⇒ *'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 $u = v$
⟨*proof*⟩

lemma *max-border-ex*: *max-borderP* (*max-border w*) *w*
⟨*proof*⟩

lemma *max-borderP-max-border*: *max-borderP u w* $\Longrightarrow$ *max-border w* $= u$

⟨*proof*⟩

lemma *max-border-len-rev*: $|max\text{-}border\ u| = |max\text{-}border\ (rev\ u)|$
⟨*proof*⟩

lemma *max-border-border*: **assumes** *bordered w* **shows** $max\text{-}border\ w \leq b\ w$
⟨*proof*⟩

theorem *max-border-border'*: $max\text{-}border\ w \neq \varepsilon \Longrightarrow max\text{-}border\ w \leq b\ w$
⟨*proof*⟩

lemma *max-border-sing-emp*: $max\text{-}border\ [a] = \varepsilon$
⟨*proof*⟩

lemma *max-border-suf*: $max\text{-}border\ w \leq s\ w$
⟨*proof*⟩

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

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 $max\text{-}border\ w = u$
⟨*proof*⟩

lemma *max-border-less-len*: **assumes** $w \neq \varepsilon$ **shows** $|max\text{-}border\ w| < |w|$
⟨*proof*⟩

theorem *max-border-max-pref*: **assumes** $u \leq b\ w$ **shows** $u \leq p\ max\text{-}border\ w$
⟨*proof*⟩

theorem *max-border-max-suf*: **assumes** $u \leq b\ w$ **shows** $u \leq s\ max\text{-}border\ w$
⟨*proof*⟩

lemma *bordered-max-bord-nemp-conv*[*code*]: *bordered w* $\longleftrightarrow max\text{-}border\ w \neq \varepsilon$
⟨*proof*⟩

lemma *max-bord-take*: $max\text{-}border\ w = take\ |max\text{-}border\ w|\ w$
⟨*proof*⟩

2.19.2 The shortest period

definition *min-period-root* :: *'a list* ⇒ *'a list* (‹π›) **where**
$min\text{-}period\text{-}root\ w = take\ (LEAST\ n.\ period\ w\ n)\ w$

definition *min-period* :: *'a list* ⇒ *nat* **where**
$min\text{-}period\ w = |\pi\ w|$

lemma *min-per-emp*[*simp*]: $\pi\ \varepsilon = \varepsilon$

⟨*proof*⟩

lemma *min-per-zero*[*simp*]: *min-period* ε = *0*
⟨*proof*⟩

lemma *min-per-per*: $w \neq \varepsilon \Longrightarrow$ *period* w (*min-period* w)
⟨*proof*⟩

lemma *min-per-pos*: $w \neq \varepsilon \Longrightarrow$ *0* < *min-period* w
⟨*proof*⟩

lemma *min-per-len*: *min-period* $w \leq |w|$
⟨*proof*⟩

lemmas *min-per-root-len* = *min-per-len*[*unfolded min-period-def*]

lemma *min-per-sing*: *min-period* [a] = *1*
⟨*proof*⟩

lemma *min-per-root-per-root*: **assumes** $w \neq \varepsilon$ **shows** w <*p* (π w) $\cdot$ w
⟨*proof*⟩

lemma *min-per-pref*: π w ≤*p* w
⟨*proof*⟩

lemma *min-per-nemp*: $w \neq \varepsilon \Longrightarrow \pi\ w \neq \varepsilon$
⟨*proof*⟩

lemma *min-per-min*: **assumes** w <*p* $r \cdot w$ **shows** π w ≤*p* r
⟨*proof*⟩

lemma *lq-min-per-pref*: $\pi\ w^{-1>}w$ ≤*p* w
⟨*proof*⟩

lemma *max-bord-emp*: *max-border* $\varepsilon = \varepsilon$
⟨*proof*⟩

theorem *min-per-max-border*: π w $\cdot$ *max-border* w = w
⟨*proof*⟩

lemma *min-per-len-diff*: *min-period* w = $|w|$ − |*max-border* w|
⟨*proof*⟩

lemma *min-per-root-take* [*code*]: π w = *take* ($|w|$ − |*max-border* w|) w
⟨*proof*⟩

2.20 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* $\Rightarrow$ *bool*
where *primitive* $u = (\forall\ r\ k.\ r^{@}k = u \longrightarrow k = 1)$

lemma *emp-not-prim*[*simp*]: $\neg$ *primitive* ε
⟨*proof*⟩

lemma *primI*[*intro*]: $(\bigwedge\ r\ k.\ r^{@}k = u \Longrightarrow k = 1) \Longrightarrow$ *primitive* u
⟨*proof*⟩

lemma *prim-nemp*: *primitive* $u \Longrightarrow u \neq \varepsilon$
⟨*proof*⟩

lemma *prim-exp-one*: *primitive* $u \Longrightarrow r^{@}k = u \Longrightarrow k = 1$
⟨*proof*⟩

lemma *pow-nemp-imprim*[*intro*]: $2 \leq k \Longrightarrow \neg$ *primitive* $(u^{@}k)$
⟨*proof*⟩

lemma *pow-not-prim*: $\neg$ *primitive* $(u^{@}Suc(Suc\ k))$
⟨*proof*⟩

lemma *pow-non-prim*: $k \neq 1 \Longrightarrow \neg$ *primitive* $(w^{@}k)$
⟨*proof*⟩

lemma *prim-exp-eq*: *primitive* $u \Longrightarrow r^{@}k = u \Longrightarrow u = r$
⟨*proof*⟩

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*⟩

lemma *prim-triv-root*: *primitive* $u \Longrightarrow u \in t* \Longrightarrow t = u$
⟨*proof*⟩

lemma *prim-comm-root*[*elim*]: **assumes** *primitive* r **and** $u \cdot r = r \cdot u$ **shows** $u \in r*$
⟨*proof*⟩

lemma *prim-comm-exp*[*elim*]: **assumes** *primitive* r **and** $u{\cdot}r = r{\cdot}u$ **obtains** k **where** $r^{@}k = u$
⟨*proof*⟩

lemma *pow-prim-root*: **assumes** $w^{@}k = r^{@}n$ **and** $0 < n$ *primitive* r
shows $w \in r*$

⟨*proof*⟩

lemma *prim-root-drop-exp*[*elim*]: **assumes** $u^{@}k \in r*$ **and** $0 < k$ **and** *primitive r*
shows $u \in r*$
⟨*proof*⟩

lemma *prim-card-set*: **assumes** *primitive u* **and** $|u| \neq 1$ **shows** $1 < card\ (set\ u)$
⟨*proof*⟩

lemma *comm-not-prim*: **assumes** $u \neq \varepsilon$ $v \neq \varepsilon$ $u \cdot v = v \cdot u$ **shows** ¬ *primitive* $(u \cdot v)$
⟨*proof*⟩

lemma *prim-rotate-conv*: *primitive w* ⟷ *primitive* (*rotate n w*)
⟨*proof*⟩

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*⟩

lemma *prim-no-rotate*: **assumes** *primitive w* **and** $0 < n$ **and** $n < |w|$
shows *rotate n w* $\neq w$
⟨*proof*⟩

lemma *no-rotate-prim*: **assumes** $w \neq \varepsilon$ **and** $\bigwedge n.\ 0 < n \Longrightarrow n < |w| \Longrightarrow$ *rotate n w* $\neq w$
shows *primitive w*
⟨*proof*⟩

corollary *prim-iff-rotate*: **assumes** $w \neq \varepsilon$ **shows**
primitive w ⟷ $(\forall\ n.\ 0 < n \wedge n < |w| \longrightarrow$ *rotate n w* $\neq w)$
⟨*proof*⟩

lemma *prim-sing*: *primitive* [*a*]
⟨*proof*⟩

lemma *sing-pow-conv* [*simp*]: $[u] = t^{@}k \longleftrightarrow t = [u] \wedge k = 1$
⟨*proof*⟩

lemma *prim-rev-iff*[*reversal-rule*]: *primitive* (*rev u*) ⟷ *primitive u*
⟨*proof*⟩

lemma *prim-map-prim*: *primitive* (*map f ws*) ⟹ *primitive ws*
⟨*proof*⟩

lemma *inj-map-prim*: **assumes** *inj-on f A* **and** $u \in$ *lists A* **and**
primitive u
shows *primitive* (*map f u*)
⟨*proof*⟩

lemma *prim-map-iff* [*reversal-rule*]:
assumes *inj f* **shows** *primitive* (*map f ws*) = *primitive* (*ws*)
⟨*proof*⟩

lemma *prim-concat-prim*: *primitive* (*concat ws*) $\Longrightarrow$ *primitive ws*
⟨*proof*⟩

lemma *eq-append-not-prim*: $x = y \Longrightarrow \neg$ *primitive* $(x \cdot y)$
⟨*proof*⟩

2.21 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* $x \neq \varepsilon$ *then* (*THE* r. *primitive* $r \wedge (\exists\ k.\ x = r^{@}k)$) *else* ε)

definition *primitive-root-exp* :: *'a list* ⇒ *nat* (‹e_ϱ›) **where**
primitive-root-exp x = (*if* $x \neq \varepsilon$ *then* (*THE* k. $x = (\varrho\ x)^{@}k$) *else* 0)

lemma *primroot-emp*[*simp*]: $\varrho\ \varepsilon = \varepsilon$
⟨*proof*⟩

lemma *comm-prim*: **assumes** *primitive* r **and** *primitive* s **and** $r \cdot s = s \cdot r$
shows $r = s$
⟨*proof*⟩

lemma *primroot-ex*: **assumes** $x \neq \varepsilon$ **shows** $\exists\ r\ k.$ *primitive* $r \wedge k \neq 0 \wedge x = r^{@}k$
⟨*proof*⟩

lemma *primroot-exE*: **assumes**$x \neq \varepsilon$
obtains $r\ k$ **where** *primitive* r **and** $0 < k$ **and** $x = r^{@}k$
⟨*proof*⟩

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** ϱ $u = r$
⟨*proof*⟩

lemma *primroot-unique'*: **assumes** $0 < k$ *primitive* r **and** $u = r^{@}k$ **shows** $\varrho\ u = r$
⟨*proof*⟩

lemma *prim-self-root*[*intro*]: *primitive* $x \Longrightarrow \varrho\ x\ = x$

⟨*proof*⟩

lemma *primroot-exp-unique*: **assumes** $u \neq \varepsilon$ **and** $(\varrho\ u)^{@}k = u$ **shows** $e_\varrho\ u = k$
⟨*proof*⟩

lemma *primroot-prim*[*intro*]: $x \neq \varepsilon \Longrightarrow$ *primitive* $(\varrho\ x)$
⟨*proof*⟩

Existence and uniqueness of the primitive root justifies the function ϱ: it indeed yields the primitive root of a nonempty word.

lemma *primroot-is-root*[*simp*]: $x \in (\varrho\ x)*$
⟨*proof*⟩

lemma *primroot-expE*: **obtains** k **where** $(\varrho\ x)^{@}k = x$ **and** $0 < k$
⟨*proof*⟩

lemma *primroot-exp-eq* [*simp*]: $(\varrho\ u)^{@}(e_\varrho\ u) = u$
⟨*proof*⟩

lemma *primroot-exp-len*:
shows $e_\varrho\ w * |\varrho\ w| = |w|$
⟨*proof*⟩

lemma *primroot-exp-nemp* [*intro*]: $u \neq \varepsilon \Longrightarrow 0 < e_\varrho\ u$
⟨*proof*⟩

lemma *primroot-nemp*[*intro*!]: $x \neq \varepsilon \Longrightarrow \varrho\ x \neq \varepsilon$
⟨*proof*⟩

lemma *primroot-idemp*[*simp*]: $\varrho\ (\varrho\ x) = \varrho\ x$
⟨*proof*⟩

lemma *prim-primroot-conv*: **assumes** $w \neq \varepsilon$ **shows** *primitive* $w \longleftrightarrow \varrho\ w = w$
⟨*proof*⟩

lemma *not-prim-primroot-expE*: **assumes** $\neg$ *primitive* w
obtains k **where** $\varrho\ w\ ^{@}k = w$ **and** $2 \leq k$
⟨*proof*⟩

lemma *not-prim-expE*: **assumes** $\neg$ *primitive* x **and** $x \neq \varepsilon$
obtains $r\ k$ **where** *primitive* r **and** $2 \leq k$ **and** $r^{@}k = x$
⟨*proof*⟩

lemma *primroot-of-root*: **assumes** $u \neq \varepsilon$ **and** $u \in q*$ **shows** $\varrho\ q = \varrho\ u$
⟨*proof*⟩

lemma *primroot-shorter-root*: **assumes** $u \neq \varepsilon$ **and** $u \in q*$ **shows** $|\varrho\ u| \leq |q|$
⟨*proof*⟩

lemma *primroot-len-le*: $u \neq \varepsilon \Longrightarrow |\varrho\ u| \leq |u|$
⟨*proof*⟩

lemma *primroot-take*: **assumes** $u \neq \varepsilon$ **shows** $\varrho\ u = (take\ (\ |\varrho\ u|\ )\ u)$
⟨*proof*⟩

lemma *primroot-rotate-comm*: **assumes** $w \neq \varepsilon$ **shows** $\varrho\ (rotate\ n\ w) = rotate\ n\ (\varrho\ w)$
⟨*proof*⟩

lemma *primroot-rotate*: $\varrho\ w = r \longleftrightarrow \varrho\ (rotate\ (k*|r|)\ w) = r$ (**is** *?L* $\longleftrightarrow$ *?R*)
⟨*proof*⟩

lemma *primrootI*[*intro*]: **assumes** *pow*: $u = r^{@}(Suc\ k)$ **and** *primitive* r **shows** $\varrho\ u = r$
⟨*proof*⟩

lemma *primroot-pref*: $\varrho\ u \leq p\ u$
⟨*proof*⟩

lemma *short-primroot*: **assumes** $u \neq \varepsilon\ \neg$ *primitive* u **shows** $|\varrho\ u| < |u|$
⟨*proof*⟩

lemma *prim-primroot-cases*: **obtains** $u = \varepsilon$ | *primitive* u | $|\varrho\ u| < |u|$
⟨*proof*⟩

We also have the standard characterization of commutation for nonempty words.

lemma *comm-rootE*: **assumes** $x \cdot y = y \cdot x$
obtains t **where** $x \in t*$ **and** $y \in t*$ **and** $t \neq \varepsilon$
⟨*proof*⟩

theorem *comm-primroots*: **assumes** $u \neq \varepsilon$ **and** $v \neq \varepsilon$ **shows** $u \cdot v = v \cdot u \longleftrightarrow \varrho\ u = \varrho\ v$
⟨*proof*⟩

lemma *comm-primroots'*: $u \neq \varepsilon \Longrightarrow v \neq \varepsilon \Longrightarrow u \cdot v = v \cdot u \Longrightarrow \varrho\ u = \varrho\ v$
⟨*proof*⟩

lemma *same-primroots-comm*: $\varrho\ x = \varrho\ y \Longrightarrow x \cdot y = y \cdot x$
⟨*proof*⟩

lemma *pow-primroot*: **assumes** $x \neq \varepsilon$ **shows** $\varrho\ (x^{@}Suc\ k) = \varrho\ x$
⟨*proof*⟩

lemma *comm-primroot-exp*: **assumes** $v \neq \varepsilon$ **and** $u \cdot v = v \cdot u$
obtains n **where** $(\varrho\ v)^{@}n = u$
⟨*proof*⟩

lemma *comm-primrootE*: **assumes** $x \cdot y = y \cdot x$
obtains t **where** $x \in t*$ **and** $y \in t*$ **and** *primitive t*
⟨*proof*⟩

lemma *primE*: **obtains** t **where** *primitive t*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *comm-primroot-conv*: $u \cdot v = v \cdot u \longleftrightarrow u \cdot \varrho\ v = \varrho\ v \cdot u$
⟨*proof*⟩

lemma *comm-primroot* [*simp*, *intro*]: $u \cdot \varrho\ u = \varrho\ u \cdot u$
⟨*proof*⟩

lemma *comp-primroot-conv'*: **shows** $u \cdot v = v \cdot u \longleftrightarrow \varrho\ u \cdot \varrho\ v = \varrho\ v \cdot \varrho\ u$
⟨*proof*⟩

lemma *per-root-primroot*: $w <p\ r \cdot w \Longrightarrow w <p\ \varrho\ r \cdot w$
⟨*proof*⟩

lemma *primroot-per-root*: $r \neq \varepsilon \Longrightarrow r <p\ \varrho\ r \cdot r$
⟨*proof*⟩

lemma *prim-comm-short-emp*: **assumes** *primitive p* **and** $u{\cdot}p{=}p{\cdot}u$ **and** $|u| < |p|$
shows $u = \varepsilon$
⟨*proof*⟩

lemma *primroot-rev*[*reversal-rule*]: **shows** $\varrho\ (rev\ u) = rev\ (\varrho\ u)$
⟨*proof*⟩

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*⟩

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** $s \neq \varepsilon$
⟨*proof*⟩

2.21.1 Primitivity and the shortest period

lemma *min-per-primitive*: **assumes** $w \neq \varepsilon$ **shows** *primitive* $(\pi\ w)$
⟨*proof*⟩

lemma *min-per-short-primroot*: **assumes** $w \neq \varepsilon$ **and** $(\varrho\ w)^{@}k = w$ **and** $k \neq 1$
shows $\pi\ w = \varrho\ w$
⟨*proof*⟩

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*⟩

2.22 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** ‹∼› *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** $s \cdot r = v$
⟨*proof*⟩

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** $s \neq \varepsilon$
⟨*proof*⟩

lemma *conjugE1* [*elim*]:
assumes $u \sim v$
obtains r **where** $u \cdot r = r \cdot v$
⟨*proof*⟩

lemma *conjug-rev-conv* [*reversal-rule*]: *rev* $u \sim$ *rev* $v \longleftrightarrow u \sim v$

⟨*proof*⟩

lemma *conjug-rotate-iff*: $u \sim v \longleftrightarrow (\exists\ n.\ v = rotate\ n\ u)$
⟨*proof*⟩

lemma *rotate-conjug*: $w \sim rotate\ n\ w$
⟨*proof*⟩

lemma *conjug-rotate-iff-le*:
shows $u \sim v \longleftrightarrow (\exists\ n \leq |u| - 1.\ v = rotate\ n\ u)$
⟨*proof*⟩

lemma *conjugI* [*intro*]: $r \cdot s = u \Longrightarrow s \cdot r = v \Longrightarrow u \sim v$
⟨*proof*⟩

lemma *conjugI'* [*intro*!]: $r \cdot s \sim s \cdot r$
⟨*proof*⟩

lemma *conjug-refl*: $u \sim u$
⟨*proof*⟩

lemma *conjug-sym*[*sym*]: $u \sim v \Longrightarrow v \sim u$
⟨*proof*⟩

lemma *conjug-swap*: $u \sim v \longleftrightarrow v \sim u$
⟨*proof*⟩

lemma *conjug-nemp-iff*: $u \sim v \Longrightarrow u = \varepsilon \longleftrightarrow v = \varepsilon$
⟨*proof*⟩

lemma *conjug-len*: $u \sim v \Longrightarrow |u| = |v|$
⟨*proof*⟩

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*⟩

lemma *conjug-set*: **assumes** $u \sim v$ **shows** $set\ u = set\ v$
⟨*proof*⟩

lemma *conjug-concat-conjug*: $xs \sim ys \Longrightarrow concat\ xs \sim concat\ ys$
⟨*proof*⟩

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** $v \neq \varepsilon$
$\langle proof \rangle$

theorem *conjugation*: **assumes** $x{\cdot}z = z{\cdot}y$ **and** $x \neq \varepsilon$
shows $\exists\ u\ v\ k.\ u \cdot v = x \wedge v \cdot u = y \wedge (u \cdot v)^{@}k \cdot u = z$
$\langle proof \rangle$

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* $(r \cdot s)$
$\langle proof \rangle$

lemma *conjugI1* [*intro*]:
assumes *eq*: $u \cdot r = r \cdot v$
shows $u \sim v$
$\langle proof \rangle$

lemma *pow-conjug-conjug-conv*: **assumes** $0 < k$ **shows** $u^{@}k \sim v^{@}k \longleftrightarrow u \sim v$
$\langle proof \rangle$

lemma *conjug-trans* [*trans*]:
assumes *uv*: $u \sim v$ **and** *vw*: $v \sim w$
shows $u \sim w$
$\langle proof \rangle$

lemma *conjug-trans′*: **assumes** *uv′*: $u \cdot r = r \cdot v$ **and** *vw′*: $v \cdot s = s \cdot w$ **shows** $u \cdot (r \cdot s) = (r \cdot s) \cdot w$
$\langle proof \rangle$

Of course, conjugacy is an equivalence relation.

lemma *conjug-equiv*: *equivp* $(\sim)$
$\langle proof \rangle$

lemma *rotate-fac-pref*: **assumes** $u \le f\ w$
obtains w' **where** $w' \sim w$ **and** $u \le p\ w'$
$\langle proof \rangle$

lemma *rotate-into-pos-sq*: **assumes** $s{\cdot}p \le f\ w{\cdot}w$ **and** $|s| \le |w|$ **and** $|p| \le |w|$
obtains w' **where** $w \sim w'\ p \le p\ w'\ s \le s\ w'$
$\langle proof \rangle$

lemma *rotate-into-pref-sq*: **assumes** $p \le f\ w{\cdot}w$ **and** $|p| \le |w|$
obtains w' **where** $w \sim w'\ p \le p\ w'$
$\langle proof \rangle$

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*⟩

lemma *rotate-into-pos-conjug*: **assumes** $w \sim v$ **and** $s{\cdot}p \le f\ v$
obtains w' **where** $w \sim w'$ $p \le p\ w'$ $s \le s\ w'$
⟨*proof*⟩

lemma *nconjug-neq*: $\neg\ u \sim v \Longrightarrow u \neq v$
⟨*proof*⟩

lemma *prim-conjug*:
assumes *prim*: *primitive* u **and** *conjug*: $u \sim v$
shows *primitive* v
⟨*proof*⟩

lemma *conjug-prim-iff*: **assumes** $u \sim v$ **shows** *primitive* u = *primitive* v
⟨*proof*⟩

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* $(s \cdot r)$
⟨*proof*⟩

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*⟩

lemma *root-conjug*: $u \le p\ r \cdot u \Longrightarrow u^{-1>}(r{\cdot}u) \sim r$
⟨*proof*⟩

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*⟩

lemma *conjug-primroot*:
assumes $u \sim v$
shows $\varrho\ u \sim \varrho\ v$
⟨*proof*⟩

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*⟩

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*⟩

lemma *conjug-add-exp*: $u \sim v \Longrightarrow u^{@}k \sim v^{@}k$
⟨*proof*⟩

lemma *conjug-primroot-iff*:
assumes *nemp*:$u \neq \varepsilon$ **and** *len*: $|u| = |v|$
shows $\varrho\ u \sim \varrho\ v \longleftrightarrow u \sim v$
⟨*proof*⟩

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*⟩

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* $(s \cdot r)$
⟨*proof*⟩

lemma *fac-pow-pref-conjug*:
assumes $u \leq f\ t^{@}k$
obtains t' **where** $t \sim t'$ **and** $u \leq p\ t'^{@}k$
⟨*proof*⟩

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*⟩

lemma *conjug-fac-sq*:
$u \sim v \Longrightarrow u \leq f\ v \cdot v$
⟨*proof*⟩

lemma *conjug-fac-pow-conv*: **assumes** $|u| = |v|$ **and** $2 \leq k$
shows $u \sim v \longleftrightarrow u \leq f\ v^{@}k$
⟨*proof*⟩

lemma *conjug-fac-Suc*: **assumes** $t \sim v$
shows $t^{@}k \leq f\ v^{@}Suc\ k$
⟨*proof*⟩

lemma *fac-pow-conjug*: **assumes** $u \leq f\ v^{@}k$ **and** $t \sim v$
shows $u \leq f\ t^{@}Suc\ k$
⟨*proof*⟩

lemma *border-conjug*: $x \leq b\ w \Longrightarrow w^{<-1}x \sim x^{-1>}w$
⟨*proof*⟩

lemma *count-list-conjug*: **assumes** $u \sim v$ **shows** *count-list* $u\ a$ = *count-list* $v\ a$
⟨*proof*⟩

lemma *conjug-in-lists*: $us \sim vs \Longrightarrow vs \in$ *lists* $A \Longrightarrow us \in$ *lists* A
⟨*proof*⟩

lemma *conjug-in-lists'*: $us \sim vs \Longrightarrow us \in$ *lists* $A \Longrightarrow vs \in$ *lists* A
⟨*proof*⟩

lemma *conjug-in-lists-iff*: $us \sim vs \Longrightarrow us \in$ *lists* $A \longleftrightarrow vs \in$ *lists* A
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

lemma *fac-per-conjug*: **assumes** *period* $w\ n$ **and** $v \leq f\ w$ **and** $|v| = n$
shows $v \sim$ *take* $n\ w$
⟨*proof*⟩

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 $v \sim u$
⟨*proof*⟩

lemma *conjug-pow-powE*: **assumes** $w \sim r^{@}k$ **obtains** s **where** $w = s^{@}k$
⟨*proof*⟩

lemma *find-second-letter*: **assumes** $a \neq b$ **and** *set ws* $= \{a,b\}$
shows *dropWhile* $(\lambda\ c.\ c = a)$ *ws* $\neq \varepsilon$ **and** *hd* (*dropWhile* $(\lambda\ c.\ c = a)$ *ws*) $= b$
⟨*proof*⟩

lemma *fac-conjuq-sq*:
assumes $u \sim v$ **and** $|w| \leq |u|$ **and** $w \leq f\ u \cdot u$
shows $w \leq f\ v \cdot v$
⟨*proof*⟩

lemma *fac-conjuq-sq-iff*:
assumes $u \sim v$ **shows** $|w| \leq |u| \Longrightarrow w \leq f\ u \cdot u \longleftrightarrow w \leq f\ v \cdot v$
⟨*proof*⟩

lemma *map-conjug*:
$u \sim v \Longrightarrow$ *map* $f\ u \sim$ *map* $f\ v$
⟨*proof*⟩

lemma *map-conjug-iff* [*reversal-rule*]:
assumes *inj* f **shows** *map* $f\ u \sim$ *map* $f\ v \longleftrightarrow u \sim v$
⟨*proof*⟩

lemma *card-conjug*: **assumes** $w \neq \varepsilon$
shows *card* (*Collect* (*conjugate* w)) $= |\varrho\ w|$
⟨*proof*⟩

lemma *finite-Bex-conjug*: **assumes** *finite* A
shows *finite* $\{r.\ Bex\ A\ (conjugate\ r)\}$

⟨*proof*⟩

2.22.1 Enumerating conjugates

definition *bounded-conjug*
where $bounded\text{-}conjug\ w'\ w\ k \equiv (\exists\ n \leq k.\ w = rotate\ n\ w')$

named-theorems *bounded-conjug*

lemma[*bounded-conjug*]: $bounded\text{-}conjug\ w'\ w\ 0 \longleftrightarrow w = w'$
⟨*proof*⟩

lemma[*bounded-conjug*]: $bounded\text{-}conjug\ w'\ w\ (Suc\ k) \longleftrightarrow bounded\text{-}conjug\ w'\ w\ k \lor w = rotate\ (Suc\ k)\ w'$
⟨*proof*⟩

lemma[*bounded-conjug*]: $w' \sim w \longleftrightarrow bounded\text{-}conjug\ w\ w'\ (|w|-1)$
⟨*proof*⟩

lemma $w \sim [a,b,c] \longleftrightarrow w = [a,b,c] \lor w = [b,c,a] \lor w = [c,a,b]$
⟨*proof*⟩

