

Combinatorics on Words formalized Basics

Štěpán Holub
Martin Raška
Štěpán Starosta

May 20, 2026

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

Contents

0.1	Arithmetical hints	5
1	Reverse symmetry	8
1.1	Quantifications and maps	8
1.1.1	Quantifications and reverse	9
1.2	Attributes	9
1.2.1	Cons reversion	9
1.2.2	Final corrections	9
1.2.3	Declaration attribute <i>reversal-rule</i>	10
1.2.4	Tracing attribute	10
1.2.5	Rule attribute <i>reversed</i>	10
1.3	Declaration of basic reversal rules	11
1.3.1	Pure	11
1.3.2	<i>HOL.HOL</i>	11
1.3.3	<i>HOL.Set</i>	11
1.3.4	<i>HOL.List</i>	11
1.3.5	Reverse Symmetry	13
1.4	13
1.5	Examples	14
1.5.1	Cons and append	14
1.5.2	Induction rules	15
1.5.3	hd, tl, last, butlast	15
1.5.4	set	15
1.5.5	rotate	16
2	Basics of Combinatorics on Words	17
2.1	Definitions and notations	17
2.1.1	Empty and nonempty word	18
2.1.2	Prefix	18
2.1.3	Suffix	19
2.1.4	Factor	20
2.2	Various elementary lemmas	21
2.2.1	General facts	25
2.2.2	Map injective function	26

2.2.3	Orderings on lists: prefix, suffix, factor	26
2.2.4	On the empty word	26
2.2.5	Counting letters	27
2.2.6	Set inspection method	27
2.3	Length and its properties	28
2.4	List inspection method	31
2.5	Prefix and prefix comparability properties	32
2.5.1	Prefix comparability	34
2.5.2	Minimal and maximal prefix with a given property	39
2.6	Suffix and suffix comparability properties	39
2.6.1	Suffix comparability	41
2.7	Left and Right Quotient	42
2.7.1	Left Quotient	43
2.7.2	Right quotient	45
2.7.3	Left and right quotients combined	46
2.8	Equidivisibility	47
2.9	Longest common prefix	48
2.9.1	Longest common prefix and prefix comparability	51
2.9.2	Longest common suffix	52
2.10	Mismatch	53
2.11	Factor properties	54
2.12	Power and its properties	56
2.12.1	Comparison	62
2.13	Rotation	63
2.14	Lists of words and their concatenation	64
2.15	Commutation	66
2.16	Periods	67
2.16.1	periodic root	67
2.16.2	Maximal root-prefix	70
2.16.3	Period - numeric	70
2.16.4	Period: overview	73
2.16.5	Singleton and its power	73
2.17	Border	76
2.17.1	The shortest border	77
2.17.2	Relation to period and conjugation	78
2.18	The longest border and the shortest period	79
2.18.1	The longest border	79
2.18.2	The shortest period	81
2.19	Primitive words	82
2.20	Primitive root	84
2.20.1	Primitivity and the shortest period	89
2.21	Conjugation	89
2.21.1	Enumerating conjugates	96
2.21.2	General lemmas using conjugation	97

2.22	Element of lists: a method for testing if a word is in lists A . . .	97
2.23	Reversed mappings	99
2.24	Overlapping powers, periods, prefixes and suffixes	100
2.25	Testing primitivity	105
2.26	Factor interpretation	105
2.26.1	Auxiliary lemmas on suffix and border extension . . .	108
2.27	Computing the Border Array	109
2.27.1	Verification of the computation	111
2.28	Border array	111
3	Submonoids of a free monoid	114
3.1	Generated monoid (also called hull)	114
3.2	Factorization into generators	118
3.2.1	Refinement into a specific decomposition	119
3.3	Basis	120
3.4	Code	122
3.5	Prefix code	125
3.5.1	Suffix code	126
3.5.2	Bifix code	127
3.6	Marked code	127
3.7	Non-overlapping code	128
3.8	Binary code	129
3.8.1	Binary code locale	130
3.9	Two words hull (not necessarily a code)	137
3.9.1	Binary Mismatch tools	139
3.9.2	Applied mismatch	144
3.10	Free hull	144
3.11	Reversing hulls and decompositions	146
3.12	Lists as the free hull of singletons	147
3.13	Various additional lemmas	148
3.13.1	Roots of binary set	148
3.13.2	Other	149
4	Morphisms	150
4.1	One morphism	150
4.1.1	Morphism, core map and extension	150
4.1.2	periodic morphism	153
4.1.3	Non-erasing morphism	154
4.1.4	Code morphism	155
4.1.5	Prefix code morphism	156
4.1.6	Marked morphism	157
4.1.7	Image length	157
4.1.8	Endomorphism	158
4.1.9	Decomposition into primitive roots	160

4.2	Primitivity preserving morphisms	161
4.3	Two morphisms	161
4.3.1	Solutions	161
4.3.2	Coincidence pairs	162
4.3.3	Basics	163
4.3.4	Factor intepretation of morphic images	165
4.3.5	Two nonerasing morphisms	165
4.3.6	Two marked morphisms	168
5	The periodicity Lemma	172
5.1	Main claim	172
5.2	Optimality of the bound by construction of the Fine and Wilf word.	173
5.3	Optimality of the Fine and Wilf word.	175
5.4	Other variants of the periodicity lemma	175
6	Lyndon-Schützenberger Equation	177
6.1	The original result	177
6.2	The original result	177
6.2.1	Some alternative formulations.	178
6.3	Parametric solution of the equation $x^{\textcircled{a}} j \cdot y^{\textcircled{a}} k = z^{\textcircled{a}} l$.	179
6.3.1	Auxiliary lemmas	179
6.3.2	x is longer	179
6.3.3	Putting things together	180
6.3.4	Uniqueness of the imprimitivity witness	182
6.4	The bound on the exponent in Lyndon-Schützenberger equation	182
7	Binary alphabet and binary morphisms	184
7.1	Datatype of a binary alphabet	184
7.2	Binary morphisms	188
7.2.1	Binary periodic morphisms	189
7.3	From two words to a binary morphism	190
7.4	Two binary morphism	191
7.5	Binary code morphism	192
7.5.1	Locale - binary code morphism	192
7.5.2	More translations	196
7.6	Marked binary morphism	197
7.7	Marked version	198
7.8	Two binary code morphisms	200
7.9	Two marked binary morphisms with blocks	204
7.10	Binary primitivity preserving morphism given by a pair of words	205
7.10.1	Translating to list concatenation	205
7.10.2	Basic properties of <i>bin-prim</i>	206

8	Equations on words - basics	207
8.1	Miscellanea	207
8.1.1	Mismatch additions	207
8.1.2	Conjugate words with conjugate periods	207
8.1.3	Covering uvvu	208
8.1.4	Conjugate words	209
8.1.5	Predicate “commutes”	209
8.1.6	Strong elementary lemmas	210
8.1.7	Binary words without a letter square	210
8.1.8	Three covers	211
8.1.9	Binary Equality Words	212
	References	214

```

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:  $i \leq j \implies (0 :: nat) = j - i \implies j = i$ 
  <proof>

```

```

lemma zero-less-diff':  $i < j \implies j - i \neq (0 :: nat)$ 
  <proof>

```

```

lemma nat-prod-le:  $m \neq (0 :: nat) \implies m * n \leq k \implies n \leq k$ 
  <proof>

```

```

lemma get-div:  $(p :: nat) < a \implies m = (m * a + p) \text{ div } a$ 
  <proof>

```

```

lemma get-mod:  $(p :: nat) < a \implies p = (m * a + p) \text{ mod } a$ 
  <proof>

```

```

lemma plus-one-between:  $(a :: nat) < b \implies \neg b < a + 1$ 
  <proof>

```

```

lemma quotient-smaller:  $k \neq (0 :: nat) \implies b \leq k * b$ 
  <proof>

```

```

thm mult-right-le-imp-le

```

lemma *add-lessD2*: $k + m < (n::nat) \implies m < n$
<proof>

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

lemma **assumes** $q \neq (0::nat)$ **shows** $p \leq p + q - \text{gcd } p \ q$
<proof>

lemma *pos-div-gcd-pos* [*intro*]: **assumes** $(0 :: nat) < p$ **shows** $0 < p \text{ div } \text{gcd } p \ q$
<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*: $1 \leq k \iff 0 < (k::nat)$
<proof>

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

lemma *Suc-less-iff-Suc-le*: $\text{Suc } n < k \iff \text{Suc } n \leq k - 1$
<proof>

lemma *nat-induct-pair*: $P \ 0 \ 0 \implies (\bigwedge m \ n. P \ m \ n \implies P \ m \ (\text{Suc } n)) \implies (\bigwedge m \ n. P \ m \ n \implies P \ (\text{Suc } m) \ n) \implies P \ m \ n$
<proof>

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

lemma *at-least2-Suc*: **assumes** $2 \leq k$
obtains k' **where** $k = \text{Suc}(\text{Suc } k')$
<proof>

lemma *at-least3-Suc*: **assumes** $3 \leq k$
obtains k' **where** $k = \text{Suc}(\text{Suc}(\text{Suc } k'))$
<proof>

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

lemma *le1-SucE*: **assumes** $1 \leq n$
obtains k **where** $n = \text{Suc } k$ *<proof>*

lemma *Suc-minus*: $k \neq 0 \implies \text{Suc}(k - 1) = k$
<proof>

lemma *Suc-minus'*: $1 \leq k \implies \text{Suc}(k - 1) = k$
<proof>

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

lemma *Suc-minus2*: $2 \leq k \implies \text{Suc}(\text{Suc}(k - 2)) = k$
<proof>

lemma *Suc-leE*: **assumes** $\text{Suc } k \leq n$ **obtains** m **where** $n = \text{Suc } m$ **and** $k \leq m$
<proof>

lemma *two-three-add-le-mult*: $2 \leq (l::\text{nat}) \implies 3 \leq k \implies k + l + 1 \leq k * l$
<proof>

lemma *almost-equal-equal*: **assumes** $(a::\text{nat}) \neq 0$ **and** $b \neq 0$ **and** *eq*: $k*(a+b) + a = m*(a+b) + b$
shows $k = m$ **and** $a = b$
<proof>

lemma *add-le-cancel-ge-right*: $a + b \leq c + d \implies d \leq b \implies a \leq (c::\text{nat})$ **for** a
 b c d
<proof>

lemma *add-less-cancel-ge-right*: $a + b \leq c + d \implies d < b \implies a < (c::\text{nat})$ **for** a
 b c d
<proof>

lemma *add-le-cancel-ge-left*: $a + b \leq c + d \implies c \leq a \implies b \leq (d::\text{nat})$ **for** a b
 c d
<proof>

lemma *add-less-cancel-ge-left*: $a + b \leq c + d \implies c < a \implies b < (d::\text{nat})$ **for** a
 b c d
<proof>

lemma *crossproduct-le*: **assumes** $(a::\text{nat}) \leq b$ **and** $c \leq d$
shows $a*d + b*c \leq a*c + b*d$
<proof>

lemma (**in** *linorder*) *le-less-cases*: $(a \leq b \implies P) \implies (b < a \implies P) \implies P$
<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 \forall, \exists and $\exists!$ quantifiers; each bounded quantification of 'a list' type variable $\forall xs \in A. P\ xs$ is substituted by (logically equivalent) quantification $\forall xs \in rev\ A. P\ (rev\ xs)$, similarly for bounded \exists quantifier; 4. simultaneous rewrites according to 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** $(\forall y \in f^{-1} A. P (inv f y)) \longleftrightarrow (\forall x \in A. P x)$
<proof>

lemma *Bex-inj-conv*: **assumes** *inj f* **shows** $(\exists y \in f^{-1} A. P (inv f y)) \longleftrightarrow (\exists x \in A. P x)$
<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* \Rightarrow *'a list* \Rightarrow *'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*

$\langle ML \rangle$

1.2.4 Tracing attribute

$\langle ML \rangle$

1.2.5 Rule attribute *reversed*

private lemma *rev-Nil*: $rev \ [] \equiv []$

$\langle proof \rangle$ **lemma** *map-Nil*: $map \ f \ [] \equiv []$

$\langle proof \rangle$ **lemma** *image-empty*: $f \ ' \ Set.empty \equiv Set.empty$

$\langle proof \rangle$

definition *COMP* :: $('b \Rightarrow prop) \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow prop$ (**infixl** *oo* 55)

where $F \ oo \ g \equiv (\lambda x. F \ (g \ x))$

lemma *COMP-assoc*: $F \ oo \ (f \ o \ g) \equiv (F \ oo \ f) \ oo \ g$

$\langle proof \rangle$ **lemma** *image-comp-image*: $(\cdot) \ f \ o \ (\cdot) \ g \equiv (\cdot) \ (f \ o \ g)$

$\langle proof \rangle$ **lemma** *rev-involution*: $rev \ o \ rev \equiv id$

$\langle proof \rangle$ **lemma** *map-involution*: **assumes** $f \ o \ f \equiv id$ **shows** $(map \ f) \ o \ (map \ f) \equiv id$

$\langle proof \rangle$ **lemma** *image-involution*: **assumes** $f \ o \ f \equiv id$ **shows** $(image \ f) \ o \ (image \ f) \equiv id$

$\langle proof \rangle$ **lemma** *rev-map-comm*: $rev \ o \ map \ f \equiv map \ f \ o \ rev$

$\langle proof \rangle$ **lemma** *involut-comm-comp*: **assumes** $f \ o \ f \equiv id$ **and** $g \ o \ g \equiv id$ **and** $f \ o \ g \equiv g \ o \ f$

shows $(f \ o \ g) \ o \ (f \ o \ g) \equiv id$

$\langle proof \rangle$ **lemma** *rev-map-involution*: **assumes** $g \ o \ g \equiv id$

shows $(rev \ o \ map \ g) \ o \ (rev \ o \ map \ g) \equiv id$

$\langle proof \rangle$ **lemma** *prop-abs-subst*: **assumes** $f \ o \ f \equiv id$ **shows** $(\lambda x. F \ (f \ x)) \ oo \ f \equiv (\lambda x. F \ x)$

$\langle proof \rangle$ **lemma** *prop-abs-subst-comm*: **assumes** $f \ o \ f \equiv id$ **and** $g \ o \ g \equiv id$ **and** $f \ o \ g \equiv g \ o \ f$

shows $(\lambda x. F \ (f \ (g \ x))) \ oo \ (f \ o \ g) \equiv (\lambda x. F \ x)$

$\langle proof \rangle$ **lemma** *prop-abs-subst-rev-map*: **assumes** $g \ o \ g \equiv id$

shows $(\lambda x. F \ (rev \ (map \ g \ x))) \ oo \ (rev \ o \ map \ g) \equiv (\lambda x. F \ x)$

$\langle proof \rangle$ **lemma** *obj-abs-subst*: **assumes** $f \ o \ f \equiv id$ **shows** $(\lambda x. F \ (f \ x)) \ o \ f \equiv (\lambda x. F \ x)$

$\langle proof \rangle$ **lemma** *obj-abs-subst-comm*: **assumes** $f \ o \ f \equiv id$ **and** $g \ o \ g \equiv id$ **and** $f \ o \ g \equiv g \ o \ f$

shows $(\lambda x. F \ (f \ (g \ x))) \ o \ (f \ o \ g) \equiv (\lambda x. F \ x)$

$\langle proof \rangle$ **lemma** *obj-abs-subst-rev-map*: **assumes** $g \ o \ g \equiv id$

shows $(\lambda x. F \ (rev \ (map \ g \ x))) \ o \ (rev \ o \ map \ g) \equiv (\lambda x. F \ x)$

$\langle proof \rangle$

$\langle ML \rangle$

end

1.3 Declaration of basic reversal rules

1.3.1 Pure

lemma *all-surj-conv'* [reversal-rule]: **assumes** *surj f* **shows** $Pure.all (P \circ f) \equiv Pure.all P$
<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** $All (P \circ f) = All P$
<proof>

lemma *Ex-surj-conv'* [reversal-rule]: **assumes** *surj f* **shows** $Ex (P \circ f) \longleftrightarrow Ex P$
<proof>

lemma *Ex1-bij-conv'* [reversal-rule]: **assumes** *bij f* **shows** $Ex1 (P \circ f) \longleftrightarrow Ex1 P$
<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*: $Collect (P \circ f) = f^{-1} (Collect P)$
<proof>

lemma *collect-image'* [reversal-rule]: **assumes** $f \circ f = id$ **shows** $Collect (P \circ f) = f^{-1} (Collect P)$
<proof>

lemma *Ball-image* [reversal-rule]: **assumes** $(g \circ f)^{-1} A = A$ **shows** $Ball (f^{-1} A) (P \circ g) = Ball A P$
<proof>

lemma *Bex-image-comp*: $Bex (f^{-1} A) g = Bex A (g \circ f)$
<proof>

lemma *Bex-image* [reversal-rule]: **assumes** $(g \circ f)^{-1} A = A$ **shows** $Bex (f^{-1} A) (P \circ g) = Bex A P$
<proof>

lemma *insert-image* [reversal-rule]: $insert (f x) (f^{-1} X) = f^{-1} (insert x X)$
<proof>

lemmas [reversal-rule] = *inj-image-mem-iff*

— (\subseteq)

lemmas [reversal-rule] = *inj-image-subset-iff*

1.3.4 HOL.List

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

— (#)
lemma *Cons-rev*: $a \# \text{rev } u = \text{rev } (\text{snocs } u [a])$
 ⟨*proof*⟩

lemma *Cons-map*: $(f x) \# (\text{map } f xs) = \text{map } f (x \# xs)$
 ⟨*proof*⟩

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

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

— *tl*
lemma *tl-rev*: $\text{tl } (\text{rev } xs) = \text{rev } (\text{butlast } xs)$
 ⟨*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*: $\text{List.coset } (\text{rev } xs) = \text{List.coset } xs$
 ⟨*proof*⟩

lemma *coset-map*: **assumes** *bij f shows* $\text{List.coset } (\text{map } f xs) = f ' \text{List.coset } xs$
 ⟨*proof*⟩