2.22.2 General lemmas using conjugation

lemma *switch-fac*: **assumes** $x \neq y$ **and** $set\ ws = \{x,y\}$ **shows** $[x,y] \leq f\ ws \cdot ws$
⟨*proof*⟩

lemma *imprim-ext-pref-comm*: **assumes** $\neg\ primitive\ (u \cdot v)$ **and** $\neg\ primitive\ (u \cdot v \cdot u)$
shows $u \cdot v = v \cdot u$
⟨*proof*⟩

lemma *imprim-ext-suf-comm*:
$\neg\ primitive\ (u \cdot v) \Longrightarrow \neg\ primitive\ (u \cdot v \cdot v) \Longrightarrow u \cdot v = v \cdot u$
⟨*proof*⟩

lemma *prim-xyky*: **assumes** $2 \leq k$ **and** $\neg\ primitive\ ((x \cdot y)^{@}k \cdot y)$ **shows** $x \cdot y = y \cdot x$
⟨*proof*⟩

lemma *fac-pow-div*: **assumes** $u \leq f\ w^{@}l$ $primitive\ w$
shows $w^{@}((|u|\ div\ |w|) - 1) \leq f\ u$
⟨*proof*⟩

2.23 Element of lists: a method for testing if a word is in lists A

lemma *append-in-lists*[*simp*, *intro*]: $u \in lists\ A \Longrightarrow v \in lists\ A \Longrightarrow u \cdot v \in lists\ A$
⟨*proof*⟩

lemma *pref-in-lists*: $u \leq p\ v \Longrightarrow v \in lists\ A \Longrightarrow u \in lists\ A$
⟨*proof*⟩

lemmas *suf-in-lists* = *pref-in-lists*[*reversed*]

lemma *fac-in-lists*: $ws \in lists\ S \Longrightarrow vs \leq f\ ws \Longrightarrow vs \in lists\ S$
⟨*proof*⟩

lemma *lq-in-lists*: $v \in lists\ A \Longrightarrow u^{-1>}v \in lists\ A$
⟨*proof*⟩

lemmas *rq-in-lists* = *lq-in-lists*[*reversed*]

lemma *take-in-lists*: $w \in lists\ A \Longrightarrow take\ j\ w \in lists\ A$
⟨*proof*⟩

lemma *drop-in-lists*: $w \in lists\ A \Longrightarrow drop\ j\ w \in lists\ A$
⟨*proof*⟩

lemma *lcp-in-lists*: $u \in lists\ A \Longrightarrow\ u \wedge_p v \in lists\ A$
⟨*proof*⟩

lemma *lcp-in-lists'*: $v \in lists\ A \Longrightarrow\ u \wedge_p v \in lists\ A$
⟨*proof*⟩

lemma *append-in-lists-dest*: $u \cdot v \in lists\ A \Longrightarrow u \in lists\ A$
⟨*proof*⟩

lemma *append-in-lists-dest'*: $u \cdot v \in lists\ A \Longrightarrow v \in lists\ A$
⟨*proof*⟩

lemma *pow-in-lists*: $u \in lists\ A \Longrightarrow u^{@}k \in lists\ A$
⟨*proof*⟩

lemma *takeWhile-in-list*: $u \in lists\ A \Longrightarrow takeWhile\ P\ u \in lists\ A$
⟨*proof*⟩

lemma *rev-in-lists*: $u \in lists\ A \Longrightarrow rev\ u \in lists\ A$
⟨*proof*⟩

lemma *append-in-lists-dest1*: $u \cdot v = w \Longrightarrow w \in lists\ A \Longrightarrow u \in lists\ A$
⟨*proof*⟩

lemma *append-in-lists-dest2*: $u \cdot v = w \Longrightarrow w \in lists\ A \Longrightarrow v \in lists\ A$
⟨*proof*⟩

lemma *pow-in-lists-dest1*: $u \cdot v = w^{@}n \Longrightarrow w \in lists\ A \Longrightarrow u \in lists\ A$
⟨*proof*⟩

lemma *pow-in-lists-dest1-sym*: $w^{@}n = u \cdot v \Longrightarrow w \in lists\ A \Longrightarrow u \in lists\ A$
⟨*proof*⟩

lemma *pow-in-lists-dest2*: $u \cdot v = w^{@}n \Longrightarrow w \in lists\ A \Longrightarrow v \in lists\ A$
⟨*proof*⟩

lemma *pow-in-lists-dest2-sym*: $w^{@}n = u \cdot v \Longrightarrow w \in lists\ A \Longrightarrow v \in lists\ A$
⟨*proof*⟩

lemma *per-in-lists*: $w <p\ r \cdot w \Longrightarrow r \in lists\ A \Longrightarrow w \in lists\ A$
⟨*proof*⟩

lemma *nth-in-lists*: $j < |w| \Longrightarrow w \in lists\ A \Longrightarrow w\ !\ j \in A$
⟨*proof*⟩

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 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*))+›)

2.24 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*
⟨*proof*⟩

lemma *rev-map-arg*: *rev-map f u* = *rev* (*f* (*rev u*))
⟨*proof*⟩

lemma *rev-map-arg'*: *rev* ((*rev-map f*) *w*) = *f* (*rev w*)
⟨*proof*⟩

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*])
⟨*proof*⟩

lemma *rev-maps-eq-iff*[*simp*]: *rev-map g* = *rev-map h* ⟷ *g* = *h*
⟨*proof*⟩

lemma *rev-map-funpow*[*reversal-rule*]: (*rev-map* (f::'*a list* ⇒'*a list*)) $\frown\frown k$ = *rev-map* ($f \frown\frown k$)
⟨*proof*⟩

2.25 Overlapping powers, periods, prefixes and suffixes

lemma *pref-suf-overlapE*: **assumes** $p \le p\ w$ **and** $s \le s\ w$ **and** $|w| \le |p| + |s|$
obtains *p1 u s1* **where** $p1 \cdot u \cdot s1 = w$ **and** $p1 \cdot u = p$ **and** $u \cdot s1 = s$
⟨*proof*⟩

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*⟩

lemma *mid-sq'*: **assumes** $p{\cdot}x{\cdot}q{=}x{\cdot}x$ **shows** $q \cdot p = x$ **and** $p \cdot q = x$
⟨*proof*⟩

lemma *mid-sq-pref*: $p \cdot u \le p\ u \cdot u \Longrightarrow p \cdot u = u \cdot p$
⟨*proof*⟩

lemmas *mid-sq-suf* = *mid-sq-pref*[*reversed*]

lemma *mid-sq-pref-suf*: **assumes** $p{\cdot}x{\cdot}q{=}x{\cdot}x$ **shows** $p \le p\ x$ **and** $p \le s\ x$ **and** $q \le p\ x$ **and** $q \le s\ x$
⟨*proof*⟩

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*⟩

lemma *root-suf-comm*: **assumes** $x \le p\ r \cdot x$ **and** $r \le s\ r \cdot x$ **shows** $r \cdot x = x \cdot r$
⟨*proof*⟩

lemma *pref-marker*: **assumes** $w \le p\ v \cdot w$ **and** $u \cdot v \le p\ w$
shows $u \cdot v = v \cdot u$
⟨*proof*⟩

lemma *pref-marker-ext*: **assumes** $|x| \le |y|$ **and** $v \ne \varepsilon$ **and** $y \cdot v \le p\ x \cdot v^{@}k$
obtains n **where** $y = x \cdot (\varrho\ v)^{@}n$
⟨*proof*⟩

lemma *pref-marker-sq*: $p \cdot x \le p\ x \cdot x \Longrightarrow p \cdot x = x \cdot p$
⟨*proof*⟩

lemmas *suf-marker-sq* = *pref-marker-sq*[*reversed*]

lemma *pref-marker-conjug*: **assumes** $w \ne \varepsilon$ **and** $w \cdot r \cdot s \le p\ s \cdot (r \cdot s)^{@}m$ **and**

primitive $(r \cdot s)$
obtains n **where** $w = s \cdot (r \cdot s)^@n$
⟨*proof*⟩

lemmas *pref-marker-reversed* = *pref-marker*[*reversed*]

lemma *suf-marker-per-root*: **assumes** $w \le p\ v \cdot w$ **and** $p \cdot v \cdot u \le p\ w$
shows $u \le p\ v \cdot u$
⟨*proof*⟩

lemma *suf-marker-per-root'*: **assumes** $w \le p\ v \cdot w$ **and** $p \cdot v \cdot u \le p\ w$ **and** $v \neq \varepsilon$
shows $u \le p\ p \cdot u$
⟨*proof*⟩

lemma *marker-fac-pref*: **assumes** $u \le f\ r^@k$ **and** $r \le p\ u$ **shows** $u \le p\ r^@k$
⟨*proof*⟩

lemma *marker-fac-pref-len*: **assumes** $u \le f\ r^@k$ **and** $t \le p\ u$ **and** $|t| = |r|$
shows $u \le p\ t^@k$
⟨*proof*⟩

lemma *root-suf-comm'*: $x \le p\ r \cdot x \Longrightarrow r \le s\ x \Longrightarrow r \cdot x = x \cdot r$
⟨*proof*⟩

lemmas *suf-root-pref-comm* = *root-suf-comm'*[*reversed*]

lemma *marker-pref-suf-fac*: **assumes** $u \le p\ v$ **and** $u \le s\ v$ **and** $v \le f\ u^@k$
shows $u \cdot v = v \cdot u$
⟨*proof*⟩

lemma *pref-suf-per-fac-comm*:
assumes $v \le p\ u \cdot v$ **and** $v \le s\ v \cdot u$ **and** $u \le f\ v^@k$ **shows** $u \cdot v = v \cdot u$
⟨*proof*⟩

lemma *mid-long-pow*: **assumes** *eq*: $y^@m = u \cdot x^@(Suc\ k) \cdot v$ **and** $|y| \le |x^@k|$
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*⟩

lemma *mid-pow-pref-suf'*: **assumes** $s{\cdot}w^@(Suc\ l){\cdot}p \le f\ w^@k$ **shows** $p \le p\ w^@k$ **and** $s \le s\ w^@k$
⟨*proof*⟩

lemma *mid-pow-pref-suf*: **assumes** $s{\cdot}w{\cdot}p \le f\ w^@k$ **shows** $p \le p\ w^@k$ **and** $s \le s\ w^@k$
⟨*proof*⟩

lemma *fac-marker-pref*: $y \cdot x \le f\ y^@k \Longrightarrow x \le p\ y \cdot x$
⟨*proof*⟩

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

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*⟩

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*) $p = r$
  ⟨*proof*⟩

lemma *prim-overlap-sq*:
  assumes *prim*: *primitive* r **and** *eq*: $p \cdot r \cdot q = r \cdot r$
  shows $p = \varepsilon \lor q = \varepsilon$
  ⟨*proof*⟩

lemma *prim-overlap-sq′*:
  assumes *prim*: *primitive* r **and** *pref*: $p \cdot r \le p\ r \cdot r$ **and** *len*: $|p| < |r|$
  shows $p = \varepsilon$
  ⟨*proof*⟩

lemma *prim-overlap-pow*:
  assumes *prim*: *primitive* r **and** *pref*: $u \cdot r \le p\ r^{@}k$
  obtains i **where** $u = r^{@}i$ **and** $i < k$
⟨*proof*⟩

lemma *prim-overlap-pow′*:
  assumes *prim*: *primitive* r **and** *pref*: $u \cdot r \le p\ r^{@}k$ **and** *less*: $|u| < |r|$
  shows $u = \varepsilon$
⟨*proof*⟩

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*⟩

lemma *drop-pref-prim*: **assumes** *Suc* $n < |w|$ **and** $w \le p$ *drop* (*Suc* n) ($w \cdot w$) **and** *primitive* w
  shows *False*
  ⟨*proof*⟩

lemma *root-suf-conjug*: **assumes** *primitive* $(s \cdot r)$ **and** $y \le p\ (s \cdot r) \cdot y$ **and** $y \le s\ y \cdot (r \cdot s)$ **and** $|s \cdot r| \le |y|$
    obtains l **where** $y = (s \cdot r)^{@}l \cdot s$
⟨*proof*⟩

lemma *pref-suf-pows-comm*: **assumes** $x \le p\ y^{@}(Suc\ k) \cdot x^{@}l$ **and** $y \le s\ y^{@}m \cdot x^{@}(Suc\ n)$
shows $x \cdot y = y \cdot x$
⟨*proof*⟩

lemma *root-suf-pow-comm*: **assumes** $x \le p\ r \cdot x$ **and** $r \le s\ x^{@}(Suc\ k)$ **shows** $r \cdot x = x \cdot r$
⟨*proof*⟩

lemma *suf-pow-short-suf*: $r \le s\ x^{@}k \Longrightarrow |x| \le |r| \Longrightarrow x \le s\ r$
⟨*proof*⟩

thm *suf-pow-short-suf*[*reversed*]

lemma *sq-short-per*: **assumes** $|u| \le |v|$ **and** $v \cdot v \le p\ u \cdot (v \cdot v)$
shows $u \cdot v = v \cdot u$
⟨*proof*⟩

lemma *fac-marker*: **assumes** $w \le p\ u \cdot w$ **and** $u \cdot v \cdot u \le f\ w$
shows $u \cdot v = v \cdot u$
⟨*proof*⟩

lemma $4 = Suc(Suc(Suc(Suc\ 0)))$
⟨*proof*⟩

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*⟩

lemma *per-glue*: **assumes** *period* $u\ n$ **and** *period* $v\ n$ **and** $u \le p\ w$ **and** $v \le s\ w$ **and**
$|w| + n \le |u| + |v|$
shows *period* $w\ n$
⟨*proof*⟩

lemma *per-glue-facs*: **assumes** $u \cdot z \le f\ w^{@}k$ **and** $z \cdot v \le f\ w^{@}m$ **and** $|w| \le |z|$
obtains l **where** $u \cdot z \cdot v \le f\ w^{@}l$
⟨*proof*⟩

lemma *per-fac-pow-fac*: **assumes** *period* $w\ n$ **and** $v \le f\ w$ **and** $|v| = n$
obtains k **where** $w \le f\ v^{@}k$
⟨*proof*⟩

lemma *refine-per*: **assumes** *period* $w\ n$ **and** $v \le f\ w$ **and** $n \le |v|$ **and** *period* $v\ k$ **and** k *dvd* n
shows *period* $w\ k$
⟨*proof*⟩

lemma *xy-per-comp*: **assumes** $x{\cdot}y \le p\ q{\cdot}x{\cdot}y$
and $q \neq \varepsilon$ **and** $q \bowtie y$
shows $x \bowtie y$
⟨*proof*⟩

lemma *prim-xyxyy*: $x \cdot y \neq y \cdot x \Longrightarrow$ *primitive* $(x \cdot y \cdot x \cdot y \cdot y)$
⟨*proof*⟩

lemma *prim-min-per-suf-eq*: **assumes** *primitive* x **and** $\pi\ x \le s\ x$ **shows** $\pi\ x = x$
⟨*proof*⟩

lemma *primroot-code*[*code*]: $\varrho\ x = ($*if* $x \neq \varepsilon$ *then* $($*if* $\pi\ x \le s\ x$ *then* $\pi\ x$ *else* $x)$ *else* *Code.abort* (*STR ''Empty word has no primitive root.''*) (λ-. ($\varrho\ x$)))
⟨*proof*⟩

lemma *per-lemma-pref-suf*: **assumes** $w <p\ p \cdot w$ **and** $w <s\ w \cdot q$ **and**
fw: $|p| + |q| \le |w|$
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*⟩

lemma *fac-two-conjug-primroot*:
assumes *facs*: $w \le f\ p^{@}k\ w \le f\ q^{@}l$ **and** *nemps*: $p \neq \varepsilon\ q \neq \varepsilon$ **and** *len*: $|p| + |q| \le |w|$
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*⟩

corollary *fac-two-conjug-primroot'*:
assumes *facs*: $u \le f\ r^{@}k\ u \le f\ s^{@}l$ **and** *nemps*: $r \neq \varepsilon\ s \neq \varepsilon$ **and** *len*: $|r| + |s| \le |u|$
shows $\varrho\ r \sim \varrho\ s$
⟨*proof*⟩

lemma *fac-two-conjug-primroot''*:
assumes *facs*: $u \le f\ r^{@}k\ u \le f\ s^{@}l$ **and** $u \neq \varepsilon$ **and** *len*: $|r| + |s| \le |u|$
shows $\varrho\ r \sim \varrho\ s$
⟨*proof*⟩

lemma *fac-two-prim-conjug*:
assumes $w \le f\ u^{@}n\ w \le f\ v^{@}m$ *primitive* u *primitive* v $|u| + |v| \le |w|$
shows $u \sim v$
⟨*proof*⟩

lemma *fac-pow-conjug-primroot*: **assumes** $u^{@}k \le f\ v^{@}l$ **and** $|u^{@}k| \ge 2{*}|v|$ **and** $2 \le k$ **and** $u \neq \varepsilon$
shows $\varrho\ u \sim \varrho\ v$
⟨*proof*⟩

2.26 Testing primitivity

This section defines a proof method used to prove that a word is primitive.

lemma *primitive-iff* [*code*]: *primitive w* $\longleftrightarrow$ $\neg$ *w* $\leq f$ *tl w* $\cdot$ *butlast w*
⟨*proof*⟩

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*⟩

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*⟩

lemma *ext-suf-Cons*:
$Suc\ i + |u| = |w| \Longrightarrow u \leq s\ w \Longrightarrow (w!i)\#u \leq s\ (w!i)\#w$
⟨*proof*⟩

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*⟩

lemma *ext-border-Cons*:
  $Suc\ i + |u| = |w| \Longrightarrow u \leq b\ w \Longrightarrow (w!i)\#u \leq b\ (w!i)\#w$
  ⟨*proof*⟩

lemma *border-add-Cons-len*: **assumes** *max-borderP u w* **and** $v \leq b\ (a\#w)$ **shows** $|v| \leq Suc\ |u|$
⟨*proof*⟩

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*))
⟨*proof*⟩
termination
⟨*proof*⟩

lemma *KMP-len*: |*KMP w arr bord pos*| = |*arr*| + *pos*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *KMP-valid-base*: **assumes** *w* ≠ *ε* **shows** *KMP-valid w* [*0*] *0* (|*w*|−*1*)
⟨*proof*⟩

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*⟩

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*)
⟨*proof*⟩

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*|
⟨*proof*⟩

theorem *bord-array*: **assumes** *Suc k* ≤ |*w*| **shows** (*border-array w*)!*k* = |*max-border*

(take (Suc k) w)|
⟨*proof*⟩

lemma *max-border-comp* [*code*]: *max-border w* = *take* ((*border-array w*)!(|*w*|−*1*)) *w*
⟨*proof*⟩

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*,*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*
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 Hull

First, we define the hull of a set of words, that is, the monoid generated by them.

inductive-set *hull* :: $'a\ list\ set \Rightarrow {}'a\ list\ set$ (‹⟨-⟩›)
  for G **where**
    emp-in[*simp*]: $\varepsilon \in \langle G\rangle$ |
    prod-cl: $w1 \in G \Longrightarrow w2 \in \langle G\rangle \Longrightarrow w1 \cdot w2 \in \langle G\rangle$

lemmas [*intro*] = *hull.intros*

lemma *hull-closed*[*intro*]: $w1 \in \langle G\rangle \Longrightarrow w2 \in \langle G\rangle \Longrightarrow w1 \cdot w2 \in \langle G\rangle$
  ⟨*proof*⟩

lemma *gen-in* [*intro*]: $w \in G \Longrightarrow w \in \langle G\rangle$
  ⟨*proof*⟩

lemma *hull-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** $P\ x$
  ⟨*proof*⟩

lemma *genset-sub*[*simp*]: $G \subseteq \langle G\rangle$
  ⟨*proof*⟩

lemma *genset-sub-lists*: $ws \in lists\ G \Longrightarrow ws \in lists\ \langle G\rangle$
  ⟨*proof*⟩

lemma *in-lists-conv-set-subset*: $set\ ws \subseteq G \longleftrightarrow ws \in lists\ G$

⟨*proof*⟩

lemma *concat-in-hull* [*intro*]:
assumes *set ws* $\subseteq$ *G*
shows *concat ws* $\in \langle G \rangle$
⟨*proof*⟩

lemma *concat-in-hull'* [*intro*]:
assumes *ws* $\in$ *lists G*
shows *concat ws* $\in \langle G \rangle$
⟨*proof*⟩

lemma *hull-concat-lists0*: $w \in \langle G \rangle \Longrightarrow (\exists\ ws \in lists\ G.\ concat\ ws = w)$
⟨*proof*⟩

lemma *hull-concat-listsE*: **assumes** $w \in \langle G \rangle$
obtains *ws* **where** *ws* $\in$ *lists G* **and** *concat ws* $= w$
⟨*proof*⟩

lemma *hull-concat-lists*: $\langle G \rangle = concat$ ‘ *lists G*
⟨*proof*⟩

lemma *concat-tl*: x # $xs \in lists\ G \Longrightarrow concat\ xs \in \langle G \rangle$
⟨*proof*⟩

lemma *nemp-concat-hull*: **assumes** $us \neq \varepsilon$ **and** $us \in lists\ (G - \{\varepsilon\})$
shows *concat us* $\in \langle G \rangle$ **and** *concat us* $\neq \varepsilon$
⟨*proof*⟩

lemma *hull-mono*: $A \subseteq B \Longrightarrow \langle A \rangle \subseteq \langle B \rangle$
⟨*proof*⟩

lemma *emp-gen-set*: $\langle \{\} \rangle = \{\varepsilon\}$
⟨*proof*⟩

lemma *concat-lists-minus*[*simp*]: *concat* ‘ *lists* $(G - \{\varepsilon\})$ = *concat* ‘ *lists G*
⟨*proof*⟩

lemma *hull-drop-one*: $\langle G - \{\varepsilon\} \rangle = \langle G \rangle$
⟨*proof*⟩

lemma *sing-gen-power*: $u \in \langle \{x\} \rangle \Longrightarrow \exists\, k.\ u = x^{@}k$
⟨*proof*⟩

lemma *sing-gen*[*intro*]: $w \in \langle \{z\} \rangle \Longrightarrow w \in z*$
⟨*proof*⟩

lemma *pow-sing-gen*[*simp*]: $x^{@}k \in \langle \{x\} \rangle$
⟨*proof*⟩

lemma *root-sing-gen*: $w \in z* \Longrightarrow w \in \langle\{z\}\rangle$
⟨*proof*⟩

lemma *sing-genE*[*elim*]:
assumes $u \in \langle\{x\}\rangle$
obtains k **where** $x^{@}k = u$
⟨*proof*⟩

lemma *sing-gen-root-conv*: $w \in \langle\{z\}\rangle \longleftrightarrow w \in z*$
⟨*proof*⟩

lemma *lists-gen-to-hull*: $us \in lists\ (G - \{\varepsilon\}) \Longrightarrow us \in lists\ (\langle G\rangle - \{\varepsilon\})$
⟨*proof*⟩

lemma *rev-hull*: $rev`\langle G\rangle = \langle rev`G\rangle$
⟨*proof*⟩

lemma *power-in*[*intro*]: $x \in \langle G\rangle \Longrightarrow x^{@}k \in \langle G\rangle$
⟨*proof*⟩

lemma *hull-closed-lists*: $us \in lists\ \langle G\rangle \Longrightarrow concat\ us \in \langle G\rangle$
⟨*proof*⟩

lemma *hull-I* [*intro*]:
$\varepsilon \in H \Longrightarrow (\bigwedge x\ y.\ x \in H \Longrightarrow y \in H \Longrightarrow x \cdot y \in H) \Longrightarrow \langle H\rangle = H$
⟨*proof*⟩

lemma *self-gen*: $\langle\langle G\rangle\rangle = \langle G\rangle$
⟨*proof*⟩

lemma *hull-mono'*[*intro*]: $A \subseteq \langle B\rangle \Longrightarrow \langle A\rangle \subseteq \langle B\rangle$
⟨*proof*⟩

lemma *hull-conjug* [*elim*]: $w \in \langle\{r{\cdot}s,s{\cdot}r\}\rangle \Longrightarrow w \in \langle\{r,s\}\rangle$
⟨*proof*⟩

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*⟩

lemma *hull-keeps-root*: $\forall\ u \in A.\ u \in r* \Longrightarrow\ w \in \langle A\rangle \Longrightarrow w \in r*$
⟨*proof*⟩

lemma *bin-hull-keeps-root* [*intro*]: $u \in r* \Longrightarrow v \in r* \Longrightarrow w \in \langle\{u,v\}\rangle \Longrightarrow w \in r*$
⟨*proof*⟩

lemma *bin-comm-hull-comm*: $x \cdot y = y \cdot x \Longrightarrow u \in \langle\{x,y\}\rangle \Longrightarrow v \in \langle\{x,y\}\rangle \Longrightarrow u \cdot v = v \cdot u$

⟨*proof*⟩

lemma[*reversal-rule*]: rev ‘ $\langle\{rev\ u,\ rev\ v\}\rangle = \langle\{u,v\}\rangle$
⟨*proof*⟩

lemma[*reversal-rule*]: $rev\ w \in \langle rev$ ‘ $G\rangle \equiv w \in \langle G\rangle$
⟨*proof*⟩

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 $G\ u = (SOME\ us.\ us \in lists\ (G - \{\varepsilon\}) \wedge concat\ us = u)$

lemma *dec-ex*: **assumes** $u \in \langle G\rangle$ **shows** $\exists\ us.\ (us \in lists\ (G - \{\varepsilon\}) \wedge concat\ us = u)$
⟨*proof*⟩

lemma *dec-in-lists'*: $u \in \langle G\rangle \Longrightarrow (Dec\ G\ u) \in lists\ (G - \{\varepsilon\})$
⟨*proof*⟩

lemma *concat-dec*[*simp*, *intro*] : $u \in \langle G\rangle \Longrightarrow concat\ (Dec\ G\ u) = u$
⟨*proof*⟩

lemma *dec-emp* [*simp*]: $Dec\ G\ \varepsilon = \varepsilon$
⟨*proof*⟩

lemma *dec-nemp*: $u \in \langle G\rangle - \{\varepsilon\} \Longrightarrow Dec\ G\ u \neq \varepsilon$
⟨*proof*⟩

lemma *dec-nemp'*[*simp*, *intro*]: $u \neq \varepsilon \Longrightarrow u \in \langle G\rangle \Longrightarrow Dec\ G\ u \neq \varepsilon$
⟨*proof*⟩

lemma *dec-eq-emp-iff* [*simp*]: **assumes** $u \in \langle G\rangle$ **shows** $Dec\ G\ u = \varepsilon \longleftrightarrow u = \varepsilon$
⟨*proof*⟩

lemma *dec-in-lists*[*simp*]: $u \in \langle G\rangle \Longrightarrow Dec\ G\ u \in lists\ G$
⟨*proof*⟩

lemma *set-dec-sub*: $x \in \langle G\rangle \Longrightarrow set\ (Dec\ G\ x) \subseteq G$
⟨*proof*⟩

lemma *dec-hd*: $u \neq \varepsilon \Longrightarrow u \in \langle G\rangle \Longrightarrow hd\ (Dec\ G\ u) \in G$
⟨*proof*⟩

lemma *non-gen-dec*: **assumes** $u \in \langle G \rangle$ $u \notin G$ **shows** *Dec G u* $\neq [u]$
⟨*proof*⟩

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* - -› [*51*,*51*] *65*)
where
refine G us = *concat*(*map* (*decompose G*) *us*)

lemma *ref-morph*: $us \in lists\ \langle G \rangle \Longrightarrow vs \in lists\ \langle G \rangle \Longrightarrow refine\ G\ (us \cdot vs) = refine\ G\ us \cdot refine\ G\ vs$
⟨*proof*⟩

lemma *ref-conjug*:
$u \sim v \Longrightarrow (Ref\ G\ u) \sim Ref\ G\ v$
⟨*proof*⟩

lemma *ref-morph-plus*: $us \in lists\ (\langle G \rangle - \{\varepsilon\}) \Longrightarrow vs \in lists\ (\langle G \rangle - \{\varepsilon\}) \Longrightarrow refine\ G\ (us \cdot vs) = refine\ G\ us \cdot refine\ G\ vs$
⟨*proof*⟩

lemma *ref-pref-mono*: $ws \in lists\ \langle G \rangle \Longrightarrow us \le p\ ws \Longrightarrow Ref\ G\ us \le p\ Ref\ G\ ws$
⟨*proof*⟩

lemma *ref-suf-mono*: $ws \in lists\ \langle G \rangle \Longrightarrow us \le s\ ws \Longrightarrow (Ref\ G\ us) \le s\ Ref\ G\ ws$
⟨*proof*⟩

lemma *ref-fac-mono*: $ws \in lists\ \langle G \rangle \Longrightarrow us \le f\ ws \Longrightarrow (Ref\ G\ us) \le f\ Ref\ G\ ws$
⟨*proof*⟩

lemma *ref-pop-hd*: $us \neq \varepsilon \Longrightarrow us \in lists\ \langle G \rangle \Longrightarrow refine\ G\ us = decompose\ G\ (hd\ us) \cdot refine\ G\ (tl\ us)$
⟨*proof*⟩

lemma *ref-in*: $us \in lists\ \langle G \rangle \Longrightarrow (Ref\ G\ us) \in lists\ (G - \{\varepsilon\})$
⟨*proof*⟩

lemma *ref-in'*[*intro*]: $us \in lists\ \langle G \rangle \Longrightarrow (Ref\ G\ us) \in lists\ G$
⟨*proof*⟩

lemma *concat-ref*: $us \in lists\ \langle G \rangle \Longrightarrow concat\ (Ref\ G\ us) = concat\ us$
⟨*proof*⟩

lemma *ref-gen*: $us \in lists\ B \Longrightarrow B \subseteq \langle G \rangle \Longrightarrow Ref\ G\ us \in \langle decompose\ G\ `\ B \rangle$
⟨*proof*⟩

lemma *ref-set*: $ws \in lists\ \langle G \rangle \Longrightarrow set\ (Ref\ G\ ws) = \bigcup\ (set`(decompose\ G)`set\ ws)$
⟨*proof*⟩

lemma *emp-ref*: **assumes** $us \in lists\ (\langle G \rangle - \{\varepsilon\})$ **and** $Ref\ G\ us = \varepsilon$ **shows** $us = \varepsilon$
⟨*proof*⟩

lemma *sing-ref-sing*:
assumes $us \in lists\ (\langle G \rangle - \{\varepsilon\})$ **and** $refine\ G\ us = [b]$
shows $us = [b]$
⟨*proof*⟩

lemma *ref-ex*: **assumes** $Q \subseteq \langle G \rangle$ **and** $us \in lists\ Q$
shows $Ref\ G\ us \in lists\ (G - \{\varepsilon\})$ **and** $concat\ (Ref\ G\ us) = concat\ us$
⟨*proof*⟩

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.

The simple element is defined as a word which has only trivial decomposition into generators: a singleton.

definition *simple-element* :: $'a\ list \Rightarrow\ 'a\ list\ set \Rightarrow bool$ (‹ - ∈B - › [51,51] 50)
where
$simple\text{-}element\ b\ G = (b \in G \wedge (\forall\ us.\ us \in lists\ (G - \{\varepsilon\}) \wedge concat\ us = b \longrightarrow |us| = 1))$

lemma *simp-el-el*: $b \in B\ G \Longrightarrow b \in G$
⟨*proof*⟩

lemma *simp-elD*: $b \in B\ G \Longrightarrow us \in lists\ (G - \{\varepsilon\}) \Longrightarrow concat\ us = b \Longrightarrow |us| = 1$
⟨*proof*⟩

lemma *simp-el-sing*: **assumes** $b \in B\ G\ us \in lists\ (G - \{\varepsilon\})\ concat\ us = b$ **shows** $us = [b]$
⟨*proof*⟩

lemma *nonsimp*: $us \in lists\ (G - \{\varepsilon\}) \Longrightarrow concat\ us \in B\ G \Longrightarrow\ us = [concat\ us]$
⟨*proof*⟩

lemma *emp-nonsimp*: **assumes** $b \in B\ G$ **shows** $b \neq \varepsilon$
⟨*proof*⟩

lemma *basis-no-fact*: **assumes** $u \in \langle G \rangle$ **and** $v \in \langle G \rangle$ **and** $u \cdot v \in B\ G$ **shows** $u = \varepsilon \vee v = \varepsilon$
⟨*proof*⟩

lemma *simp-elI*:
assumes $b \in G$ **and** $b \neq \varepsilon$ **and** *all*: $\forall\ u\ v.\ u \neq \varepsilon \wedge u \in \langle G\rangle \wedge v \neq \varepsilon \wedge v \in \langle G\rangle \longrightarrow u \cdot v \neq b$
shows $b \in B\ G$
⟨*proof*⟩

lemma *simp-el-indecomp*:
assumes $b \in B\ G\ u \neq \varepsilon\ u \in \langle G\rangle\ v \neq \varepsilon\ v \in \langle G\rangle$
shows $u \cdot v \neq b$
⟨*proof*⟩

We are ready to define the *basis* as the set of all simple elements.

definition *basis* :: $'a$ *list set* $\Rightarrow$ $'a$ *list set* (‹$\mathfrak{B}$ -› [*51*]) **where**
basis $G = \{x.\ x \in B\ G\}$

lemma *basis-inI*: $x \in B\ G \Longrightarrow x \in \mathfrak{B}\ G$
⟨*proof*⟩

lemma *basisD*: $x \in \mathfrak{B}\ G \Longrightarrow x \in B\ G$
⟨*proof*⟩

lemma *emp-not-basis*: $x \in \mathfrak{B}\ G \Longrightarrow x \neq \varepsilon$
⟨*proof*⟩

lemma *basis-sub*: $\mathfrak{B}\ G \subseteq G$
⟨*proof*⟩

lemma *basis-drop-emp*: $(\mathfrak{B}\ G) - \{\varepsilon\} = \mathfrak{B}\ G$
⟨*proof*⟩

lemma *simp-el-hull'*: **assumes** $b \in B\ \langle G\rangle$ **shows** $b \in B\ G$
⟨*proof*⟩

lemma *simp-el-hull*: **assumes** $b \in B\ G$ **shows** $b \in B\ \langle G\rangle$
⟨*proof*⟩

lemma *concat-tl-basis*: $x \# xs \in$ *lists* $\mathfrak{B}\ G \Longrightarrow$ *concat xs* $\in \langle G\rangle$
⟨*proof*⟩

The basis generates the hull

lemma *set-concat-len*: **assumes** *us* $\in$ *lists* $(G - \{\varepsilon\})$ $1 < |us|$ $u \in$ *set us* **shows** $|u| < |concat\ us|$
⟨*proof*⟩

lemma *non-simp-dec*: **assumes** $w \notin \mathfrak{B}\ G\ w \neq \varepsilon\ w \in G$
obtains *us* **where** *us* $\in$ *lists* $(G - \{\varepsilon\})$ $1 < |us|$ *concat us* $= w$
⟨*proof*⟩

lemma *basis-gen*: $w \in G \Longrightarrow \; w \in \langle \mathfrak{B}\ G \rangle$
⟨*proof*⟩

lemmas *basis-concat-listsE* = *hull-concat-listsE*[*OF basis-gen*]

theorem *basis-gen-hull*: $\langle \mathfrak{B}\ G \rangle = \langle G \rangle$
⟨*proof*⟩

lemma *basis-gen-hull'*: $\langle \mathfrak{B}\ \langle G \rangle \rangle = \langle G \rangle$
⟨*proof*⟩

theorem *basis-of-hull*: $\mathfrak{B}\ \langle G \rangle = \mathfrak{B}\ G$
⟨*proof*⟩

lemma *basis-hull-sub*: $\mathfrak{B}\ \langle G \rangle \subseteq G$
⟨*proof*⟩

The basis is the smallest generating set.

theorem *basis-sub-gen*: $\langle S \rangle = \langle G \rangle \Longrightarrow \mathfrak{B}\ G \subseteq S$
⟨*proof*⟩

lemma *basis-min-gen*: $S \subseteq \mathfrak{B}\ G \Longrightarrow \langle S \rangle = G \Longrightarrow S = \mathfrak{B}\ G$
⟨*proof*⟩

lemma *basisI*: $(\bigwedge B.\ \langle B \rangle = \langle C \rangle \Longrightarrow C \subseteq B) \Longrightarrow \mathfrak{B}\ \langle C \rangle = C$
⟨*proof*⟩

thm *basis-inI*

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** $\langle S \rangle = \langle G \rangle$
⟨*proof*⟩

... and has the same basis

lemma *basis-sets*: $\mathfrak{B}\ G \subseteq S \Longrightarrow S \subseteq \langle G \rangle \Longrightarrow \mathfrak{B}\ G = \mathfrak{B}\ S$
⟨*proof*⟩

Any nonempty composed element has a decomposition into basis elements with many useful properties

lemma *non-simp-fac*: **assumes** $w \neq \varepsilon$ **and** $w \in \langle G \rangle$ **and** $w \notin \mathfrak{B}\ G$
obtains *us* **where** $1 < |us|$ **and** $us \neq \varepsilon$ **and** $us \in lists\ \mathfrak{B}\ G$ **and**
$hd\ us \neq \varepsilon$ **and** $hd\ us \in \langle G \rangle$ **and**
$concat(tl\ us) \neq \varepsilon$ **and** $concat(tl\ us) \in \langle G \rangle$ **and**
$w = hd\ us \cdot concat(tl\ us)$
⟨*proof*⟩

lemma *basis-dec*: $p \in \langle G \rangle \Longrightarrow s \in \langle G \rangle \Longrightarrow p \cdot s \in \mathfrak{B}\ G \Longrightarrow p = \varepsilon \vee s = \varepsilon$

⟨*proof*⟩

lemma *non-simp-fac′*: $w \notin \mathfrak{B}\ G \Longrightarrow w \neq \varepsilon \Longrightarrow w \in \langle G\rangle \Longrightarrow \exists\, us.\ us \in lists\ (G - \{\varepsilon\}) \land w = concat\ us \land |us| \neq 1$
⟨*proof*⟩

lemma *emp-gen-iff*: $(G - \{\varepsilon\}) = \{\} \longleftrightarrow \langle G\rangle = \{\varepsilon\}$
⟨*proof*⟩

lemma *emp-basis-iff*: $\mathfrak{B}\ G = \{\} \longleftrightarrow G - \{\varepsilon\} = \{\}$
⟨*proof*⟩

3.4 Code

locale *nemp-words* =
fixes *G*
assumes *emp-not-in*: $\varepsilon \notin G$

begin
lemma *drop-empD*: $G - \{\varepsilon\} = G$
⟨*proof*⟩

lemmas *emp-concat-emp′* = *emp-concat-emp*[*of - G*, *unfolded drop-empD*]

thm *disjE*[*OF ruler*[*OF take-is-prefix take-is-prefix*]]

lemma *concat-take-mono*: **assumes** $ws \in lists\ G$ **and** $concat\ (take\ i\ ws) \leq p\ concat\ (take\ j\ ws)$
shows $take\ i\ ws \leq p\ take\ j\ ws$
⟨*proof*⟩

lemma *nemp*: $x \in G \Longrightarrow x \neq \varepsilon$
⟨*proof*⟩

lemma *code-concat-eq-emp-iff* [*simp*]: $us \in lists\ G \Longrightarrow concat\ us = \varepsilon \longleftrightarrow us = \varepsilon$
⟨*proof*⟩

lemma *root-dec-inj-on*: $inj\text{-}on\ (\lambda\ x.\ [\varrho\ x]^{@}(e_\varrho\ x))\ G$
⟨*proof*⟩

lemma *concat-root-dec-eq-concat*:
assumes $ws \in lists\ G$
shows $concat\ (concat\ (map\ (\lambda\ x.\ [\varrho\ x]^{@}(e_\varrho\ x))\ ws)) = concat\ ws$
(**is** $concat(concat\ (map\ ?R\ ws)) = concat\ ws$)
⟨*proof*⟩

end

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-not-comm*: $x \in \mathcal{C} \Longrightarrow y \in \mathcal{C} \Longrightarrow x \neq y \Longrightarrow x \cdot y \neq y \cdot x$
  ⟨*proof*⟩

lemma *emp-not-in*: $\varepsilon \notin \mathcal{C}$
⟨*proof*⟩

lemma *nemp*: $u \in \mathcal{C} \Longrightarrow u \neq \varepsilon$
  ⟨*proof*⟩

sublocale *nemp-words* $\mathcal{C}$
  ⟨*proof*⟩

lemma *code-simple*: $c \in \mathcal{C} \Longrightarrow c \in B\ \mathcal{C}$
  ⟨*proof*⟩

lemma *code-is-basis*: $\mathfrak{B}\ \mathcal{C} = \mathcal{C}$
  ⟨*proof*⟩

lemma *code-unique-dec′*: $us \in lists\ \mathcal{C} \Longrightarrow Dec\ \mathcal{C}\ (concat\ us) = us$
  ⟨*proof*⟩

lemma *code-unique-dec* [*intro*!]: $us \in lists\ \mathcal{C} \Longrightarrow concat\ us = u \Longrightarrow Dec\ \mathcal{C}\ u = us$
  ⟨*proof*⟩

lemma *triv-refine*[*intro*!] : $us \in lists\ \mathcal{C} \Longrightarrow concat\ us = u \Longrightarrow Ref\ \mathcal{C}\ [u] = us$
  ⟨*proof*⟩

lemma *code-unique-ref*: $us \in lists\ \langle\mathcal{C}\rangle \Longrightarrow refine\ \mathcal{C}\ us = decompose\ \mathcal{C}\ (concat\ us)$
⟨*proof*⟩

lemma *refI* [*intro*]: $us \in lists\ \langle\mathcal{C}\rangle \Longrightarrow vs \in lists\ \mathcal{C} \Longrightarrow concat\ vs = concat\ us \Longrightarrow Ref\ \mathcal{C}\ us = vs$
  ⟨*proof*⟩

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*⟩

lemma *dec-pow*: $rs \in \langle\mathcal{C}\rangle \Longrightarrow Dec\ \mathcal{C}\ (rs^{@}k) = (Dec\ \mathcal{C}\ rs)^{@}k$

$\langle proof \rangle$

lemma *code-el-dec*: $c \in \mathcal{C} \Longrightarrow decompose\ \mathcal{C}\ c = [c]$
$\langle proof \rangle$

lemma *code-ref-list*: $us \in lists\ \mathcal{C} \Longrightarrow refine\ \mathcal{C}\ us = us$
$\langle proof \rangle$

lemma *code-ref-gen*: **assumes** $G \subseteq \langle \mathcal{C} \rangle$ $u \in \langle G \rangle$
shows $Dec\ \mathcal{C}\ u \in \langle decompose\ \mathcal{C}\ `\ G \rangle$
$\langle proof \rangle$

find-theorems $\varrho\ ?x\ ^{@}\ ?k = ?x\ 0 < ?k$

lemma *code-rev-code*: $code\ (rev\ `\ \mathcal{C})$
$\langle proof \rangle$

lemma *dec-rev* [*simp*, *reversal-rule*]:
$u \in \langle \mathcal{C} \rangle \Longrightarrow Dec\ rev\ `\ \mathcal{C}\ (rev\ u) = rev\ (map\ rev\ (Dec\ \mathcal{C}\ u))$
$\langle proof \rangle$

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 $set\ ws = \{u\}$
$\langle proof \rangle$

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 $primitive\ w$
$\langle proof \rangle$

lemma *inj-on-dec*: $inj\text{-}on\ (decompose\ \mathcal{C})\ \langle \mathcal{C} \rangle$
$\langle proof \rangle$

end — end context code

lemma *emp-is-code*: $code\ \{\}$
$\langle proof \rangle$

lemma *code-induct-hd*: **assumes** $\varepsilon \notin C$ **and**
$\bigwedge xs\ ys.\ xs \in lists\ C \Longrightarrow ys \in lists\ C \Longrightarrow concat\ xs = concat\ ys \Longrightarrow hd\ xs = hd\ ys$
shows $code\ C$
$\langle proof \rangle$

lemma *ref-set-primroot*: **assumes** $ws \in lists\ (G - \{\varepsilon\})$ **and** $code\ (\varrho`G)$
shows $set\ (Ref\ \varrho`G\ ws) = \varrho`(set\ ws)$
$\langle proof \rangle$

3.5 Prefix code

locale *pref-code* =
  fixes $\mathcal{C}$
  assumes
    emp-not-in: $\varepsilon \notin \mathcal{C}$ **and**
    pref-free: $u \in \mathcal{C} \Longrightarrow v \in \mathcal{C} \Longrightarrow u \leq p\ v \Longrightarrow u = v$

begin

lemma *nemp*: $u \in \mathcal{C} \Longrightarrow u \neq \varepsilon$
  ⟨*proof*⟩

lemma *concat-pref-concat*:
  assumes $us \in lists\ \mathcal{C}\ \ vs \in lists\ \mathcal{C}\ \ concat\ us \leq p\ concat\ vs$
  shows $us \leq p\ vs$
⟨*proof*⟩

lemma *concat-pref-concat-conv*:
  assumes $us \in lists\ \mathcal{C}\ \ vs \in lists\ \mathcal{C}$
  shows $concat\ us \leq p\ concat\ vs \longleftrightarrow us \leq p\ vs$
⟨*proof*⟩

sublocale *code*
  ⟨*proof*⟩

lemmas *is-code* = *is-code* **and**
  code = *code-axioms*

lemma *dec-pref-unique*:
  $w \in \langle\mathcal{C}\rangle \Longrightarrow p \in \langle\mathcal{C}\rangle \Longrightarrow p \leq p\ w \Longrightarrow Dec\ \mathcal{C}\ p \leq p\ Dec\ \mathcal{C}\ w$
  ⟨*proof*⟩

end

3.5.1 Suffix code

locale *suf-code* = *pref-code* (rev ‘ $\mathcal{C}$) **for** $\mathcal{C}$
begin

thm *dec-rev*
    code

sublocale *code*
  ⟨*proof*⟩

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*]

thm *is-code*
code-axioms
code

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*: $u \in \mathcal{C} \Longrightarrow u \neq \varepsilon$
⟨*proof*⟩

sublocale *pref-code*
⟨*proof*⟩

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*⟩

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*⟩

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*: $u \in \mathcal{C} \Longrightarrow u \neq \varepsilon$

⟨*proof*⟩

sublocale *pref-code*
⟨*proof*⟩

lemma *rev-non-overlapping*: *non-overlapping* (*rev* ‘ $\mathcal{C}$)
⟨*proof*⟩

sublocale *suf*: *suf-code* $\mathcal{C}$
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

theorem *prim-morph*:
assumes $ws \in lists\ \mathcal{C}$
and $|ws| \neq 1$
and *primitive ws*
shows *primitive* (*concat ws*)
⟨*proof*⟩

lemma *prim-concat-conv*:
assumes $ws \in lists\ \mathcal{C}$
and $|ws| \neq 1$
shows *primitive* (*concat ws*) $\longleftrightarrow$ *primitive ws*
⟨*proof*⟩

end

lemma (**in** *code*) *code-roots-non-overlapping*: *non-overlapping* $((\lambda\ x.\ [\varrho\ x]^{@}(e_{\varrho}\ x))$ ‘ $\mathcal{C})$
⟨*proof*⟩

theorem (**in** *code*) *roots-prim-morph*:
assumes $ws \in lists\ \mathcal{C}$

and $|ws| \neq 1$
and *primitive ws*
shows *primitive* (*concat* (*map* (λ *x*. [ϱ *x*]$^{@}$(e_ϱ *x*)) *ws*))
(**is** *primitive* (*concat* (*map* *?R* *ws*)))
⟨*proof*⟩

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* = $x \cdot y \wedge_p y \cdot x$
definition *bin-lcs* **where** *bin-lcs x y* = $x \cdot y \wedge_s y \cdot x$

definition *bin-mismatch* **where** *bin-mismatch x y* = $(x \cdot y)!|$*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*))
⟨*proof*⟩

lemma *bin-lcp-sym*: *bin-lcp x y* = *bin-lcp y x*
⟨*proof*⟩

lemma *bin-mismatch-comm*: (*bin-mismatch x y* = *bin-mismatch y x*) $\longleftrightarrow$ ($x \cdot y = y \cdot x$)
⟨*proof*⟩

lemma *bin-lcp-pref-fst-snd*: *bin-lcp x y* $\leq p$ $x \cdot y$
⟨*proof*⟩

lemma *bin-lcp-pref-snd-fst*: *bin-lcp x y* $\leq p$ $y \cdot x$
⟨*proof*⟩

lemma *bin-lcp-bin-lcs* [*reversal-rule*]: *bin-lcp* (*rev x*) (*rev y*) = *rev* (*bin-lcs x y*)
⟨*proof*⟩

lemmas *bin-lcs-sym* = *bin-lcp-sym*[*reversed*]

lemma *bin-lcp-len*: $x \cdot y \neq y \cdot x \Longrightarrow$ $|$*bin-lcp x y*$| < |x \cdot y|$
⟨*proof*⟩

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*

⟨*proof*⟩

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*: $\{uu\ a, uu\ (\neg\ a)\} = \{u_0, u_1\}$
⟨*proof*⟩

lemma *bin-code-set-bool'*: $\{uu\ a, uu\ (\neg\ a)\} = \{u_1, u_0\}$
⟨*proof*⟩

lemma *bin-code-swap*: *binary-code* u_1 u_0
⟨*proof*⟩

lemma *bin-code-bool*: *binary-code* (*uu a*) (*uu* (¬ *a*))
⟨*proof*⟩

lemma *bin-code-neq*: $u_0 \neq u_1$
⟨*proof*⟩

lemma *bin-code-neq-bool*: $uu\ a \neq uu\ (\neg\ a)$
⟨*proof*⟩

lemma *bin-fst-nemp*: $u_0 \neq \varepsilon$ **and** *bin-snd-nemp*: $u_1 \neq \varepsilon$ **and** *bin-nemp-bool*: $uu\ a \neq \varepsilon$
⟨*proof*⟩

lemma *bin-not-comp*: $\neg\ u_0 \cdot u_1 \bowtie u_1 \cdot u_0$
⟨*proof*⟩

lemma *bin-not-comp-bool*: $\neg\ (uu\ a \cdot uu\ (\neg\ a) \bowtie uu\ (\neg\ a) \cdot uu\ a)$
⟨*proof*⟩

lemma *bin-not-comp-suf*: $\neg\ u_0 \cdot u_1 \bowtie_s u_1 \cdot u_0$
⟨*proof*⟩

lemma *bin-not-comp-suf-bool*: $\neg\ (uu\ a \cdot uu\ (\neg\ a) \bowtie_s uu\ (\neg\ a) \cdot uu\ a)$
⟨*proof*⟩

lemma *bin-mismatch-neq*: *bin-mismatch* u_0 u_1 ≠ *bin-mismatch* u_1 u_0
⟨*proof*⟩

abbreviation *bin-code-lcp* (‹α›) **where** *bin-code-lcp* ≡ *bin-lcp* u_0 u_1
abbreviation *bin-code-lcs* **where** *bin-code-lcs* ≡ *bin-lcs* u_0 u_1
abbreviation *bin-code-mismatch-fst* (‹c_0›) **where** *bin-code-mismatch-fst* ≡ *bin-mismatch* u_0 u_1
abbreviation *bin-code-mismatch-snd* (‹c_1›) **where** *bin-code-mismatch-snd* ≡ *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* (¬ *a*)) = *bin-code-lcp*
⟨*proof*⟩

lemma *bin-lcp-spref*: α <p u_0 · u_1
⟨*proof*⟩

lemma *bin-lcp-spref'*: α <p u_1 · u_0
⟨*proof*⟩

lemma *bin-lcp-spref-bool*: α <p *uu a* · *uu* (¬ *a*)
⟨*proof*⟩

lemma *bin-mismatch-bool'*: α · [*cc a*] ≤p *uu a* · *uu* (¬ *a*)
⟨*proof*⟩

lemma *bin-mismatch-bool*: α · [*cc a*] ≤p *uu a* · α
⟨*proof*⟩

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} ⟹ α ≤p *concat xs* · α
⟨*proof*⟩

lemma *bin-lcp-pref-all-hull*: *w* ∈ ⟨{u_0,u_1}⟩ ⟹ α ≤p *w* · α

⟨*proof*⟩

lemma *bin-lcp-mismatch-pref-all-bool*: **assumes** $q \le p\ w$ **and** $w \in \langle\{uu\ b, uu\ (\neg\ b)\}\rangle$ **and** $|\alpha| < |uu\ a \cdot q|$
shows $\alpha \cdot [cc\ a] \le p\ uu\ a \cdot q$
⟨*proof*⟩

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** $q \le p\ w$ **and** $w \in \langle\{u_0, u_1\}\rangle$ **and** $|\alpha| \le |q|$
shows $\alpha \le p\ q$
⟨*proof*⟩

lemma *bin-mismatch-all-bool*: **assumes** $xs \in lists\ \{uu\ b,\ uu\ (\neg\ b)\}$ **shows** $\alpha \cdot [cc\ a] \le p\ (uu\ a) \cdot concat\ xs \cdot \alpha$
⟨*proof*⟩

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** $\alpha \cdot [cc\ a] \le p\ uu\ a \cdot w \cdot \alpha$
⟨*proof*⟩

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** $q \le p\ uu\ a \cdot w$ **and** $w \in \langle\{uu\ b, uu\ (\neg\ b)\}\rangle$ **and** $|\alpha| < |q|$
shows $\alpha \cdot [cc\ a] \le p\ q$
⟨*proof*⟩

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** $|\alpha| \le |q_0|$ **and** $|\alpha| \le |q_1|$ **and**
$q_0 \le p\ u_0 \cdot w_0$ **and** $q_1 \le p\ u_1 \cdot w_1$ **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*⟩

lemma *hd-lq-mismatch-fst*: $hd\ (\alpha^{-1>}(u_0 \cdot \alpha)) = c_0$
⟨*proof*⟩

lemma *hd-lq-mismatch-snd*: $hd\ (\alpha^{-1>}(u_1 \cdot \alpha)) = c_1$
⟨*proof*⟩

lemma *hds-bin-mismatch-neq*: $hd\ (\alpha^{-1>}(u_0 \cdot \alpha)) \neq hd\ (\alpha^{-1>}(u_1 \cdot \alpha))$
⟨*proof*⟩

lemma *bin-lcp-fst-pow-pref*: **assumes** $0 < k$ **shows** $\alpha \cdot [c_0] \leq p\ u_0{}^{@}k \cdot u_1 \cdot z$
⟨*proof*⟩

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*: $\alpha \leq p\ u_1 \cdot \alpha$
⟨*proof*⟩

lemma *bin-lcp-pref-all-set*: **assumes** *set ws* = $\{u_0,u_1\}$
shows $\alpha \leq p$ *concat ws*
⟨*proof*⟩

lemma *bin-lcp-conjug-morph*:
assumes $u \in \langle\{u_0,u_1\}\rangle$ **and** $v \in \langle\{u_0,u_1\}\rangle$
shows $\alpha^{-1>}(u \cdot \alpha) \cdot \alpha^{-1>}(v \cdot \alpha) = \alpha^{-1>}((u \cdot v) \cdot \alpha)$
⟨*proof*⟩

lemma *lcp-bin-conjug-prim-iff*:
set ws = $\{u_0,u_1\} \Longrightarrow$ *primitive* $(\alpha^{-1>}(concat\ ws) \cdot \alpha) \longleftrightarrow$ *primitive* (*concat ws*)
⟨*proof*⟩

lemma *bin-lcp-conjug-inj-on*: *inj-on* $(\lambda u.\ \alpha^{-1>}(u \cdot \alpha))\ \langle\{u_0,u_1\}\rangle$
⟨*proof*⟩