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

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

— *concat*
lemma *concat-rev-map-rev*: $\text{concat } (\text{rev } (\text{map } \text{rev } xs)) = \text{rev } (\text{concat } xs)$
 ⟨*proof*⟩

lemma *concat-rev-map-rev'*: $\text{concat } (\text{rev } (\text{map } (\text{rev } \circ f) xs)) = \text{rev } (\text{concat } (\text{map } f xs))$
 ⟨*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*: $\text{bij } f \implies \text{bij } (\text{image } f)$
<proof>

lemma *inj-image*: $\text{inj } f \implies \text{inj } (\text{image } f)$
<proof>

lemma *surj-image*: $\text{surj } f \implies \text{surj } (\text{image } f)$
<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:  $xs \neq [] \implies \text{rotate } 1 \text{ } xs = \text{tl } xs @ [\text{hd } xs]$ 
  <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
  List-Power.List-Power
begin
```

Chapter 2

Basics of Combinatorics on Words

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

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

The most natural way to represent finite words is by the type '*a list*'. From an algebraic viewpoint, lists are free monoids. On the other hand, any free monoid is isomorphic 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 (\circ) 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 \cdot , which we use and add here.

notation *append* (**infix** \cdot 65)

lemmas *rassoc* = *append-assoc*

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

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

notation

length ($|$ -| 1000) — note that it's bold |

notation (*latex output*)

length ($|$ -|)

notation *longest-common-prefix* (**infixr** \wedge_p 61) — provided by Sublist.thy

2.1.1 Empty and nonempty word

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

notation *Nil* (ε)

named-theorems *emp-simps pow-list-Nil*

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

2.1.2 Prefix

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

notation *prefix* (**infixl** \leq_p 50)

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

lemma *prefI* [*intro*]: $p \cdot s = w \implies p \leq_p w$
<proof>

lemma *prefD*: $u \leq_p v \implies \exists z. v = u \cdot z$
<proof>

definition *prefix-comparable* :: *'a list* \Rightarrow *'a list* \Rightarrow *bool* (**infixl** \bowtie 50)

where (*prefix-comparable* u v) $\equiv u \leq_p v \vee v \leq_p u$

lemma *pref-compI1*: $u \leq_p v \implies u \bowtie v$
<proof>

lemma *pref-compI2*: $v \leq_p u \implies u \bowtie v$
<proof>

lemma *pref-compE* [*elim*]: **assumes** $u \bowtie v$ **obtains** $u \leq_p v \mid v \leq_p u$
<proof>

lemma *pref-compI* [*intro*]: $u \leq_p v \vee v \leq_p u \implies u \bowtie v$

<proof>

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

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

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

interpretation *lcp*: *semilattice-order* (\wedge_p) *prefix strict-prefix*

<proof>

lemma *spreI1[intro]*: $v = u \cdot z \implies z \neq \varepsilon \implies u <_p v$

<proof>

lemma *spreI1'[intro]*: $u \cdot z = v \implies z \neq \varepsilon \implies u <_p v$

<proof>

lemma *spreI2[intro]*: $u \leq_p v \implies |u| < |v| \implies u <_p v$

<proof>

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

lemma *spreD*: $u <_p v \implies u \leq_p v \wedge u \neq v$

<proof>

lemmas *spreD1[dest]* = *prefix-order.strict-implies-order* **and**

spreD2 = *prefix-order.less-imp-neq*

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

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

<proof>

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

<proof>

2.1.3 Suffix

notation *suffix* (**infixl** \leq_s 50)

notation (*latex output*) *suffix* (\leq_s)

lemma *sufI[intro]*: $p \cdot s = w \implies s \leq_s w$

<proof>

lemma *sufD[elim]*: $u \leq_s v \implies \exists z. z \cdot u = v$

<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*]: $u \cdot v = w \implies u \neq \varepsilon \implies v <_s w$
<proof>

lemma *ssufI2*[*intro*]: $u \leq_s v \implies \text{length } u < \text{length } v \implies u <_s v$
<proof>

lemma *ssufE*: $u <_s v \implies (u \leq_s v \implies u \neq v \implies \text{thesis}) \implies \text{thesis}$
<proof>

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

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

definition *suffix-comparable* :: 'a list \Rightarrow 'a list \Rightarrow bool (**infixl** \bowtie_s 50)
where (*suffix-comparable* $u\ v$) \longleftrightarrow (*rev* u) \bowtie (*rev* v)

lemma *suf-compI1*[*intro*]: $u \leq_s v \implies u \bowtie_s v$
<proof>

lemma *suf-compI2*[*intro*]: $v \leq_s u \implies u \bowtie_s v$
<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** \leq_f 50)
notation (*latex output*) *sublist* (\leq_f)
lemmas *fac-def* = *sublist-def*

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

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

notation (*latex output*) *nonempty-factor* (\leq_{nf})

lemmas *facI* = *sublist-appendI*

lemma *facI'*: $a \cdot u \cdot b = w \implies u \leq_f w$
<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

thm *drop-eq-Nil*

lemma *exE2*: $\exists x \ y. P \ x \ y \implies (\bigwedge x \ y. P \ x \ y \implies thesis) \implies thesis$
<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[simp]* = *list.map-comp[of rev rev, unfolded rev-involution'*
list.map-id]

lemma *map-rev-lists-rev*: $map \ rev \ ' (lists \ (rev \ 'A)) = lists \ A$
<proof>

lemma *inj-on-map-lists*: **assumes** *inj-on f A*
shows *inj-on (map f) (lists A)*
<proof>

lemma *bij-lists*: $bij\text{-betw } f \ X \ Y \implies bij\text{-betw } (map \ f) \ (lists \ X) \ (lists \ Y)$
<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 *hd-Cons-append*[*intro,simp*]: $hd ((a\#v) \cdot u) = a$
<proof>

lemma *pref-singE*: **assumes** $p \leq_p [a]$ **obtains** $p = \varepsilon \mid p = [a]$
<proof>

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

lemma *hd-tl*: $w \neq \varepsilon \implies [hd w] \cdot tl w = w$
<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 *long-list-tl*[*intro*]: **assumes** $1 < |w|$
shows $tl w \neq \varepsilon$
<proof>

lemma *long-last-tl*: **assumes** $1 < |w|$ **shows** $last (tl w) = last w$
<proof>

lemma *hd-middle-last*: **assumes** $1 < |w|$
shows $[hd w] \cdot butlast (tl w) \cdot [last w] = w$
<proof>

lemma *hd-pref*[*intro*]: $w \neq \varepsilon \implies [hd w] \leq_p w$
<proof>

lemma *pref-hd-pref*[*intro*]: $u \cdot v = w \implies u \neq w \implies u \cdot [hd v] \leq_p w$
<proof>

lemma *pref-hd-pref'*[*intro*]: $u \cdot v = w \implies v \neq \varepsilon \implies u \cdot [hd v] \leq_p w$
<proof>

lemma Nil-take-Nil [simp]: $u = \varepsilon \implies \text{take } p \ u = \varepsilon$
<proof>

lemma add-nth: **assumes** $n < |w|$ **shows** $(\text{take } n \ w) \cdot [w!n] \leq_p w$
<proof>

lemma hd-pref': **assumes** $w \neq \varepsilon$ **shows** $[w!0] \leq_p w$
<proof>

lemma sub-lists-mono: $A \subseteq B \implies x \in \text{lists } A \implies x \in \text{lists } B$
<proof>

lemma lists-hd-in-set[simp]: $us \neq \varepsilon \implies us \in \text{lists } Q \implies \text{hd } us \in Q$
<proof>

lemma lists-last-in-set[simp]: $us \neq \varepsilon \implies us \in \text{lists } Q \implies \text{last } us \in Q$
<proof>

lemma replicate-in-lists: $\text{replicate } k \ z \in \text{lists } \{z\}$
<proof>

lemma tl-in-lists: **assumes** $us \in \text{lists } A$ **shows** $\text{tl } us \in \text{lists } A$
<proof>

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

lemma tl-set: $x \in \text{set } (\text{tl } a) \implies x \in \text{set } a$
<proof>

lemma drop-take-inv: $n \leq |u| \implies \text{drop } n \ (\text{take } n \ u \cdot w) = w$
<proof>

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

lemma flatten-lists: $G \subseteq \text{lists } B \implies xs \in \text{lists } G \implies \text{concat } xs \in \text{lists } B$
<proof>

lemma concat-map-sing-ident: $\text{concat } (\text{map } (\lambda x. [x]) \ xs) = xs$
<proof>

lemma hd-concat-tl: **assumes** $ws \neq \varepsilon$ **shows** $\text{hd } ws \cdot \text{concat } (\text{tl } ws) = \text{concat } ws$
<proof>

lemma concat-butlast-last: **assumes** $n \text{emp}$: $ws \neq \varepsilon$ **shows** $\text{concat } (\text{butlast } ws) \cdot \text{last } ws = \text{concat } ws$
<proof>

lemma *sprel-butlast-pref*: **assumes** $u \leq_p v$ **and** $u \neq v$ **shows** $u \leq_p \text{butlast } v$
 ⟨proof⟩

lemma *last-concat*: $xs \neq \varepsilon \implies \text{last } xs \neq \varepsilon \implies \text{last } (\text{concat } xs) = \text{last } (\text{last } xs)$
 ⟨proof⟩

lemma *concat-last-suf*: $ws \neq \varepsilon \implies \text{last } ws \leq_s \text{concat } ws$
 ⟨proof⟩

lemma *concat-hd-pref*: $ws \neq \varepsilon \implies \text{hd } ws \leq_p \text{concat } ws$
 ⟨proof⟩

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

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

lemma *drop-neq [intro]*: $w \neq \varepsilon \implies 0 < n \implies \text{drop } n \ w \neq w$
 ⟨proof⟩

lemma *suf-drop-conv*: $u \leq_s w \longleftrightarrow \text{drop } (|w| - |u|) \ w = u$
 ⟨proof⟩

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

lemma *rev-induct2*:
 [$P [] []$;
 $\bigwedge x \ xs. P (xs \cdot [x]) []$;
 $\bigwedge y \ ys. P [] (ys \cdot [y])$;
 $\bigwedge x \ xs \ y \ ys. P \ xs \ ys \implies P (xs \cdot [x]) (ys \cdot [y])$]
 $\implies P \ xs \ ys$
 ⟨proof⟩

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

lemma *rev-rev-image-eq [reversal-rule]*: $\text{rev } \text{' } \text{rev } \text{' } X = X$
 ⟨proof⟩

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

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

(map f u)
⟨proof⟩

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

lemma hd-hdE: (u = ε ⇒ thesis) ⇒ (u = [hd u] ⇒ thesis) ⇒ (u = [hd u,
hd (tl u)] · tl (tl u) ⇒ thesis) ⇒ thesis
⟨proof⟩

lemma same-sing-pref: u · [a] ≤p v ⇒ u · [b] ≤p v ⇒ a = b
⟨proof⟩

lemma compow-Suc: (f[~](Suc k)) w = f ((f[~]k) w)
⟨proof⟩

lemma compow-Suc': (f[~](Suc k)) w = (f[~]k) (f w)
⟨proof⟩

2.2.1 General facts

lemma doubleton-neq-eq-iff [simp]: x ≠ y ⇒ {a,b} = {x,y} ↔ a ≠ b ∧ a ∈
{x,y} ∧ b ∈ {x,y}
⟨proof⟩

lemma two-elem-sub: x ∈ A ⇒ y ∈ A ⇒ {x,y} ⊆ A
⟨proof⟩

thm fun.inj-map[THEN injD]

lemma inj-comp: assumes inj (f :: 'a list ⇒ 'b list) shows (g w = h w ↔ (f
○ g) w = (f ○ h) w)
⟨proof⟩

lemma inj-comp-eq: assumes inj (f :: 'a list ⇒ 'b list) shows (g = h ↔ f ○ g
= f ○ h)
⟨proof⟩

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

lemma two-elem-cases2[elim]: assumes u ∈ {x, y} v ∈ {x,y} u ≠ v
shows (u = x ⇒ v = y ⇒ thesis) ⇒ (u = y ⇒ v = x ⇒ thesis) ⇒ thesis
⟨proof⟩

lemma two-elemP: u ∈ {x, y} ⇒ P x ⇒ P y ⇒ P u
⟨proof⟩

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

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

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

2.2.2 Map injective function

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

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

lemma *map-fac-conv* [*reversal-rule*]: **assumes** *inj f* **shows** $map f u \leq_f map f v \longleftrightarrow u \leq_f v$
 ⟨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*]: $u \in S \implies u \neq \varepsilon \implies u \in S - \{\varepsilon\}$
 ⟨proof⟩

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

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

lemma *pref-nemp* [*intro*]: $u \neq \varepsilon \implies u \cdot v \neq \varepsilon$
<proof>

lemma *suf-nemp* [*intro*]: $v \neq \varepsilon \implies u \cdot v \neq \varepsilon$
<proof>

lemma *pref-of-emp*: $u \cdot v = \varepsilon \implies u = \varepsilon$
<proof>

lemma *suf-of-emp*: $u \cdot v = \varepsilon \implies v = \varepsilon$
<proof>

lemma *nemp-comm*: $(u \neq \varepsilon \implies v \neq \varepsilon \implies u \neq v \implies u \cdot v = v \cdot u) \implies u \cdot v = v \cdot u$
<proof>

lemma *non-triv-comm* [*intro*]: $(u \neq \varepsilon \implies v \neq \varepsilon \implies u \neq v \implies u \cdot v = v \cdot u) \implies u \cdot v = v \cdot u$
<proof>

lemma *split-list'*: $a \in \text{set } ws \implies \exists p s. ws = p \cdot [a] \cdot s$
<proof>

lemma *split-listE*: **assumes** $a \in \text{set } w$
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 possibilities 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 $u \in \{x, y\} \implies P u$
<proof>

lemma $\bigwedge u. u \in \{x, y\} \implies u = x \vee u = y$
<proof>

2.3 Length and its properties

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

lemma *lenarg-not*: $|u| \neq |v| \implies u \neq v$
<proof>

lemma *len-less-neq*: $|u| < |v| \implies u \neq v$
<proof>

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

lemma *npos-len*: $|u| \leq 0 \implies u = \varepsilon$
<proof>

lemma *nemp-len[intro]*: $w \neq \varepsilon \implies 0 < |w|$
<proof>

lemma *nemp-len-not0 [intro]*: $w \neq \varepsilon \implies |w| \neq 0$
<proof>

lemma *nemp-le-len*: $r \neq \varepsilon \implies 1 \leq |r|$
<proof>

lemma *nemp-spref-len*: **assumes** $u \neq \varepsilon$ $u <_p v$ **shows** $1 < |v|$
<proof>

lemma *nemp-ssuf-len*: **assumes** $u \neq \varepsilon$ $u <_s v$ **shows** $1 < |v|$
<proof>

lemma *tl-emp [intro]*: $|w| \leq 1 \implies tl\ w = \varepsilon$
<proof>

lemma *butlast-emp [intro]*: $|w| \leq 1 \implies butlast\ w = \varepsilon$
<proof>

lemma *butlast-tl-emp[intro]*: $|w| \leq 2 \implies butlast(tl\ w) = \varepsilon$
<proof>

lemma *tl-butlast-emp[intro]*: $|w| \leq 2 \implies tl(butlast\ w) = \varepsilon$
<proof>

lemma *swap-len*: $|u \cdot v| = |v \cdot u|$
<proof>

lemma *len-after-drop*: $p + q \leq |w| \implies q \leq |\text{drop } p \ w|$
<proof>

lemma *short-take-append*: $n \leq |u| \implies \text{take } n \ (u \cdot v) = \text{take } n \ u$
<proof>

lemma *sing-word*: $|us| = 1 \implies [\text{hd } us] = us$
<proof>

lemma *sing-word-concat*: **assumes** $|us| = 1$ **shows** $[\text{concat } us] = us$
<proof>

lemma *len-one-concat-in*: $ws \in \text{lists } A \implies |ws| = 1 \implies \text{concat } ws \in A$
<proof>

lemma *concat-nemp*: $\text{concat } us \neq \varepsilon \implies us \neq \varepsilon$
<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*: $u \leq_f v \implies |u| \leq |v|$
<proof>

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

lemma *fac-len-eq*: $u \leq_f v \implies |u| = |v| \implies u = v$
<proof>

thm *length-take*

lemma *len-take1*: $|\text{take } p \ w| \leq p$
<proof>

lemma *len-take2*: $|\text{take } p \ w| \leq |w|$

<proof>

lemma *drop-len*: $|u \cdot w| \leq |u \cdot v \cdot w|$
<proof>

lemma *drop-pref*: $\text{drop } |u| (u \cdot w) = w$
<proof>

lemma *take-len*: $p \leq |w| \implies |\text{take } p \ w| = p$
<proof>

lemma *eq-conjug-len*: $p \cdot x = x \cdot s \implies |p| = |s|$
<proof>

lemma *take-nemp-len*: $u \neq \varepsilon \implies r \neq \varepsilon \implies \text{take } |r| \ u \neq \varepsilon$
<proof>

lemma $r \neq \varepsilon \implies w \neq \varepsilon \implies \text{drop } |r| \ w \neq w$
<proof>

lemma *emp-len*: $w = \varepsilon \implies |w| = 0$
<proof>

lemma *take-self*: $\text{take } |w| \ w = w$
<proof>

lemma *len-le-concat*: $\varepsilon \notin \text{set } ws \implies |ws| \leq |\text{concat } ws|$
<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*: $|w| = \text{Suc } n \implies w \neq \varepsilon$
<proof>

lemma *same-suffix-nil*: **assumes** $z \cdot u \leq_p u$ **shows** $z = \varepsilon$
<proof>

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

lemma *bin-len-concat*: **assumes** $u \neq v \ ws \in \text{lists } \{u, v\}$

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