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*⟩
lemma **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*⟩

lemma *bin-code-lcp-concat*: **assumes** *us* ∈ *lists* $\{u_0,u_1\}$ **and** *vs* ∈ *lists* $\{u_0,u_1\}$ **and** ¬ *us* ⋈ *vs*
shows *concat us* $\cdot \alpha \wedge_p$ *concat vs* $\cdot \alpha$ = *concat* (*us* $\wedge_p$ *vs*) $\cdot \alpha$
⟨*proof*⟩

lemma *bin-code-lcp-concat′*: **assumes** *us* ∈ *lists* $\{u_0,u_1\}$ **and** *vs* ∈ *lists* $\{u_0,u_1\}$

and $\neg$ *concat us* $\bowtie$ *concat vs*
shows *concat us* $\wedge_p$ *concat vs* = *concat* (*us* $\wedge_p$ *vs*) $\cdot$ α
⟨*proof*⟩

lemma *bin-lcp-pows*: $0 < k \Longrightarrow 0 < l \Longrightarrow u_0{}^{@}k \cdot u_1 \cdot z \wedge_p u_1{}^{@}l \cdot u_0 \cdot z' = \alpha$
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *code-bin-roots*: *binary-code* (ϱ u_0) (ϱ u_1)
⟨*proof*⟩

sublocale *code* $\{u_0,u_1\}$
⟨*proof*⟩

lemma *primroot-dec*: (*Dec* $\{\varrho\ u_0,\ \varrho\ u_1\}$ u_0) = $[\varrho\ u_0]^{@}e_\varrho\ u_0$ (*Dec* $\{\varrho\ u_0,\ \varrho\ u_1\}$ u_1) = $[\varrho\ u_1]^{@}e_\varrho\ u_1$
⟨*proof*⟩

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 $\neg\ u_0 \cdot\ p \le p\ u_1 \cdot w'$
⟨*proof*⟩

lemma *bin-code-rev*: *binary-code* (*rev* u_0) (*rev* u_1)
⟨*proof*⟩

lemma *bin-mismatch-pows*: $\neg\ u_0{}^{@}Suc\ k \cdot u_1 \cdot z = u_1{}^{@}Suc\ l \cdot u_0 \cdot z'$
⟨*proof*⟩

lemma *bin-lcp-pows-lcp*: $0 < k \Longrightarrow 0 < l \Longrightarrow u_0{}^{@}k \cdot u_1{}^{@}l \wedge_p u_1{}^{@}l \cdot u_0{}^{@}k = u_0 \cdot u_1 \wedge_p u_1 \cdot u_0$
⟨*proof*⟩

lemma *bin-mismatch*: $u_0 \cdot \alpha \wedge_p u_1 \cdot \alpha = \alpha$
⟨*proof*⟩

lemma *not-comp-bin-fst-snd*: $\neg\ u_0 \cdot \alpha \bowtie u_1 \cdot\ \alpha$
⟨*proof*⟩

theorem *bin-bounded-delay*: **assumes** $z \le p\ u_0 \cdot w_0$ **and** $z \le p\ u_1 \cdot w_1$
and $w_0 \in \langle\{u_0,u_1\}\rangle$ **and** $w_1 \in \langle\{u_0,u_1\}\rangle$
shows $|z| \le |\alpha|$
⟨*proof*⟩

thm *binary-code.bin-lcp-pows-lcp*

lemma *prim-roots-lcp*: $\varrho\ u_0 \cdot \varrho\ u_1 \wedge_p \varrho\ u_1 \cdot \varrho\ u_0 = \alpha$
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

lemma *per-root-lcp-per-root*: $u_0 \cdot u_1 <p\ r \cdot u_0 \cdot u_1 \Longrightarrow \alpha \cdot [c_0] \leq p\ r \cdot \alpha$
⟨*proof*⟩

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*⟩

end
lemmas *no-comm-bin-code* = *binary-code.bin-code*[*unfolded binary-code-def*]

theorem *bin-code-code*: **assumes** $u \cdot v \neq v \cdot u$ **shows** *code* $\{u,\ v\}$
⟨*proof*⟩

lemma *code-bin-code*: $u \neq v \Longrightarrow code\ \{u,v\} \Longrightarrow u \cdot v \neq v \cdot u$
⟨*proof*⟩

lemma *lcp-roots-lcp*: $x \cdot y \neq y \cdot x \Longrightarrow x \cdot y \wedge_p y \cdot x = \varrho\ x \cdot \varrho\ y \wedge_p \varrho\ y \cdot \varrho\ x$
⟨*proof*⟩

3.8.2 Binary Mismatch tools

thm *binary-code.bin-mismatch-pows*[*unfolded binary-code-def*]

lemma *bin-mismatch*: $u^{@}Suc\ k \cdot v \cdot z = v^{@}Suc\ l \cdot u \cdot z' \Longrightarrow u \cdot v = v \cdot u$
⟨*proof*⟩

definition *bin-mismatch-pref* :: *'a list* ⇒ *'a list* ⇒ *'a list* ⇒ *bool* **where**
bin-mismatch-pref x y w ≡ ∃ *k*. $x^{@}k \cdot y \le p\ w$

— Binary mismatch elims

lemma *bm-pref-letter*: **assumes** $x \cdot y \neq y \cdot x$ **and** *bin-mismatch-pref x y* $(w1 \cdot y)$
shows *bin-lcp x y* · [*bin-mismatch x y*] ≤*p x* · *w1* · *bin-lcp x y*
⟨*proof*⟩

lemma *bm-eq-hard*: **assumes** $x \cdot w1 = y \cdot w2$ **and** *bin-mismatch-pref x y* (*w1* · *y*) **and** *bin-mismatch-pref y x* (*w2* · *x*)
shows $x \cdot y = y \cdot x$
⟨*proof*⟩

lemma *bm-hard-lcp*: **assumes** $x \cdot y \neq y \cdot x$ **and** *bin-mismatch-pref x y w1* **and** *bin-mismatch-pref y x w2*
shows $x \cdot w1 \wedge_p y \cdot w2 = x \cdot y \wedge_p y \cdot x$
⟨*proof*⟩

lemma *bm-pref-hard*: **assumes** *x* · *w1* ≤*p y* · *w2* **and** *bin-mismatch-pref x y w1* **and** *bin-mismatch-pref y x* (*w2* · *x*)
shows $x \cdot y = y \cdot x$
⟨*proof*⟩

named-theorems *bm-elims*
lemmas [*bm-elims*] = *bm-eq-hard bm-eq-hard*[*symmetric*] *bm-pref-hard bm-pref-hard*[*symmetric*]
bm-hard-lcp bm-hard-lcp[*symmetric*]
arg-cong2[*of* - - - - λ *x y*. $x \wedge_p y$]

named-theorems *bm-elims-rev*
lemmas [*bm-elims-rev*] = *bm-elims*[*reversed*]

— Binary mismatch predicate evaluation
named-theorems *bm-simps*
lemma [*bm-simps*]: *bin-mismatch-pref x y* (*y* · *v*)
⟨*proof*⟩
lemma [*bm-simps*]: *bin-mismatch-pref x y y*
⟨*proof*⟩
lemma [*bm-simps*]:
w1 ∈ ⟨{*x*,*y*}⟩ ⟹ *bin-mismatch-pref x y w* ⟹ *bin-mismatch-pref x y* (*w1* · *w*)
⟨*proof*⟩

lemmas [*bm-simps*] = *lcp-ext-left*

named-theorems *bm-simps-rev*
lemmas [*bm-simps-rev*] = *bm-simps*[*reversed*]

— Binary hull membership evaluation

named-theorems *bin-hull-in*
lemma[*bin-hull-in*]: $x \in \langle\{x,y\}\rangle$
⟨*proof*⟩
lemma[*bin-hull-in*]: $y \in \langle\{x,y\}\rangle$
⟨*proof*⟩
lemma[*bin-hull-in*]: $w \in \langle\{x,y\}\rangle \longleftrightarrow w \in \langle\{y,x\}\rangle$
⟨*proof*⟩
lemmas[*bin-hull-in*] = *hull-closed power-in rassoc*

named-theorems *bin-hull-in-rev*
lemmas [*bin-hull-in-rev*] = *bin-hull-in*[*reversed*]

method *mismatch0* =
((*simp only*: *shifts bm-simps*)*?*,
(*elim bm-elims*)*?*;
(*simp-all only*: *bm-simps bin-hull-in*))

method *mismatch-rev* =
((*simp only*: *shifts-rev bm-simps-rev*)*?*,
(*elim bm-elims-rev*)*?*;
(*simp-all only*: *bm-simps-rev bin-hull-in-rev*))

method *mismatch* =
(*insert method-facts*, *use nothing* **in**
‹(*mismatch0*;*fail*)|(*mismatch-rev*)›
)

thm *bm-elims*

Mismatch method demonstrations

lemma $y \cdot x \leq p\ x^{@}k \cdot x \cdot y \cdot w \Longrightarrow x \cdot y = y \cdot x$
⟨*proof*⟩

lemma $w1 \in \langle\{x,y\}\rangle \Longrightarrow w2 \in \langle\{x,y\}\rangle \Longrightarrow x \cdot w2 \cdot y \cdot z = y \cdot w1 \cdot x \cdot v \Longrightarrow x \cdot y = y \cdot x$
⟨*proof*⟩

lemma $w1 \in \langle\{x,y\}\rangle \Longrightarrow y \cdot x \cdot w2 \cdot z = x \cdot w1 \Longrightarrow x \cdot y = y \cdot x$
⟨*proof*⟩

lemma $w1 \in \langle\{x,y\}\rangle \Longrightarrow w2 \in \langle\{x,y\}\rangle \Longrightarrow x \cdot y \cdot w2 \cdot x \leq s\ x \cdot w1 \cdot y \Longrightarrow x \cdot y = y \cdot x$
$\langle proof \rangle$

lemma assumes $x \cdot y \cdot z = y \cdot y \cdot x \cdot v$ **shows** $x \cdot y = y \cdot x$
$\langle proof \rangle$

lemma assumes $y \cdot x \cdot x \cdot y \cdot z = y \cdot x \cdot y \cdot y \cdot x \cdot v$ **shows** $x \cdot y = y \cdot x$
$\langle proof \rangle$

lemma $y \cdot y \cdot x \cdot v = x \cdot x \cdot y \cdot z \Longrightarrow x \cdot y = y \cdot x$
$\langle proof \rangle$

lemma $x \cdot x \cdot y \cdot z = y \cdot y \cdot x \cdot z' \Longrightarrow x \cdot y = y \cdot x$
$\langle proof \rangle$

lemma $z \cdot x \cdot y \cdot x \cdot x = v \cdot x \cdot y \cdot y \Longrightarrow y \cdot x = x \cdot y$
$\langle proof \rangle$

lemma $x \cdot y \leq p\ y \cdot y \cdot x \Longrightarrow x \cdot y = y \cdot x$
$\langle proof \rangle$

lemma $y \cdot x \cdot x \cdot x \cdot y \leq p\ y \cdot x \cdot x \cdot y \cdot y \cdot x \Longrightarrow x \cdot y = y \cdot x$
$\langle proof \rangle$

lemma $x \cdot y \leq p\ y \cdot y \cdot x \cdot z \Longrightarrow y \cdot x = x \cdot y$
$\langle proof \rangle$

lemma $x \cdot x \cdot y \cdot y \cdot y \leq s\ z \cdot y \cdot y \cdot x \cdot x \Longrightarrow x \cdot y = y \cdot x$
$\langle proof \rangle$

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** $x \cdot y = y \cdot x$
$\langle proof \rangle$

lemma $k \neq 0 \Longrightarrow j \neq 0 \Longrightarrow (x\ ^{@}\ j \cdot y\ ^{@}\ ka) \cdot y = y^{@}k \cdot x\ ^{@}\ j \cdot y\ ^{@}\ (k - 1) \Longrightarrow x \cdot y = y \cdot x$
$\langle proof \rangle$

lemma $dif \neq 0 \Longrightarrow j \neq 0 \Longrightarrow (x\ ^{@}\ j \cdot y\ ^{@}\ ka) \cdot y\ ^{@}\ dif = y\ ^{@}\ dif \cdot x\ ^{@}\ j \cdot y\ ^{@}\ ka \Longrightarrow x \cdot y = y \cdot x$
$\langle proof \rangle$

lemma assumes $x \cdot y \neq y \cdot x$
shows $x \cdot x \cdot y \wedge_p y \cdot y \cdot x = (x \cdot y \wedge_p y \cdot x)$
$\langle proof \rangle$

lemma assumes $x \cdot y \neq y \cdot x$
shows $w \cdot z \cdot x \cdot x \cdot y \wedge_p w \cdot z \cdot y \cdot y \cdot x = (w \cdot z) \cdot (x \cdot y \wedge_p y \cdot x)$

⟨*proof*⟩

3.8.3 Applied 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*⟩

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**
$count\text{-}list\ ws\ x + count\text{-}list\ ws\ y = |ws|$
⟨*proof*⟩

lemma *two-elem-first-block*: **assumes** $w \in \langle\{u,v\}\rangle$
obtains m **where** $u^{@}m \cdot v \bowtie w$
⟨*proof*⟩

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*⟩

lemmas *two-elem-suf* = *two-elem-pref*[*reversed*]

lemma *gen-drop-exp*: **assumes** $p \in \langle\{u,v^{@}(Suc\ k)\}\rangle$ **shows** $p \in \langle\{u,v\}\rangle$
⟨*proof*⟩

lemma *gen-drop-exp-pos*: **assumes** $p \in \langle\{u,v^{@}k\}\rangle$ $0 < k$ **shows** $p \in \langle\{u,v\}\rangle$
⟨*proof*⟩

lemma *gen-prim*: $p \in \langle\{u,v\}\rangle \Longrightarrow p \in \langle\{u,\varrho\ v\}\rangle$
⟨*proof*⟩

lemma *roots-hull*: **assumes** $w \in \langle\{u^{@}k,v^{@}m\}\rangle$ **shows** $w \in \langle\{u,v\}\rangle$
⟨*proof*⟩

lemma *roots-hull-sub*: $\langle\{u^{@}k,v^{@}m\}\rangle \subseteq \langle\{u,v\}\rangle$
⟨*proof*⟩

lemma *primroot-gen*[*intro*]: $v \in \langle\{u,\ \varrho\ v\}\rangle$
⟨*proof*⟩

lemma *primroot-gen'*[*intro*]: $u \in \langle\{\varrho\ u,\ v\}\rangle$
⟨*proof*⟩

lemma *set-lists-primroot*: $set\ ws \subseteq \{x,y\} \Longrightarrow ws \in lists\ \langle\{\varrho\ x,\ \varrho\ y\}\rangle$
⟨*proof*⟩

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* (‹$\langle$-$\rangle_F$›)
for G **where**
$\varepsilon \in \langle G\rangle_F$
| *free-gen-in*: $w \in G \Longrightarrow w \in \langle G\rangle_F$
| $w1 \in \langle G\rangle_F \Longrightarrow w2 \in \langle G\rangle_F \Longrightarrow w1 \cdot w2 \in \langle G\rangle_F$
| $p \in \langle G\rangle_F \Longrightarrow q \in \langle G\rangle_F \Longrightarrow p \cdot w \in \langle G\rangle_F \Longrightarrow w \cdot q \in \langle G\rangle_F \Longrightarrow w \in \langle G\rangle_F$ — the stability condition

lemmas [*intro*] = *free-hull.intros*

The defined set indeed is a hull.

lemma *free-hull-hull*[*simp*]: $\langle\langle G\rangle_F\rangle = \langle G\rangle_F$
⟨*proof*⟩

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*⟩

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.

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*: $\langle \mathfrak{B}_F\ G\rangle = \langle G\rangle_F$
⟨*proof*⟩

lemma *genset-sub-free*: $G \subseteq \langle G\rangle_F$
⟨*proof*⟩

We have developed two points of view on freeness:

- being a free hull, that is, 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*⟩

lemma *gen-in-free-hull*: $x \in G \Longrightarrow x \in \langle \mathfrak{B}_F\ G \rangle$
⟨*proof*⟩

Second, a code generates its free hull.

lemma (**in** *code*) *code-gen-free-hull*: $\langle \mathcal{C} \rangle_F = \langle \mathcal{C} \rangle$
⟨*proof*⟩

That is, a code is its own free basis

lemma (**in** *code*) *code-free-basis*: $\mathcal{C} = \mathfrak{B}_F\ \mathcal{C}$
⟨*proof*⟩

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*: $p \in \langle \mathcal{C} \rangle \Longrightarrow q \in \langle \mathcal{C} \rangle \Longrightarrow p \cdot w \in \langle \mathcal{C} \rangle \Longrightarrow w \cdot q \in \langle \mathcal{C} \rangle \Longrightarrow w \in \langle \mathcal{C} \rangle$
⟨*proof*⟩

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*: $G \subseteq H \Longrightarrow \langle G \rangle_F \subseteq \langle H \rangle_F$
⟨*proof*⟩

lemma *free-hull-idem*: $\langle \langle G \rangle_F \rangle_F = \langle G \rangle_F$
⟨*proof*⟩

lemma *hull-gen-free-hull*: $\langle \langle G \rangle \rangle_F = \langle G \rangle_F$
⟨*proof*⟩

Code is also the free basis of its hull.

lemma (**in** *code*) *code-free-basis-hull*: $\mathcal{C} = \mathfrak{B}_F\ \langle \mathcal{C} \rangle$
⟨*proof*⟩

The minimality of the free hull easily follows.

theorem (**in** *code*) *free-hull-min*: **assumes** $G \subseteq \langle \mathcal{C} \rangle$ **shows** $\langle G \rangle_F \subseteq \langle \mathcal{C} \rangle$
⟨*proof*⟩

theorem *free-hull-inter*: $\langle G \rangle_F = \bigcap\ \{M.\ G \subseteq M \wedge M = \langle M \rangle_F\}$
⟨*proof*⟩

Decomposition into the free basis is a morphism.

lemma *free-basis-dec-morph*: $u \in \langle G \rangle_F \Longrightarrow v \in \langle G \rangle_F \Longrightarrow$
$Dec\ (\mathfrak{B}_F\ G)\ (u \cdot v) = (Dec\ (\mathfrak{B}_F\ G)\ u) \cdot (Dec\ (\mathfrak{B}_F\ G)\ v)$
⟨*proof*⟩

3.11 Reversing hulls and decompositions

lemma *basis-rev-commute*[*reversal-rule*]: $\mathfrak{B}$ (*rev* ‘ *G*) = *rev* ‘ ($\mathfrak{B}$ *G*)
⟨*proof*⟩

lemma *rev-free-hull-comm*: $\langle rev \text{ ‘ } X\rangle_F$ = *rev* ‘ $\langle X\rangle_F$
⟨*proof*⟩

lemma *free-basis-rev-commute* [*reversal-rule*]: $\mathfrak{B}_F$ *rev* ‘ *X* = *rev* ‘ ($\mathfrak{B}_F$ *X*)
⟨*proof*⟩

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*⟩

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*⟩

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*))
⟨*proof*⟩

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*. $b \in B$}

term *Set.filter P A*

lemma *sings-image*: *sings B* = (λ *x*. [*x*]) ‘ *B*
⟨*proof*⟩

lemma *lists-sing-map-concat-ident*: *xs* ∈ *lists* (*sings B*) ⟹ *xs* = *map* (λ *x*. [*x*]) (*concat xs*)
⟨*proof*⟩

lemma *code-sings*: *code* (*sings B*)
⟨*proof*⟩

lemma *sings-gen-lists*: ⟨*sings B*⟩ = *lists B*
⟨*proof*⟩

lemma *sing-gen-lists*: *lists* {*x*} = ⟨{[*x*]}⟩
⟨*proof*⟩

lemma *bin-gen-lists*: *lists* {*x*, *y*} = ⟨{[*x*],[*y*]}⟩

⟨*proof*⟩

lemma $sings\ B = \mathfrak{B}_F\ (lists\ B)$
⟨*proof*⟩

lemma *map-sings*: $xs \in lists\ B \Longrightarrow map\ (\lambda x.\ x\ \#\ \varepsilon)\ xs \in lists\ (sings\ B)$
⟨*proof*⟩

lemma *dec-sings*: $xs \in lists\ B \Longrightarrow Dec\ (sings\ B)\ xs = map\ (\lambda\ x.\ [x])\ xs$
⟨*proof*⟩

lemma *sing-lists-exp*: **assumes** $ws \in lists\ \{x\}$
obtains k **where** $ws = [x]^{@}k$
⟨*proof*⟩

lemma *sing-lists-exp-len*: $ws \in lists\ \{x\} \Longrightarrow [x]^{@}|ws| = ws$
⟨*proof*⟩

lemma *sing-lists-exp-count*: $ws \in lists\ \{x\} \Longrightarrow [x]^{@}(count\text{-}list\ ws\ x) = ws$
⟨*proof*⟩

lemma *sing-set-pow-count-list*: $set\ ws \subseteq \{a\} \Longrightarrow [a]^{@}(count\text{-}list\ ws\ a) = ws$
⟨*proof*⟩

lemma *sing-set-pow*: $set\ ws \subseteq \{a\} \Longrightarrow [a]^{@}|ws| = ws$
⟨*proof*⟩

lemma *count-sing-exp*[*simp*]: $count\text{-}list\ ([a]^{@}k)\ a = k$
⟨*proof*⟩

lemma *count-sing-exp'*[*simp*]: $count\text{-}list\ ([a])\ a = 1$
⟨*proof*⟩

lemma *count-sing-distinct*[*simp*]: $a \neq b \Longrightarrow count\text{-}list\ ([a]^{@}k)\ b = 0$
⟨*proof*⟩

lemma *count-sing-distinct'*[*simp*]: $a \neq b \Longrightarrow count\text{-}list\ ([a])\ b = 0$
⟨*proof*⟩

lemma *sing-letter-imp-prim*: **assumes** $count\text{-}list\ w\ a = 1$ **shows** $primitive\ w$
⟨*proof*⟩

lemma *prim-abk*: $a \neq b \Longrightarrow primitive\ ([a] \cdot [b]\ ^{@}\ k)$
⟨*proof*⟩

lemma *sing-code*: $x \neq \varepsilon \Longrightarrow code\ \{x\}$
⟨*proof*⟩

lemma *sings-card*: $card\ A = card\ (sings\ A)$

⟨*proof*⟩

lemma *sings-finite*: *finite A* = *finite* (*sings A*)
⟨*proof*⟩

lemma *sings-conv*: $A = B \longleftrightarrow$ *sings A* = *sings B*
⟨*proof*⟩

3.13 Various additional lemmas

3.13.1 Roots of binary set

lemma *two-roots-code*: **assumes** $x \neq \varepsilon$ **and** $y \neq \varepsilon$ **shows** *code* $\{\varrho\ x,\ \varrho\ y\}$
⟨*proof*⟩

lemma *primroot-in-set-dec*: **assumes** $x \neq \varepsilon$ **and** $y \neq \varepsilon$ **shows** $\varrho\ x \in$ *set* (*Dec* $\{\varrho\ x,\ \varrho\ y\}$ x)
⟨*proof*⟩

lemma *primroot-dec*: **assumes** $x \cdot y \neq y \cdot x$
shows (*Dec* $\{\varrho\ x,\ \varrho\ y\}$ x) = $[\varrho\ x]^{@} e_{\varrho}\ x$ (*Dec* $\{\varrho\ x,\ \varrho\ y\}$ y) = $[\varrho\ y]^{@} e_{\varrho}\ y$
⟨*proof*⟩

lemma (**in** *binary-code*) *bin-roots-sings-code*: *non-overlapping* $\{$*Dec* $\{\varrho\ u_0,\ \varrho\ u_1\}$ u_0, *Dec* $\{\varrho\ u_0,\ \varrho\ u_1\}$ $u_1\}$
⟨*proof*⟩

3.13.2 Other

lemma *bin-count-one-decompose*: **assumes** $ws \in$ *lists* $\{x,y\}$ **and** $x \neq y$ **and** *count-list ws y* = *1*
obtains *k m* **where** $[x]^{@}k \cdot [y] \cdot [x]^{@}m = ws$
⟨*proof*⟩

lemma *bin-count-one-conjug*: **assumes** $ws \in$ *lists* $\{x,y\}$ **and** $x \neq y$ **and** *count-list ws y* = *1*
shows $ws \sim [x]^{@}$(*count-list ws x*) $\cdot [y]$
⟨*proof*⟩

lemma *bin-prim-long-set*: **assumes** $ws \in$ *lists* $\{x,y\}$ **and** *primitive ws* **and** $2 \leq |ws|$
shows *set ws* = $\{x,y\}$
⟨*proof*⟩

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*⟩

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* :: $('a \Rightarrow 'b\ list) \Rightarrow ('a\ list \Rightarrow 'b\ list)$ (‹-$^{\mathcal{L}}$› [*1000*] *1000*)
where $t^{\mathcal{L}} \equiv (\lambda\ x.\ concat\ (map\ t\ x))$

definition *morphism-core* :: $('a\ list \Rightarrow 'b\ list) \Rightarrow ('a \Rightarrow 'b\ list)$ (‹-$^{\mathcal{C}}$› [*1000*] *1000*)
where *core-def*: $f^{\mathcal{C}} \equiv (\lambda\ x.\ f\ [x])$

lemma *core-sing*: $f^{\mathcal{C}}\ a = f\ [a]$
⟨*proof*⟩

lemma *range-map-core*: $range\ (map\ f^{\mathcal{C}}) = lists\ (range\ f^{\mathcal{C}})$
⟨*proof*⟩

lemma *map-core-lists*: $(map\ f^{\mathcal{C}}\ w) \in lists\ (range\ f^{\mathcal{C}})$
⟨*proof*⟩

lemma *comp-core*: $(f \circ g)^{\mathcal{C}} = f \circ g^{\mathcal{C}}$
⟨*proof*⟩

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*[*simp*]: $f\ \varepsilon = \varepsilon$
⟨*proof*⟩

lemma *emp-to-emp'*: $w = \varepsilon \Longrightarrow f\ w = \varepsilon$
⟨*proof*⟩

lemma *morph-concat-concat-map*: *ws* ∈ *lists* ⟨*A*⟩ ⟹ *f* (*concat ws*) = *concat* (*map f ws*)
⟨*proof*⟩

lemma *hull-im-hull*:
shows ⟨*f* ‘ *A*⟩ = *f* ‘ ⟨*A*⟩
⟨*proof*⟩

lemma *inj-basis-to-basis*: **assumes** *inj-on f* ⟨*A*⟩
shows *f* ‘ (𝔅 ⟨*A*⟩) = 𝔅 (*f*‘⟨*A*⟩)
⟨*proof*⟩

lemma *inj-code-to-code*: **assumes** *inj-on f* ⟨*A*⟩ **and** *code A*
shows *code* (*f* ‘ *A*)
⟨*proof*⟩

end

locale *morphism* =
fixes *f* :: '*a list* ⇒ '*b list*
assumes *morph*: *f* (*u* · *v*) = *f u* · *f v*
begin

sublocale *morphism-on f UNIV*
⟨*proof*⟩

lemma *map-core-lists*[*simp*]: *map* $f^{\mathcal{C}}$ *xs* ∈ *lists* (*range* $f^{\mathcal{C}}$)
⟨*proof*⟩

lemma *pow-morph*: *f* ($x^{@}k$) = $(f\ x)^{@}k$
⟨*proof*⟩

lemma *rev-map-pow*: (*rev-map f*) ($w^{@}n$) = *rev* ($(f\ (rev\ w))^{@}n$)
⟨*proof*⟩

lemma *pop-hd*: *f* (*a*#*u*) = *f* [*a*] · *f u*
⟨*proof*⟩

lemma *pop-hd-nemp*: *u* ≠ ε ⟹ *f* (*u*) = *f* [*hd u*] · *f* (*tl u*)
⟨*proof*⟩

lemma *pop-last-nemp*: *u* ≠ ε ⟹ *f* (*u*) = *f* (*butlast u*) · *f* [*last u*]
⟨*proof*⟩

lemma *pref-mono*: *u* ≤*p v* ⟹ *f u* ≤*p f v*
⟨*proof*⟩

lemma *suf-mono*: *u* ≤*s v* ⟹ *f u* ≤*s f v*

⟨*proof*⟩

lemma *morph-concat-map*: $concat\ (map\ f^{\mathcal{C}}\ x) = f\ x$
⟨*proof*⟩

lemma *morph-concat-map'*: $(\lambda\ x.\ concat\ (map\ f^{\mathcal{C}}\ x)) = f$
⟨*proof*⟩

lemma *morph-to-concat*:
obtains *xs* **where** $xs \in lists\ (range\ f^{\mathcal{C}})$ **and** $f\ x = concat\ xs$
⟨*proof*⟩

lemma *range-hull*: $range\ f = \langle(range\ f^{\mathcal{C}})\rangle$
⟨*proof*⟩

lemma *im-in-hull*: $f\ w \in \langle(range\ f^{\mathcal{C}})\rangle$
⟨*proof*⟩

lemma *core-ext-id*: $f^{\mathcal{C}\mathcal{L}} = f$
⟨*proof*⟩

lemma *rev-map-morph*: $morphism\ (rev\text{-}map\ f)$
⟨*proof*⟩

lemma *morph-rev-len*: $|f\ (rev\ u)| = |f\ u|$
⟨*proof*⟩

lemma *rev-map-len*: $|rev\text{-}map\ f\ u| = |f\ u|$
⟨*proof*⟩

lemma *in-set-morph-len*: **assumes** $a \in set\ w$ **shows** $|f\ [a]| \leq |f\ w|$
⟨*proof*⟩

lemma *morph-lq-comm*: $u \leq p\ v \Longrightarrow f\ (u^{-1>}v) = (f\ u)^{-1>}(f\ v)$
⟨*proof*⟩

lemma *morph-rq-comm*: **assumes** $v \leq s\ u$
shows $f\ (u^{<-1}v) = (f\ u)^{<-1}(f\ v)$
⟨*proof*⟩

lemma *code-set-morph*: **assumes** *c*: $code\ (f^{\mathcal{C}}\ `(set\ (u \cdot v)))$ **and** *i*: $inj\text{-}on\ f^{\mathcal{C}}\ (set\ (u \cdot v))$
and $f\ u = f\ v$
shows $u = v$
⟨*proof*⟩

lemma *morph-concat-concat-map*: $f\ (concat\ ws) = concat\ (map\ f\ ws)$
⟨*proof*⟩

lemma *morph-on*: *morphism-on f A*
⟨*proof*⟩

lemma *noner-sings-conv*: $(\forall\ w.\ w = \varepsilon \longleftrightarrow f\ w = \varepsilon) \longleftrightarrow (\forall\ a.\ f\ [a] \neq \varepsilon)$
⟨*proof*⟩

lemma *fac-mono*: $u \leq f\ w \Longrightarrow f\ u \leq f\ f\ w$
⟨*proof*⟩

lemma *set-core-set*: $set\ (f\ w) = \bigcup\ (set\ `\ f^{\mathcal{C}}\ `\ (set\ w))$
⟨*proof*⟩

end

lemma *morph-map*: *morphism* (*map f*)
⟨*proof*⟩

lemma *list-ext-morph*: *morphism* $t^{\mathcal{L}}$
⟨*proof*⟩

lemma *ext-def-on-set*: $(\bigwedge\ a.\ a \in set\ u \Longrightarrow g\ a = f\ a) \Longrightarrow g^{\mathcal{L}}\ u = f^{\mathcal{L}}\ u$
⟨*proof*⟩

lemma *morph-def-on-set*: $morphism\ f \Longrightarrow morphism\ g \Longrightarrow (\bigwedge\ a.\ a \in set\ u \Longrightarrow g^{\mathcal{C}}\ a = f^{\mathcal{C}}\ a) \Longrightarrow g\ u = f\ u$
⟨*proof*⟩

lemma *morph-compose*: $morphism\ f \Longrightarrow morphism\ g \Longrightarrow morphism\ (f \circ g)$
⟨*proof*⟩

4.1.2 Periodic morphism

locale *periodic-morphism* = *morphism* +
assumes *ims-comm*: $\bigwedge\ u\ v.\ f\ u \cdot f\ v = f\ v \cdot f\ u$ **and**
not-triv-emp: $\neg\ (\forall\ c.\ f\ [c] = \varepsilon)$
begin

lemma *per-morph-root-ex*:
$\exists\ r.\ \forall\ u.\ \exists\ n.\ f\ u = r^{@}n \wedge primitive\ r$
⟨*proof*⟩

definition *mroot* **where** $mroot \equiv (SOME\ r.\ (\forall\ u.\ \exists\ n.\ f\ u = r^{@}n) \wedge primitive\ r)$
definition *mexp* :: $'a \Rightarrow nat$ **where** $mexp\ c \equiv (SOME\ n.\ f\ [c] = mroot^{@}n)$

lemma *per-morph-rootI*: $\forall\ u.\ \exists\ n.\ f\ u = mroot^{@}n$ **and**
per-morph-root-prim: *primitive mroot*
⟨*proof*⟩

lemma *per-morph-expI'*: $f\ [c] = mroot^{@}(mexp\ c)$

⟨*proof*⟩

lemma *per-morph-expE*:
obtains n **where** $f\ u = mroot^{@}n$
⟨*proof*⟩

interpretation *mirror*: *periodic-morphism rev-map f*
⟨*proof*⟩

lemma *mroot-rev*: *mirror.mroot* = *rev mroot*
⟨*proof*⟩

end

4.1.3 Non-erasing morphism

locale *nonerasing-morphism* = *morphism* +
assumes *nonerasing*: $f\ w = \varepsilon \Longrightarrow w = \varepsilon$
begin

lemma *core-nemp*: $f^{\mathcal{C}}\ a \neq \varepsilon$
⟨*proof*⟩

lemma *nemp-to-nemp*: $w \neq \varepsilon \Longrightarrow f\ w \neq \varepsilon$
⟨*proof*⟩

lemma *sing-to-nemp*: $f\ [a] \neq \varepsilon$
⟨*proof*⟩

lemma *pref-morph-pref-eq*: $u \leq p\ v \Longrightarrow f\ v \leq p\ f\ u \Longrightarrow u = v$
⟨*proof*⟩

lemma *comm-eq-im-eq*:
$u \cdot v = v \cdot u \Longrightarrow f\ u = f\ v \Longrightarrow u = v$
⟨*proof*⟩

lemma *comm-eq-im-iff* :
assumes $u \cdot v = v \cdot u$
shows $f\ u = f\ v \longleftrightarrow u = v$
⟨*proof*⟩

lemma *rev-map-nonerasing*: *nonerasing-morphism* (*rev-map f*)
⟨*proof*⟩

lemma *first-of-first*: $(f\ (a\ \#\ ws))!0 = f\ [a]!0$
⟨*proof*⟩

lemma *hd-im-hd-hd*: **assumes** $u \neq \varepsilon$ **shows** $hd\ (f\ u) = hd\ (f\ [hd\ u])$
⟨*proof*⟩

lemma *ssuf-mono*: $u <s\ v \Longrightarrow f\ u <s\ f\ v$
⟨*proof*⟩

lemma *im-len-le*: $|u| \leq |f\ u|$
⟨*proof*⟩

lemma *im-len-eq-iff*: $|u| = |f\ u| \longleftrightarrow (\forall\ c.\ c \in set\ u \longrightarrow |f\ [c]| = 1)$
⟨*proof*⟩

lemma *im-len-less*: $a \in set\ u \Longrightarrow |f\ [a]| \neq 1 \Longrightarrow |u| < |f\ u|$
⟨*proof*⟩

end

lemma (**in** *morphism*) *nonerI*[*intro*]: **assumes** $(\bigwedge\ a.\ f^{\mathcal{C}}\ a \neq \varepsilon)$
shows *nonerasing-morphism f*
⟨*proof*⟩

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*⟩

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*⟩

locale *code-morphism* = *morphism f* **for** *f* +
assumes *code-morph*: *inj f*

begin

lemma *inj-core*: $inj\ f^{\mathcal{C}}$
⟨*proof*⟩

lemma *sing-im-core*: $f\ [a] \in (range\ f^{\mathcal{C}})$
⟨*proof*⟩

lemma *code-im*: $code\ (range\ f^{\mathcal{C}})$
⟨*proof*⟩

sublocale *code range* $f^{\mathcal{C}}$
⟨*proof*⟩

sublocale *nonerasing-morphism*
⟨*proof*⟩

lemma *code-morph-code*: **assumes** $f\ r = f\ s$ **shows** $r = s$
⟨*proof*⟩

lemma *code-morph-bij*: *bij-betw f UNIV* ⟨(*range* $f^{\mathcal{C}}$)⟩
⟨*proof*⟩

lemma *code-morphism-rev-map*: *code-morphism* (*rev-map f*)
⟨*proof*⟩

lemma *morph-on-inj-on*:
morphism-on f A inj-on f A
⟨*proof*⟩

end

lemma (**in** *morphism*) *code-morphismI*: *inj f* $\Longrightarrow$ *code-morphism f*
⟨*proof*⟩

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*⟩

4.1.5 Prefix code morphism

locale *pref-code-morphism* = *nonerasing-morphism* +
assumes
pref-free: $f^{\mathcal{C}}\ a \le p\ f^{\mathcal{C}}\ b \Longrightarrow a = b$

begin

interpretation *prefrange*: *pref-code* (*range* $f^{\mathcal{C}}$)
⟨*proof*⟩

lemma *inj-core*: *inj* $f^{\mathcal{C}}$
⟨*proof*⟩

sublocale *code-morphism*
⟨*proof*⟩

thm *nonerasing*

lemma *pref-free-morph*: **assumes** $f\ r \le p\ f\ s$ **shows** $r \le p\ s$

⟨*proof*⟩

end

4.1.6 Marked morphism

locale *marked-morphism* = *nonerasing-morphism* +
assumes
marked-core: $hd\ (f^{\mathcal{C}}\ a) = hd\ (f^{\mathcal{C}}\ b) \Longrightarrow a = b$

begin

lemma *marked-im*: *marked-code* (*range* $f^{\mathcal{C}}$)
⟨*proof*⟩

interpretation *marked-code* (*range* $f^{\mathcal{C}}$)
⟨*proof*⟩

lemmas *marked-morph* = *marked-core*[*unfolded core-sing*]

sublocale *pref-code-morphism*
⟨*proof*⟩

lemma *hd-im-eq-hd-eq*: **assumes** $u \neq \varepsilon$ **and** $v \neq \varepsilon$ **and** $hd\ (f\ u) = hd\ (f\ v)$
shows $hd\ u = hd\ v$
⟨*proof*⟩

lemma *marked-morph-lcp*: $f\ (r \wedge_p s) = f\ r \wedge_p f\ s$
⟨*proof*⟩

lemma *marked-inj-map*: *inj* $e \Longrightarrow$ *marked-morphism* ((*map* e) $\circ$ f)
⟨*proof*⟩

end

thm *morphism.nonerI*

lemma (**in** *morphism*) *marked-morphismI*:
$(\bigwedge a.\ f[a] \neq \varepsilon) \Longrightarrow (\bigwedge a\ b.\ a \neq b) \Longrightarrow hd\ (f[a]) \neq hd\ (f[b]) \Longrightarrow$ *marked-morphism* f
⟨*proof*⟩

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*: $\lceil id::('a\ list \Rightarrow 'a\ list)\rceil = 1$ **and** *min-im-len-id*: $\lfloor id::('a\ list \Rightarrow 'a\ list)\rfloor = 1$
⟨*proof*⟩

context *morphism*
begin

lemma *max-im-len-le*: *finite* (*length*'*range* $f^{\mathcal{C}}$) $\Longrightarrow |f\ z| \leq |z| * \lceil f\rceil$
⟨*proof*⟩

lemma *max-im-len-le-sing*: **assumes** *finite* (*length*'*range* $f^{\mathcal{C}}$)
shows $|f\ [a]| \leq \lceil f\rceil$
⟨*proof*⟩

lemma *min-im-len-ge*: *finite* (*length*'*range* $f^{\mathcal{C}}$) $\Longrightarrow |z| * \lfloor f\rfloor \leq |f\ z|$
⟨*proof*⟩

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*⟩

lemma *max-im-len-emp*: **assumes** *finite* (*length* ' *range* $f^{\mathcal{C}}$)
shows $\lceil f\rceil = 0 \longleftrightarrow (f = (\lambda w.\ \varepsilon))$
⟨*proof*⟩

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*]

end

4.1.8 Endomorphism

locale *endomorphism* = *morphism f* **for** *f*:: *'a list* ⇒ *'a list*
begin

lemma *pow-endomorphism*: *endomorphism* (*f*^^*k*)
⟨*proof*⟩

interpretation *pow-endm*: *endomorphism* (*f*^^*k*)
⟨*proof*⟩

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* ⟹ *set* $((f\widehat{\ }\widehat{\ }k)\ w)$ = *set* $((f\widehat{\ }\widehat{\ }k)\ v)$
⟨*proof*⟩

lemma *fin-len-ran-pow*: *finite* (*length* ‘ *range* $f^{\mathcal{C}}$) ⟹ *finite* (*length* ‘ *range* $(f\widehat{\ }\widehat{\ }k)^{\mathcal{C}}$)
⟨*proof*⟩

lemma *max-im-len-pow-le*: **assumes** *finite* (*length* ‘ *range* $f^{\mathcal{C}}$) **shows** $\lceil f\widehat{\ }\widehat{\ }k\rceil \leq \lceil f\rceil\widehat{\ }k$
⟨*proof*⟩

lemma *max-im-len-pow-le'*: *finite* (*length* ‘ *range* $f^{\mathcal{C}}$) ⟹ $|(f\widehat{\ }\widehat{\ }k)\ w| \leq |w| * \lceil f\rceil\widehat{\ }k$
⟨*proof*⟩

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* $(f\widehat{\ }\widehat{\ }k)$
⟨*proof*⟩

lemma *im-len-pow-mono*: **assumes** *nonerasing-morphism f i* ≤ *j*
shows $(|(f\widehat{\ }\widehat{\ }i)\ w| \leq |(f\widehat{\ }\widehat{\ }j)\ w|)$
⟨*proof*⟩

lemma *fac-mono-pow*: $u \leq f\ (f\widehat{\ }\widehat{\ }k)\ w \Longrightarrow (f\widehat{\ }\widehat{\ }l)\ u \leq f\ (f\widehat{\ }\widehat{\ }(l+k))\ w$
⟨*proof*⟩

lemma *rev-map-endomorph*: *endomorphism* (*rev-map f*)
⟨*proof*⟩

end

4.2 Primitivity preserving morphisms

locale *primitivity-preserving-morphism* = *nonerasing-morphism* +
assumes *prim-morph* : $2 \leq |u| \Longrightarrow$ *primitive u* ⟹ *primitive* (*f u*)
begin

sublocale *code-morphism*
⟨*proof*⟩

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* :: $'a\ list \Rightarrow ('a\ list \Rightarrow 'b\ list) \Rightarrow ('a\ list \Rightarrow 'b\ list) \Rightarrow bool$
(‹- ∈ - $=_M$ -› [80,80,80] 51)
where *min-sol-def*: $minimal\text{-}solution\ s\ g\ h \equiv s \neq \varepsilon \wedge g\ s = h\ s$
$\wedge (\forall\ s'.\ s' \neq \varepsilon \wedge s' \leq p\ s \wedge g\ s' = h\ s' \longrightarrow s' = s)$

lemma *min-solD*: $s \in g =_M h \Longrightarrow g\ s = h\ s$
⟨*proof*⟩

lemma *min-solD'*: $s \in g =_M h \Longrightarrow s \neq \varepsilon$
⟨*proof*⟩

lemma *min-solD-min*: $s \in g =_M h \Longrightarrow p \neq \varepsilon \Longrightarrow p \leq p\ s \Longrightarrow g\ p = h\ p \Longrightarrow p = s$
⟨*proof*⟩

lemma *min-solI*[*intro*]: $s \neq \varepsilon \Longrightarrow g\ s = h\ s \Longrightarrow (\bigwedge s'.\ s' \leq p\ s \Longrightarrow s' \neq \varepsilon \Longrightarrow g\ s' = h\ s' \Longrightarrow s' = s) \Longrightarrow s \in g =_M h$
⟨*proof*⟩

lemma *min-sol-sym-iff*: $s \in g =_M h \longleftrightarrow s \in h =_M g$
⟨*proof*⟩

lemma *min-sol-sym*[*sym*]: $s \in g =_M h \Longrightarrow s \in h =_M g$
⟨*proof*⟩

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*⟩

4.3.2 Coincidence pairs

definition *coincidence-set* :: $('a\ list \Rightarrow 'b\ list) \Rightarrow ('a\ list \Rightarrow 'b\ list) \Rightarrow ('a\ list \times 'a\ list)\ set$ (‹$\mathfrak{C}$›)
where *coincidence-set* $g\ h \equiv \{(r,s).\ g\ r = h\ s\}$

lemma *coin-set-eq*: $(g \circ fst)`(\mathfrak{C}\ g\ h) = (h \circ snd)`(\mathfrak{C}\ g\ h)$
⟨*proof*⟩

lemma *coin-setD*: $pair \in \mathfrak{C}\ g\ h \Longrightarrow g\ (fst\ pair) = h\ (snd\ pair)$
⟨*proof*⟩

lemma *coin-setD-iff*: $pair \in \mathfrak{C}\ g\ h \longleftrightarrow g\ (fst\ pair) = h\ (snd\ pair)$
⟨*proof*⟩

lemma *coin-set-sym*: $fst`(\mathfrak{C}\ g\ h) = snd\ `(\mathfrak{C}\ h\ g)$
⟨*proof*⟩

lemma *coin-set-inter-fst*: $(g \circ fst)`(\mathfrak{C}\ g\ h) = range\ g \cap range\ h$
⟨*proof*⟩

lemmas *coin-set-inter-snd* = *coin-set-inter-fst*[*unfolded coin-set-eq*]

definition *minimal-coincidence* :: $('a\ list \Rightarrow 'b\ list) \Rightarrow 'a\ list \Rightarrow ('a\ list \Rightarrow 'b\ list) \Rightarrow\ 'a\ list \Rightarrow bool$ (‹(- -) $=_m$ (- -)› [*80,81,80,81*] *51*)
where *min-coin-def*: $minimal\text{-}coincidence\ g\ r\ h\ s \equiv r \neq \varepsilon \wedge s \neq \varepsilon \wedge g\ r = h\ s \wedge (\forall\ r'\ s'.\ r' \leq np\ r \wedge s' \leq np\ s \wedge g\ r' = h\ s' \longrightarrow r' = r \wedge s' = s)$

definition *min-coincidence-set* :: $('a\ list \Rightarrow 'b\ list) \Rightarrow ('a\ list \Rightarrow 'b\ list) \Rightarrow ('a\ list \times 'a\ list)\ set$ (‹$\mathfrak{C}_m$›)
where *min-coincidence-set* $g\ h \equiv (\{(r,s)\ .\ g\ r =_m h\ s\})$

lemma *min-coin-minD*: $g\ r =_m h\ s \Longrightarrow r' \leq np\ r \Longrightarrow s' \leq np\ s \Longrightarrow g\ r' = h\ s' \Longrightarrow r' = r \wedge s' = s$
⟨*proof*⟩

lemma *min-coin-setD*: $p \in \mathfrak{C}_m\ g\ h \Longrightarrow g\ (fst\ p) =_m h\ (snd\ p)$
⟨*proof*⟩

lemma *min-coinD*: $g\ r =_m h\ s \Longrightarrow g\ r = h\ s$
⟨*proof*⟩

lemma *min-coinD'*: $g\ r =_m h\ s \Longrightarrow r \neq \varepsilon \wedge s \neq \varepsilon$
⟨*proof*⟩

lemma *min-coin-sub*: $\mathfrak{C}_m\ g\ h \subseteq \mathfrak{C}\ g\ h$
⟨*proof*⟩

lemma *min-coin-defI*: **assumes** $r \neq \varepsilon$ **and** $s \neq \varepsilon$ **and** $g\ r = h\ s$ **and**
$(\bigwedge r'\ s'.\ r' \leq np\ r \Longrightarrow s' \leq np\ s \Longrightarrow g\ r' = h\ s' \Longrightarrow r' = r \wedge s' = s)$
shows $g\ r =_m h\ s$
⟨*proof*⟩

lemma *min-coin-sym*[*sym*]: $g\ r =_m h\ s \Longrightarrow h\ s =_m g\ r$
⟨*proof*⟩

lemma *min-coin-sym-iff*: $g\ r =_m h\ s \longleftrightarrow h\ s =_m g\ r$
⟨*proof*⟩

lemma *min-coin-set-sym*: $fst`(\mathfrak{C}_m\ g\ h) = snd\ `(\mathfrak{C}_m\ h\ g)$
⟨*proof*⟩

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 $g\ u = h\ u$
⟨*proof*⟩

lemma *def-on-sings-eq*:
assumes $\bigwedge a.\ g\ [a] = h\ [a]$
shows $g = h$
⟨*proof*⟩

lemma *ims-prefs-comp*:
assumes $u \le p\ u'$ **and** $v \le p\ v'$ **and** $g\ u' \bowtie h\ v'$ **shows** $g\ u \bowtie h\ v$
⟨*proof*⟩

lemma *ims-sufs-comp*:
assumes $u \le s\ u'$ **and** $v \le s\ v'$ **and** $g\ u' \bowtie_s h\ v'$ **shows** $g\ u \bowtie_s h\ v$
⟨*proof*⟩

lemma *ims-hd-eq-comp*:
assumes $u \neq \varepsilon$ **and** $g\ u = h\ u$ **shows** $g\ [hd\ u] \bowtie h\ [hd\ u]$
⟨*proof*⟩

lemma *ims-last-eq-suf-comp*:
assumes $u \neq \varepsilon$ **and** $g\ u = h\ u$ **shows** $g\ [last\ u] \bowtie_s h\ [last\ u]$
⟨*proof*⟩

lemma *len-im-le*:
assumes $(\bigwedge a.\ a \in set\ s \Longrightarrow |g\ [a]| \le |h\ [a]|)$
shows $|g\ s| \le |h\ s|$
⟨*proof*⟩

lemma *len-im-less*:
assumes $\bigwedge a.\ a \in set\ s \Longrightarrow |g\ [a]| \le |h\ [a]|$ **and**
$b \in set\ s$ **and** $|g\ [b]| < |h\ [b]|$
shows $|g\ s| < |h\ s|$

⟨*proof*⟩

lemma *solution-eq-len-eq*:
assumes $g\ s = h\ s$ **and** $\bigwedge a.\ a \in set\ s \Longrightarrow |g\ [a]| = |h\ [a]|$
shows $\bigwedge a.\ a \in set\ s \Longrightarrow g\ [a] = h\ [a]$
⟨*proof*⟩

lemma *rev-maps*: *two-morphisms* (*rev-map g*) (*rev-map h*)
⟨*proof*⟩

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*⟩

lemma *min-sol-rev*[*reversal-rule*]:
assumes $s \in g =_M h$
shows $(rev\ s) \in (rev\text{-}map\ g) =_M (rev\text{-}map\ h)$
⟨*proof*⟩

lemma *coin-set-lists-concat*: $ps \in lists\ (\mathfrak{C}\ g\ h) \Longrightarrow g\ (concat\ (map\ fst\ ps)) = h\ (concat\ (map\ snd\ ps))$
⟨*proof*⟩

lemma *coin-set-hull*: $\langle snd\ `(\mathfrak{C}\ g\ h)\rangle = snd\ `(\mathfrak{C}\ g\ h)$
⟨*proof*⟩

lemma *min-sol-sufE*:
assumes $g\ r = h\ r$ **and** $r \neq \varepsilon$
obtains *e* **where** $e \in g =_M h$ **and** $e \leq s\ r$
⟨*proof*⟩

lemma *min-sol-primitive*: **assumes** $sol \in g =_M h$ **shows** *primitive sol*
⟨*proof*⟩

lemma *prim-sol-two-sols*:
assumes $g\ u = h\ u$ **and** $\neg\ u \in g =_M h$ **and** *primitive u*
obtains *s1 s2* **where** $s1 \in g =_M h$ **and** $s2 \in g =_M h$ **and** $s1 \neq s2$
⟨*proof*⟩

lemma *prim-sols-two-sols*:
assumes $g\ r = h\ r$ **and** $g\ s = h\ s$ **and** *primitive s* **and** *primitive r* **and** $r \neq s$
obtains *s1 s2* **where** $s1 \in g =_M h$ **and** $s2 \in g =_M h$ **and** $s1 \neq s2$
⟨*proof*⟩

end

4.3.4 Two nonerasing morphisms

Minimal coincidence pairs and minimal solutions make good sense for non-erasing 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*
⟨*proof*⟩

lemma *noner-eq-emp-iff*: $g\ u = h\ v \Longrightarrow u = \varepsilon \longleftrightarrow v = \varepsilon$
⟨*proof*⟩

lemma *min-coin-rev*:
assumes $g\ r =_m h\ s$
shows $(\mathit{rev\text{-}map}\ g)\ (\mathit{rev}\ r) =_m (\mathit{rev\text{-}map}\ h)\ (\mathit{rev}\ s)$
⟨*proof*⟩

lemma *min-coin-pref-eq*:
assumes $g\ e =_m h\ f$ **and** $g\ e' = h\ f'$ **and** $e' \le np\ e$ **and** $f' \bowtie f$
shows $e' = e$ **and** $f' = f$
⟨*proof*⟩

lemma *min-coin-prefE*:
assumes $g\ r = h\ s$ **and** $r \neq \varepsilon$
obtains *e f* **where** $g\ e =_m h\ f$ **and** $e \le p\ r$ **and** $f \le p\ s$ **and** $\mathit{hd}\ e = \mathit{hd}\ r$
⟨*proof*⟩

lemma *min-coin-dec*: **assumes** $g\ e = h\ f$
obtains *ps* **where** $\mathit{concat}\ (\mathit{map}\ \mathit{fst}\ \mathit{ps}) = e$ **and** $\mathit{concat}\ (\mathit{map}\ \mathit{snd}\ \mathit{ps}) = f$ **and**
$\bigwedge p.\ p \in \mathit{set}\ \mathit{ps} \Longrightarrow g\ (\mathit{fst}\ p) =_m h\ (\mathit{snd}\ p)$
⟨*proof*⟩

lemma *min-coin-code*:
assumes $\mathit{xs} \in \mathit{lists}\ (\mathfrak{C}_m\ g\ h)$ **and** $\mathit{ys} \in \mathit{lists}\ (\mathfrak{C}_m\ g\ h)$ **and**
$\mathit{concat}\ (\mathit{map}\ \mathit{fst}\ \mathit{xs}) = \mathit{concat}\ (\mathit{map}\ \mathit{fst}\ \mathit{ys})$ **and**
$\mathit{concat}\ (\mathit{map}\ \mathit{snd}\ \mathit{xs}) = \mathit{concat}\ (\mathit{map}\ \mathit{snd}\ \mathit{ys})$
shows $\mathit{xs} = \mathit{ys}$
⟨*proof*⟩

lemma *coin-closed*: $\mathit{ps} \in \mathit{lists}\ (\mathfrak{C}\ g\ h) \Longrightarrow (\mathit{concat}\ (\mathit{map}\ \mathit{fst}\ \mathit{ps}), \mathit{concat}\ (\mathit{map}\ \mathit{snd}\ \mathit{ps})) \in \mathfrak{C}\ g\ h$
⟨*proof*⟩

lemma *min-coin-gen-snd*: ⟨*snd* ‘ ($\mathfrak{C}_m$ *g h*)⟩ = *snd* ‘($\mathfrak{C}$ *g h*)
⟨*proof*⟩

lemma *min-coin-gen-fst*: ⟨*fst* ‘ ($\mathfrak{C}_m$ *g h*)⟩ = *fst* ‘($\mathfrak{C}$ *g h*)
⟨*proof*⟩

lemma *min-coin-code-snd*:
assumes *code-morphism g* **shows** *code* (*snd* ‘ ($\mathfrak{C}_m$ *g h*))
⟨*proof*⟩

lemma *min-coin-code-fst*:
code-morphism h ⟹ *code* (*fst* ‘ ($\mathfrak{C}_m$ *g h*))
⟨*proof*⟩

lemma *min-coin-basis-snd*:
assumes *code-morphism g*
shows $\mathfrak{B}$ (*snd* ‘($\mathfrak{C}$ *g h*)) = *snd* ‘ ($\mathfrak{C}_m$ *g h*)
⟨*proof*⟩

lemma *min-coin-basis-fst*:
code-morphism h ⟹ $\mathfrak{B}$ (*fst* ‘($\mathfrak{C}$ *g h*)) = *fst* ‘ ($\mathfrak{C}_m$ *g h*)
⟨*proof*⟩

lemma *sol-im-len-less*: **assumes** $g\ u = h\ u$ **and** $g \neq h$ **and** *set u* = *UNIV*
shows $|u| < |g\ u|$
⟨*proof*⟩