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

List inspection demonstrations

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

lemma $|w| = 2 \wedge w \in \text{lists } \{x,y,z\} \implies P w$
<proof>

The required premise as 2 separate premises.

lemma $|w| \leq 2 \implies w \in \text{lists } \{x,y,z\} \implies P w$
<proof>

The required premise as 2 separate assumptions.

lemma assumes $|w| \leq 2$ **and** $w \in \text{lists } \{x,y,z\}$ **shows** $P w$
<proof>

lemma $w \leq_p w \implies |w| \leq 2 \implies w \in \text{lists } \{a,b\} \implies \text{hd } w = a \implies w \neq \varepsilon \implies w = [a, b] \vee w = [a, a] \vee w = [a]$
<proof>

lemma $w \leq_p w \implies |w| = 2 \implies w \in \text{lists } \{a,b\} \implies \text{hd } w = a \implies w = [a, b] \vee w = [a, a]$
<proof>

lemma $w \leq_p w \implies |w| = 2 \wedge w \in \text{lists } \{a,b\} \implies \text{hd } w = a \implies w = [a, b] \vee w = [a, a]$
<proof>

lemma $w \leq_p w \implies w \in \text{lists } \{a,b\} \wedge |w| = 2 \implies \text{hd } w = a \implies w = [a, b] \vee w = [a, a]$
<proof>

end

2.5 Prefix and prefix comparability properties

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

lemma *triv-pref*: $r \leq_p r \cdot s$
<proof>

lemma *triv-spref*: $s \neq \varepsilon \implies r <_p r \cdot s$
<proof>

lemma *pref-cancel*: $z \cdot u \leq_p z \cdot v \implies u \leq_p v$
<proof>

lemma *pref-cancel'* [*intro*]: $u \leq_p v \implies z \cdot u \leq_p z \cdot v$
<proof>

lemma *spref-cancel*: $z \cdot u <_p z \cdot v \implies u <_p v$
<proof>

lemma *spref-of-nemp* [*intro*]: $u <_p w \implies w \neq \varepsilon$
<proof>

lemma *spref-cancel'* [*intro*]: $u <_p v \implies z \cdot u <_p z \cdot v$
<proof>

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

lemma *pref-cancel-hd-conv*: $a \# u \leq_p a \# v \longleftrightarrow u \leq_p v$
<proof>

lemma *spref-cancel-conv*: $z \cdot x <_p z \cdot y \longleftrightarrow x <_p y$
<proof>

lemma *spref-snoc-iff* [*simp*]: $u <_p v \cdot [a] \longleftrightarrow u \leq_p v$
<proof>

lemma *butlast-not-pref* [*intro, simp*]: **assumes** $u \neq \varepsilon$ **shows** $\neg u \leq_p \text{butlast } u$
<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*: $x \cdot y <_p z \implies x <_p z$
<proof>

lemma *spref-ext*: $r <_p u \implies r <_p u \cdot v$
<proof>

lemma *pref-ext-nemp*: $r \leq_p u \implies v \neq \varepsilon \implies r <_p u \cdot v$
<proof>

lemma *pref-take*: $p \leq_p w \implies \text{take } |p| \ w = p$
<proof>

lemma *pref-take-conv*: $\text{take } (|r|) \ w = r \longleftrightarrow r \leq_p w$

<proof>

lemma *le-suf-drop*: **assumes** $i \leq j$ **shows** $\text{drop } j \ w \leq_s \text{drop } i \ w$
<proof>

lemma *spref-take*: $p <_p w \implies \text{take } |p| \ w = p$
<proof>

lemma *pref-same-len*: $u \leq_p v \implies |u| = |v| \implies u = v$
<proof>

lemma *pref-same-len'*: $u \cdot z \leq_p v \cdot w \implies |u| = |v| \implies u = v$
<proof>

lemma *pref-comp-eq*: $u \bowtie v \implies |u| = |v| \implies u = v$
<proof>

lemma *ruler-eq-len*: $u \leq_p w \implies v \leq_p w \implies |u| = |v| \implies u = v$
<proof>

lemma *pref-prod-eq*: $u \leq_p v \cdot z \implies |u| = |v| \implies u = v$
<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*: $|u| \leq |w| \implies (\forall i < |u|. \ u!i = w!i) \implies u \leq_p w$
<proof>

lemma *pref-index*: **assumes** $u \leq_p w$ $i < |u|$ **shows** $u!i = w!i$
<proof>

lemma *pref-drop*: $u \leq_p v \implies \text{drop } p \ u \leq_p \text{drop } p \ v$
<proof>

lemma *pref-prod-short*: $u \leq_p v \cdot w \implies |u| < |v| \implies u <_p v$
<proof>

2.5.1 Prefix comparability

lemma *pref-comp-sym*[*sym*]: $u \bowtie v \implies v \bowtie u$
<proof>

lemma *not-pref-comp-sym*[*sym*]: $\neg u \bowtie v \implies \neg v \bowtie u$
<proof>

lemma *pref-comp-sym-iff*: $u \bowtie v \iff v \bowtie u$
<proof>

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

lemma *ruler-prefs*: **assumes** $L = R \mid u \mid \leq \mid v \mid \mid u \leq_p L \mid v \leq_p R$
shows $u \leq_p v$
<proof>

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

lemma *ruler-eq*: $u \cdot x = v \cdot y \implies u \leq_p v \vee v \leq_p u$
<proof>

lemma *ruler-eq'*: $u \cdot x = v \cdot y \implies u \leq_p v \vee v <_p u$
<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*: $u \leq_p v \implies u' \leq_p v' \implies v \bowtie v' \implies u \bowtie u'$
<proof>

lemma *ruler-pref'*: $w \leq_p v \cdot z \implies w \leq_p v \vee v \leq_p w$
<proof>

lemma *ruler-pref''*: $w \leq_p v \cdot z \implies w \bowtie v$
<proof>

lemma *pref-prod-pref-short*: $u \leq_p z \cdot w \implies v \leq_p w \implies \mid u \mid \leq \mid z \cdot v \mid \implies u \leq_p z \cdot v$
<proof>

lemma *pref-prod-pref*: $u \leq_p z \cdot w \implies u \leq_p w \implies u \leq_p z \cdot u$
<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*: $u \leq_p v \cdot w \implies \mid v \mid \leq \mid u \mid \implies v \leq_p u$

<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 *ruler-less*: $ps \leq_p xs \implies qs \leq_p xs \implies |ps| < |qs| \implies ps <_p qs$
<proof>

lemma *ruler-le-neq*: $ps \leq_p xs \implies qs \leq_p xs \implies |ps| \leq |qs| \implies ps \neq qs \implies ps <_p qs$
<proof>

lemma *pref-keeps-per-root*: $u \leq_p r \cdot u \implies v \leq_p u \implies v \leq_p r \cdot v$
<proof>

lemma *pref-keeps-per-root'*: $u <_p r \cdot u \implies v \leq_p u \implies v <_p r \cdot v$
<proof>

lemma *per-root-pref-sing*: $w <_p r \cdot w \implies u \cdot [a] \leq_p w \implies u \cdot [a] \leq_p r \cdot u$
<proof>

lemma *pref-prolong*: $w \leq_p z \cdot r \implies r \leq_p s \implies w \leq_p z \cdot s$
<proof>

lemma *spref--pref-prolong*: $w <_p z \cdot r \implies r \leq_p s \implies w <_p z \cdot s$
<proof>

lemma *pref-spref-prolong*: $w \leq_p z \cdot r \implies r <_p s \implies w <_p z \cdot s$
<proof>

lemma *spref-spref-prolong*: $w <_p z \cdot r \implies r <_p s \implies w <_p z \cdot s$
<proof>

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

lemma *pref-prolong'*: $u \leq_p w \cdot z \implies v \cdot u \leq_p z \implies u \leq_p w \cdot v \cdot u$
<proof>

lemma *pref-prolong-per-root*: $u \leq_p r \cdot s \implies s \leq_p r \cdot s \implies u \leq_p r \cdot u$
<proof>

lemma *pref-prod-le[intro]*: $u \leq_p v \cdot w \implies |u| \leq |v| \implies u \leq_p v$
<proof>

lemma *prod-pref-prod-le*: $u \cdot v \leq_p x \cdot y \implies |u| \leq |x| \implies u \leq_p x$
<proof>

lemma *pref-cancel-right-len*: $u \cdot z \leq_p v \cdot z' \implies |z| = |z'| \implies u \leq_p v$
 ⟨proof⟩

lemma *spref-cancel-right-len*: **assumes** $u \cdot z <_p v \cdot z' \quad |z| = |z'|$
shows $u <_p v$
 ⟨proof⟩

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

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

lemma *pref-less-spref*: $u \cdot w \leq_p v \cdot z \implies |u| < |v| \implies u <_p v$
 ⟨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*: $\neg u \bowtie v \iff u \parallel v$
 ⟨proof⟩

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

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

lemma *comm-ruler*: $r \cdot s \leq_p w1 \implies s \cdot r \leq_p w2 \implies w1 \bowtie w2 \implies r \cdot s = s \cdot r$
 ⟨proof⟩

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

lemma *pref-share-take*: $u \leq_p v \implies q \leq |u| \implies \text{take } q \ u = \text{take } q \ v$
 ⟨proof⟩

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

lemma *pref-comp-not-pref*: $u \bowtie v \implies \neg v \leq_p u \implies u <_p v$
<proof>

lemma *pref-comp-not-spref*: $u \bowtie v \implies \neg u <_p v \implies v \leq_p u$
<proof>

lemma *hd-prod*: $u \neq \varepsilon \implies (u \cdot v)!0 = u!0$
<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'*: $u <_p w \implies w \leq_p u \cdot [a] \implies w = u \cdot [a]$
<proof>

lemma *comp-shorter*: $v \bowtie w \implies |v| \leq |w| \implies v \leq_p w$
<proof>

lemma *comp-shorter-conv*: $|u| \leq |v| \implies u \bowtie v \longleftrightarrow u \leq_p v$
<proof>

lemma *pref-comp-len-trans*: $w \leq_p v \implies u \bowtie v \implies |w| \leq |u| \implies w \leq_p u$
<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*: $u \neq \varepsilon \implies \varepsilon <_p u$
<proof>

lemma *long-pref*: $u \leq_p v \implies |v| \leq |u| \implies u = v$
<proof>

lemma *not-comp-ext*: $\neg w1 \bowtie w2 \implies \neg w1 \cdot z \bowtie w2 \cdot z'$
<proof>

lemma *mismatch-incopm*: $|u| = |v| \implies x \neq y \implies \neg u \cdot [x] \bowtie v \cdot [y]$
<proof>

lemma *comp-prefs-comp*: $u \cdot z \bowtie v \cdot w \implies u \bowtie v$
<proof>

lemma *comp-hd-eq*: $u \bowtie v \implies u \neq \varepsilon \implies v \neq \varepsilon \implies \text{hd } u = \text{hd } v$
<proof>

lemma *pref-hd-eq'*: $p \leq_p u \implies p \leq_p v \implies p \neq \varepsilon \implies \text{hd } u = \text{hd } v$
 ⟨proof⟩

lemma *pref-hd-eq*: $u \leq_p v \implies u \neq \varepsilon \implies \text{hd } u = \text{hd } v$
 ⟨proof⟩

lemma *sing-pref-hd*: $[a] \leq_p v \implies \text{hd } v = a$
 ⟨proof⟩

lemma *suf-last-eq*: $p \leq_s u \implies p \leq_s v \implies p \neq \varepsilon \implies \text{last } u = \text{last } v$
 ⟨proof⟩

lemma *comp-hd-eq'*: $u \cdot r \bowtie v \cdot s \implies u \neq \varepsilon \implies v \neq \varepsilon \implies \text{hd } u = \text{hd } v$
 ⟨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** $\bigwedge y. y \leq_p w \implies P y \implies v \leq_p y$
 ⟨proof⟩

lemma *min-pref'*: **assumes** $u \leq_p w$ **and** $P u$
obtains v **where** $v \leq_p w$ **and** $P v$ **and** $\bigwedge y. y \leq_p v \implies P y \implies y = v$
 ⟨proof⟩

lemma *max-pref*: **assumes** $u \leq_p w$ **and** $P u$
obtains v **where** $v \leq_p w$ **and** $P v$ **and** $\bigwedge y. y \leq_p w \implies P y \implies y \leq_p v$
 ⟨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*: $u \neq \varepsilon \implies \varepsilon <_s u$
 ⟨proof⟩

lemma *suf-cancel*: $u \cdot v \leq_s w \cdot v \implies u \leq_s w$
 ⟨proof⟩

lemma *suf-cancel'* [*intro*]: $u \leq_s w \implies u \cdot v \leq_s w \cdot v$
 ⟨proof⟩

lemma *ssuf-cancel'* [intro]: $u <_s w \implies u \cdot v <_s w \cdot v$
 ⟨proof⟩

lemma *ssuf-cancel-conv*: $x \cdot z <_s y \cdot z \iff x <_s y$
 ⟨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*: $u \leq_p v \iff \text{rev } u \leq_s \text{rev } v$
 ⟨proof⟩

lemma *spref-rev-suf-iff*: $s <_p w \iff \text{rev } s <_s \text{rev } w$
 ⟨proof⟩

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

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

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

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

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

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

lemma *ssuf-Cons-iff* [simp]: $u <_s a \# v \iff u \leq_s v$
 ⟨proof⟩

lemma *ssuf-induct* [case-names *ssuf*]:
 assumes $\bigwedge u. (\bigwedge v. v <_s u \implies P v) \implies P u$

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

lemmas $comp-sufs-comp = comp-prefs-comp[reversed]$ **and**

$suf\text{-}comp\text{-}not\text{-}suf = pref\text{-}comp\text{-}not\text{-}pref[reversed]$ **and**
 $suf\text{-}comp\text{-}not\text{-}ssuf = pref\text{-}comp\text{-}not\text{-}spref[reversed]$ **and**
 $suf\text{-}comp\text{-}cancel = comp\text{-}cancel[reversed]$ **and**
 $suf\text{-}not\text{-}comp\text{-}ext = not\text{-}comp\text{-}ext[reversed]$ **and**
 $mismatch\text{-}suf\text{-}incopm = mismatch\text{-}incopm[reversed]$ **and**
 $suf\text{-}comp\text{-}sym[sym] = pref\text{-}comp\text{-}sym[reversed]$ **and**
 $suf\text{-}comp\text{-}refl = comp\text{-}refl[reversed]$

lemma $suf\text{-}comp\text{-}or$: $u \bowtie_s v \longleftrightarrow u \leq_s v \vee v \leq_s u$
 $\langle proof \rangle$

lemma $comm\text{-}comp\text{-}eq\text{-}conv$: $r \cdot s \bowtie s \cdot r \longleftrightarrow r \cdot s = s \cdot r$
 $\langle proof \rangle$

lemma $comm\text{-}comp\text{-}eq\text{-}conv\text{-}suf$: $r \cdot s \bowtie_s s \cdot r \longleftrightarrow r \cdot s = s \cdot r$
 $\langle proof \rangle$

lemma $suf\text{-}comp\text{-}last\text{-}eq$: **assumes** $u \bowtie_s v$ $u \neq \varepsilon$ $v \neq \varepsilon$
shows $last\ u = last\ v$
 $\langle proof \rangle$

lemma $suf\text{-}comp\text{-}last\text{-}eq'$: $r \cdot u \bowtie_s s \cdot v \implies u \neq \varepsilon \implies v \neq \varepsilon \implies last\ u = last\ v$
 $\langle proof \rangle$

2.7 Left and Right Quotient

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

definition $left\text{-}quotient$:: $'a\ list \Rightarrow 'a\ list \Rightarrow 'a\ list$ $((^{-1}>)(-)$ [75,74] 74)

where $left\text{-}quotient\ u\ v = drop\ |u|\ v$

notation (*latex output*) $left\text{-}quotient$ $(-^{-1} \cdot -)$

Analogously, we define the right quotient.

definition $right\text{-}quotient$:: $'a\ list \Rightarrow 'a\ list \Rightarrow 'a\ list$ $((-)(^{<-1})$ [76,77] 76)

where $right\text{-}quotient\ u\ v = rev\ ((rev\ v)^{-1}>(rev\ u))$

notation (*latex output*) $right\text{-}quotient$ $(- \cdot -^{-1})$

lemmas $lq\text{-}def = left\text{-}quotient\text{-}def$ **and**
 $rq\text{-}def = right\text{-}quotient\text{-}def$

Priorities of these operations are as follows:

lemma $u^{<-1}v^{<-1}w = (u^{<-1}v)^{<-1}w$
 $\langle proof \rangle$

lemma $u^{-1}>v^{-1}>w = u^{-1}>(v^{-1}>w)$
 $\langle proof \rangle$

lemma $u^{-1} \triangleright v \triangleleft^{-1} w = u^{-1} \triangleright (v \triangleleft^{-1} w)$
 ⟨proof⟩

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

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

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

2.7.1 Left Quotient

lemma lqI : $u \cdot z = v \implies u^{-1} \triangleright v = z$
 ⟨proof⟩

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

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

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

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

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

lemma $lq\text{-}pref$ [simp]: $u \leq_p v \implies u \cdot (u^{-1} \triangleright v) = v$
 ⟨proof⟩

lemmas $lq\text{-}spref = lq\text{-}pref$ [OF $sprefD1$]

lemma $lq\text{-}pref\text{-}conv$: $|u| \leq |v| \implies u \leq_p v \iff u \cdot u^{-1} \triangleright v = v$
 ⟨proof⟩

lemma $lq\text{-}len$: $|u^{-1} \triangleright v| = |v| - |u|$
 ⟨proof⟩

lemmas $lcp\text{-}lq = lq\text{-}pref$ [OF $longest\text{-}common\text{-}prefix\text{-}prefix1$] $lq\text{-}pref$ [OF $longest\text{-}common\text{-}prefix\text{-}prefix2$]

lemma $lq\text{-}pref\text{-}cancel$: $u \leq_p v \implies v \cdot r = u \cdot s \implies (u^{-1} \triangleright v) \cdot r = s$
 ⟨proof⟩

lemma $lq\text{-}the$: **assumes** $u \leq_p v$

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

lemma *lq-same-len*: $|u| = |v| \implies u^{-1} \triangleright v = \varepsilon$
<proof>

lemma *lq-assoc*: $|u| \leq |v| \implies (u^{-1} \triangleright v)^{-1} \triangleright w = v^{-1} \triangleright (u \cdot w)$
<proof>

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

lemma *lq-reassoc*: $u \leq_p v \implies (u^{-1} \triangleright v) \cdot w = u^{-1} \triangleright (v \cdot w)$
<proof>

lemma *lq-trans*: $u \leq_p v \implies v \leq_p w \implies (u^{-1} \triangleright v) \cdot (v^{-1} \triangleright w) = u^{-1} \triangleright w$
<proof>

lemma *lq-rq-reassoc-suf*: **assumes** $u \leq_p z$ $u \leq_s w$ **shows** $w \cdot u^{-1} \triangleright z = w^{<-1} u \cdot z$
<proof>

lemma *lq-ne*: **assumes** $u \neq \varepsilon$ **shows** $p^{-1} \triangleright (u \cdot p) \neq \varepsilon$
<proof>

lemma *lq-spref-nemp*: $u <_p v \implies u^{-1} \triangleright v \neq \varepsilon$