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*
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *min-coin-im-basis*:
$\mathfrak{B}$ (*h*‘ (*snd* ‘($\mathfrak{C}$ *g h*))) = *h* ‘ *snd* ‘ ($\mathfrak{C}_m$ *g h*)
$\mathfrak{B}$ (*g*‘ (*fst* ‘($\mathfrak{C}$ *g h*))) = *g* ‘ *fst* ‘ ($\mathfrak{C}_m$ *g h*)
⟨*proof*⟩

lemma *range-inter-basis-snd*:

shows $\mathfrak{B}$ (*range* $g \cap$ *range* h) $= h$ ‘ (*snd* ‘ $\mathfrak{C}_m$ g h)
$\mathfrak{B}$ (*range* $g \cap$ *range* h) $= g$ ‘ (*fst* ‘ $\mathfrak{C}_m$ g h)
⟨*proof*⟩

lemma *range-inter-code*:
shows *code* $\mathfrak{B}$ (*range* $g \cap$ *range* h)
⟨*proof*⟩

end

4.3.5 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
⟨*proof*⟩

lemmas *ne-g* = *g.nonerasing* **and**
ne-h = *h.nonerasing*

lemma *unique-continuation*:
$z \cdot g\ r = z' \cdot h\ s \Longrightarrow z \cdot g\ r' = z' \cdot h\ s' \Longrightarrow z \cdot g\ (r \wedge_p r') = z' \cdot h\ (s \wedge_p s')$
⟨*proof*⟩

lemmas *empty-sol* = *noner-eq-emp-iff*

lemma *comparable-min-sol-eq*: **assumes** $r \le p\ r'$ **and** $g\ r =_m h\ s$ **and** $g\ r' =_m h\ s'$
shows $r = r'$ **and** $s = s'$
⟨*proof*⟩

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 $e \le p\ e'$ **and** $f \le p\ f'$
⟨*proof*⟩

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*⟩

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*⟩

definition *pre-block* :: $'a \Rightarrow 'a$ *list* $\times$ $'a$ *list*
where *pre-block* a = (*THE* p. (g (*fst* p) $=_m$ h (*snd* p)) $\wedge$ *hd* (*fst* p) = a)
— *pre-block a* may not be a block, if no such exists

definition *blockP* :: $'a \Rightarrow$ *bool*
where *blockP* $a \equiv g$ (*fst* (*pre-block* a)) $=_m$ h (*snd* (*pre-block* a)) $\wedge$ *hd* (*fst* (*pre-block* a)) = a
— Predicate: the *pre-block* of the letter a is indeed a block

lemma *pre-blockI*: $g\ u =_m h\ v \Longrightarrow$ *pre-block* (*hd* u) = (u,v)
⟨*proof*⟩

lemma *blockI*: **assumes** $g\ u =_m h\ v$ **and** *hd* $u = a$
shows *blockP* a
⟨*proof*⟩

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*⟩

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*⟩

lemma *block-ex*: **assumes** $g\ u =_m h\ v$ **shows** *blockP* (*hd* u)
⟨*proof*⟩

lemma *sol-block-ex*: **assumes** $g\ u = h\ v$ **and** $u \neq \varepsilon$ **shows** *blockP* (*hd* u)
⟨*proof*⟩

definition *suc-fst* **where** *suc-fst* $\equiv \lambda$ x. *concat*(*map* (λ y. *fst* (*pre-block* y)) x)
definition *suc-snd* **where** *suc-snd* $\equiv \lambda$ x. *concat*(*map* (λ y. *snd* (*pre-block* y)) x)

lemma *blockP-D*: *blockP* $a \Longrightarrow g$ (*suc-fst* [a]) $=_m$ h (*suc-snd* [a])
⟨*proof*⟩

lemma *blockP-D-hd*: *blockP* $a \Longrightarrow$ *hd* (*suc-fst* [a]) = a
⟨*proof*⟩

abbreviation *blocks* $\tau \equiv (\forall\ a.\ a \in$ *set* $\tau \longrightarrow$ *blockP* a)

sublocale *sucs*: *two-morphisms suc-fst suc-snd*
⟨*proof*⟩

lemma *blockP-D-hd-hd*: **assumes** *blockP a*
shows $hd\ (h^{\mathcal{C}}\ (hd\ (suc\text{-}snd\ [a]))) = hd\ (g^{\mathcal{C}}\ a)$
⟨*proof*⟩

lemma *suc-morph-sings*: **assumes** $g\ e =_m h\ f$
shows $suc\text{-}fst\ [hd\ e] = e$ **and** $suc\text{-}snd\ [hd\ e] = f$
⟨*proof*⟩

lemma *blocks-eq*: $blocks\ \tau \Longrightarrow g\ (suc\text{-}fst\ \tau)\ = h\ (suc\text{-}snd\ \tau)$
⟨*proof*⟩

lemma *suc-eq'*: **assumes** $\bigwedge a.\ blockP\ a$ **shows** $g(suc\text{-}fst\ w) = h(suc\text{-}snd\ w)$
⟨*proof*⟩

lemma *suc-eq*: **assumes** *all-blocks*: $\bigwedge a.\ blockP\ a$ **shows** $g \circ suc\text{-}fst = h \circ suc\text{-}snd$
⟨*proof*⟩

lemma *block-eq*: $blockP\ a \Longrightarrow g\ (suc\text{-}fst\ [a]) = h\ (suc\text{-}snd\ [a])$
⟨*proof*⟩

lemma *blocks-inj-suc*: **assumes** $blocks\ \tau$ **shows** $inj\text{-}on\ suc\text{-}fst^{\mathcal{C}}\ (set\ \tau)$
⟨*proof*⟩

lemma *blocks-inj-suc'*: **assumes** $blocks\ \tau$ **shows** $inj\text{-}on\ suc\text{-}snd^{\mathcal{C}}\ (set\ \tau)$
⟨*proof*⟩

lemma *blocks-marked-code*: **assumes** $blocks\ \tau$
shows $marked\text{-}code\ (suc\text{-}fst^{\mathcal{C}}\ `(set\ \tau))$
⟨*proof*⟩

lemma *blocks-marked-code'*: **assumes** *all-blocks*: $\bigwedge a.\ a \in set\ \tau \Longrightarrow blockP\ a$
shows $marked\text{-}code\ (suc\text{-}snd^{\mathcal{C}}\ `(set\ \tau))$
⟨*proof*⟩

lemma *sucs-marked-morphs*: **assumes** *all-blocks*: $\bigwedge a.\ blockP\ a$
shows *two-marked-morphisms suc-fst suc-snd*
⟨*proof*⟩

lemma *pre-blocks-range*: $\{(e,f).\ g\ e =_m h\ f\ \} \subseteq range\ pre\text{-}block$
⟨*proof*⟩

corollary *card-blocks*: **assumes** $finite\ (UNIV :: 'a\ set)$ **shows** $card\ \{(e,f).\ g\ e =_m h\ f\ \} \leq card\ (UNIV :: 'a\ set)$
⟨*proof*⟩

lemma *block-decomposition*: **assumes** $g\ e = h\ f$
obtains τ **where** $suc\text{-}fst\ \tau = e$ **and** $suc\text{-}snd\ \tau = f$ **and** $blocks\ \tau$
⟨*proof*⟩

lemma *block-decomposition-unique*: **assumes** $g\ e = h\ f$ **and**
suc-fst $\tau = e$ **and** *suc-fst* $\tau' = e$ **and** *blocks* τ **and** *blocks* τ' **shows** $\tau = \tau'$
⟨*proof*⟩

lemma *block-decomposition-unique'*: **assumes** $g\ e = h\ f$ **and**
suc-snd $\tau = f$ **and** *suc-snd* $\tau' = f$ **and** *blocks* τ **and** *blocks* τ'
shows $\tau = \tau'$
⟨*proof*⟩

lemma *comm-sings-block*: **assumes** $g[a] \cdot h[b] = h[b] \cdot g[a]$
obtains $m\ n$ **where** *suc-fst* $[a] = [a]^{@}Suc\ m$ **and** *suc-snd* $[a] = [b]^{@}Suc\ n$
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *encoding-inj*: *inj sucs-encoding*
⟨*proof*⟩

lemma *map-encoding-inj*: *inj* (*map sucs-encoding*)
⟨*proof*⟩

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'*
⟨*proof*⟩

lemma *encoded-sucs-eq-conv'*: *suc-fst* = *suc-snd* ⟷ *suc-fst'* = *suc-snd'*
⟨*proof*⟩

lemma *encoded-sucs*: **assumes** ⋀ *c*. *blockP c* **shows** *two-marked-morphisms suc-fst'* *suc-snd'*
⟨*proof*⟩

lemma *encoded-sucs-len*: |*suc-fst w*| = |*suc-fst' w*| **and** |*suc-snd w*| = |*suc-snd' w*|
⟨*proof*⟩

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 *Combinatorics-Words.CoWBasic* as *two-pers-root*: $[\![<_p w\ (u \cdot w); <_p w\ (v \cdot w); |u| + |v| \leq |w|]\!] \Longrightarrow u \cdot v = v \cdot u$.

theorem *per-lemma-relaxed*:
  assumes *period w p* **and** *period w q* **and** $p + q \leq |w|$
  shows (*take p w*)·(*take q w*) = (*take q w*)·(*take p w*)
  ⟨*proof*⟩

Also in terms of the numeric period:

thm *two-periods*

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.

theorem *per-lemma-comm*:
  assumes $w \leq p\ r \cdot w$ **and** $w \leq p\ s \cdot w$
    and *len*: $|r| + |s| - (gcd\ |r|\ |s|) \leq |w|$
  shows $r \cdot s = s \cdot r$
  ⟨*proof*⟩

lemma *per-lemma-comm-pref*:
  assumes $u \leq p\ r^{@}k\ u \leq p\ s^{@}l$
    and *len*: $|r| + |s| - gcd\ (|r|)\ (|s|) \leq |u|$

shows $r \cdot s = s \cdot r$
⟨*proof*⟩

We can now prove the numeric version.

theorem *per-lemma*: **assumes** *period w p* **and** *period w q* **and** *len*: $p + q - gcd\ p\ q \leq |w|$
shows *period w* (*gcd p q*)
⟨*proof*⟩

5.2 Optimality

FW-word (where FW stands for Fine and Wilf) yields a word which show the optimality of the bound in the Periodicity lemma. Moreover, the obtained word has maximum possible letters (each equality of letters is forced by periods). The latter is not proved here.

term *butlast* ([*0..<*(*gcd p q*)]$^{@}$(*p div* (*gcd p q*)))·[*gcd p q*]·(*butlast* ([*0..<*(*gcd p q*)]$^{@}$(*p div* (*gcd p q*))))

— an auxiliary claim
lemma *ext-per-sum*: **assumes** *period w p* **and** *period w q* **and** $p \leq |w|$
shows *period* ((*take p w*) · *w*) (*p*+*q*)
⟨*proof*⟩

definition *fw-p-per p q* ≡ *butlast* ([*0..<*(*gcd p q*)]$^{@}$(*p div* (*gcd p q*)))
definition *fw-base p q* ≡ *fw-p-per p q* · [*gcd p q*]· *fw-p-per p q*

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* = *q then* ε *else*
— base case *if gcd p q* = *q* − *p then fw-base p q*
— step *else* (*take p* (*FW-word p* (*q*−*p*))) · *FW-word p* (*q*−*p*))

lemma *FW-sym*: *FW-word p q* = *FW-word q p*
⟨*proof*⟩

theorem *fw-word′*: ¬ *p dvd q* ⟹ ¬ *q dvd p* ⟹
|*FW-word p q*| = *p* + *q* − *gcd p q* − *1* ∧ *period* (*FW-word p q*) *p* ∧ *period* (*FW-word p q*) *q* ∧ ¬ *period* (*FW-word p q*) (*gcd p q*)
⟨*proof*⟩

theorem *fw-word*: **assumes** ¬ *p dvd q* ¬ *q dvd p*
shows |*FW-word p q*| = *p* + *q* − *gcd p q* − *1* **and** *period* (*FW-word p q*) *p* **and** *period* (*FW-word p q*) *q* **and** ¬ *period* (*FW-word p q*) (*gcd p q*)
⟨*proof*⟩

Calculation examples

5.3 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 $[\![<_p\ ?w\ (?p \cdot ?w);\ <_s\ ?w\ (?w \cdot ?q);\ |?p| + |?q| \leq |?w|;\ \bigwedge r\ s\ k\ l\ m.\ [\![?p = (r \cdot s)\ ^@\ k;\ ?q = (s \cdot r)\ ^@\ l;\ ?w = (r \cdot s)\ ^@\ m \cdot r;\ primitive\ (r \cdot s)]\!] \Longrightarrow ?thesis]\!] \Longrightarrow ?thesis$

$[\![\leq_f\ ?w\ (?p\ ^@\ ?k);\ \leq_f\ ?w\ (?q\ ^@\ ?l);\ ?p \neq \varepsilon;\ ?q \neq \varepsilon;\ |?p| + |?q| \leq |?w|;\ \bigwedge r\ s\ m.\ [\![\varrho\ ?p \sim r \cdot s;\ \varrho\ ?q \sim r \cdot s;\ ?w = (r \cdot s)\ ^@\ m \cdot r;\ primitive\ (r \cdot s)]\!] \Longrightarrow ?thesis]\!] \Longrightarrow ?thesis$

$[\![\leq_f\ ?u\ (?r\ ^@\ ?k);\ \leq_f\ ?u\ (?s\ ^@\ ?l);\ ?r \neq \varepsilon;\ ?s \neq \varepsilon;\ |?r| + |?s| \leq |?u|]\!] \Longrightarrow \varrho\ ?r \sim \varrho\ ?s$

$[\![\leq_f\ ?u\ (?r\ ^@\ ?k);\ \leq_f\ ?u\ (?s\ ^@\ ?l);\ ?u \neq \varepsilon;\ |?r| + |?s| \leq |?u|]\!] \Longrightarrow \varrho\ ?r \sim \varrho\ ?s$

$[\![\leq_f\ ?w\ (?u\ ^@\ ?n);\ \leq_f\ ?w\ (?v\ ^@\ ?m);\ primitive\ ?u;\ primitive\ ?v;\ |?u| + |?v| \leq |?w|]\!] \Longrightarrow ?u \sim ?v$ that have a relaxed length assumption $|p| + |q| \leq |w|$ instead of $|p| + |q| - gcd\ |p|\ |q| \leq |w|$ (and which follow from the relaxed version of periodicity lemma $[\![\leq_p\ ?w\ (?u \cdot ?w);\ \leq_p\ ?w\ (?v \cdot ?w);\ |?u| + |?v| \leq |?w|]\!] \Longrightarrow ?u \cdot ?v = ?v \cdot ?u$.

lemma *per-lemma-pref-suf-gcd*: **assumes** $w <p\ p \cdot w$ **and** $w <s\ w \cdot q$ **and**
fw: $|p| + |q| - (gcd\ |p|\ |q|) \leq |w|$
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*⟩

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*⟩

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 $\varrho\ r \sim \varrho\ s$
⟨*proof*⟩

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*⟩

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 $u \sim v$
⟨*proof*⟩

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*⟩

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*⟩

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** $x \cdot z = z \cdot x$
⟨*proof*⟩

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*⟩

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'*:
$[\![\ x^{@}a \cdot y^{@}b = z^{@}c;\ \ 2 \leq a;\ \ 2 \leq b;\ 2 \leq c\ ]\!]$
$\Longrightarrow x \cdot y = y \cdot x$
⟨*proof*⟩

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*⟩
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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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** $j = 1 \mid k = 1$
⟨*proof*⟩

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*⟩

case $j = 1$

lemma *case-j1k2-primitive*: **assumes** $j = 1$ $2 \leq k$
shows *primitive* x
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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 $2 \leq k$ 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 $2 \leq k$ 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 @ $j \cdot y$ with $2 \leq j$
$r\ q\ t$ **where**
$x = (r \cdot q)$ @ $t \cdot r$ **and**
$y = q \cdot r \cdot r \cdot q$ **and**
$z = (r \cdot q)$ @ $t \cdot r \cdot r \cdot q$ **and**
$j = 2$ **and** $l = 2$ **and** $2 \leq t$ **and** $r{\cdot}q \neq q{\cdot}r$ **and**
primitive x **and** *primitive* y
⟨*proof*⟩

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*: $|y| \leq |x|$
and *j-min*: $1 \leq j$ **and** *k-min*: $1 \leq k$ **and** *l-min*: $2 \leq l$
shows
$x^@j \cdot y^@k = z^@l$
$\longleftrightarrow$
$(\exists\, r\ m\ n\ t.$
$x = r^@m \wedge y = r^@n \wedge z = r^@t \wedge m*j+n*k=t*l)$ — Case A: x,y is not a code
$\vee\ (j = 1 \wedge k = 1) \wedge$
$(\exists\, r\ q\ m\ n.$
$x = (r{\cdot}q)^@m{\cdot}r \wedge y = q{\cdot}(r{\cdot}q)^@n \wedge z = r{\cdot}q \wedge m + n + 1 = l \wedge r{\cdot}q \neq q{\cdot}r)$ — Case B
$\vee\ (j = 1 \wedge 2 \leq k) \wedge$
$(\exists\, r\ q.$
$x = (q{\cdot}r^@k)^@(l-1){\cdot}q \wedge y = r \wedge z = q{\cdot}r^@k \wedge r{\cdot}q \neq q{\cdot}r)$ — Case C
$\vee\ (j = 1 \wedge 2 \leq k) \wedge$
$(\exists\, r\ q\ t.\ 0 < t \wedge$
$x = ((q \cdot r) \cdot (r \cdot (q \cdot r)$ @ $t)$ @ $(k - 1))$ @ $(l - 2){\cdot}(((q \cdot r) \cdot$
$(r \cdot (q \cdot r)$ @ $t)$ @ $(k - 2)) \cdot r) \cdot q$
$\wedge\ y = r \cdot (q \cdot r)$ @ t
$\wedge\ z = (q \cdot r) \cdot (r \cdot (q \cdot r)$ @ $t)$ @ $(k - 1)$
$\wedge\ r{\cdot}q \neq q{\cdot}r)$ — Case D
$\vee\ (j = 2 \wedge k = 1 \wedge l = 2) \wedge$
$(\exists\, r\ q\ t.\ 2 \leq t \wedge$
$x = (r \cdot q)$ @ $t \cdot r \wedge y = q \cdot r \cdot r \cdot q$
$\wedge\ z = (r \cdot q)$ @ $t \cdot r \cdot r \cdot q\ \wedge r{\cdot}q \neq q{\cdot}r\ )$ — Case E

(**is** *?eq* =
(*?sol-per* $\vee$ (*?cond-j1k1* $\wedge$ *?sol-j1k1*) $\vee$
(*?cond-j1k2* $\wedge$ *?sol-j1k2-b*) $\vee$

(*?cond-j1k2* ∧ *?sol-j1k2-a*) ∨
(*?cond-j2k1l2* ∧ *?sol-j2k1l2*)))
⟨*proof*⟩

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** $x \cdot y \neq y \cdot x$
and $1 \leq j$ **and** $1 \leq k$ **and** $\neg$ *primitive*$(x^{@}j{\cdot}y^{@}k)$
and $1 \leq k'$ **and** $\neg$ *primitive*$(x^{@}j{\cdot}y^{@}k')$
shows $k = k'$
⟨*proof*⟩

lemma *LS-unique-distinct-le*: **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*$(x{\cdot}y^{@}k)$
and $|y| \leq |x|$
shows *False*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *LS-unique'*: **assumes** $x \cdot y \neq y \cdot x$
and $1 \leq j$ **and** $1 \leq k$ **and** $\neg$ *primitive*$(x^{@}j{\cdot}y^{@}k)$
and $1 \leq j'$ **and** $1 \leq k'$ **and** $\neg$ *primitive*$(x^{@}j'{\cdot}y^{@}k')$
shows $k = k'$
⟨*proof*⟩

lemma *LS-unique*: **assumes** $x \cdot y \neq y \cdot x$
and $1 \leq j$ **and** $1 \leq k$ **and** $\neg$ *primitive*$(x^{@}j{\cdot}y^{@}k)$
and $1 \leq j'$ **and** $1 \leq k'$ **and** $\neg$ *primitive*$(x^{@}j'{\cdot}y^{@}k')$
shows $j = j'$ **and** $k = k'$
⟨*proof*⟩

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*⟩

lemma (**in** *LS-len-le*) *case-j2k1-exp-le*:
  assumes $2 \leq j$ $k = 1$
  shows $j*|x| + 4 \leq |y| + 2*|x|$
⟨*proof*⟩

theorem *LS-exp-le-one*:
  assumes *eq*: $x \cdot y^{@} k = z^{@} l$
    and $2 \leq l$
    and $x \cdot y \neq y \cdot x$
    and $1 \leq k$
    shows $k*|y| + 4 \leq |x|+2*|y|$
⟨*proof*⟩

lemma *LS-exp-le-conv-rat*:
  fixes *x y k*::*'a*::*linordered-field*
  assumes $y > 0$
  shows $k * y + 4 \leq x + 2 * y \longleftrightarrow k \leq (x - 4)/y + 2$
  ⟨*proof*⟩

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 (*input*) *binUNIV* :: *binA set* **where** *binUNIV* ≡ *UNIV*

lemma *binUNIV-I* [*simp*, *intro*]: *bina* ∈ *A* ⟹ *binb* ∈ *A* ⟹ *A* = *UNIV*
  ⟨*proof*⟩

lemma *bin-basis-code*: *code* {𝔞,𝔟}
  ⟨*proof*⟩

lemma *bin-num*: *bina* = *0* *binb* = *1*
  ⟨*proof*⟩

lemma *binA-simps* [*simp*]: *bina* − *binb* = *binb* *binb* − *bina* = *binb* *1* − *bina* = *binb* *1* − *binb* = *bina* *a* − *a* = *bina* *1* − (*1* − *a*) = *a*
⟨*proof*⟩

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*)
⟨*proof*⟩

definition *bin-swap-morph* **where** *bin-swap-morph* ≡ *map* *bin-swap*

lemma *alphabet-or*[*simp*]: *a* = *bina* ∨ *a* = *binb*
⟨*proof*⟩

lemma *bin-im-or*: *f* [*a*] = *f* 𝔞 ∨ *f* [*a*] = *f* 𝔟
⟨*proof*⟩

thm *triv-forall-equality*

lemma *binUNIV-card*: *card* *binUNIV* = *2*
⟨*proof*⟩

lemma *other-letter*: **obtains** *b* **where** *b* ≠ (*a* :: *binA*)
⟨*proof*⟩

lemma *alphabet-or-neq*: *x* ≠ *y* ⟹ *x* = (*a* :: *binA*) ∨ *y* = *a*
⟨*proof*⟩

lemma *binA-neq-cases*: **assumes** *neq*: *a* ≠ *b*
obtains *a* = *bina* **and** *b* = *binb* | *a* = *binb* **and** *b* = *bina*
⟨*proof*⟩

lemma *bin-neq-sym-pred*: **assumes** *a* ≠ *b* **and** *P* *bina* *binb* **and** *P* *binb* *bina* **shows** *P* *a* *b*
⟨*proof*⟩

lemma *no-third*: (*c* :: *binA*) ≠ *a* ⟹ *b* ≠ *a* ⟹ *b* = *c*
⟨*proof*⟩

lemma *two-in-bin-UNIV*: **assumes** *a* ≠ *b* **and** *a* ∈ *S* **and** *b* ∈ *S* **shows** *S* = *binUNIV*
⟨*proof*⟩

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*⟩

lemma *bin-basis-singletons*: {[*q*] |*q*. *q* ∈ {*bina*, *binb*}} = {𝔞,𝔟}

⟨*proof*⟩

lemma *bin-basis-generates*: $\langle\{\mathfrak{a},\mathfrak{b}\}\rangle = UNIV$
⟨*proof*⟩

lemma *a-in-bin-basis*: $[a] \in \{\mathfrak{a},\mathfrak{b}\}$
⟨*proof*⟩

lemma *lcp-zero-one-emp*: $\mathfrak{a} \wedge_p \mathfrak{b} = \varepsilon$ **and** *lcp-one-zero-emp*: $\mathfrak{b} \wedge_p \mathfrak{a} = \varepsilon$
⟨*proof*⟩

lemma *bin-neq-induct*: $(a::binA) \neq b \Longrightarrow P\ a \Longrightarrow P\ b \Longrightarrow P\ c$
⟨*proof*⟩

lemma *bin-neq-induct'*: **assumes**$(a::binA) \neq b$ **and** $P\ a$ **and** $P\ b$ **shows** $\bigwedge c.\ P\ c$
⟨*proof*⟩

lemma *neq-exhaust*: **assumes** $(a::binA) \neq b$ **obtains** $c = a \mid c = b$
⟨*proof*⟩

lemma *bin-swap-neq* [*simp*]: $1-(a :: binA) \neq a$
⟨*proof*⟩
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'*: $P\ (a :: binA) \Longrightarrow P\ (1-a) \Longrightarrow (\bigwedge c.\ P\ c)$
⟨*proof*⟩

lemma *swap-UNIV*: $\{a,\ 1-a\} = binUNIV$ (**is** *?P a*)
⟨*proof*⟩

lemma *bin-neq-swap'*[*intro*]: $a \neq b \Longrightarrow 1 - b = (a :: binA)$
⟨*proof*⟩

lemma *bin-neq-swap*[*intro*]: $a \neq b \Longrightarrow 1 - a = (b :: binA)$
⟨*proof*⟩

lemma *bin-neq-swap''*[*intro*]: $a \neq b \Longrightarrow b = 1-(a:: binA)$
⟨*proof*⟩

lemma *bin-neq-swap'''*[*intro*]: $a \neq b \Longrightarrow a = 1-(b:: binA)$
⟨*proof*⟩

lemma *bin-neq-iff*: $c \neq d \longleftrightarrow 1 - d = (c :: binA)$
⟨*proof*⟩

lemma *bin-neq-iff'*: $c \neq d \longleftrightarrow 1 - c = (d :: binA)$

⟨*proof*⟩

lemma *binA-neq-cases-swap*: **assumes** *neq*: $a \neq (b :: binA)$
obtains $a = c$ **and** $b = 1 - c$ | $a = 1 - c$ **and** $b = c$
⟨*proof*⟩

lemma *im-swap-neq*: $f\ a = f\ b \Longrightarrow f\ bina \neq f\ binb \Longrightarrow a = b$
⟨*proof*⟩

lemma *bin-without-letter*: **assumes** $(a1 :: binA) \notin set\ w$
obtains k **where** $w = [1-a1]^{@}k$
⟨*proof*⟩

lemma *bin-empty-iff*: $S = \{\} \longleftrightarrow (a :: binA) \notin S \wedge 1-a \notin S$
⟨*proof*⟩

lemma *bin-UNIV-iff*: $S = binUNIV \longleftrightarrow a \in S \wedge 1-a \in S$
⟨*proof*⟩

lemma *bin-UNIV-I*: $a \in S \Longrightarrow 1-a \in S \Longrightarrow S = binUNIV$
⟨*proof*⟩

lemma *bin-sing-iff*: $A = \{a :: binA\} \longleftrightarrow a \in A \wedge 1-a \notin A$
⟨*proof*⟩

lemma *bin-set-cases*: **obtains** $S = \{\}$ | $S = \{bina\}$ | $S = \{binb\}$ | $S = binUNIV$
⟨*proof*⟩

lemma *not-UNIV-E*: **assumes** $A \neq binUNIV$ **obtains** a **where** $A \subseteq \{a\}$
⟨*proof*⟩

lemma *not-UNIV-nempE*: **assumes** $A \neq binUNIV$ **and** $A \neq \{\}$ **obtains** a **where** $A = \{a\}$
⟨*proof*⟩

lemma *bin-sing-gen-iff*: $x \in \langle\{[a]\}\rangle \longleftrightarrow 1-(a :: binA) \notin set\ x$
⟨*proof*⟩

lemma *set-hd-pow-conv*: $w \in [hd\ w]* \longleftrightarrow set\ w \neq binUNIV$
⟨*proof*⟩

lemma *not-swap-eq*: $P\ a\ b \Longrightarrow (\bigwedge (c :: binA).\ \neg\ P\ c\ (1-c)) \Longrightarrow a = b$
⟨*proof*⟩

lemma *bin-distinct-letter*: **assumes** $set\ w = binUNIV$
obtains $k\ w'$ **where** $[hd\ w]^{@}Suc\ k \cdot [1-hd\ w] \cdot w' = w$
⟨*proof*⟩

lemma $P\ a \longleftrightarrow P\ (1-a) \Longrightarrow P\ a \Longrightarrow (\bigwedge (b :: binA).\ P\ b)$

⟨*proof*⟩

lemma *bin-sym-all*: $P\ (a :: binA) \longleftrightarrow P\ (1{-}a) \Longrightarrow P\ a \Longrightarrow P\ x$
⟨*proof*⟩

lemma *bin-sym-all-comm*:
$f\ [a] \cdot f\ [1{-}a] \neq f\ [1{-}a] \cdot f\ [a] \Longrightarrow f\ [b] \cdot f\ [1{-}b] \neq f\ [1{-}b] \cdot f\ [(b :: binA)]$ (**is** $?P\ a \Longrightarrow ?P\ b$)
⟨*proof*⟩

lemma *bin-sym-all-neq*:
$f\ [(a :: binA)] \neq f\ [1{-}a] \Longrightarrow f\ [b] \neq f\ [1{-}b]$ (**is** $?P\ a \Longrightarrow ?P\ b$)
⟨*proof*⟩

lemma *bin-len-count*:
fixes w :: *binA list*
shows $|w|$ = *count-list* w a + *count-list* w $(1{-}a)$
⟨*proof*⟩

lemma *bin-len-count′*:
fixes w :: *binA list*
shows $|w|$ = *count-list* w *bina* + *count-list* w *binb*
⟨*proof*⟩

7.2 Binary morphisms

lemma *bin-map-core-lists*: (*map* $f^{\mathcal{C}}$ w) ∈ *lists* $\{f\ \mathfrak{a}, f\ \mathfrak{b}\}$
⟨*proof*⟩

lemma *bin-range*: *range* f = {f *bina*, f *binb*}
⟨*proof*⟩

lemma *bin-core-range*: *range* $f^{\mathcal{C}} = \{f\ \mathfrak{a}, f\ \mathfrak{b}\}$
⟨*proof*⟩

lemma *bin-core-range-swap*: *range* $f^{\mathcal{C}} = \{f\ [(a :: binA)], f\ [1{-}a]\}$ (**is** $?P\ a$)
⟨*proof*⟩

lemma *bin-map-core-lists-swap*: (*map* $f^{\mathcal{C}}$ w) ∈ *lists* $\{f\ [(a :: binA)], f\ [1{-}a]\}$
⟨*proof*⟩

locale *binary-morphism* = *morphism f*
for f :: *binA list* ⇒ *′a list*
begin

lemma *bin-len-count-im*:
fixes a :: *binA*
shows $|f\ w|$ = *count-list* w a ∗ $|f\ [a]|$ + *count-list* w $(1{-}a)$ ∗ $|f\ [1{-}a]|$
⟨*proof*⟩

lemma *bin-len-count-im'*:
shows $|f\ w| = \textit{count-list}\ w\ \textit{bina} * |f\ \mathfrak{a}| + \textit{count-list}\ w\ \textit{binb} * |f\ \mathfrak{b}|$
⟨*proof*⟩

lemma *bin-neq-inj-core*: **assumes** $f\ [a] \neq f\ [1{-}a]$ **shows** $\textit{inj}\ f^{\mathcal{C}}$
⟨*proof*⟩

lemma *bin-code-morphism-inj*: **assumes** $f\ [a] \cdot f\ [1{-}a] \neq f\ [1{-}a] \cdot f\ [a]$
shows *inj f*
⟨*proof*⟩

lemma *bin-code-morphismI*: $f\ [a] \cdot f\ [1{-}a] \neq f\ [1{-}a] \cdot f\ [a] \Longrightarrow \textit{code-morphism}\ f$
⟨*proof*⟩

end

7.2.1 Binary periodic morphisms

locale *binary-periodic-morphism* = *periodic-morphism f*
for *f* :: *binA list* ⇒ *'a list*
begin

sublocale *binary-morphism*
⟨*proof*⟩

definition *fn0* **where** $\textit{fn0} \equiv (\textit{SOME}\ n.\ f\ \mathfrak{a} = \textit{mroot}^{@}n)$
definition *fn1* **where** $\textit{fn1} \equiv (\textit{SOME}\ n.\ f\ \mathfrak{b} = \textit{mroot}^{@}n)$

lemma *bin0-im*: $f\ \mathfrak{a} = \textit{mroot}^{@}\textit{fn0}$
⟨*proof*⟩

lemma *bin1-im*: $f\ \mathfrak{b} = \textit{mroot}^{@}\textit{fn1}$
⟨*proof*⟩

lemma *sorted-image* : $f\ w = (f\ [a])^{@}(\textit{count-list}\ w\ a) \cdot (f\ [1{-}a])^{@}(\textit{count-list}\ w\ (1{-}a))$
⟨*proof*⟩

lemma *bin-per-morph-expI*: $f\ u = \textit{mroot}^{@}((\textit{mexp}\ \textit{bina}) * (\textit{count-list}\ u\ \textit{bina}) + (\textit{mexp}\ \textit{binb}) * (\textit{count-list}\ u\ \textit{binb}))$
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *bin-morph-of-case-def*: *bin-morph-of x y u* = *concat* (*map* (λ *a*. (*case a of bina* ⇒ *x* | *binb* ⇒ *y*)) *u*)
⟨*proof*⟩

lemma *case-finiteD*: *case-finite-2* (*f* 𝔞) (*f* 𝔟) = $f^{\mathcal{C}}$
⟨*proof*⟩

lemma *case-finiteD'*: *case-finite-2* (*f* 𝔞) (*f* 𝔟) *u* = $f^{\mathcal{C}}$ *u*
⟨*proof*⟩

lemma *bin-morph-of-maps*: *bin-morph-of x y* = *List.maps* (*case-finite-2 x y*)
⟨*proof*⟩

lemma *bin-morph-ofD*: (*bin-morph-of x y*) 𝔞 = *x* (*bin-morph-of x y*) 𝔟 = *y*
⟨*proof*⟩

lemma *bin-range-swap*: *range f* = {*f* (*a*::*binA*), *f* (*1*−*a*)} (**is** *?P a*)
⟨*proof*⟩

lemma *bin-morph-of-core-range*: *range* (*bin-morph-of x y*)$^{\mathcal{C}}$ = {*x*,*y*}
⟨*proof*⟩

lemma *bin-morph-of-morph*: *morphism* (*bin-morph-of x y*)
⟨*proof*⟩

lemma *bin-morph-of-bin-morph*: *binary-morphism* (*bin-morph-of x y*)
⟨*proof*⟩

lemma *bin-morph-of-range*: *range* (*bin-morph-of x y*) = ⟨{*x*,*y*}⟩
⟨*proof*⟩

context *binary-code*
begin

lemma *code-morph-of*: *code-morphism* (*bin-morph-of* u_0 u_1)
⟨*proof*⟩

lemma *inj-morph-of*: *inj* (*bin-morph-of* u_0 u_1)

⟨*proof*⟩

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*: $g\ \mathfrak{a} = h\ \mathfrak{a} \Longrightarrow g\ \mathfrak{b} = h\ \mathfrak{b} \Longrightarrow g = h$
⟨*proof*⟩

sublocale *g*: *binary-morphism g*
⟨*proof*⟩
sublocale *h*: *binary-morphism h*
⟨*proof*⟩

lemma *rev-morphs*: *two-binary-morphisms* (*rev-map g*) (*rev-map h*)
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*⟩

lemma *len-ims-sing-neq*:
assumes $g\ s = h\ s\ g \neq h\ set\ s = binUNIV$
shows $|g\ [c]| \neq |h\ [c]|$
⟨*proof*⟩

end

lemma *two-binary-morphismsI*: *binary-morphism g* $\Longrightarrow$ *binary-morphism h* $\Longrightarrow$ *two-binary-morphisms g h*
⟨*proof*⟩

7.5 Binary code morphism

7.5.1 Locale - binary code morphism

locale *binary-code-morphism* = *code-morphism f* :: *binA list* ⇒ *'a list* **for** *f*

begin

lemma *morph-bin-morph-of*: f = *bin-morph-of* (f $\mathfrak{a}$) (f $\mathfrak{b}$)
⟨*proof*⟩

lemma *non-comm-morph* [*simp*]: f [a] · f [*1*−a] ≠ f [*1*−a] · f [a]
⟨*proof*⟩

lemma *non-comp-morph*: ¬ f [a] · f [*1*−a] ⋈ f [*1*−a] · f [a]
⟨*proof*⟩

lemma *swap-non-comm-morph* [*simp*, *intro*]: $a \neq b \Longrightarrow f$ [a] · f [b] ≠ f [b] · f [a]
⟨*proof*⟩

thm *bin-core-range*[*of f*]

lemma *bin-code-morph-rev-map*: *binary-code-morphism* (*rev-map f*)
⟨*proof*⟩

sublocale *swap*: *binary-code f* $\mathfrak{b}$ *f* $\mathfrak{a}$
⟨*proof*⟩

sublocale *binary-code f* $\mathfrak{a}$ *f* $\mathfrak{b}$
⟨*proof*⟩

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 $\mathfrak{a}$) (f $\mathfrak{b}$)
abbreviation *bin-morph-mismatch* (‹$\mathfrak{c}$›)
where *bin-morph-mismatch a* ≡ *bin-mismatch* (f[a]) (f[*1*−a])
abbreviation *bin-morph-mismatch-suf* (‹$\mathfrak{d}$›)
where *bin-morph-mismatch-suf a* ≡ *bin-mismatch-suf* (f[*1*−a]) (f[a])

lemma *bin-lcp-def'*: α = f ([a] · [*1*−a]) $\wedge_p$ f ([*1*−a] · [a])
⟨*proof*⟩

lemma *bin-lcp-neq*: $a \neq b \Longrightarrow \alpha$ = f ([a] · [b]) $\wedge_p$ f ([b] · [a])

⟨*proof*⟩

lemma *sing-im*: $f\ [a] \in\ \{f\ \mathfrak{a},\ f\ \mathfrak{b}\}$
⟨*proof*⟩

lemma *bin-mismatch-inj*: *inj* $\mathfrak{c}$
⟨*proof*⟩

lemma *map-in-lists*: $map\ (\lambda x.\ f\ [x])\ w \in lists\ \{f\ \mathfrak{a},\ f\ \mathfrak{b}\}$
⟨*proof*⟩

lemma *bin-morph-lcp-short*: $|\alpha| < |f\ [a]| + |f[1-a]|$
⟨*proof*⟩

lemma *swap-not-pref-bin-lcp*: $\neg\ f([a] \cdot [1-a]) \leq p\ \alpha$
⟨*proof*⟩

thm *local.bin-mismatch-inj*

lemma *bin-mismatch-suf-inj*: *inj* $\mathfrak{d}$
⟨*proof*⟩

lemma *bin-lcp-sing*: $bin\text{-}lcp\ (f\ [a])\ (f\ [1-a]) = \alpha$
⟨*proof*⟩

lemma *bin-lcs-sing*: $bin\text{-}lcs\ (f\ [a])\ (f\ [1-a]) = \beta$
⟨*proof*⟩

lemma *bin-code-morph-sing*: $binary\text{-}code\ (f\ [a])\ (f\ [1-a])$
⟨*proof*⟩

lemma *bin-mismatch-swap-neq*: $\mathfrak{c}\ a \neq \mathfrak{c}\ (1-a)$
⟨*proof*⟩

lemma *long-bin-lcp-hd*: **assumes** $|f\ w| \leq |\alpha|$
shows $w \in [hd\ w]*$
⟨*proof*⟩

thm *nonerasing*
nonerasing-morphism.nonerasing
lemmas *nonerasing* = *nonerasing*
thm *nonerasing-morphism.nonerasing*
binary-code-morphism.nonerasing

lemma *bin-morph-lcp-mismatch-pref*:
$\alpha \cdot [\mathfrak{c}\ a] \leq p\ f\ [a] \cdot \alpha$
⟨*proof*⟩

lemma $[\mathfrak{d}\ a] \cdot \beta \leq s\ \beta \cdot f\ [a]$ ⟨*proof*⟩

lemma *bin-lcp-pref-all*: $\alpha \le p\ f\ w \cdot \alpha$
⟨*proof*⟩

lemma *bin-lcp-spref-all*: $w \neq \varepsilon \Longrightarrow \alpha <p\ f\ w \cdot \alpha$
⟨*proof*⟩

lemma *pref-mono-lcp*: **assumes** $w \le p\ w'$ **shows** $f\ w \cdot \alpha \le p\ f\ w' \cdot \alpha$
⟨*proof*⟩

lemma *long-bin-lcp*: **assumes** $w \neq \varepsilon$ **and** $|f\ w| \le |\alpha|$
shows $w \in [hd\ w]*$
⟨*proof*⟩

thm *sing-to-nemp*
nonerasing

lemma *bin-mismatch-code-morph*: $c_0 = \mathfrak{c}\ 0\ c_1 = \mathfrak{c}\ 1$
⟨*proof*⟩

lemma *bin-lcp-mismatch-pref-all*: $\alpha \cdot [\mathfrak{c}\ a] \le p\ f\ [a] \cdot f\ w \cdot \alpha$
⟨*proof*⟩

lemma *bin-fst-mismatch-all*: $\alpha \cdot [c_0] \le p\ f\ \mathfrak{a} \cdot f\ w \cdot \alpha$
⟨*proof*⟩

lemma *bin-snd-mismatch-all*: $\alpha \cdot [c_1] \le p\ f\ \mathfrak{b} \cdot f\ w \cdot \alpha$
⟨*proof*⟩

lemma *bin-long-mismatch*: **assumes** $|\alpha| < |f\ w|$ **shows** $\alpha \cdot [\mathfrak{c}\ (hd\ w)] \le p\ f\ w$
⟨*proof*⟩

lemma *sing-pow-mismatch*: **assumes** $f\ [a] = [b]^{@} Suc\ n$ **shows** $\mathfrak{c}\ a = b$
⟨*proof*⟩

lemma *sing-pow-mismatch-suf*: $f\ [a] = [b]^{@} Suc\ n \Longrightarrow \mathfrak{d}\ a = b$
⟨*proof*⟩

lemma *bin-lcp-swap-hd*: $f\ [a] \cdot f\ w \cdot \alpha\ \wedge_p\ f\ [1-a] \cdot f\ w' \cdot \alpha = \alpha$
⟨*proof*⟩

lemma *bin-lcp-neq-hd*: $a \neq b \Longrightarrow f\ [a] \cdot f\ w \cdot \alpha\ \wedge_p\ f\ [b] \cdot f\ w' \cdot \alpha = \alpha$
⟨*proof*⟩

lemma *bin-mismatch-swap-not-comp*: $\neg\ f\ [a] \cdot f\ w \cdot \alpha \bowtie f\ [1-a] \cdot f\ w' \cdot \alpha$
⟨*proof*⟩

lemma *bin-lcp-root*: $\alpha <p\ f\ [a] \cdot \alpha$

⟨*proof*⟩

lemma *bin-lcp-pref*: **assumes** $w \notin \mathfrak{b}*$ **and** $w \notin \mathfrak{a}*$
shows $\alpha \leq p$ (*f w*)
⟨*proof*⟩

lemma *bin-lcp-pref''*: $[a] \leq f\ w \Longrightarrow [1-a] \leq f\ w \Longrightarrow \alpha \leq p\ (f\ w)$
⟨*proof*⟩
lemma *bin-lcp-pref'*: $\mathfrak{a} \leq f\ w \Longrightarrow \mathfrak{b} \leq f\ w \Longrightarrow \alpha \leq p\ (f\ w)$
⟨*proof*⟩

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*⟩

lemma *bin-lcp-comp-hd*: $\alpha \bowtie f\ (\mathfrak{a} \cdot w0) \wedge_p\ f\ (\mathfrak{b} \cdot w1)$
⟨*proof*⟩

lemma *sing-mismatch*: **assumes** $f\ \mathfrak{a} \in [a]*$ **shows** $c_0 = a$
⟨*proof*⟩

lemma *sing-mismatch'*: **assumes** $f\ \mathfrak{b} \in [a]*$ **shows** $c_1 = a$
⟨*proof*⟩

lemma *bin-lcp-comp-all*: $\alpha \bowtie (f\ w)$
⟨*proof*⟩

lemma *not-comp-bin-swap*: $\neg\ f\ [a] \cdot \alpha \bowtie f\ [1-a] \cdot \alpha$
⟨*proof*⟩

lemma *mismatch-pref*:
assumes $\alpha \leq p\ f\ ([a] \cdot w0)$ **and** $\alpha \leq p\ f\ ([1-a] \cdot w1)$
shows $\alpha = f\ ([a] \cdot w0) \wedge_p f\ ([1-a] \cdot w1)$
⟨*proof*⟩

lemma *bin-set-UNIV-length*: **assumes** $set\ w = UNIV$ **shows** $|f\ [a]| + |f\ [1-a]| \leq |f\ w|$
⟨*proof*⟩

lemma *set-UNIV-bin-lcp-pref*: **assumes** $set\ w = UNIV$ **shows** $\alpha \cdot [\mathfrak{c}\ (hd\ w)] \leq p\ f\ w$
⟨*proof*⟩

lemmas *not-comp-bin-lcp-pref* = *bin-not-comp-set-UNIV*[*THEN set-UNIV-bin-lcp-pref*]

lemma *marked-lcp-conv*: *marked-morphism f* $\longleftrightarrow \alpha = \varepsilon$
⟨*proof*⟩

lemma *im-comm-lcp*: $f\ w \cdot \alpha = \alpha \cdot f\ w \Longrightarrow (\forall\ a.\ a \in set\ w \longrightarrow f\ [a] \cdot \alpha = \alpha \cdot f\ [a])$
⟨*proof*⟩

lemma *im-comm-lcp-nemp*: **assumes** $f\ w \cdot \alpha = \alpha \cdot f\ w$ **and** $w \neq \varepsilon$ **and** $\alpha \neq \varepsilon$
obtains k **where** $w = [hd\ w]^{@}Suc\ k$
⟨*proof*⟩

lemma *bin-lcp-ims-im-lcp*: $f\ w \cdot \alpha \wedge_p f\ w' \cdot \alpha = f\ (w \wedge_p w') \cdot \alpha$
⟨*proof*⟩

lemma *per-comp*:
assumes $r <p\ f\ w \cdot r$
shows $r \bowtie f\ w \cdot \alpha$
⟨*proof*⟩

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*⟩

lemma *bin-code-morph-iff*: *binary-code-morphism* (*bin-morph-of* $x\ y$) $\longleftrightarrow x \cdot y \neq y \cdot x$
⟨*proof*⟩

lemma *bin-noner-morph-iff*: *nonerasing-morphism* (*bin-morph-of* $x\ y$) $\longleftrightarrow x \neq \varepsilon \wedge y \neq \varepsilon$
⟨*proof*⟩

lemma *morph-bin-morph-of*: *morphism* $f \longleftrightarrow$ *bin-morph-of* $(f\ \mathfrak{a})\ (f\ \mathfrak{b}) = f$
⟨*proof*⟩

lemma *two-bin-code-morphs-nonerasing-morphs*: *binary-code-morphism g* $\Longrightarrow$ *binary-code-morphism h* $\Longrightarrow$ *two-nonerasing-morphisms g h*
⟨*proof*⟩

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*⟩

locale *marked-binary-morphism* = *marked-morphism f* :: *binA list* ⇒ *'a list* **for** *f*

begin

lemma *bin-marked*: *hd* (*f* 𝔞) $\neq$ *hd* (*f* 𝔟)
⟨*proof*⟩

lemma *bin-marked-sing*: *hd* (*f* [*a*]) $\neq$ *hd* (*f* [*1−a*])
⟨*proof*⟩

sublocale *binary-code-morphism*
⟨*proof*⟩

lemma *marked-lcp-emp*: $\alpha = \varepsilon$
⟨*proof*⟩

lemma *bin-marked'*: (*f* 𝔞)!*0* $\neq$ (*f* 𝔟)!*0*
⟨*proof*⟩

lemma *marked-bin-morph-pref-code*: *r* ⋈ *s* ∨ *f* (*r* · *z1*) $\wedge_p$ *f* (*s* · *z2*) = *f* (*r* $\wedge_p$ *s*)
⟨*proof*⟩

end

lemma *bin-marked-preimg-hd*:
assumes *marked-binary-morphism* (*f* :: *binA list* ⇒ *binA list*)
obtains *c* **where** *hd* (*f* [*c*]) = *a*
⟨*proof*⟩

7.7 Marked version

context *binary-code-morphism*

begin

definition *marked-version* ($\langle f_m \rangle$) **where** $f_m = (\lambda\ w.\ \alpha^{-1>}(f\ w \cdot \alpha))$

lemma *marked-version-conjugates*: $\alpha \cdot f_m\ w = f\ w \cdot \alpha$
⟨*proof*⟩

lemma *marked-eq-conv*: $f\ w = f\ w' \longleftrightarrow f_m\ w = f_m\ w'$
⟨*proof*⟩

lemma *marked-marked*: **assumes** *marked-morphism* f **shows** $f_m = f$
⟨*proof*⟩

lemma *marked-version-all-nemp*: $w \neq \varepsilon \Longrightarrow f_m\ w \neq \varepsilon$
⟨*proof*⟩

lemma *marked-version-binary-code-morph*: *binary-code-morphism* f_m
⟨*proof*⟩

interpretation *mv-bcm*: *binary-code-morphism* f_m
⟨*proof*⟩

lemma *marked-lcs*: *bin-lcs* $(f_m\ \mathfrak{a})\ (f_m\ \mathfrak{b}) = \beta \cdot \alpha$
⟨*proof*⟩

lemma *bin-lcp-shift*: **assumes** $|\alpha| < |f\ w|$ **shows** $(f\ w)!|\alpha| = hd\ (f_m\ w)$
⟨*proof*⟩

lemma *mismatch-fst*: $hd\ (f_m\ \mathfrak{a}) = c_0$
⟨*proof*⟩

lemma *mismatch-snd*: $hd\ (f_m\ \mathfrak{b}) = c_1$
⟨*proof*⟩

lemma *marked-hd-neq*: $hd\ (f_m\ [a]) \neq hd\ (f_m\ [1{-}a])$ (**is** *?P* $(a :: binA)$)
⟨*proof*⟩

lemma *marked-version-marked-morph*: *marked-morphism* f_m
⟨*proof*⟩

interpretation *mv-mbm*: *marked-binary-morphism* f_m
⟨*proof*⟩

lemma *bin-code-pref-morph*: $f\ u \cdot \alpha \leq p\ f\ w \cdot \alpha \Longrightarrow u \leq p\ w$
⟨*proof*⟩

lemma *mismatch-pref0*: $[c_0] \le p\ f_m\ \mathfrak{a}$
⟨*proof*⟩

lemma *mismatch-pref1*: $[c_1] \le p\ f_m\ \mathfrak{b}$
⟨*proof*⟩

lemma *marked-version-len*: $|f_m\ w| = |f\ w|$
⟨*proof*⟩

lemma *bin-code-lcp*: $(f\ r \cdot \alpha) \wedge_p (f\ s \cdot \alpha) = f\ (r \wedge_p s) \cdot \alpha$
⟨*proof*⟩

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*⟩

lemma *bin-morph-pref-conv*: $f\ u \cdot \alpha \le p\ f\ v \cdot \alpha \longleftrightarrow u \le p\ v$
⟨*proof*⟩

lemma *bin-morph-compare-conv*: $f\ u \cdot \alpha \bowtie f\ v \cdot \alpha \longleftrightarrow u \bowtie v$
⟨*proof*⟩

lemma *code-lcp'*: $\neg\ r \bowtie s \Longrightarrow \alpha \le p\ f\ z \Longrightarrow \alpha \le p\ f\ z' \Longrightarrow f\ (r \cdot z) \wedge_p f\ (s \cdot z')$
$= f\ (r \wedge_p s) \cdot \alpha$
⟨*proof*⟩

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*⟩

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*: $\alpha' \le p\ \alpha$

begin

definition *shifted-f* **where** *shifted-f* = $(\lambda\ w.\ \alpha'^{<-1>}(f\ w \cdot \alpha'))$

lemma *shift-pref-all*: $\alpha' \le p\ f\ w \cdot \alpha'$
⟨*proof*⟩

sublocale *shifted*: *binary-code-morphism shifted-f*
⟨*proof*⟩

lemma *shifted-lcp*: $\alpha' \cdot$ *shifted.bin-code-lcp* $= \alpha$
⟨*proof*⟩

lemma $\alpha' = \alpha \Longrightarrow$ *shifted-f* $= f_m$
⟨*proof*⟩

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
⟨*proof*⟩

sublocale *hm*: *marked-binary-morphism* h_m
⟨*proof*⟩

sublocale *two-binary-morphisms g h*⟨*proof*⟩

sublocale *marked*: *two-marked-morphisms* g_m h_m⟨*proof*⟩

sublocale *code*: *two-code-morphisms g h*
⟨*proof*⟩

lemma *marked-two-binary-code-morphisms*: *two-binary-code-morphisms* g_m h_m
⟨*proof*⟩

lemma *revs-two-binary-code-morphisms*: *two-binary-code-morphisms* (*rev-map g*) (*rev-map h*)
⟨*proof*⟩

lemma *swap-two-binary-code-morphisms*: *two-binary-code-morphisms h g*
⟨*proof*⟩