<proof>

lemma *lq-is-suf[simp]*: $r^{-1} \triangleright s \leq_s s$
<proof>

lemma *lq-short-len*: $r \leq_p s \implies |r| + |r^{-1} \triangleright s| = |s|$
<proof>

lemma *pref-lq*: $v \leq_p w \implies u^{-1} \triangleright v \leq_p u^{-1} \triangleright w$
<proof>

lemma *spref-lq*: $u \leq_p v \implies v <_p w \implies u^{-1} \triangleright v <_p u^{-1} \triangleright w$
<proof>

lemma *pref-gcd-lq*: **assumes** $u \leq_p v$ **shows** $(\text{gcd } |u| \ |u^{-1} \triangleright v|) = \text{gcd } |u| \ |v|$
<proof>

lemma *conjug-lq*: $x \cdot z = z \cdot y \implies y = z^{-1} \triangleright (x \cdot z)$
<proof>

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

lemma *hd-lq-conv-nth*: **assumes** $u <_p v$ **shows** $\text{hd}(u^{-1} \triangleright v) = v!|u|$

<proof>

lemma *concat-morph-lq*: $us \leq_p ws \implies \text{concat}(us^{-1}\triangleright ws) = (\text{concat } us)^{-1}\triangleright(\text{concat } ws)$

<proof>

lemma *pref-suf-len-split*: **assumes** $p \leq_p u$ $s \leq_s u$ $|p \cdot s| = |u|$
shows $p \cdot s = u$

<proof>

lemma *pref-cancel-lq*: **assumes** $u \leq_p x \cdot y$
shows $x^{-1}\triangleright u \leq_p y$

<proof>

lemma *pref-cancel-lq-ext*: **assumes** $u \cdot v \leq_p x \cdot y$ **and** $|x| \leq |u|$ **shows** $x^{-1}\triangleright u \cdot v \leq_p y$

<proof>

lemma *pref-cancel-lq-ext'*: **assumes** $u \cdot v \leq_p x \cdot y$ **and** $|u| \leq |x|$ **shows** $v \leq_p u^{-1}\triangleright x \cdot y$

<proof>

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

lemma *empty-lq-eq*: $r \leq_p z \implies r^{-1}\triangleright z = \varepsilon \implies r = z$

<proof>

lemma *le-if-then-lq*: $|u| \leq |v| \implies (\text{if } |v| \leq |u| \text{ then } v^{-1}\triangleright u \text{ else } u^{-1}\triangleright v) = u^{-1}\triangleright v$

<proof>

lemma *append-comp-lq*: $u \cdot v \bowtie w \implies v \bowtie u^{-1}\triangleright w$

<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[*simp*] = *lq-spref*[*reversed*] **and**
rq-ssuf-nemp = *lq-spref-nemp*[*reversed*] **and**
rq-reassoc = *lq-reassoc*[*reversed*] **and**
rq-len = *lq-len*[*reversed*] **and**
rq-trans = *lq-trans*[*reversed*] **and**
rq-lq-reassoc-suf = *lq-rq-reassoc-suf*[*reversed*] **and**
rq-ne = *lq-ne*[*reversed*] **and**

$rq\text{-is-pref}[simp] = lq\text{-is-suf}[reversed]$ **and**
 $suf\text{-}rq = pref\text{-}lq[reversed]$ **and**
 $ssuf\text{-}rq = spref\text{-}lq[reversed]$ **and**
 $conjug\text{-}rq = conjug\text{-}lq[reversed]$ **and**
 $conjug\text{-}emp\text{-}emp' = conjug\text{-}emp\text{-}emp[reversed]$ **and**
 $rq\text{-take} = lq\text{-def}[reversed]$ **and**
 $empty\text{-}rq\text{-}eq = empty\text{-}lq\text{-}eq[reversed]$ **and**
 $append\text{-}rq = append\text{-}lq[reversed]$ **and**
 $rq\text{-same-len} = lq\text{-same-len}[reversed]$ **and**
 $rq\text{-assoc} = lq\text{-assoc}[reversed]$ **and**
 $rq\text{-assoc}' = lq\text{-assoc}'[reversed]$ **and**
 $le\text{-if-then-rq} = le\text{-if-then-lq}[reversed]$ **and**
 $append\text{-}comp\text{-}rq = append\text{-}comp\text{-}lq[reversed]$

2.7.3 Left and right quotients combined

lemma $pref\text{-}lq\text{-}rq\text{-}id$: $p \leq_p w \implies w^{<-1}(p^{-1} > w) = p$
<proof>

lemmas $suf\text{-}rq\text{-}lq\text{-}id = pref\text{-}lq\text{-}rq\text{-}id[reversed]$

lemma $rev\text{-}lq'$: $r \leq_p s \implies rev (r^{-1} > s) = (rev s)^{<-1}(rev r)$
<proof>

lemma $pref\text{-}rq\text{-}suf\text{-}lq$: $s \leq_s u \implies r \leq_p (u^{<-1} s) \implies s \leq_s (r^{-1} > u)$
<proof>

lemmas $suf\text{-}lq\text{-}pref\text{-}rq = pref\text{-}rq\text{-}suf\text{-}lq[reversed]$

lemma $w \cdot s = v \implies v^{<-1} s = w$ *<proof>*

lemma $lq\text{-}rq\text{-}assoc$: $s \leq_s u \implies r \leq_p (u^{<-1} s) \implies (r^{-1} > u)^{<-1} s = r^{-1} > (u^{<-1} s)$
<proof>

lemmas $rq\text{-}lq\text{-}assoc = lq\text{-}rq\text{-}assoc[reversed]$

lemma $lq\text{-}prod$: $u \leq_p v \cdot u \implies u \leq_p w \implies u^{-1} > (v \cdot u) \cdot u^{-1} > w = u^{-1} > (v \cdot w)$
<proof>

lemmas $rq\text{-}prod = lq\text{-}prod[reversed]$

lemma $pref\text{-}suf\text{-}mid$: **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-pref*: $x \cdot y = u \cdot v \implies |x| \leq |u| \implies x \cdot (x^{-1} \triangleright u) = u \wedge (x^{-1} \triangleright u) \cdot v = y$
 = y
 <proof>

lemma *eqd*: $x \cdot y = u \cdot v \implies |x| \leq |u| \implies \exists t. x \cdot t = u \wedge t \cdot v = y$
 <proof>

lemma *eqd-or*: $x \cdot y = u \cdot v \implies \exists t. x \cdot t = u \wedge t \cdot v = y \vee u \cdot t = x \wedge t \cdot y = v$
 <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-less*: **assumes** $x \cdot y = u \cdot v$ **and** $|x| < |u|$
shows $\exists t. x \cdot t = u \wedge t \cdot v = y \wedge t \neq \varepsilon$
 <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** $x \leq_p u \iff v \leq_s y$
 <proof>

lemma *eqd-spref-ssuf-iff*: **assumes** $x \cdot y = u \cdot v$ **shows** $x <_p u \iff v <_s y$
 <proof>

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

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

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

lemma *eqd-eq-suf*: $x \cdot y = u \cdot v \implies |y| = |v| \implies x = u \wedge y = v$
<proof>

lemma *eqd-comp*: **assumes** $x \cdot y = u \cdot v$ **shows** $x \bowtie u$
<proof>

lemma *eqd-linear-cases* [*elim*]: **assumes** $x \cdot y = u \cdot v$ **and**
 $\bigwedge t. t \neq \varepsilon \implies x \cdot t = u \implies t \cdot v = y \implies \textit{thesis}$ **and**
 $\bigwedge t. t \neq \varepsilon \implies u \cdot t = x \implies t \cdot y = v \implies \textit{thesis}$ **and**
 $x = u \implies y = v \implies \textit{thesis}$
shows *thesis*
<proof>

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

— not equal to *eqd_pref1*[*reversed*]

lemma *eqd-suf1*: $x \cdot y = u \cdot v \implies |x| \leq |u| \implies (y^{<-1}v) \cdot v = y$
<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*: $u \leq_p v \implies u \leq_p (u \cdot w) \wedge_p (v \cdot z)$
 ⟨*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*: $\text{take}(|u \wedge_p v|) u = \text{take}(|u \wedge_p v|) v$
 ⟨*proof*⟩

lemma *lcp-pref-conv*: $u \wedge_p v = u \iff u \leq_p v$
 ⟨*proof*⟩

lemma *lcp-pref-conv'*: $u \wedge_p v = v \iff v \leq_p u$
 ⟨*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*: $w \leq_p u \wedge_p v \implies w \leq_p u$
 ⟨*proof*⟩

lemma *pref-lcp-pref'*: $w \leq_p u \wedge_p v \implies w \leq_p v$
 ⟨*proof*⟩

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

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

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

lemma *lcp-len'*: $\neg u \leq_p v \implies |u \wedge_p v| < |u|$

$\langle \text{proof} \rangle$

lemma *incomp-lcp-len*: $\neg u \bowtie v \implies |u \wedge_p v| < \min |u| |v|$
 $\langle \text{proof} \rangle$

lemma *lcp-ext-right-conv*: $\neg r \bowtie r' \implies (r \cdot u) \wedge_p (r' \cdot v) = r \wedge_p r'$
 $\langle \text{proof} \rangle$

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

lemma *lcp-same-len*: $|u| = |v| \implies u \neq v \implies u \cdot w \wedge_p v \cdot w' = u \wedge_p v$
 $\langle \text{proof} \rangle$

lemma *lcp-mismatch*: $|u \wedge_p v| < |u| \implies |u \wedge_p v| < |v| \implies u! |u \wedge_p v| \neq v! |u \wedge_p v|$
 $\langle \text{proof} \rangle$

lemma *lcp-mismatch'*: $\neg u \bowtie v \implies u! |u \wedge_p v| \neq v! |u \wedge_p v|$
 $\langle \text{proof} \rangle$

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** $hd\ us' \neq hd\ vs'$
 $\langle \text{proof} \rangle$

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**
 $hd\ ((u \wedge_p v)^{-1} > u) \neq hd\ ((u \wedge_p v)^{-1} > v)$
 $\langle \text{proof} \rangle$

lemma *lcp-ext-left*: $(z \cdot u) \wedge_p (z \cdot v) = z \cdot (u \wedge_p v)$
 $\langle \text{proof} \rangle$

lemma *lcp-first-letters*: $u!0 \neq v!0 \implies u \wedge_p v = \varepsilon$
 $\langle \text{proof} \rangle$

lemma *lcp-first-mismatch*: $a \neq b \implies w \cdot [a] \cdot u \wedge_p w \cdot [b] \cdot v = w$
 $\langle \text{proof} \rangle$

lemma *lcp-first-mismatch'*: $a \neq b \implies u \cdot [a] \wedge_p u \cdot [b] = u$
 $\langle \text{proof} \rangle$

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*: $hd\ u \neq hd\ v \implies u \wedge_p v = \varepsilon$
<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*: $|u \cdot w \wedge_p v \cdot w'| < |u| \implies |u \cdot w \wedge_p v \cdot w'| < |v| \implies u \wedge_p v = u \cdot w \wedge_p v \cdot w'$
<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** $u \wedge_p v \leq_p w \mid u \wedge_p w = v \wedge_p w$
<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

lemma *lexord-cancel-right*: $(u \cdot z, v \cdot w) \in lexord\ r \implies \neg u \bowtie v \implies (u, v) \in lexord\ r$
<proof>

lemma *lcp-rulersE*: **assumes** $r \leq_p s$ **and** $r' \leq_p s'$ **obtains** $r \bowtie r' \mid s \wedge_p s' = r \wedge_p r'$
<proof>

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

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

lemma *lcp-ruler*: $r \bowtie w1 \implies r \bowtie w2 \implies \neg w1 \bowtie w2 \implies r \leq_p w1 \wedge_p w2$
 ⟨proof⟩

lemma *comp-monotone*: $w \bowtie r \implies u \leq_p w \implies u \bowtie r$
 ⟨proof⟩

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

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

lemma *double-ruler*: **assumes** $w \bowtie r$ **and** $w' \bowtie r'$ **and** $\neg r \bowtie r'$
shows $w \wedge_p w' = \text{take } (\min |w| |w'|) (r \wedge_p r')$
 ⟨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 $u v \equiv \text{rev } (rev u \wedge_p rev v)$

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

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

function *mismatch-pair* :: 'a list \Rightarrow 'a list \Rightarrow ('a \times 'a) **where**
mismatch-pair ε v = ($\varepsilon!0$, v!0) |
mismatch-pair v ε = (v!0, $\varepsilon!0$) |
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*: \neg u \bowtie v \Longrightarrow (*mismatch-pair* u v) \notin Id
 <proof>

lemma *mismatch-ext-left*: \neg u \bowtie v \Longrightarrow *mismatch-pair* u v = *mismatch-pair* (p.u)
 (p.v)
 <proof>

lemma *mismatch-ext-right*: **assumes** \neg u \bowtie v
shows *mismatch-pair* u v = *mismatch-pair* (u.z) (v.w)
 <proof>

lemma *mismatchI*: \neg u \bowtie v \Longrightarrow i < min |u| |v| \Longrightarrow take i u = take i v \Longrightarrow u!i
 \neq 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** \neg u \bowtie v **and** *mismatch-pair* u v \in r
shows (u,v) \in 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** \neg u \bowtie v **and** irrefl r
shows *mismatch-pair* u v \in r \longleftrightarrow (u,v) \in lexord r
 <proof>

2.11 Factor properties

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

lemma *rev-fac*[*reversal-rule*]: $\text{rev } u \leq_f \text{rev } v \longleftrightarrow u \leq_f v$
 ⟨*proof*⟩

lemma *fac-pref*: $u \leq_f v \equiv \exists p. p \cdot u \leq_p v$
 ⟨*proof*⟩

lemma *fac-pref-suf*: $u \leq_f v \implies \exists p. p \leq_p v \wedge u \leq_s p$
 ⟨*proof*⟩

lemma *pref-suf-fac*: $r \leq_p v \implies u \leq_s r \implies u \leq_f v$
 ⟨*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*: $s \leq_s p \implies p \leq_p s \implies p = s$
 ⟨*proof*⟩

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

lemma *fac-triv'*: $p \cdot x \cdot q = x \implies q = \varepsilon$
 ⟨*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 $f \leq_f m \cdot (\text{take } (|f| - 1) \ s)$
 ⟨*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 $f \leq_f (\text{drop } (|p| - (|f| - 1)) \ p) \cdot m \cdot (\text{take } (|f| - 1) \ s)$
 ⟨*proof*⟩

lemma *fac-ext-pref*: $u \leq_f w \implies u \leq_f p \cdot w$

<proof>

lemma *fac-ext-suf*: $u \leq_f w \implies u \leq_f w \cdot s$
<proof>

lemma *fac-ext*: $u \leq_f w \implies u \leq_f p \cdot w \cdot s$
<proof>

lemma *fac-ext-hd*: $u \leq_f w \implies u \leq_f a\#w$
<proof>

lemma *card-switch-fac*: **assumes** $2 \leq \text{card } (set\ ws)$
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 $u \leq_f x \cdot y \vee u \leq_f y \cdot z$
<proof>

2.12 Power and its properties

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

abbreviation *listpow* :: 'a list \Rightarrow nat \Rightarrow 'a list (**infixr** $\langle^\text{@}\rangle$ 80)
where $f^\text{@} n \equiv \text{compow } n\ f$

some simplified names

lemmas

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

lemma *pow-pos*: $0 < k \implies a^\text{@}k = a \cdot a^\text{@}(k-1)$
<proof>

lemma *pow-pos2*: $0 < k \implies a^\text{@}k = a^\text{@}(k-1) \cdot a$
<proof>

lemma *pow-diff*: $k < n \implies a^{\textcircled{}}(n - k) = a \cdot a^{\textcircled{}}(n-k-1)$
<proof>

lemma *pow-list-3*: $u^{\textcircled{}}3 = u \cdot u \cdot u$
<proof>

named-theorems *exp-simps*

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

named-theorems *cow-simps*

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

— more power properties

lemma *nemp-pref-nemp*: $u \leq_p v \implies u \neq \varepsilon \implies v \neq \varepsilon$
<proof>

lemma *pref-concat-emp[intro]*: $us \leq_p vs \implies \text{concat } vs = \varepsilon \implies \text{concat } us = \varepsilon$
<proof>

lemma *nemp-exp-pos*: $w \neq \varepsilon \implies r^{\textcircled{}}k = w \implies 0 < k$
<proof>

lemma *nemp-pow-nemp*: $t^{\textcircled{}}m \neq \varepsilon \implies t \neq \varepsilon$
<proof>

lemma *emp-pow-emp*: $t = \varepsilon \implies t^{\textcircled{}}m = \varepsilon$
<proof>

lemma *list-pow-emp [intro]*: $t = \varepsilon \vee m = 0 \implies t^{\textcircled{}}m = \varepsilon$
<proof>

lemma *list-pow-emp-iff [simp]*: $t^{\textcircled{}}m = \varepsilon \iff t = \varepsilon \vee m = 0$
<proof>

lemma *hd-nemp-pow[intro]*: $v^{\textcircled{}}k \neq \varepsilon \implies \text{hd } (v^{\textcircled{}}k) = \text{hd } v$
<proof>

lemma *hd-pref-pow[intro]*: **assumes** $u \leq_p v^{\textcircled{}}k$ **and** $u \neq \varepsilon$

shows $hd\ u = hd\ v$
<proof>

lemma *pop-pow*: $m \leq k \implies u^{\textcircled{m}} \cdot u^{\textcircled{k-m}} = u^{\textcircled{k}}$
<proof>

lemma *pop-pow-2*: $1 < k \implies u \cdot u \cdot u^{\textcircled{k-2}} = u^{\textcircled{k}}$
<proof>

lemma *pop-pow-cancel*: $u^{\textcircled{k}} \cdot v = u^{\textcircled{m}} \cdot w \implies m \leq k \implies u^{\textcircled{k-m}} \cdot v = w$
<proof>

lemma *pows-comm*: $t^{\textcircled{k}} \cdot t^{\textcircled{m}} = t^{\textcircled{m}} \cdot t^{\textcircled{k}}$
<proof>

lemma *comm-pows-comm*: **assumes** $r \cdot u = u \cdot r$ **shows** $r^{\textcircled{m}} \cdot u^{\textcircled{k}} = u^{\textcircled{k}} \cdot r^{\textcircled{m}}$
<proof>

lemma *pows-comp*: $x^{\textcircled{i}} \bowtie x^{\textcircled{j}}$
<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-list-eq-if*[*reversed*] **and**
pop-pow-list-one' = *pop-pos*[*reversed*] **and**
pop-pow' = *pop-pow*[*reversed*] **and**
pop-pow-cancel' = *pop-pow-cancel*[*reversed*]

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

lemma *pow-nemp-pos-nemp*[*intro*]: $w = u^{\textcircled{k}} \implies u \neq \varepsilon \implies 0 < k \implies w \neq \varepsilon$
<proof>

lemma *nemp-emp-pow*: **assumes** $u \neq \varepsilon$ **shows** $u^{\textcircled{m}} = \varepsilon \iff m = 0$
<proof>

lemma *nemp-pow-nemp-pos-conv*: **assumes** $u \neq \varepsilon$ **shows** $u^{\textcircled{m}} \neq \varepsilon \iff 0 < m$
<proof>

lemma *nemp-Suc-pow-nemp*: $u \neq \varepsilon \implies u^{\textcircled{\text{Suc } k}} \neq \varepsilon$
<proof>

lemma *Suc-pow-eq-eq*[*elim*]: $u^{\textcircled{\text{Suc } k}} = v^{\textcircled{\text{Suc } k}} \implies u = v$

<proof>

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

named-theorems *concat-simps*

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

lemma *pow-sing-nemp [elim]*: $w = [a]^{\textcircled{a}} k \implies 0 < k \implies w \neq \varepsilon$
<proof>

lemma *nemp-pow-emp[intro]*: $u \neq \varepsilon \implies u^{\textcircled{a}} n = \varepsilon \implies n = 0$
<proof>

lemma *long-pow*: $r \neq \varepsilon \implies m \leq |r^{\textcircled{a}} m|$
<proof>

lemma *long-pow-exp'*: $r \neq \varepsilon \implies m < |r^{\textcircled{a}}(\text{Suc } m)|$
<proof>

lemma *long-pow-expE*: **assumes** $r \neq \varepsilon$ **obtains** n **where** $m \leq |r^{\textcircled{a}} \text{Suc } n|$
<proof>

lemma *pref-pow-ext*: $x \leq_p r^{\textcircled{a}} k \implies x \leq_p r^{\textcircled{a}} \text{Suc } k$
<proof>

lemma *pref-pow-ext'*: $u \leq_p r^{\textcircled{a}} k \implies u \leq_p r \cdot r^{\textcircled{a}} k$
<proof>

lemma *pref-pow-root-ext*: $x \leq_p r^{\textcircled{a}} k \implies r \cdot x \leq_p r^{\textcircled{a}} \text{Suc } k$
<proof>

lemma *pref-pow-root*: $u \leq_p r^{\textcircled{a}} k \implies u \leq_p r \cdot u$
<proof>

lemma *le-exps-pref [intro]*: $k \leq l \implies r^{\textcircled{a}} k \leq_p r^{\textcircled{a}} l$
<proof>

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

lemma *pref-exp-le*: **assumes** $u \neq \varepsilon$ $u^{\textcircled{a}} m \leq_p u^{\textcircled{a}} n$ **shows** $m \leq n$
<proof>

lemma *sing-exp-pref-iff*: **assumes** $a \neq b$

shows $[a]^{\textcircled{a}} i \leq_p [a]^{\textcircled{a}} k \cdot [b] \cdot w \longleftrightarrow i \leq k$
 ⟨proof⟩

lemmas

$\text{suf-pow-ext} = \text{pref-pow-ext}[\text{reversed}]$ **and**
 $\text{suf-pow-ext}' = \text{pref-pow-ext}'[\text{reversed}]$ **and**
 $\text{suf-pow-root-ext} = \text{pref-pow-root-ext}[\text{reversed}]$ **and**
 $\text{suf-prod-root} = \text{pref-pow-root}[\text{reversed}]$ **and**
 $\text{suf-exps-pow} = \text{le-exps-pref}[\text{reversed}]$ **and**
 $\text{suf-exp-le} = \text{pref-exp-le}[\text{reversed}]$ **and**
 $\text{sing-exp-suf-iff} = \text{sing-exp-pref-iff}[\text{reversed}]$

lemma *comm-common-pow-list-iff*: **assumes** $r \cdot u = u \cdot r$ **shows** $r^{\textcircled{a}} |u| = u^{\textcircled{a}} |r|$
 ⟨proof⟩

lemma *concat-morph-power*: $xs \in \text{lists } B \implies xs = ts^{\textcircled{a}} k \implies \text{concat } ts^{\textcircled{a}} k = \text{concat } xs$
 ⟨proof⟩

lemma *per-exp-pref*: $u \leq_p r \cdot u \implies u \leq_p r^{\textcircled{a}} k \cdot u$
 ⟨proof⟩

lemmas

$\text{per-exp-suf} = \text{per-exp-pref}[\text{reversed}]$

lemma *hd-sing-pow*: $k \neq 0 \implies \text{hd}([a]^{\textcircled{a}} k) = a$
 ⟨proof⟩

lemma *sing-pref-comp-mismatch*:

assumes $b \neq a$ **and** $c \neq a$ **and** $[a]^{\textcircled{a}} k \cdot [b] \bowtie [a]^{\textcircled{a}} l \cdot [c]$
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 $[a]^{\textcircled{a}} r \cdot [b] \cdot u \wedge_p [a]^{\textcircled{a}} s \cdot [c] \cdot v = [a]^{\textcircled{a}} (\min r s)$

⟨proof⟩

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

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

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

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

lemma *sing-pow-len* [*simp*]: $|[r]^{\textcircled{l}}| = l$
<proof>

lemma *take-sing-pow*: $k \leq l \implies \text{take } k ([r]^{\textcircled{l}}) = [r]^{\textcircled{k}}$
<proof>

lemma *concat-take-sing*: **assumes** $k \leq l$
shows $\text{concat } (\text{take } k ([r]^{\textcircled{l}})) = r^{\textcircled{k}}$
<proof>

lemma *sing-set-word*: $\text{set } w \subseteq \{a\} \implies w = [a]^{\textcircled{|w|}}$
<proof>

lemma *sing-set-concat*: $\text{set } w \subseteq \{a\} \implies \text{concat } w = a^{\textcircled{|w|}}$
<proof>

lemma *sing-set-word-hd*: $\text{set } w \subseteq \{a\} \implies w = [\text{hd } w]^{\textcircled{|w|}}$
<proof>

lemma *sing-set-wordE*[*elim*]: **assumes** $\text{set } w \subseteq \{a\}$ **obtains** k **where** $w = [a]^{\textcircled{k}}$
<proof>

lemma *sing-set-wordE'*[*elim*]: **assumes** $\text{set } w = \{a\}$ **obtains** k **where** $w = [a]^{\textcircled{k}}$
and $0 < k$
<proof>

lemma *conjug-pow*: $x \cdot z = z \cdot y \implies x^{\textcircled{k}} \cdot z = z \cdot y^{\textcircled{k}}$
<proof>

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

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

lemma *pow-pref-root-one*: **assumes** $0 < k$ **and** $r \neq \varepsilon$ **and** $r^{\textcircled{k}} \leq p$ r
shows $k = 1$
<proof>

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

lemma *comp-pows-pref'*: **assumes** $v \neq \varepsilon$ **and** $(u \cdot v)^{\textcircled{k}} \leq p$ $(u \cdot v)^{\textcircled{m}} \cdot u$ **shows**
 $k \leq m$
<proof>

lemma *comp-pows-not-pref*: $\neg (u \cdot v)^{\textcircled{a}}k \cdot u \leq_p (u \cdot v)^{\textcircled{a}}m \implies m \leq k$
 ⟨proof⟩

lemma *comp-pows-spref*: $u^{\textcircled{a}}k <_p u^{\textcircled{a}}m \implies k < m$
 ⟨proof⟩

lemma *comp-pows-spref-ext*: $(u \cdot v)^{\textcircled{a}}k \cdot u <_p (u \cdot v)^{\textcircled{a}}m \implies k < m$
 ⟨proof⟩

lemma *comp-pows-pref-zero*: $(u \cdot v)^{\textcircled{a}}k <_p u \implies k = 0$
 ⟨proof⟩

lemma *comp-pows-spref'*: $(u \cdot v)^{\textcircled{a}}k <_p (u \cdot v)^{\textcircled{a}}m \cdot u \implies k < \text{Suc } m$
 ⟨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*]: $(u \cdot v)^{\textcircled{a}}k \cdot u = u \cdot (v \cdot u)^{\textcircled{a}}k$
 ⟨proof⟩

lemma[*shifts*]: $(u \cdot v)^{\textcircled{a}}k \cdot u \cdot z = u \cdot (v \cdot u)^{\textcircled{a}}k \cdot z$
 ⟨proof⟩

lemma[*shifts*]: $u^{\textcircled{a}}k \cdot u \cdot z = u \cdot u^{\textcircled{a}}k \cdot z$
 ⟨proof⟩

lemma[*shifts*]: $r^{\textcircled{a}}k \leq_p r \cdot r^{\textcircled{a}}k$
 ⟨proof⟩

lemma [*shifts*]: $r^{\textcircled{a}}k \leq_p r \cdot r^{\textcircled{a}}k \cdot z$
 ⟨proof⟩

lemma [*shifts*]: $(r \cdot q)^{\textcircled{a}}k \leq_p r \cdot q \cdot (r \cdot q)^{\textcircled{a}}k \cdot z$
 ⟨proof⟩

lemma [*shifts*]: $(r \cdot q)^{\textcircled{a}}k \leq_p r \cdot q \cdot (r \cdot q)^{\textcircled{a}}k$
 ⟨proof⟩

lemma[*shifts*]: $r^{\textcircled{a}}k \cdot u \leq_p r \cdot r^{\textcircled{a}}k \cdot v \longleftrightarrow u \leq_p r \cdot v$
 ⟨proof⟩

lemma[*shifts*]: $u \cdot u^{\textcircled{a}}k \cdot z = u^{\textcircled{a}}k \cdot w \longleftrightarrow u \cdot z = w$
 ⟨proof⟩

lemma[*shifts*]: $(r \cdot q)^{\textcircled{a}}k \cdot u \leq_p r \cdot q \cdot (r \cdot q)^{\textcircled{a}}k \cdot v \longleftrightarrow u \leq_p r \cdot q \cdot v$
 ⟨proof⟩

lemma[*shifts*]: $(r \cdot q)^{\textcircled{a}}k \cdot u = r \cdot q \cdot (r \cdot q)^{\textcircled{a}}k \cdot v \longleftrightarrow u = r \cdot q \cdot v$
 ⟨proof⟩

lemma_[shifts]: $r \cdot q \cdot (r \cdot q)^{\textcircled{k}} \cdot v = (r \cdot q)^{\textcircled{k}} \cdot u \longleftrightarrow r \cdot q \cdot v = u$
 ⟨proof⟩
lemma *shifts-spec* _[shifts]: $(u^{\textcircled{k}} \cdot v)^{\textcircled{l}} \cdot u \cdot u^{\textcircled{k}} \cdot z = u^{\textcircled{k}} \cdot (v \cdot u^{\textcircled{k}})^{\textcircled{l}} \cdot u \cdot z$
 ⟨proof⟩
lemmas _[shifts] = *shifts-spec*[of - - $r \cdot q$, *unfolded rassoc*] **for** $r \ q$
lemmas _[shifts] = *shifts-spec*[of - - $r \cdot q - \varepsilon$, *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 - - $r \cdot q \ r \cdot q \ \varepsilon$, *unfolded rassoc emp-simps*] **for** $r \ q$
lemma_[shifts]: $(u \cdot (v \cdot u)^{\textcircled{k}})^{\textcircled{j}} \cdot (u \cdot v)^{\textcircled{k}} = (u \cdot v)^{\textcircled{k}} \cdot (u \cdot (u \cdot v)^{\textcircled{k}})^{\textcircled{j}}$
 ⟨proof⟩
lemma_[shifts]: $(u \cdot (v \cdot u)^{\textcircled{k}} \cdot z)^{\textcircled{j}} \cdot (u \cdot v)^{\textcircled{k}} = (u \cdot v)^{\textcircled{k}} \cdot (u \cdot z \cdot (u \cdot v)^{\textcircled{k}})^{\textcircled{j}}$
 ⟨proof⟩
lemmas_[shifts] = *pow-comm cancel rassoc pow-Suc pref-cancel-conv suf-cancel-conv pow-add cancel-right numeral-nat pow-zero emp-simps*
lemmas_[shifts] = *less-eq-Suc-le*
lemmas_[shifts] = *neq0-conv*
lemma *shifts-hd-hd* _[shifts]: $a \# b \# v = [a] \cdot b \# v$
 ⟨proof⟩
lemmas _[shifts] = *shifts-hd-hd*[of - - ε]
lemma_[shifts]: $n \leq k \implies x^{\textcircled{k}} = x^{\textcircled{n}}(n + (k - n))$
 ⟨proof⟩
lemma_[shifts]: $n < k \implies x^{\textcircled{k}} = x^{\textcircled{n}}(n + (k - n))$
 ⟨proof⟩
lemmas_[shifts] = *cancel cancel-right pref-cancel-conv suf-cancel-conv triv-pref*
lemmas_[shifts] = *pow-diff*
lemmas_[shifts] = *pows-comm*
lemma_[shifts]: $m < l \implies x^{\textcircled{m}} \cdot w \leq_p x^{\textcircled{l}} \cdot w' \longleftrightarrow w \leq_p x^{\textcircled{l-m}} \cdot w'$
 ⟨proof⟩
lemma_[shifts]: $x \cdot x^{\textcircled{m}} \cdot w \leq_p x^{\textcircled{m}} \cdot w' \longleftrightarrow x \cdot w \leq_p w'$
 ⟨proof⟩

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

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

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

lemma *take-root*: $0 < k \implies \text{take } |ws| \ (ws^{\textcircled{k}}) = ws$
 ⟨proof⟩

2.13 Rotation

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

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

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

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

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

lemma *rotate-back*: $\text{rotate } (|u| - n \bmod |u|) (\text{rotate } n u) = u$
 ⟨proof⟩

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

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

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

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

lemma *rotate-comp-eq*: $w \bowtie \text{rotate } n w \implies \text{rotate } n w = w$
 ⟨proof⟩

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

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

lemma *pref-mod-list'*: **assumes** $u <_p \text{concat } ws$

obtains $j\ r$ **where** $j < |ws|$ **and** $r <_p ws!$ j **and** $\text{concat } (\text{take } j\ ws) \cdot r = u$
 ⟨*proof*⟩

lemma *pref-mod-list*: **assumes** $u \leq_p \text{concat } ws$ $u \neq \varepsilon$
obtains $ps\ p\ s$ **where** $ps \leq_p ws$ **and** $p \neq \varepsilon$ **and** $p \cdot s = \text{last } ps$
 $\text{concat } ps = u \cdot s$ **and** $\bigwedge y. y \leq_p ws \implies u \leq_p \text{concat } y \implies ps \leq_p y$
 ⟨*proof*⟩

lemma *pref-mod-pow*: **assumes** $u \leq_p w^{\textcircled{a}}l$ **and** $w \neq \varepsilon$
obtains $k\ z$ **where** $k \leq l$ **and** $z <_p w$ **and** $w^{\textcircled{a}}k \cdot z = u$
 ⟨*proof*⟩

lemma *pref-mod-pow'*: **assumes** $u <_p w^{\textcircled{a}}l$
obtains $k\ z$ **where** $k < l$ **and** $z <_p w$ **and** $w^{\textcircled{a}}k \cdot z = u$
 ⟨*proof*⟩

lemma *split-pow*: **assumes** $u \cdot v = w^{\textcircled{a}}k$ $0 < k$ $v \neq \varepsilon$
obtains $p\ s\ i\ j$ **where** $w = p \cdot s$ $s \neq \varepsilon$ $u = (p \cdot s)^{\textcircled{a}}i$ $p \cdot v = (s \cdot p)^{\textcircled{a}}j$ $s \cdot k = i + j + 1$
 ⟨*proof*⟩

lemma *del-emp-concat* [*simp*]: $\text{concat } (\text{filter } (\lambda x. x \neq \varepsilon) us) = \text{concat } us$
 ⟨*proof*⟩

lemma *lists-minus*: $us \in \text{lists } (C - A) \implies us \in \text{lists } C$
 ⟨*proof*⟩

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

lemma *pref-concat-pref*[*intro*]: $us \leq_p ws \implies \text{concat } us \leq_p \text{concat } ws$
 ⟨*proof*⟩

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

lemma *concat-mono-fac*: $us \leq_f ws \implies \text{concat } us \leq_f \text{concat } ws$
 ⟨*proof*⟩

lemma *ruler-concat-less*: **assumes** $us \leq_p ws$ **and** $vs \leq_p ws$ **and** $|\text{concat } us| < |\text{concat } vs|$
shows $us <_p vs$
 ⟨*proof*⟩

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

lemma *take-pp-less*: **assumes** $\text{take } k\ ws <_p \text{take } n\ ws$ **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*: $j \leq k \implies \text{take } j (\text{take } k \text{ xs}) = \text{take } j \text{ xs}$
 ⟨*proof*⟩

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

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

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

2.15 Commutation

The solution of the easiest nontrivial word equation, $x \cdot y = y \cdot x$, is contained
 AFP theory *List-Power.List-Power* as $(u \cdot v = v \cdot u) = (\exists r k m. u = r^{\textcircled{a}} k \wedge v = r^{\textcircled{a}} m)$.

lemmas *comm = comm-append-pow-list-iff*

lemma *commE[elim]*: **assumes** $x \cdot y = y \cdot x$
obtains $t m k$ **where** $x = t^{\textcircled{a}} k$ **and** $y = t^{\textcircled{a}} m$ **and** $t \neq \varepsilon$
 ⟨*proof*⟩

lemma *comm-nemp-egE*: **assumes** $u \cdot v = v \cdot u$ $u \neq \varepsilon$ $v \neq \varepsilon$
obtains $k m$ **where** $u^{\textcircled{a}} k = v^{\textcircled{a}} m$ $0 < k$ $0 < m$
 ⟨*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 *comm-cancel-pref*: **assumes** $r \cdot u = u \cdot r$ $r \cdot u \cdot v = u \cdot v \cdot r$
shows $r \cdot v = v \cdot r$
 ⟨*proof*⟩

lemma *comm-cancel-suf*: **assumes** $r \cdot v = v \cdot r$ $r \cdot u \cdot v = u \cdot v \cdot r$
shows $r \cdot u = u \cdot r$
 ⟨*proof*⟩

lemma *LS-comm*:
assumes $y^{\textcircled{a}} k \cdot x = z^{\textcircled{a}} l$
and $z \cdot y = y \cdot z$

shows $x \cdot y = y \cdot x$
<proof>

2.16 Periods

periodicity is probably the most studied property of words. It captures the fact that a word overlaps with itself. Another possible point of view is that the periodic word is a prefix of an (infinite) power of some nonempty word, which can be called its period word. Both these points of view are expressed by the following definition.

2.16.1 periodic root

lemma $u <_p r \cdot u \iff u \leq_p r \cdot u \wedge r \neq \varepsilon$
<proof>

lemma *per-rootI*[intro]: $u \leq_p r \cdot u \implies r \neq \varepsilon \implies u <_p r \cdot u$
<proof>

lemma *per-rootI'*[intro]: **assumes** $u \leq_p r^{\textcircled{k}}$ **and** $r \neq \varepsilon$ **shows** $u <_p r \cdot u$
<proof>

lemma *per-root-nemp*[dest]: $u <_p r \cdot u \implies r \neq \varepsilon$
<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 $w <_p r \cdot u \implies |r| \leq |w| \implies r \leq_p w$
<proof>

periodic words are prefixes of the power of the root, which motivates the notation

lemma *pref-pow-ext-take*: **assumes** $u \leq_p r^{\textcircled{k}}$ **shows** $u \leq_p \text{take } |r| \ u \cdot r^{\textcircled{k}}$
<proof>

lemma *pref-pow-take*: **assumes** $u \leq_p r^{\textcircled{k}}$ **shows** $u \leq_p \text{take } |r| \ u \cdot u$
<proof>

lemma *per-root-powE*: **assumes** $u \leq_p r \cdot u$ $r \neq \varepsilon$
obtains k **where** $u <_p r^{\textcircled{k}}$ **and** $0 < k$
<proof>

thm *per-rootI per-rootI'*

lemma *per-root-modE'* [elim]: **assumes** $u \leq_p r \cdot u$ $r \neq \varepsilon$
obtains p **where** $p <_p r$ **and** $r^{\textcircled{}}(|u| \text{ div } |r|) \cdot p = u$
⟨proof⟩

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

lemma *nemp-per-root-conv*: $r \neq \varepsilon \implies u <_p r \cdot u \iff u \leq_p r \cdot u$
⟨proof⟩

lemma *root-ruler*: **assumes** $w \leq_p u \cdot w$ $v \leq_p u \cdot v$ $u \neq \varepsilon$ $v \neq \varepsilon$
shows $w \bowtie v$
⟨proof⟩

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** $u <_p r^{\textcircled{}}m \cdot u$
⟨proof⟩

theorem *per-root-pow-conv*: $x <_p r \cdot x \iff (\exists k. x \leq_p r^{\textcircled{}}k) \wedge r \neq \varepsilon$
⟨proof⟩

lemma [intro]: $r = \varepsilon \implies u \leq_p r \implies u = \varepsilon$
⟨proof⟩

lemma [intro]: $u = \varepsilon \implies u \leq_p w$