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*⟩

lemma *two-equals*:
  assumes $g\ r = h\ r$ **and** $g\ s = h\ s$ **and** $\neg\ r \bowtie s$
  shows $g\ (r \wedge_p\ s) \cdot \alpha_g = h\ (r \wedge_p\ s) \cdot \alpha_h$
  ⟨*proof*⟩

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*⟩

lemma *alphas-pref*: **assumes** $|\alpha_h| \leq |\alpha_g|$ **and** $g\ r =_m\ h\ s$ **shows** $\alpha_h \leq p\ \alpha_g$
⟨*proof*⟩

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*: $\alpha_h \leq p\ \alpha_g$
  ⟨*proof*⟩

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*: $\alpha_h \cdot \alpha = \alpha_g$
  ⟨*proof*⟩

lemma *solution-marked-version-conv*: $g\ r = h\ s \longleftrightarrow \alpha \cdot\ \ g_m\ r = h_m\ s \cdot \alpha$
  ⟨*proof*⟩

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** $h\ v =_m\ g\ u$
  ⟨*proof*⟩

definition α' **where** $\alpha' = (if\ |\alpha_g| \leq |\alpha_h|\ then\ \alpha_g\ else\ \alpha_h)$
definition g' **where** $g' = (if\ |\alpha_g| \leq |\alpha_h|\ then\ (\lambda\ w.\ \alpha'^{-1>}(g\ w \cdot \alpha'))\ else\ (\lambda\ w.$

$\alpha'^{-1>}(h\ w \cdot \alpha')))$
definition h' **where** $h' = ($*if* $|\alpha_g| \leq |\alpha_h|$ *then* $(\lambda\ w.\ \alpha'^{-1>}(h\ w \cdot \alpha'))$ *else* $(\lambda\ w.\ \alpha'^{-1>}(g\ w \cdot \alpha')))$

lemma *shift-pref-fst*: $\alpha' \leq p\ \alpha_g$
⟨*proof*⟩

interpretation *gshift*: *binary-code-morphism-shift* $g\ \alpha'$
⟨*proof*⟩

interpretation *swap*: *two-binary-code-morphisms* $h\ g$
⟨*proof*⟩

lemma *shift-pref-snd*: $\alpha' \leq p\ \alpha_h$
⟨*proof*⟩

interpretation *hshift*: *binary-code-morphism-shift* $h\ \alpha'$
⟨*proof*⟩

lemma *shifted-eq-conv*:$g\ r = h\ s \longleftrightarrow g'\ r = h'\ s$
⟨*proof*⟩

lemma *shifted-eq-conv*:$g\ r = h\ r \longleftrightarrow g'\ r = h'\ r$
⟨*proof*⟩

lemma *shifted-eq-conv'*:$g = h \longleftrightarrow g' = h'$
⟨*proof*⟩

interpretation *shifted-g*: *binary-code-morphism* $(\lambda\ w.\ \alpha'^{-1>}(g\ w \cdot \alpha'))$
⟨*proof*⟩

interpretation *shifted-h*: *binary-code-morphism* $(\lambda\ w.\ \alpha'^{-1>}(h\ w \cdot \alpha'))$
⟨*proof*⟩

lemma *shifted-min-sol-conv*: $r \in g =_M h \longleftrightarrow r \in g' =_M h'$
⟨*proof*⟩

lemma *shifted-not-triv*: $g = h \longleftrightarrow g' = h'$
⟨*proof*⟩

sublocale *shifted*: *two-binary-code-morphisms* $g'\ h'$
⟨*proof*⟩

lemma *shifted-fst-lcp-emp*: *shifted.g.bin-code-lcp* $= \varepsilon$
⟨*proof*⟩

lemma *shifted-alphas*: **assumes** *le*: $|\alpha_g| \leq |\alpha_h|$
shows $\alpha' \cdot$ *shifted.g.bin-code-lcp* $= \alpha_g$ **and** $\alpha' \cdot$ *shifted.h.bin-code-lcp* $= \alpha_h$
⟨*proof*⟩

interpretation *swapped*: *binary-code-coincidence-sym h g*
⟨*proof*⟩

lemma *eq-len-eq-conv*: $\alpha_g = \alpha_h \longleftrightarrow |\alpha_g| = |\alpha_h|$
⟨*proof*⟩

lemma *shift-swapped*: *swapped*.$\alpha' = \alpha'$
⟨*proof*⟩

lemma *morphs-swapped*: **assumes** $|\alpha_g| \neq |\alpha_h|$ **shows** *swapped.g'* = *g'* *swapped.h'* = *h'*
⟨*proof*⟩

lemma *morphs-swapped'*: **assumes** $|\alpha_g| = |\alpha_h|$ **shows** *swapped.g'* = *h'* *swapped.h'* = *g'*
⟨*proof*⟩

lemma *shifted-lcp-len-eq*: |*shifted.g.bin-code-lcp*| = |*shifted.h.bin-code-lcp*| $\longleftrightarrow |\alpha_g| = |\alpha_h|$ **and**
shifted-lcp-len-le: |*shifted.g.bin-code-lcp*| ≤ |*shifted.h.bin-code-lcp*|
⟨*proof*⟩

end

locale *two-marked-binary-morphisms* = *two-marked-morphisms g h*
for *g h* :: *binA list* ⇒ *'a list*
begin

sublocale *two-binary-code-morphisms g h* ⟨*proof*⟩

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*⟩

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*]) · *h* ([*hd s*]) ≠ *h* ([*hd s*]) · *g* ([*hd s*])
⟨*proof*⟩

sublocale *g*: *marked-binary-morphism g*
⟨*proof*⟩

sublocale *h*: *marked-binary-morphism h*
⟨*proof*⟩

sublocale *revs*: *two-binary-code-morphisms rev-map g rev-map h*
⟨*proof*⟩

end

7.9 Two marked binary morphisms with blocks

locale *two-binary-marked-blocks* = *two-marked-binary-morphisms* +
assumes *both-blocks*: ⋀ *a*. *blockP a*

begin

sublocale *sucs*: *two-marked-binary-morphisms suc-fst suc-snd*
⟨*proof*⟩

sublocale *sucs-enc*: *two-marked-binary-morphisms suc-fst′ suc-snd′*
⟨*proof*⟩

lemma *bin-blocks-swap*: *two-binary-marked-blocks h g*
⟨*proof*⟩

lemma *blocks-all-letters-fst*: [*b*] ≤*f suc-fst* ([*a*] · [*1*−*a*])
⟨*proof*⟩

lemma *blocks-all-letters-snd*: [*b*] ≤*f suc-snd* ([*a*] · [*1*−*a*])
⟨*proof*⟩

lemma *lcs-suf-blocks-fst*: *g.bin-code-lcs* ≤*s g* (*suc-fst* ([*a*] · [*1*−*a*]))
⟨*proof*⟩

lemma *lcs-suf-blocks-snd*: *h.bin-code-lcs* ≤*s h* (*suc-snd* ([*a*] · [*1*−*a*]))
⟨*proof*⟩

lemma *lcs-fst-suf-snd*: *g.bin-code-lcs* ≤*s h.bin-code-lcs* · *h sucs.h.bin-code-lcs*
⟨*proof*⟩

lemma *suf-comp-lcs*: *g.bin-code-lcs* $\bowtie_s$ *h.bin-code-lcs*
⟨*proof*⟩

end

7.10 Binary primitivity preserving morphism given by a pair of words

definition *bin-prim* :: *'a list* ⇒ *'a list* ⇒ *bool*
where *bin-prim x y* ⟷ *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*⟩

7.10.1 Translating to to list concatenation

lemma *bin-concat-prim-pres-noner1*:
assumes $x \neq y$
and *prim-pres*: ⋀ *ws. ws* ∈ *lists* {*x*,*y*} ⟹ *2* ≤ |*ws*| ⟹ *primitive ws* ⟹ *primitive* (*concat ws*)
shows $x \neq \varepsilon$
⟨*proof*⟩

lemma *bin-concat-prim-pres-noner*:
assumes $x \neq y$
and *prim-pres*: ⋀ *ws. ws* ∈ *lists* {*x*,*y*} ⟹ *2* ≤ |*ws*| ⟹ *primitive ws* ⟹ *primitive* (*concat ws*)
shows *nonerasing-morphism* (*bin-morph-of x y*)
⟨*proof*⟩

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*⟩

lemma *bin-prim-concat-prim-pres*:
assumes *bin-prim x y*
shows *ws* ∈ *lists* {*x*, *y*} ⟹ *2* ≤ |*ws*| ⟹ *primitive ws* ⟹ *primitive* (*concat ws*)
⟨*proof*⟩

lemma *bin-prim-altdef1*:
bin-prim x y ⟷
($x \neq y$) ∧ (∀ *ws* ∈ *lists* {*x*,*y*}. *2* ≤ |*ws*| ⟶ *primitive ws* ⟶ *primitive* (*concat ws*))
⟨*proof*⟩

lemma *bin-prim-altdef2*:
bin-prim x y ⟷
($x \cdot y \neq y \cdot x$) ∧ (∀ *ws* ∈ *lists* {*x*,*y*}. *2* ≤ |*ws*| ⟶ *primitive ws* ⟶ *primitive* (*concat ws*))

⟨*proof*⟩

7.10.2 Basic properties of *bin-prim*

lemma *bin-prim-irrefl*: ¬ *bin-prim x x*
⟨*proof*⟩

lemma *bin-prim-symm* [*sym*]: *bin-prim x y* ⟹ *bin-prim y x*
⟨*proof*⟩

lemma *bin-prim-commutes*: *bin-prim x y* ⟷ *bin-prim y x*
⟨*proof*⟩

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 Factor interpretation

definition *factor-interpretation* :: $'a\ list \Rightarrow 'a\ list \Rightarrow 'a\ list \Rightarrow 'a\ list\ list \Rightarrow bool$ (‹- - - $\sim_{\mathcal{I}}$ -› [51,51,51,51] 60)
  where *factor-interpretation* $p\ u\ s\ ws = (p <p\ hd\ ws \wedge s <s\ last\ ws \wedge p \cdot u \cdot s = concat\ ws)$

lemma *fac-interp-nemp*: $u \neq \varepsilon \Longrightarrow p\ u\ s \sim_{\mathcal{I}} ws \Longrightarrow ws \neq \varepsilon$
  ⟨*proof*⟩

lemma *fac-interpD*: **assumes** $p\ u\ s \sim_{\mathcal{I}} ws$
  shows $p <p\ hd\ ws$ **and** $s <s\ last\ ws$ **and** $p \cdot u \cdot s = concat\ ws$
  ⟨*proof*⟩

lemma *fac-interpI*:
  $p <p\ hd\ ws \Longrightarrow s <s\ last\ ws \Longrightarrow p \cdot u \cdot s = concat\ ws \Longrightarrow p\ u\ s \sim_{\mathcal{I}} ws$
  ⟨*proof*⟩

lemma *obtain-fac-interp*: **assumes** $pu \cdot u \cdot su = concat\ ws$ **and** $u \neq \varepsilon$
  obtains *ps ss p s vs* **where** $p\ u\ s \sim_{\mathcal{I}} vs$ **and** $ps \cdot vs \cdot ss = ws$ **and** $concat\ ps \cdot p = pu$ **and**
    $s \cdot concat\ ss = su$
  ⟨*proof*⟩

lemma *obtain-fac-interp'*: **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*⟩

lemma *fac-pow-longE*: **assumes** $w \le f\ v^{@}k$ **and** $|v| \le |w|$
obtains *m v1 v2* **where** $v1 \le s\ v\ v2 \le p\ v\ w = v1 \cdot v^{@}m \cdot v2$
⟨*proof*⟩

lemma *obtain-fac-interp-dec*: **assumes** $w \in \langle G\rangle\ \ u \le f\ w\ u \neq \varepsilon$
obtains *p s ws* **where** $ws \in lists\ (G - \{\varepsilon\})\ p\ u\ s \sim_{\mathcal{I}} ws\ ws \le f\ Dec\ G\ w$
⟨*proof*⟩

lemma *fac-interp-inner*: **assumes** $u \neq \varepsilon$ **and** $p\ u\ s \sim_{\mathcal{I}} ws$ **and** $1 < |ws|$
shows $p^{-1>}(hd\ ws) \cdot concat(butlast\ (tl\ ws)) \cdot (last\ ws)^{<-1}s = u$
⟨*proof*⟩

lemma *fac-interp-inner-len*: **assumes** $u \neq \varepsilon$ **and** $p\ u\ s \sim_{\mathcal{I}} ws$
shows $|concat(butlast\ (tl\ ws))| < |u|$
⟨*proof*⟩

lemma *rev-in-set-map-rev-conv*: $rev\ u \in set\ (map\ rev\ ws) \longleftrightarrow u \in set\ ws$
⟨*proof*⟩

lemma *rev-fac-interp*: **assumes** $p\ u\ s \sim_{\mathcal{I}} ws$ **shows** $(rev\ s)\ (rev\ u)\ (rev\ p) \sim_{\mathcal{I}} rev\ (map\ rev\ ws)$
⟨*proof*⟩

lemma *rev-fac-interp-iff* [*reversal-rule*]: $(rev\ s)\ (rev\ u)\ (rev\ p) \sim_{\mathcal{I}} rev\ (map\ rev\ ws) \longleftrightarrow p\ u\ s \sim_{\mathcal{I}} ws$
⟨*proof*⟩

lemma *fac-interp-mid-fac*: **assumes** $p\ u\ s \sim_{\mathcal{I}} ws$
shows $concat\ (butlast\ (tl\ ws)) \le f\ u$
⟨*proof*⟩

definition *disjoint-interpretation* :: *'a list* ⇒ *'a list list* ⇒ *'a list* ⇒ *'a list list* ⇒ *bool* (‹- - - $\sim_{\mathcal{D}}$ -› [*51,51,51,51*] *60*)
where $p\ us\ s \sim_{\mathcal{D}} ws \equiv p\ (concat\ us)\ s \sim_{\mathcal{I}} ws\ \wedge$
$(\forall\ u\ v.\ u \le p\ us \wedge v \le p\ ws \longrightarrow p \cdot concat\ u \neq concat\ v)$

lemma *disjoint-interpI*: $p\ (concat\ us)\ s \sim_{\mathcal{I}} ws \Longrightarrow$
$(\forall\ u\ v.\ u \le p\ us \wedge v \le p\ ws \longrightarrow p \cdot concat\ u \neq concat\ v) \Longrightarrow p\ us\ s \sim_{\mathcal{D}} ws$
⟨*proof*⟩

lemma *disjoint-interpI'*[*intro*]: $p\ (concat\ us)\ s \sim_{\mathcal{I}} ws \Longrightarrow$
$(\bigwedge u\ v.\ u \le p\ us \Longrightarrow v \le p\ ws \Longrightarrow p \cdot concat\ u \neq concat\ v) \Longrightarrow p\ us\ s \sim_{\mathcal{D}} ws$
⟨*proof*⟩

lemma *disj-interpD*: $p\ us\ s \sim_{\mathcal{D}} ws \Longrightarrow p\ (concat\ us)\ s \sim_{\mathcal{I}} ws$
⟨*proof*⟩

lemma *disj-interpD1*: **assumes** $p\ us\ s \sim_{\mathcal{D}} ws$ **and** $us' \le p\ us$ **and** $ws' \le p\ ws$
shows $p \cdot concat\ us' \neq concat\ ws'$
⟨*proof*⟩

lemma *disj-interp-nemp*: **assumes** $p\ us\ s \sim_{\mathcal{D}} ws$
shows $p \neq \varepsilon$ **and** $s \neq \varepsilon$
⟨*proof*⟩

8.1.1 Factor intepretation of morphic images

context *morphism*
begin

lemma *image-fac-interp'*: **assumes** $w \le f\ f\ z\ w \neq \varepsilon$
obtains p *w-pred* s **where** *w-pred* $\le f\ z\ p\ w\ s \sim_{\mathcal{I}} (map\ f^{\mathcal{C}}$ *w-pred*$)$
⟨*proof*⟩

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-pred·*w-pred*·*v-pred* $= z\ p\ w\ s \sim_{\mathcal{I}} (map\ f^{\mathcal{C}}$ *w-pred*$)$
$u = (f$ *u-pred*$)\cdot p\ v = s\cdot(f$ *v-pred*$)$
⟨*proof*⟩

lemma *image-fac-interp-mid*: **assumes** $p\ w\ s \sim_{\mathcal{I}} map\ f^{\mathcal{C}}$ *w-pred* $2 \le |$*w-pred*$|$
obtains *pw sw* **where**
$w = pw \cdot (f\ (butlast\ (tl$ *w-pred*$))) \cdot sw\ p\cdot pw = f\ [hd$ *w-pred*$]\ sw\cdot s = f\ [last$ *w-pred*$]$
⟨*proof*⟩

end

8.2 Miscellanea

8.2.1 Mismatch additions

lemma *mismatch-pref-comm-len*: **assumes** $w1 \in \langle\{u,v\}\rangle$ **and** $w2 \in \langle\{u,v\}\rangle$ **and** $p \le p\ w1$
$u \cdot p \le p\ v \cdot w2$ **and** $|v| \le |p|$
shows $u \cdot v = v \cdot u$
⟨*proof*⟩

lemma *mismatch-pref-comm*: **assumes** $w1 \in \langle\{u,v\}\rangle$ **and** $w2 \in \langle\{u,v\}\rangle$ **and**
$u \cdot w1 \cdot v \le p\ v \cdot w2 \cdot u$
shows $u \cdot v = v \cdot u$
⟨*proof*⟩

lemma *mismatch-eq-comm*: **assumes** $w1 \in \langle\{u,v\}\rangle$ **and** $w2 \in \langle\{u,v\}\rangle$ **and**
$u \cdot w1 = v \cdot w2$

shows $u \cdot v = v \cdot u$
⟨*proof*⟩

lemmas *mismatch-suf-comm* = *mismatch-pref-comm*[*reversed*] **and**
mismatch-suf-comm-len = *mismatch-pref-comm-len*[*reversed*, *unfolded rassoc*]

8.2.2 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*⟩

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*⟩

hide-fact *conj-pers-conj-comm-aux*

8.2.3 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*⟩

lemmas *uv-fac-uvv-suf* = *uv-fac-uvv*[*reversed*, *unfolded rassoc*]

lemma $u \leq p\ v \Longrightarrow u' \leq p\ v' \Longrightarrow u \wedge_p u' \neq u \Longrightarrow u \wedge_p u' \neq u' \Longrightarrow u \wedge_p u' = v \wedge_p v'$
⟨*proof*⟩

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*⟩

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 $u \cdot v = v \cdot u$
⟨*proof*⟩

lemma *uvu-pref-uvv*: **assumes** $p \cdot u \cdot v \cdot v \cdot s = u \cdot v \cdot u \cdot q$ **and**

$p \le p\ u$ **and** $q \le p\ w$ **and** $s \le p\ w'$ **and**
$w \in \langle\{u,v\}\rangle$ **and** $w' \in \langle\{u,v\}\rangle$ **and** $|u| \le |s|$
shows $u \cdot v = v \cdot u$
⟨*proof*⟩

lemma *uvu-pref-uvvu*: **assumes** $p \cdot u \cdot v \cdot v \cdot u = u \cdot v \cdot u \cdot q$ **and**
$p \le p\ u$ **and** $q \le p\ w$ **and** $w \in \langle\{u,v\}\rangle$
shows $u \cdot v = v \cdot u$
⟨*proof*⟩

lemma *uvu-pref-uvvu-interp*: **assumes** *interp*: $p\ u \cdot v \cdot v \cdot u\ s \sim_{\mathcal{I}} ws$ **and**
$[u, v, u] \le p\ \ ws$ **and** $ws \in lists\ \{u,v\}$
shows $u \cdot v = v \cdot u$
⟨*proof*⟩

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*: $p\ u \cdot v \cdot v \cdot u\ s \sim_{\mathcal{I}} ws \Longrightarrow [u, v, u] \le s\ \ ws$
$\Longrightarrow ws \in lists\ \{u,v\} \Longrightarrow u \cdot v = v \cdot u$
⟨*proof*⟩

8.2.4 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 $r = s \lor r = \varepsilon \lor s = \varepsilon$
⟨*proof*⟩

lemma *conjug-conjug-primroots*: **assumes** $u \ne \varepsilon$ **and** $r \ne \varepsilon$ **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*⟩

8.2.5 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*: $commutes\ A \Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow x{\cdot}y = y{\cdot}x$
⟨*proof*⟩

lemma *commutes-root*: **assumes** *commutes A*
obtains r **where** $\bigwedge x.\ x \in A \Longrightarrow x \in r*$
⟨*proof*⟩

lemma *commutes-primroot*: **assumes** *commutes A*
obtains r **where** $\bigwedge x.\ x \in A \Longrightarrow x \in r*$ **and** *primitive r*

⟨*proof*⟩

lemma *commutesI* [*intro*]: $(\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow x{\cdot}y = y{\cdot}x) \Longrightarrow$ *commutes* A
⟨*proof*⟩

lemma *commutesI'*: **assumes** $x \neq \varepsilon$ **and** $\bigwedge y.\ y \in A \Longrightarrow x{\cdot}y = y{\cdot}x$
shows *commutes* A
⟨*proof*⟩

lemma *commutesI-root*[*intro*]: $\forall x \in A.\ x \in t*\Longrightarrow$ *commutes* A
⟨*proof*⟩

lemma *commutes-sub*: *commutes* $A \Longrightarrow B \subseteq A \Longrightarrow$ *commutes* B
⟨*proof*⟩

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)
⟨*proof*⟩

lemma *commutes-emp* [*simp*]: *commutes* $\{\varepsilon,\ w\}$
⟨*proof*⟩

lemma *commutes-emp'*[*simp*]: *commutes* $\{w,\ \varepsilon\}$
⟨*proof*⟩

lemma *commutes-cancel*: **assumes** $y \in A$ **and** $x \cdot y \in A$ **and** *commutes* A
shows *commutes* (*insert* x A)
⟨*proof*⟩

lemma *commutes-cancel'*: **assumes** $x \in A$ **and** $x \cdot y \in A$ **and** *commutes* A
shows *commutes* (*insert* y A)
⟨*proof*⟩

8.2.6 Strong elementary lemmas

Discovered by smt

lemma *xyx-per-comm*: **assumes** $x{\cdot}y{\cdot}x \le p\ q{\cdot}x{\cdot}y{\cdot}x$
and $q \neq \varepsilon$ **and** $q \le p\ y \cdot q$
shows $x{\cdot}y = y{\cdot}x$
⟨*proof*⟩

lemma *two-elem-root-suf-comm*: **assumes** $u \le p\ v \cdot u$ **and** $v \le s\ p \cdot u$ **and** $p \in \langle\{u,v\}\rangle$
shows $u \cdot v = v \cdot u$
⟨*proof*⟩

8.2.7 Binary words without a letter square

lemma *no-repetition-list*:
  assumes *set ws* ⊆ {*a*,*b*}
    and *not-per*: ¬ *ws* ≤*p* [*a*,*b*] · *ws* ¬ *ws* ≤*p* [*b*,*a*] · *ws*
    and *not-square*: ¬ [*a*,*a*] ≤*f ws* **and** ¬ [*b*,*b*] ≤*f ws*
  shows *False*
  ⟨*proof*⟩

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

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*⟩

lemma *per-root-hd-last-root*: **assumes** *ws* ≤*p* [*a*,*b*] · *ws* **and** *hd ws* ≠ *last ws*
  shows *ws* ∈ [*a*,*b*]∗
  ⟨*proof*⟩

lemma *no-cyclic-repetition-list*:
  assumes *set ws* ⊆ {*a*,*b*} *ws* ∉ [*a*,*b*]∗ *ws* ∉ [*b*,*a*]∗ *hd ws* ≠ *last ws*
    ¬ [*a*,*a*] ≤*f ws* ¬ [*b*,*b*] ≤*f ws*
  shows *False*
  ⟨*proof*⟩

8.2.8 Three covers

lemma *three-covers-example*:
  assumes
    v: *v* = 𝔞 **and**
    t: *t* = (𝔟 · 𝔞$^@$(*j*+*1*))$^@$(*m* + *l* + *1*) · 𝔟 · 𝔞 **and**
    r: *r* = 𝔞 · 𝔟 · (𝔞$^@$(*j*+*1*) · 𝔟)$^@$(*m* + *l* + *1*) **and**
    t′: *t′* = (𝔟 · 𝔞$^@$(*j* + *1*))$^@$*m* · 𝔟 · 𝔞 **and**
    r′: *r′* = 𝔞 · 𝔟 · (𝔞$^@$(*j* + *1*) · 𝔟)$^@$*l* **and**
    w: *w* = 𝔞 · (𝔟 · 𝔞$^@$(*j* + *1*))$^@$(*m* + *l* + *1*) · 𝔟 · 𝔞
  shows *w* = *v* · *t* **and** *w* = *r* · *v* **and** *w* = *r′* · *v*$^@$(*j* + *1*) · *t′* **and** *t′* <*p t* **and** *r′* <*s r*
⟨*proof*⟩

lemma *three-covers-pers*: — alias Old Good Lemma
  assumes *w* = *v* · *t* **and** *w* = *r′* · *v*$^@$*j* · *t′* **and** *w* = *r* · *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*⟩

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** $|t'| \leq |r'|$
and *primitive* v
shows *period* w (*gcd* ($|t| - |t'|$) ($|r| - |r'|$))
⟨*proof*⟩

lemma *three-covers-per*: **assumes** $w = v \cdot t$ **and** $w = r' \cdot v^{@}j \cdot t'$ **and** $w = r \cdot v$
$r' <s\ r$ **and** $t' <p\ t$ **and** $0 < j$
shows *period* w (*gcd* ($|t| - |t'|$) ($|r| - |r'|$))
⟨*proof*⟩

thm *per-root-modE'*

lemma assumes $w <p\ r \cdot w$
obtains $p\ q\ i$ **where** $w = (p \cdot q)^{@}i \cdot p\ p \cdot q = r$
⟨*proof*⟩

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*⟩

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 $x \cdot y = y \cdot x$
⟨*proof*⟩

8.2.9 Binary Equality Words

definition *binary-equality-word* :: *binA list* ⇒ *bool* **where**
binary-equality-word w = (∃ (g :: *binA list* ⇒ *nat list*) h. *binary-code-morphism* g ∧ *binary-code-morphism* h ∧ $g \neq h$ ∧ $w \in g =_M h$)

lemma *not-bew-baiba*: **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*⟩

lemma *not-bew-baibaib*: **assumes** $|x| < |u|$ **and** $1 < i$ **and**
$x \cdot y^{@}i\cdot\ x \cdot y^{@}i \cdot x = u \cdot v^{@}i\cdot\ u \cdot v^{@}i \cdot u$
shows *commutes* $\{x,y,u,v\}$
⟨*proof*⟩

theorem ¬ *binary-equality-word* $(\mathfrak{a} \cdot \mathfrak{b}^{@}Suc\ k \cdot \mathfrak{a} \cdot \mathfrak{b})$
⟨*proof*⟩

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.