⟨proof⟩

lemma *per-root-exp'*: **assumes** $x \leq_p r^{\textcircled{}}k$ **shows** $x \leq_p r^{\textcircled{}}|x|$
⟨proof⟩

lemma *per-root-exp*: **assumes** $x <_p r \cdot x$ **shows** $x \leq_p r^{\textcircled{}}|x|$
⟨proof⟩

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

lemma *per-root-exp-conv*: $u <_p (r^{\textcircled{}}\text{Suc } m) \cdot u \iff u <_p r \cdot u$
⟨proof⟩

lemma *per-root-sq-drop[simp]*: $x \leq_p y \cdot y \cdot x \implies x \leq_p y \cdot x$
⟨proof⟩

lemmas *per-root-sq-drop-suf* = *per-root-sq-drop*[reversed, unfolded rassoc]

lemma *per-drop-exp*: $0 < k \implies x \leq_p r^{\textcircled{}}k \cdot x \implies x \leq_p r \cdot x$

<proof>

lemmas *per-drop-exp-rev* = *per-drop-exp*[reversed]

corollary *comm-drop-exp*: **assumes** $0 < m$ **and** $u \cdot r^{\textcircled{m}} = r^{\textcircled{m}'} \cdot u$ **shows** $r \cdot u = u \cdot r$
<proof>

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

lemma *comm-drop-exps*: $0 < k \implies 0 < m \implies u^{\textcircled{k}} \cdot v^{\textcircled{m}} = v^{\textcircled{m}} \cdot u^{\textcircled{k}} \implies u \cdot v = v \cdot u$
<proof>

lemma *comm-pow-roots*:
assumes $0 < m$ **and** $0 < k$
shows $u^{\textcircled{m}} \cdot v^{\textcircled{k}} = v^{\textcircled{k}} \cdot u^{\textcircled{m}} \iff u \cdot v = v \cdot u$
<proof>

lemma *pow-comm-comm'*: **assumes** *comm*: $u^{\textcircled{k}}(Suc\ k) = v^{\textcircled{l}}(Suc\ l)$ **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 *drop-per-pref*: **assumes** $w \leq_p u \cdot w$ **shows** $drop\ |u|\ w \leq_p w$
<proof>

Note that $\llbracket w <_p u \cdot w; u <_p t \cdot u \rrbracket \implies w <_p t \cdot w$ does not hold.

lemma *per-root-same-prefix*: $w <_p r \cdot w \implies w' \leq_p r \cdot w' \implies w \bowtie w'$
<proof>

lemma *take-after-drop*: $|u| + q \leq |w| \implies w \leq_p u \cdot w \implies take\ q\ (drop\ |u|\ w) = take\ q\ w$
<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>

2.16.2 Maximal root-prefix

lemma *max-root-mismatch*: **assumes** $u \cdot [a] <_p r \cdot u \cdot [a]$ **and** $u \cdot [b] \leq_p w$ **and** $a \neq b$

shows $w \wedge_p r \cdot w = u$

<proof>

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 $u \wedge_p r \cdot u \leq_p r^{\textcircled{}} |u \wedge_p r \cdot u|$

<proof>

lemma *max-pref-lcp-root-pow*: **assumes** $r \neq \varepsilon$ **and** $|u \wedge_p r \cdot u| \leq k$

shows $u \wedge_p r \cdot u = u \wedge_p r^{\textcircled{}} k$ (**is** $?max = u \wedge_p r^{\textcircled{}} k$)

<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 $?u \wedge_p ?r \cdot ?u$

2.16.3 Period - numeric

Definition of a period as the length of the periodic root is often offered as the basic one. From our point of view, it is secondary, and less convenient for reasoning.

definition *period* :: 'a list \Rightarrow nat \Rightarrow bool

where [*simp*]: *period* w $n \equiv w \leq_p (\text{take } n \ w) \cdot w$

lemma *periodI*: $w \leq_p (\text{take } n \ w) \cdot w \implies \text{period } w \ n$

<proof>

lemma *periodI-pref*: **assumes** $|r| = n \ w \leq_p r \cdot w$

shows *period* $w \ n$

<proof>

lemma *periodI-pref'*: **assumes** $w \leq_p r \cdot w$

shows *period* $w \ |r|$

<proof>

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

lemma *emp-all-periods*: *period* ε n
<proof>

lemma *zero-period*: *period* w 0
<proof>

lemma *periodD*: *period* w $n \implies w \leq_p \text{take } n \ w \cdot w$
<proof>

A nonempty word has all "long" periods

lemma *all-long-pers*: $|w| \leq n \implies \text{period } w \ n$
<proof>

lemma *len-is-per*: *period* w $|w|$
<proof>

Period is preserved by reversal and taking factors

lemma *period-rev*: **assumes** *period* w p **shows** *period* $(\text{rev } w)$ p
<proof>

lemma *period-rev-conv* [*reversal-rule*]: *period* $(\text{rev } w)$ $n \longleftrightarrow \text{period } w \ n$
<proof>

lemma *period-pref-period*: **assumes** *period* w n $v \leq_p w$
shows *period* v n
<proof>

lemmas *period-suf-period* = *period-pref-period*[*reversed*] **and**
periodI-suf = *periodI-pref*[*reversed*] **and**
periodI-suf' = *periodI-pref'*[*reversed*]

lemma *period-fac-period*: **assumes** *period* $(u \cdot w \cdot v)$ p
shows *period* w p
<proof>

lemma *period-fac-period'*: *period* v $p \implies u \leq_f v \implies \text{period } u \ p$
<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 *period-mod*: *period* w $n \implies i < |w| \implies w!i = w!(i \bmod n)$
<proof>

lemma *indeces-period*:

assumes forall: $\bigwedge i. i + n < |w| \implies w!i = w!(i+n)$
shows period $w\ n$
 ⟨proof⟩

Alternative proof of $\text{period } ?w\ ?p \implies \text{period } (\text{rev } ?w)\ ?p$ using indeces

lemma assumes period $w\ p$ **shows** period $(\text{rev } w)\ p$
 ⟨proof⟩

lemma pow-per[intro]: period $(y^{\textcircled{k}})\ |y|$
 ⟨proof⟩

lemma per-fac: **assumes** $w \leq_f y^{\textcircled{k}}$ **shows** period $w\ |y|$
 ⟨proof⟩

lemma per-drop [intro]: **assumes** period $w\ n$ **shows** drop $n\ w \leq_p w$
 ⟨proof⟩

Two more characterizations of a period

theorem per-shift:
shows period $w\ n \longleftrightarrow \text{drop } n\ w \leq_p w$
 ⟨proof⟩

lemma rotate-per-root: **assumes** $w = \text{rotate } n\ w$
shows period $w\ n$
 ⟨proof⟩

Various lemmas on periods

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

lemma ext-per-left-power: period $w\ p \implies p \leq |w| \implies \text{period } ((\text{take } p\ w)^{\textcircled{k}} \cdot w)\ p$
 ⟨proof⟩

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

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

lemma per-mult: **assumes** period $w\ n$ **shows** period $w\ (m*n)$

<proof>

theorem *two-periods:*

assumes *period w p period w q p + q ≤ |w|*

shows *period w (gcd p q)*

<proof>

lemma *index-mod-per-root:* **assumes** *r ≠ ε and i: ∀ i < |w|. w!i = r!(i mod |r|)*

shows *w <ₚ r · w*

<proof>

lemma *index-pref-pow-mod:* *w ≤ₚ r^{@k} ⇒ i < |w| ⇒ w!i = r!(i mod |r|)*

<proof>

lemma *index-per-root-mod:* **assumes** *w ≤ₚ r · w r ≠ ε i < |w|*

shows *w!i = r!(i mod |r|)*

<proof>

lemma *prepend-periods:* **assumes** *p dvd n n ≤ |w| period w p*

shows *period ((take n w) · w) p*

<proof>

2.16.4 Period: overview

notepad

begin

<proof>

end

2.16.5 Singleton and its power

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

<proof>

lemma *sing-pow-hd-tl:* *set (c # w) ⊆ {a} ↔ c = a ∧ set w ⊆ {a}*

<proof>

lemma *pref-sing-pow:* **assumes** *w ≤ₚ [a]^{@m}* **shows** *w = [a]^{@|w|}*

<proof>

lemmas *suf-sing-pow = pref-sing-pow[reversed]*

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

<proof>

lemma *sing-pow-palindrom'[simp]:* *rev [a]^{@k} = [a]^{@k}*

<proof>

lemma *sing-fac-pow:* **assumes** *set w ⊆ {a}* **and** *v ≤ₚ w* **shows** *set v ⊆ {a}*

<proof>

lemma *sing-pow-fac*: **assumes** $a \neq b$ **and** $set\ w \subseteq \{a\}$ **shows** $\neg ([b] \leq^f w)$
<proof>

lemma *all-set-sing-pow*: $(\forall b. b \in set\ w \longrightarrow b = a) \longleftrightarrow set\ w \subseteq \{a\}$
<proof>

lemma *sing-fac-set*: $[a] \leq^f x \Longrightarrow a \in set\ x$
<proof>

lemma *set-sing-pow-hd* [simp]: **assumes** $0 < k$ **shows** $a \in set\ ([a]^{\textcircled{a}} k)$
<proof>

lemma *neq-set-not-root*: $a \neq b \Longrightarrow b \in set\ w \Longrightarrow \neg set\ w \subseteq \{a\}$
<proof>

lemma *sing-pow-set-Suc*[simp]: $set\ ([a]^{\textcircled{a}} Suc\ k) = \{a\}$
<proof>

lemma *per-one*: $set\ w \subseteq \{a\} \longleftrightarrow w <_p [a] \cdot w$
<proof>

lemma *per-one'*: **assumes** $set\ r \subseteq \{a\}$ $w <_p r \cdot w$
shows $set\ w \subseteq \{a\}$
<proof>

thm *pref-sing-pow*

lemma *pref-sing-pow'*: $w \leq_p [a]^{\textcircled{a}} m \Longrightarrow set\ w \subseteq \{a\}$
<proof>

lemma *sing-pow-set'*: $set\ ([a]^{\textcircled{a}} k) \subseteq \{a\}$
<proof>

lemma *sing-pow-set-sub* [elim]: $x = [a]^{\textcircled{a}} k \Longrightarrow set\ x \subseteq \{a\}$
<proof>

lemma *set-sing-nemp-eq* [intro]: $set\ w \subseteq \{a\} \Longrightarrow w \neq \varepsilon \Longrightarrow set\ w = \{a\}$
<proof>

lemma *sing-pow-set-pos-eq* [intro]: $w = [a]^{\textcircled{a}} k \Longrightarrow 0 < k \Longrightarrow set\ w = \{a\}$
<proof>

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

lemma *neq-in-set-not-pow*: **assumes** $a \neq b$ $b \in set\ x$ **shows** $x \neq [a]^{\textcircled{a}} k$

<proof>

lemma *sing-pow-card-set-Suc*: **assumes** $c = [a]^{\textcircled{a}} \text{Suc } k$ **shows** $\text{card}(\text{set } c) = 1$
<proof>

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

lemma *takeWhile-eq-set-conv[simp]*: $\text{takeWhile}(\lambda x. x = a) w = w \longleftrightarrow \text{set } w \subseteq \{a\}$
<proof>

lemma *dropWhile-distinct*: **assumes** $\neg \text{set } w \subseteq \{a\}$
shows $\text{takeWhile}(\lambda x. x = a) w \cdot [\text{hd}(\text{dropWhile}(\lambda x. x = a) w)] \leq_p w$
<proof>

lemma *takeWhile-subset*: $\text{set}(\text{takeWhile}(\lambda x. x = a) w) \subseteq \{a\}$
<proof>

lemma *takeWhile-sing-pow*: $\text{takeWhile}(\lambda x. x = a) w = w \longleftrightarrow w = [a]^{\textcircled{a}} |w|$
<proof>

lemma *letter-pref-exp-pref*: $[a]^{\textcircled{a}} |\text{takeWhile}(\lambda x. x = a) u| \cdot \text{dropWhile}(\lambda x. x = a) u = u$
<proof>

lemma *letter-pref-exp-mismatch*: $u = [a]^{\textcircled{a}} |\text{takeWhile}(\lambda x. x = a) u| \cdot v \implies v \neq \varepsilon \implies \text{hd } v \neq a$
<proof>

lemma *dropWhile-sing-pow*: $\text{dropWhile}(\lambda x. x = a) w = \varepsilon \longleftrightarrow w = [a]^{\textcircled{a}} |w|$
<proof>

lemma *sing-exp-len*: $u = [a]^{\textcircled{a}} m \implies u = [a]^{\textcircled{a}} |u|$
<proof>

lemma *nemp-takeWhile-hd*: $us \neq \varepsilon \implies \text{hd}(\text{takeWhile}(\lambda a. a = \text{hd } us) us) = \text{hd } us$
<proof>

lemma $w = [a]^{\textcircled{a}} |w| \longleftrightarrow \text{set } w \subseteq \{a\}$
<proof>

lemma *distinct-letter-in*: **assumes** $\neg \text{set } w \subseteq \{a\}$
obtains $m b q$ **where** $[a]^{\textcircled{a}} m \cdot [b] \cdot q = w$ **and** $b \neq a$
<proof>

lemma *distinct-letter-in-hd*: **assumes** $\neg \text{set } w \subseteq \{\text{hd } w\}$

obtains $m\ b\ q$ **where** $[hd\ w]^{\textcircled{m}} \cdot [b] \cdot q = w$ **and** $b \neq hd\ w$ **and** $m \neq 0$
 ⟨proof⟩

lemma *distinct-letter-in-hd'*: **assumes** $\neg\ set\ w \subseteq \{hd\ w\}$
obtains $m\ b\ q$ **where** $[hd\ w]^{\textcircled{m}} \cdot [b] \cdot q = w$ **and** $b \neq hd\ w$
 ⟨proof⟩

lemma *distinct-letter-in-suf*: **assumes** $\neg\ set\ w \subseteq \{a\}$
obtains $m\ b$ **where** $[b] \cdot [a]^{\textcircled{m}} \leq_s w$ **and** $b \neq a$
 ⟨proof⟩

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

lemma *sing-pow-exp*: $set\ w \subseteq \{a\} \longleftrightarrow w = [a]^{\textcircled{|w|}}$
 ⟨proof⟩

lemma *sing-power'*: **assumes** $set\ w \subseteq \{a\}$ **and** $i < |w|$ **shows** $w ! i = a$
 ⟨proof⟩

lemma *sing-set-palindrom*: $set\ w \subseteq \{a\} \implies rev\ w = w$
 ⟨proof⟩

lemma *lcp-letter-power*:
assumes $w \neq \varepsilon$ **and** $set\ w \subseteq \{a\}$ **and** $[a]^{\textcircled{m}} \cdot [b] \leq_p z$ **and** $a \neq b$
shows $w \cdot z \wedge_p z \cdot w = [a]^{\textcircled{m}}$
 ⟨proof⟩

2.17 Border

A non-empty word $x \neq w$ is a *border* of a word w if it is both its prefix and suffix. This elementary property captures how much the word w overlaps with itself, and it is in the obvious way related to a period of w . However, in many cases it is much easier to reason about borders than about periods.

definition *border* :: 'a list \Rightarrow 'a list \Rightarrow bool ($- \leq b - [51,51]$ 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 \Rightarrow bool
where [simp]: $bordered\ w = (\exists b. b \leq b\ w)$

lemma *borderI[intro]*: $x \leq_p w \implies x \leq_s w \implies x \neq w \implies x \neq \varepsilon \implies x \leq b\ w$
 ⟨proof⟩

lemma *borderD-pref*: $x \leq b\ w \implies x \leq_p w$
 ⟨proof⟩

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

lemma *border-sprefE* [*elim*]: **assumes** $b \leq b$ **obtains** r **where** $b \cdot r = w$ **and** $r \neq \varepsilon$
<proof>

lemma *borderD-suf*: $x \leq b$ $w \implies x \leq_s w$
<proof>

lemma *borderD-ssuf*: $x \leq b$ $w \implies x <_s w$
<proof>

lemma *borderD-nemp*: $x \leq b$ $w \implies x \neq \varepsilon$
<proof>

lemma *border-ssufE* [*elim*]: **assumes** $b \leq b$ **obtains** r **where** $r \cdot b = w$ **and** $r \neq \varepsilon$
<proof>

lemma *borderD-neq*: $x \leq b$ $w \implies x \neq w$
<proof>

lemma *borderedI*: $u \leq b$ $w \implies$ *bordered* w
<proof>

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

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

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

lemma *border-rev-conv*[*reversal-rule*]: $\text{rev } x \leq b$ $\text{rev } w \iff x \leq b$ w
<proof>

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

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

2.17.1 The shortest border

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

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*:
 $\text{bordered } w \implies \exists x. x \leq_b w \wedge \neg \text{bordered } x$
<proof>

lemma *unbordered-border-shortest*: $x \leq_b w \implies \neg \text{bordered } x \implies y \leq_b w \implies |x| \leq |y|$
<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*: $\neg \text{bordered } \varepsilon$
<proof>

lemma *bordered-nemp*: $\text{bordered } w \implies w \neq \varepsilon$
<proof>

lemma *sing-not-bordered*: $\neg \text{bordered } [a]$
<proof>

2.17.2 Relation to period and conjugation

lemma *border-conjug-eq*: $x \leq_b w \implies (w^{<-1}x) \cdot w = w \cdot (x^{-1} > w)$
<proof>

lemma *border-per-root*: $x \leq_b w \implies w \leq_p (w^{<-1}x) \cdot w$
<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*: $x \leq b w \implies \text{period } w (|w| - |x|)$
<proof>

lemma *per-border*: **assumes** $n < |w|$ **and** $n \neq 0$ **and** *period* $w n$
shows *take* $(|w| - n) w \leq b w$
<proof>

2.18 The longest border and the shortest period

2.18.1 The longest border

definition *max-borderP* :: 'a list \Rightarrow 'a list \Rightarrow bool **where**
 $\text{max-borderP } u w = (u \leq p w \wedge u \leq s w \wedge (u = w \longrightarrow w = \varepsilon) \wedge (\forall v. v \leq b w \longrightarrow v \leq p u))$

lemma *max-borderP-emp-emp*: $\text{max-borderP } \varepsilon \varepsilon$
<proof>

lemma *max-borderP-exE*: **obtains** u **where** $\text{max-borderP } u w$
<proof>

lemma *max-borderP-of-nemp*: $\text{max-borderP } u \varepsilon \implies u = \varepsilon$
<proof>

lemma *max-borderP-D-neg*: $w \neq \varepsilon \implies \text{max-borderP } u w \implies u \neq w$
<proof>

lemma *max-borderP-D-pref*: $\text{max-borderP } u w \implies u \leq p w$
<proof>

lemma *max-borderP-D-suf*: $\text{max-borderP } u w \implies u \leq s w$
<proof>

lemma *max-borderP-D-max*: $\text{max-borderP } u w \implies v \leq b w \implies v \leq p u$
<proof>

lemma *max-borderP-D-max'*: $\text{max-borderP } u w \implies v \leq b w \implies v \leq s u$
<proof>

lemma *unbordered-max-border-emp*: $\neg \text{bordered } w \implies \text{max-borderP } u w \implies u = \varepsilon$
<proof>

lemma *bordered-max-border-nemp*: $\text{bordered } w \implies \text{max-borderP } u w \implies u \neq \varepsilon$
<proof>

lemma *max-borderP-border*: $\text{max-borderP } u w \implies u \neq \varepsilon \implies u \leq b w$
<proof>

lemma *max-borderP-rev*: $\text{max-borderP } (\text{rev } u) (\text{rev } w) \implies \text{max-borderP } u w$
<proof>

lemma *max-borderP-rev-conv*: $\text{max-borderP } (\text{rev } u) (\text{rev } w) \longleftrightarrow \text{max-borderP } u w$
<proof>

definition *max-border* :: 'a list \Rightarrow 'a list **where**
max-border $w = (\text{THE } u. (\text{max-borderP } u w))$

lemma *max-border-unique*: **assumes** $\text{max-borderP } u w \text{ max-borderP } v w$
shows $u = v$
<proof>

lemma *max-border-ex*: $\text{max-borderP } (\text{max-border } w) w$
<proof>

lemma *max-borderP-max-border*: $\text{max-borderP } u w \implies \text{max-border } w = u$
<proof>

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

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

theorem *max-border-border'*: $\text{max-border } w \neq \varepsilon \implies \text{max-border } w \leq b w$
<proof>

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

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

lemma *max-border-nemp-neg*: $w \neq \varepsilon \implies \text{max-border } w \neq w$
<proof>

lemma *max-borderI*: **assumes** $u \neq w$ **and** $u \leq p w$ **and** $u \leq s w$ **and** $\forall v. v \leq b w \implies v \leq p u$
shows $\text{max-border } w = u$
<proof>

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

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

theorem *max-border-max-suf*: **assumes** $u \leq_b w$ **shows** $u \leq_s \text{max-border } w$
<proof>

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

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

2.18.2 The shortest period

definition *min-period-root* :: 'a list \Rightarrow 'a list (π) **where**
min-period-root $w = \text{take } (\text{LEAST } n. 0 < n \wedge \text{period } w \ n) w$

definition *min-period* :: 'a list \Rightarrow nat **where**
min-period $w = |\pi w|$

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

lemma *min-per-zero*[simp]: *min-period* $\varepsilon = 0$
<proof>

lemma *min-per-per*: *period* w (*min-period* w) **and**
min-per-pos: $w \neq \varepsilon \implies 0 < \text{min-period } w$
<proof>

lemma $n \leq |w| \implies |\text{take } n w| = n$
<proof>

lemma $u \leq_p w \implies \text{take } |u| w = u$
<proof>

lemma *min-per-root-take-eq* : $\text{take } |\pi w| w = \pi w$
<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*: **assumes** $w \neq \varepsilon$ **shows** $w \leq_p (\pi w) \cdot w$
<proof>

lemma *min-per-root-per-root*: **assumes** $w \neq \varepsilon$ **shows** $w <_p (\pi w) \cdot w$
<proof>

lemma *min-per-pref*: $\pi w \leq_p w$
 ⟨proof⟩

lemma *min-per-nemp*: $w \neq \varepsilon \implies \pi w \neq \varepsilon$
 ⟨proof⟩

lemma *min-per-min*: **assumes** $w <_p r \cdot w$ **shows** $\pi w \leq_p r$
 ⟨proof⟩

lemma *lq-min-per-pref*: $\pi w^{-1} > w \leq_p w$
 ⟨proof⟩

lemma *max-bord-emp*: $\text{max-border } \varepsilon = \varepsilon$
 ⟨proof⟩

theorem *min-per-max-border*: $\pi w \cdot \text{max-border } w = w$
 ⟨proof⟩

lemma *min-per-len-diff*: $\text{min-period } w = |w| - |\text{max-border } w|$
 ⟨proof⟩

lemma *min-per-root-take* [code]: $\pi w = \text{take } (|w| - |\text{max-border } w|) w$
 ⟨proof⟩

2.19 Primitive words

If a word w is not a non-trivial power of some other word, we say it is primitive.

definition *primitive* :: 'a list \Rightarrow bool
where *primitive* $u = (\forall r k. r^{\textcircled{a}}k = u \implies k = 1)$

lemma *emp-not-prim[simp]*: $\neg \text{primitive } \varepsilon$
 ⟨proof⟩

lemma *primI[intro]*: $(\bigwedge r k. r^{\textcircled{a}}k = u \implies k = 1) \implies \text{primitive } u$
 ⟨proof⟩

lemma *prim-nemp*: $\text{primitive } u \implies u \neq \varepsilon$
 ⟨proof⟩

lemma *prim-exp-one*: $\text{primitive } u \implies r^{\textcircled{a}}k = u \implies k = 1$
 ⟨proof⟩

lemma *pow-nemp-imprim[intro]*: $2 \leq k \implies \neg \text{primitive } (u^{\textcircled{a}}k)$
 ⟨proof⟩

lemma *pow-not-prim*: $\neg \text{primitive } (u^{\textcircled{a}}\text{Suc}(\text{Suc } k))$

<proof>

lemma *pow-non-prim*: $k \neq 1 \implies \neg \text{primitive } (w^{\textcircled{k}})$
<proof>

lemma *prim-exp-eq*: $\text{primitive } u \implies r^{\textcircled{k}} = u \implies u = r$
<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-comm-exp[elim]*: **assumes** *primitive r and $u \cdot r = r \cdot u$*
obtains k **where** $r^{\textcircled{k}} = u$
<proof>

lemma *set-singleton-iff*: $\text{card } (\text{set } u) = 1 \iff \text{set } u = \{\text{hd } u\}$
<proof>

lemma *set-empty-iff*: $\text{card } (\text{set } u) = 0 \iff \text{set } u = \{\}$
<proof>

lemma *set-large-iff*: $1 < \text{card } (\text{set } u) \iff \neg \text{set } u \subseteq \{\text{hd } u\}$
<proof>

lemma *prim-card-set*: **assumes** *primitive u and $|u| \neq 1$* **shows** $1 < \text{card } (\text{set } u)$
<proof>

lemma *comm-not-prim*: **assumes** $u \neq \varepsilon$ $v \neq \varepsilon$ $u \cdot v = v \cdot u$ **shows** $\neg \text{primitive } (u \cdot v)$
<proof>

lemma *prim-rotate-conv*: $\text{primitive } w \iff \text{primitive } (\text{rotate } n \ w)$
<proof>

lemma *non-prim*: **assumes** $\neg \text{primitive } w$ **and** $w \neq \varepsilon$
obtains r k **where** $r \neq \varepsilon$ **and** $1 < k$ **and** $r^{\textcircled{k}} = w$ **and** $w \neq r$
<proof>

lemma *prim-no-rotate*: **assumes** *primitive w and $0 < n$ and $n < |w|$*
shows $\text{rotate } n \ w \neq w$
<proof>

lemma *no-rotate-prim*: **assumes** $w \neq \varepsilon$ **and** $\bigwedge n. 0 < n \implies n < |w| \implies \text{rotate } n \ w \neq w$
shows *primitive w*

<proof>

corollary *prim-iff-rotate*: **assumes** $w \neq \varepsilon$ **shows**

$\text{primitive } w \longleftrightarrow (\forall n. 0 < n \wedge n < |w| \longrightarrow \text{rotate } n \ w \neq w)$

<proof>

lemma *prim-sing*: *primitive* $[a]$

<proof>

lemma *sing-pow-conv* [*simp*]: $[u] = t^{\textcircled{a}}k \longleftrightarrow t = [u] \wedge k = 1$

<proof>

lemma *prim-rev-iff* [*reversal-rule*]: *primitive* $(\text{rev } u) \longleftrightarrow \text{primitive } u$

<proof>

lemma *prim-map-prim*: *primitive* $(\text{map } f \ ws) \implies \text{primitive } ws$

<proof>

lemma *inj-map-prim*: **assumes** *inj-on* $f \ A$ **and** $u \in \text{lists } A$ **and**
primitive u

shows *primitive* $(\text{map } f \ u)$

<proof>

lemma *prim-map-iff* [*reversal-rule*]:

assumes *inj* f **shows** *primitive* $(\text{map } f \ ws) = \text{primitive } (ws)$

<proof>

lemma *prim-concat-prim*: *primitive* $(\text{concat } ws) \implies \text{primitive } ws$

<proof>

lemma *eq-append-not-prim*: $x = y \implies \neg \text{primitive } (x \cdot y)$

<proof>

2.20 Primitive root

Given a non-empty word w which is not primitive, it is natural to look for the shortest u such that $w = u^k$. Such a word is primitive, and it is the primitive root of w .

definition *primitive-root* :: 'a list \Rightarrow 'a list (ϱ) **where**

primitive-root $x = (\text{if } x \neq \varepsilon \text{ then } (\text{THE } r. \text{primitive } r \wedge (\exists k. x = r^{\textcircled{a}}k)) \text{ else } \varepsilon)$

definition *primitive-root-exp* :: 'a list \Rightarrow nat (e_ϱ) **where**

primitive-root-exp $x = (\text{if } x = \varepsilon \text{ then } 0 \text{ else } (\text{THE } k. x = (\varrho \ x)^{\textcircled{a}}k))$

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** $\exists r k. \text{primitive } r \wedge k \neq 0 \wedge x = r^{\textcircled{a}}k$
<proof>

lemma *primroot-exE*: **assumes** $x \neq \varepsilon$
obtains $r k$ **where** *primitive* r **and** $0 < k$ **and** $x = r^{\textcircled{a}}k$
<proof>

Uniqueness of the primitive root follows from the following lemma

lemma *primroot-unique*: **assumes** $u \neq \varepsilon$ **and** *primitive* r **and** $u = r^{\textcircled{a}}k$ **shows** $\varrho u = r$
<proof>

lemma *primroot-unique'*: **assumes** $0 < k$ *primitive* r **and** $u = r^{\textcircled{a}}k$ **shows** $\varrho u = r$
<proof>

lemma *prim-primroot[intro]*: **assumes** *primitive* x **shows** $\varrho x = x$
<proof>

lemma *primroot-exp-unique*: **assumes** $u \neq \varepsilon$ **and** $(\varrho u)^{\textcircled{a}}k = u$ **shows** $e_{\varrho} u = k$
<proof>

lemma *primroot-prim[intro]*: $x \neq \varepsilon \implies \text{primitive } (\varrho x)$
<proof>

Existence and uniqueness of the primitive root justifies the function ϱ : it indeed yields the primitive root of a nonempty word.

lemma *primroot-expE*: **obtains** k **where** $(\varrho x)^{\textcircled{a}}k = x$ **and** $0 < k$
<proof>

lemma *primroot-exp-eq [simp]*: $(\varrho u)^{\textcircled{a}}(e_{\varrho} u) = u$
<proof>

lemma *prim-pow-exp-one*: *primitive* $(u^{\textcircled{a}}i) \implies i = 1$
<proof>

lemma *prim-exp-emp[simp]*: $e_{\varrho} \varepsilon = 0$
<proof>

lemma *prim-exp-zero-iff[simp]*: $e_{\varrho} x = 0 \longleftrightarrow x = \varepsilon$
<proof>

lemma *prim-exp-one-iff*: *primitive* $x \longleftrightarrow e_{\varrho} x = 1$
<proof>

lemma *nemp-not-prim-exp*: $x \neq \varepsilon \implies \neg \text{primitive } x \longleftrightarrow 1 < e_\rho x$
 ⟨proof⟩

lemma *not-prim-pref-primroots*: $\neg \text{primitive } x \longleftrightarrow \rho x \cdot \rho x \leq_p x$
 ⟨proof⟩

lemma *zero-prim-exp-eq*[intro]: $n = 0 \implies e_\rho(r^\textcircled{n}) = n$
 ⟨proof⟩

lemma *primroot-exp-len*:
 shows $e_\rho w * |\rho w| = |w|$
 ⟨proof⟩

lemma *primroot-exp-nemp* [intro]: $u \neq \varepsilon \implies 0 < e_\rho u$
 ⟨proof⟩

lemma *primroot-exp-zero* [intro]: $e_\rho u = 0 \implies u = \varepsilon$
 ⟨proof⟩

lemma *primroot-exp-zero-conv* [simp]: $e_\rho u = 0 \longleftrightarrow u = \varepsilon$
 ⟨proof⟩

lemma *pop-primroot*: $u = \rho u \cdot (\rho u)^\textcircled{(e_\rho u - 1)}$
 ⟨proof⟩

lemma *pop-primroot'*: $u = (\rho u)^\textcircled{(e_\rho u - 1)} \cdot \rho u$
 ⟨proof⟩

lemma *primroot-exp-minus-Suc-eq* [simp]: $\rho x^\textcircled{(e_\rho x - \text{Suc } 0)} \cdot \rho x = x$
 ⟨proof⟩

lemma *primroot-nemp*[intro!]: $x \neq \varepsilon \implies \rho x \neq \varepsilon$
 ⟨proof⟩

lemma *primroot-idemp*[simp]: $\rho(\rho x) = \rho x$
 ⟨proof⟩

lemma *prim-primroot-conv*: **assumes** $w \neq \varepsilon$ **shows** $\text{primitive } w \longleftrightarrow \rho w = w$
 ⟨proof⟩

lemma *not-prim-primroot-expE*: **assumes** $\neg \text{primitive } w$
obtains k **where** $\rho w^\textcircled{k} = w$ **and** $2 \leq k$
 ⟨proof⟩

lemma *not-prim-expE*: **assumes** $\neg \text{primitive } x$ **and** $x \neq \varepsilon$
obtains $r k$ **where** $\text{primitive } r$ **and** $2 \leq k$ **and** $r^\textcircled{k} = x$

<proof>

lemma *primroot-len-le*: $u \neq \varepsilon \implies |\varrho u| \leq |u|$
<proof>

lemma *primroot-pref*: $\varrho u \leq_p u$
<proof>

lemma *primroot-take*: **assumes** $u \neq \varepsilon$ **shows** $\varrho u = (\text{take } (|\varrho u|) u)$
<proof>

lemma *primroot-rotate-comm*: **assumes** $w \neq \varepsilon$ **shows** $\varrho (\text{rotate } n w) = \text{rotate } n (\varrho w)$
<proof>

lemma *primroot-rotate*: $\varrho w = r \iff \varrho (\text{rotate } (k*|r|) w) = r$ (**is** $?L \iff ?R$)
<proof>

lemma *primrootI[intro]*: **assumes** *pow*: $u = r^\circledast (\text{Suc } k)$ **and primitive** r **shows** $\varrho u = r$
<proof>

lemma *sing-primroot[simp]*: $\varrho [a] = [a]$
<proof>

lemma *short-primroot*: **assumes** $u \neq \varepsilon \wedge \text{primitive } u$ **shows** $|\varrho u| < |u|$
<proof>

lemma *prim-primroot-cases*: **obtains** $u = \varepsilon \mid \text{primitive } u \mid |\varrho u| < |u|$
<proof>

We also have the standard characterization of commutation for nonempty words.

theorem *comm-primroots*: **assumes** $u \neq \varepsilon$ **and** $v \neq \varepsilon$ **shows** $u \cdot v = v \cdot u \iff \varrho u = \varrho v$
<proof>

lemma *comm-primroots'*: $u \neq \varepsilon \implies v \neq \varepsilon \implies u \cdot v = v \cdot u \implies \varrho u = \varrho v$
<proof>

lemma *primroot-add-exp*: **assumes** $0 < l$ **shows** $\varrho (x^\circledast l) = \varrho x$
<proof>

lemma *same-primroots-comm*: $\varrho x = \varrho y \implies x \cdot y = y \cdot x$
<proof>

lemma *pow-primroot*: **assumes** $x \neq \varepsilon$ **shows** $\varrho (x^\circledast \text{Suc } k) = \varrho x$
<proof>

lemma *comm-primroot-exp*: **assumes** $v \neq \varepsilon$ **and** $u \cdot v = v \cdot u$
obtains n **where** $(\varrho v)^{\textcircled{n}} = u$
<proof>

lemma *primE*: **obtains** t **where** *primitive* t
<proof>

lemma *comm-primrootE'*: **assumes** $x \cdot y = y \cdot x$
obtains $t k m$ **where** $x = t^{\textcircled{k}}$ **and** $y = t^{\textcircled{m}}$ **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** $x = t^{\textcircled{k}}$ **and** $y = t^{\textcircled{m}}$ **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 *comm-primroot-conv'*: **shows** $u \cdot v = v \cdot u \longleftrightarrow \varrho u \cdot \varrho v = \varrho v \cdot \varrho u$
<proof>

lemma *comm-primrootI [intro]*: $\varrho u \cdot \varrho v = \varrho v \cdot \varrho u \implies u \cdot v = v \cdot u$
<proof>

lemma *per-root-primroot*: $w <_p r \cdot w \implies w <_p \varrho r \cdot w$
<proof>

lemma *per-root-primroot'*: $w <_p \varrho r \cdot w \implies w <_p r \cdot w$
<proof>

lemma *per-root-primroot-iff*: $w <_p \varrho r \cdot w \longleftrightarrow w <_p r \cdot w$
<proof>

lemma *per-root-primroot-iff'*: $x \leq_p \varrho u \cdot x \longleftrightarrow x \leq_p u \cdot x$
<proof>

lemma *primroot-per-root*: $r \neq \varepsilon \implies r <_p \varrho r \cdot r$
<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** $\varrho (\text{rev } u) = \text{rev } (\varrho u)$
<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** $p \cdot s = \rho r$ **and** $(p \cdot s)^{\textcircled{n}} \cdot p = u$ **and** $s \neq \varepsilon$

<proof>

lemmas[*shifts*] = *pows-comm*

lemma *per-root-shift*: **assumes** $x = (r \cdot s)^{\textcircled{k}} \cdot r$ $y = (r \cdot s)^{\textcircled{m}}$

shows $y \cdot x = x \cdot (s \cdot r)^{\textcircled{m}}$

<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>

2.20.1 Primitivity and the shortest period

lemma *min-per-primitive*: **assumes** $w \neq \varepsilon$ **shows** *primitive* (πw)

<proof>

lemma *min-per-short-primroot*: **assumes** $w \neq \varepsilon$ **and** $(\rho w)^{\textcircled{k}} = w$ **and** $k \neq 1$

shows $\pi w = \rho w$

<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>

2.21 Conjugation

Two words x and y are conjugated if one is a rotation of the other. Or, equivalently, there exists z such that

$$xz = zy.$$

definition *conjugate* (**infix** ~ 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*]: $\text{rev } u \sim \text{rev } v \longleftrightarrow u \sim v$
<proof>

lemma *conjug-rotate-iff*: $u \sim v \longleftrightarrow (\exists n. v = \text{rotate } n u)$
<proof>

lemma *rotate-conjug*: $w \sim \text{rotate } n w$
<proof>

lemma *conjug-rotate-iff-le*:
shows $u \sim v \longleftrightarrow (\exists n \leq |u| - 1. v = \text{rotate } n u)$
<proof>

lemma *conjugI* [*intro*]: $r \cdot s = u \implies s \cdot r = v \implies u \sim v$
<proof>

lemma *conjugI'* [*intro*]: $r \cdot s \sim s \cdot r$
<proof>

lemma *conjug-refl*: $u \sim u$
<proof>

lemma *conjug-sym*[*sym*]: $u \sim v \implies v \sim u$
<proof>

lemma *conjug-swap*: $u \sim v \longleftrightarrow v \sim u$
<proof>

lemma *conjug-nemp-iff*: $u \sim v \implies u = \varepsilon \longleftrightarrow v = \varepsilon$
<proof>

lemma *conjug-len*: $u \sim v \implies |u| = |v|$
<proof>

lemma *pow-conjug*:
assumes eq: $t^{\textcircled{i}} \cdot r \cdot u = t^{\textcircled{k}}$ and $t: r \cdot s = t$
shows $u \cdot t^{\textcircled{i}} \cdot r = (s \cdot r)^{\textcircled{k}}$

<proof>

lemma *conjug-set*: **assumes** $u \sim v$ **shows** $\text{set } u = \text{set } v$
<proof>

lemma *conjug-concat-conjug*: $xs \sim ys \implies \text{concat } xs \sim \text{concat } ys$
<proof>

The solution of the equation

$$xz = zy$$

is given by the next lemma.

lemma *conjug-eqE* [*elim*, *consumes 2*]:
assumes $\text{eq: } x \cdot z = z \cdot y$ **and** $x \neq \varepsilon$
obtains $u v k$ **where** $u \cdot v = x$ **and** $v \cdot u = y$ **and** $(u \cdot v)^{\textcircled{k}} \cdot u = z$ **and** $v \neq \varepsilon$
<proof>

theorem *conjugation*: **assumes** $x \cdot z = z \cdot y$ **and** $x \neq \varepsilon$
shows $\exists u v k. u \cdot v = x \wedge v \cdot u = y \wedge (u \cdot v)^{\textcircled{k}} \cdot u = z$
<proof>

lemma *conjug-eqE'* [*elim*]:
assumes $\text{eq: } x \cdot z = z \cdot y$
obtains $u v k i$ **where** $(u \cdot v)^{\textcircled{i}} = x$ **and** $(v \cdot u)^{\textcircled{i}} = y$ **and** $(u \cdot v)^{\textcircled{k}} \cdot u = z$
<proof>

lemma *conjug-eq-primrootE'* [*elim*, *consumes 2*]:
assumes $\text{eq: } x \cdot z = z \cdot y$ **and** $x \neq \varepsilon$
obtains $r s i n$ **where**
 $(r \cdot s)^{\textcircled{i}} = x$ **and**
 $(s \cdot r)^{\textcircled{i}} = y$ **and**
 $(r \cdot s)^{\textcircled{n}} \cdot r = z$ **and**
 $s \neq \varepsilon$ **and** $0 < i$ **and** *primitive* $(r \cdot s)$
<proof>

lemma *conjugI1* [*intro*]:
assumes $\text{eq: } u \cdot r = r \cdot v$
shows $u \sim v$
<proof>

lemma *pow-conjug-conjug-conv*: **assumes** $0 < k$ **shows** $u^{\textcircled{k}} \sim v^{\textcircled{k}} \iff u \sim v$
<proof>

lemma *conjug-trans* [*trans*]:
assumes $uv: u \sim v$ **and** $vw: v \sim w$
shows $u \sim w$
<proof>

lemma *conjug-trans'*: **assumes** $uv': u \cdot r = r \cdot v$ **and** $vw': v \cdot s = s \cdot w$ **shows** $u \cdot (r \cdot s) = (r \cdot s) \cdot w$

<proof>

lemma *root-conjugE*:

assumes $x \leq_p r \cdot x$

obtains $u v k i$ **where** $(u \cdot v)^{\textcircled{i}} = r$ **and** $(u \cdot v)^{\textcircled{k}} \cdot u = x$ **and** $(v \cdot u)^{\textcircled{i}} = x^{-1} \triangleright (r \cdot x)$

<proof>

lemmas *suf-root-conjugE* = *root-conjugE*[*reversed*]

Of course, conjugacy is an equivalence relation.

lemma *conjug-equiv*: *equivp* (\sim)

<proof>

lemma *append-split-mod-primroot*:

obtains $r s i j$ **where** $(r \cdot s)^{\textcircled{i}} \cdot r = x$ $(s \cdot r)^{\textcircled{j}} \cdot s = y$ $r \cdot s = \varrho(x \cdot y)$ $s \cdot r = x^{-1} \triangleright (\varrho(x \cdot y) \cdot x)$

<proof>

lemma *rotate-fac-pref*: **assumes** $u \leq_f w$

obtains w' **where** $w' \sim w$ **and** $u \leq_p w'$

<proof>

lemma *rotate-into-pos-sq*: **assumes** $s \cdot p \leq_f w \cdot w$ **and** $|s| \leq |w|$ **and** $|p| \leq |w|$

obtains w' **where** $w \sim w'$ $p \leq_p w'$ $s \leq_s w'$

<proof>

lemma *rotate-into-pref-sq*: **assumes** $p \leq_f w \cdot w$ **and** $|p| \leq |w|$

obtains w' **where** $w \sim w'$ $p \leq_p w'$

<proof>

lemmas *rotate-into-suf-sq* = *rotate-into-pref-sq*[*reversed*]

lemma *rotate-into-pos*: **assumes** $s \cdot p \leq_f w$

obtains w' **where** $w \sim w'$ $p \leq_p w'$ $s \leq_s w'$

<proof>

lemma *rotate-into-pos-conjug*: **assumes** $w \sim v$ **and** $s \cdot p \leq_f v$

obtains w' **where** $w \sim w'$ $p \leq_p w'$ $s \leq_s w'$

<proof>

lemma *nconjug-neg*: $\neg u \sim v \implies u \neq v$

<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* $u = \text{primitive } v$
 ⟨*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**

$(r \cdot s)^{\textcircled{i}} = x$ **and**

$(s \cdot r)^{\textcircled{i}} = y$ **and**

$(r \cdot s)^{\textcircled{n}} \cdot r = z$ **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** $p = (r \cdot s)^{\textcircled{k}}$ **and** $q = (s \cdot r)^{\textcircled{l}}$ **and** *primitive* $(r \cdot s)$

⟨*proof*⟩

lemma *root-conjug*: $u \leq_p r \cdot u \implies u^{-1} \triangleright (r \cdot u) \sim r$

⟨*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*: $u \sim v \implies u^{\textcircled{k}} \sim v^{\textcircled{k}}$

⟨*proof*⟩

lemma *conjug-primroot-iff*:

assumes *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**

$u = (s \cdot r)^{\textcircled{k}} \cdot s$ **and** $v = (r \cdot s)^{\textcircled{l}} \cdot r$ **and**

$x = (s \cdot r)^{\textcircled{m}} \cdot s$ **and** $y = (r \cdot s)^{\textcircled{n}} \cdot r$ **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**

$u = (s \cdot r)^{\textcircled{k}} \cdot s$ **and** $v = (r \cdot s)^{\textcircled{l}} \cdot r$ **and**

$x = (s \cdot r)^{\textcircled{m}} \cdot s$ **and** $y = (r \cdot s)^{\textcircled{n}} \cdot r$ **and**

primitive $(r \cdot s)$ **and** *primitive* $(s \cdot r)$

<proof>

lemma *fac-pow-pref-conjug*:

assumes $u \leq f t^{\textcircled{k}}$

obtains t' **where** $t \sim t'$ **and** $u \leq p t'^{\textcircled{k}}$

<proof>

lemmas *fac-pow-suf-conjug* = *fac-pow-pref-conjug*[*reversed*]

lemma *fac-pow-len-conjug*[*intro*]: **assumes** $|u| = |v|$ **and** $u \leq f v^{\textcircled{k}}$ **shows** $v \sim u$

<proof>

lemma *conjug-fac-sq*:

$u \sim v \implies u \leq f v \cdot v$

<proof>

lemma *conjug-fac-pow-conv*: **assumes** $|u| = |v|$ **and** $2 \leq k$

shows $u \sim v \longleftrightarrow u \leq f v^{\textcircled{k}}$

<proof>

lemma *conjug-fac-Suc*: **assumes** $t \sim v$

shows $t^{\textcircled{k}} \leq f v^{\textcircled{k}} \text{Suc } k$

<proof>

lemma *fac-pow-conjug*: **assumes** $u \leq f v^{\textcircled{k}}$ **and** $t \sim v$

shows $u \leq f t^{\textcircled{k}} \text{Suc } k$

<proof>

lemma *border-conjug*: $x \leq b w \implies w^{<-1} x \sim x^{-1} > w$

<proof>

lemma *count-list-conjug*: **assumes** $u \sim v$ **shows** $\text{count-list } u \ a = \text{count-list } v \ a$
 ⟨proof⟩

lemma *conjug-in-lists*: $us \sim vs \implies vs \in \text{lists } A \implies us \in \text{lists } A$
 ⟨proof⟩

lemma *conjug-in-lists'*: $us \sim vs \implies us \in \text{lists } A \implies vs \in \text{lists } A$
 ⟨proof⟩

lemma *conjug-in-lists-iff*: $us \sim vs \implies us \in \text{lists } A \longleftrightarrow vs \in \text{lists } A$
 ⟨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** $(u \cdot v)^{\textcircled{a}} k \cdot u = z$
 ⟨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** $(r \cdot s)^{\textcircled{a}} k \cdot r = z$
 ⟨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** $u \cdot r \cdot s \leq_p (s \cdot r)^{\textcircled{a}} n$
and $r \neq \varepsilon$
obtains n **where** $(s \cdot r)^{\textcircled{a}} n \cdot s = u$
 ⟨proof⟩

lemma *fac-per-conjug*: **assumes** *period* $w \ n$ **and** $v \leq_f w$ **and** $|v| = n$
shows $v \sim \text{take } n \ w$
 ⟨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** $w \sim r^{\textcircled{a}}k$ **obtains** s **where** $w = s^{\textcircled{a}}k$
 ⟨*proof*⟩

lemma *find-second-letter*: **assumes** $a \neq b$ **and** $\text{set } ws = \{a, b\}$
shows $\text{dropWhile } (\lambda c. c = a) \text{ } ws \neq \varepsilon$ **and** $\text{hd } (\text{dropWhile } (\lambda c. c = a) \text{ } ws) = b$
 ⟨*proof*⟩

lemma *fac-conjug-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-conjug-sq-iff*:
assumes $u \sim v$ **shows** $|w| \leq |u| \implies w \leq_f u \cdot u \longleftrightarrow w \leq_f v \cdot v$
 ⟨*proof*⟩

lemma *map-conjug*:
 $u \sim v \implies \text{map } f \text{ } u \sim \text{map } f \text{ } v$
 ⟨*proof*⟩

lemma *concat-map-conjug*:
 $u \sim v \implies \text{concat } (\text{map } f \text{ } u) \sim \text{concat } (\text{map } f \text{ } v)$
 ⟨*proof*⟩

lemma *map-conjug-iff [reversal-rule]*:
assumes $\text{inj } f$ **shows** $\text{map } f \text{ } u \sim \text{map } f \text{ } v \longleftrightarrow u \sim v$
 ⟨*proof*⟩

lemma *set-conjug*: **assumes** $w \neq \varepsilon$
shows $\{w'. w \sim w'\} = \{\text{rotate } n \text{ } w \mid n. n < |\varrho w|\}$
 ⟨*proof*⟩

lemma *card-conjug*: **assumes** $w \neq \varepsilon$
shows $\text{card } \{w'. w \sim w'\} = |\varrho w|$
 ⟨*proof*⟩

lemma *finite-Bex-conjug*: **assumes** *finite* A
shows *finite* $\{r. \text{Bex } A \text{ } (\text{conjugate } r)\}$
 ⟨*proof*⟩

2.21.1 Enumerating conjugates

definition *bounded-conjug*
where $\text{bounded-conjug } w' \text{ } w \text{ } k \equiv (\exists n \leq k. w = \text{rotate } n \text{ } w')$

named-theorems *bounded-conjug*

lemma[*bounded-conjug*]: $\text{bounded-conjug } w' \text{ } w \text{ } 0 \longleftrightarrow w = w'$
 ⟨*proof*⟩

lemma_[bounded-conjug]: *bounded-conjug* $w' w (Suc\ k) \longleftrightarrow$ *bounded-conjug* $w' w k \vee w = rotate\ (Suc\ k)\ w'$
 ⟨*proof*⟩

lemma_[bounded-conjug]: $w' \sim w \longleftrightarrow$ *bounded-conjug* $w w' (|w|-1)$
 ⟨*proof*⟩

lemma $w \sim [a,b,c] \longleftrightarrow w = [a,b,c] \vee w = [b,c,a] \vee w = [c,a,b]$
 ⟨*proof*⟩

2.21.2 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*:
 $\neg primitive\ (u \cdot v) \implies \neg primitive\ (u \cdot v \cdot v) \implies u \cdot v = v \cdot u$
 ⟨*proof*⟩

lemma *prim-xyky*: **assumes** $2 \leq k$ **and** $\neg primitive\ ((x \cdot y)^{\textcircled{k}} \cdot y)$ **shows** $x \cdot y = y \cdot x$
 ⟨*proof*⟩

lemma *fac-pow-div*: **assumes** $u \leq_f w^{\textcircled{l}}$ *primitive* w
shows $w^{\textcircled{(|u|\ div\ |w| - 1)}} \leq_f u$
 ⟨*proof*⟩

2.22 Element of lists: a method for testing if a word is in lists A

lemma *append-in-lists*_[simp, intro]: $u \in lists\ A \implies v \in lists\ A \implies u \cdot v \in lists\ A$
 ⟨*proof*⟩

lemma *pref-in-lists*: $u \leq_p v \implies v \in lists\ A \implies u \in lists\ A$
 ⟨*proof*⟩

lemmas *suf-in-lists* = *pref-in-lists*_[reversed]

lemma *fac-in-lists*: $ws \in lists\ S \implies vs \leq_f ws \implies vs \in lists\ S$
 ⟨*proof*⟩

lemma *lq-in-lists*: $v \in lists\ A \implies u^{-1} > v \in lists\ A$

<proof>

lemmas $rq\text{-in-lists} = lq\text{-in-lists}[\text{reversed}]$

lemma $take\text{-in-lists}$: $w \in \text{lists } A \implies take\ j\ w \in \text{lists } A$
<proof>

lemma $drop\text{-in-lists}$: $w \in \text{lists } A \implies drop\ j\ w \in \text{lists } A$
<proof>

lemma $lcp\text{-in-lists}$: $u \in \text{lists } A \implies u \wedge_p v \in \text{lists } A$
<proof>

lemma $lcp\text{-in-lists}'$: $v \in \text{lists } A \implies u \wedge_p v \in \text{lists } A$
<proof>

lemma $append\text{-in-lists-dest}[\text{elim}]$: $u \cdot v \in \text{lists } A \implies u \in \text{lists } A$
<proof>

lemma $append\text{-in-lists-dest}'$: $u \cdot v \in \text{lists } A \implies v \in \text{lists } A$
<proof>

lemma $pow\text{-in-lists}$: $u \in \text{lists } A \implies u^{\textcircled{k}} \in \text{lists } A$
<proof>

lemma $takeWhile\text{-in-list}$: $u \in \text{lists } A \implies takeWhile\ P\ u \in \text{lists } A$
<proof>

lemma $rev\text{-in-lists}$: $u \in \text{lists } A \implies rev\ u \in \text{lists } A$
<proof>

lemma $append\text{-in-lists-dest1}[\text{elim}]$: $u \cdot v = w \implies w \in \text{lists } A \implies u \in \text{lists } A$
<proof>

lemma $append\text{-in-lists-dest2}$: $u \cdot v = w \implies w \in \text{lists } A \implies v \in \text{lists } A$
<proof>

lemma $pow\text{-in-lists-dest1}$: $u \cdot v = w^{\textcircled{n}} \implies w \in \text{lists } A \implies u \in \text{lists } A$
<proof>

lemma $pow\text{-in-lists-dest1-sym}$: $w^{\textcircled{n}} = u \cdot v \implies w \in \text{lists } A \implies u \in \text{lists } A$
<proof>

lemma $pow\text{-in-lists-dest2}$: $u \cdot v = w^{\textcircled{n}} \implies w \in \text{lists } A \implies v \in \text{lists } A$
<proof>

lemma $pow\text{-in-lists-dest2-sym}$: $w^{\textcircled{n}} = u \cdot v \implies w \in \text{lists } A \implies v \in \text{lists } A$
<proof>

lemma *per-in-lists*: $w <_p r \cdot w \implies r \in \text{lists } A \implies w \in \text{lists } A$
 ⟨proof⟩

lemma *nth-in-lists*: $j < |w| \implies w \in \text{lists } A \implies w ! j \in A$
 ⟨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 *lists-butlast* | rule *tl-in-lists* |
 rule *take-in-lists* | intro *lq-in-lists* | rule *nth-in-lists* |
 rule *append-in-lists* | elim *conjug-in-lists* | rule *pow-in-lists* | rule *takeWhile-in-list*
 | elim *append-in-lists-dest1* | elim *append-in-lists-dest2*
 | elim *pow-in-lists-dest2* | elim *pow-in-lists-dest2-sym*
 | elim *pow-in-lists-dest1* | elim *pow-in-lists-dest1-sym*)
 | (simp | fact))+)⟩

2.23 Reversed mappings

definition *rev-map* :: (*'a list* \Rightarrow *'b list*) \Rightarrow (*'a list* \Rightarrow *'b list*) **where**
rev-map *f* = *rev* \circ *f* \circ *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* \longleftrightarrow *g* = *h*
 ⟨proof⟩

lemma *rev-map-funpow*[*reversal-rule*]: (*rev-map* (*f*::*'a list* \Rightarrow *'a list*)) $\hat{\sim}$ *k* = *rev-map*
 (*f* $\hat{\sim}$ *k*)
 ⟨proof⟩

2.24 Overlapping powers, periods, prefixes and suffixes

lemma *pref-suf-overlapE*: **assumes** $p \leq_p w$ **and** $s \leq_s w$ **and** $|w| \leq |p| + |s|$
obtains $p1 \cdot u \cdot s1 = w$ **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*: $p \cdot u \leq_p u \cdot u \implies p \cdot u = u \cdot p$
<proof>

lemmas *mid-sq-suf* = *mid-sq-pref*[reversed]

lemma *mid-sq-pref-suf*: **assumes** $p \cdot x \cdot q = x \cdot x$ **shows** $p \leq_p x$ **and** $p \leq_s x$ **and** $q \leq_p x$ **and** $q \leq_s x$
<proof>

lemma *mid-sq-primroot*: **assumes** $p \cdot x <_p x \cdot x$
shows $x \cdot x = p \cdot x \cdot (\varrho x)^{\textcircled{e}} (e_{\varrho} ((p \cdot x)^{-1} \triangleright (x \cdot x)))$ **(is** $x \cdot x = p \cdot x \cdot (\varrho x)^{\textcircled{e}} e_{\varrho} ?z$
 $0 < e_{\varrho} ((p \cdot x)^{-1} \triangleright (x \cdot x))$ **(is** $0 < e_{\varrho} ?z$)
<proof>

lemma *mid-pow*: **assumes** $p \cdot x^{\textcircled{l}} (\text{Suc } l) \cdot q = x^{\textcircled{k}}$
shows $x \cdot p = p \cdot x$ **and** $x \cdot q = q \cdot x$
<proof>

lemma *root-suf-comm*: **assumes** $x \leq_p r \cdot x$ **and** $r \leq_s r \cdot x$ **shows** $r \cdot x = x \cdot r$
<proof>

lemma *pref-marker*: **assumes** $w \leq_p v \cdot w$ **and** $u \cdot v \leq_p w$
shows $u \cdot v = v \cdot u$
<proof>

lemma *pref-marker-ext*: **assumes** $|x| \leq |y|$ **and** $v \neq \varepsilon$ **and** $y \cdot v \leq_p x \cdot v^{\textcircled{k}}$
obtains n **where** $y = x \cdot (\varrho v)^{\textcircled{n}}$
<proof>

lemma *pref-marker-sq*: $p \cdot x \leq_p x \cdot x \implies p \cdot x = x \cdot p$
<proof>

lemmas *suf-marker-sq* = *pref-marker-sq*[reversed]

lemma *pref-marker-conjug*: **assumes** $w \neq \varepsilon$ **and** $w \cdot r \cdot s \leq_p s \cdot (r \cdot s)^{\textcircled{m}}$ **and**

primitive $(r \cdot s)$

obtains n **where** $w = s \cdot (r \cdot s)^{\textcircled{n}}$

<proof>

lemmas *pref-marker-reversed* = *pref-marker[reversed]*

lemma *suf-marker-per-root*: **assumes** $w \leq_p v \cdot w$ **and** $p \cdot v \cdot u \leq_p w$

shows $u \leq_p v \cdot u$

<proof>

lemma *suf-marker-per-root'*: **assumes** $w \leq_p v \cdot w$ **and** $p \cdot v \cdot u \leq_p w$ **and** $v \neq \varepsilon$

shows $u \leq_p p \cdot u$

<proof>

lemma *marker-fac-pref*: **assumes** $u \leq_f r^{\textcircled{k}}$ **and** $r \leq_p u$ **shows** $u \leq_p r^{\textcircled{k}}$

<proof>

lemma *marker-fac-pref-len*: **assumes** $u \leq_f r^{\textcircled{k}}$ **and** $t \leq_p u$ **and** $|t| = |r|$

shows $u \leq_p t^{\textcircled{k}}$

<proof>

lemma *pref-comm-cancel*: $u \cdot z \leq_p v \implies u \cdot v = v \cdot u \implies z \leq_p v$

<proof>

lemma *marker-fac-pref-ext*: **assumes** $w \leq_f v^{\textcircled{k}}$ $v = p \cdot s \cdot s \cdot v \bowtie w$ $|v| \leq |w|$

shows $p \cdot w \leq_p v \cdot p \cdot w$

<proof>

lemma *root-suf-comm'*: $x \leq_p r \cdot x \implies r \leq_s x \implies r \cdot x = x \cdot r$

<proof>

lemmas *suf-root-pref-comm* = *root-suf-comm'[reversed]*

lemma *mid-marker-root*: **assumes** $u \cdot v \leq_p r^{\textcircled{k}}$ $r \leq_s u$ **shows** $v \leq_p r \cdot v$

<proof>

lemmas *mid-marker-root'* = *mid-marker-root[reversed]*

lemma *marker-pref-suf-fac*: **assumes** $u \leq_p v$ **and** $u \leq_s v$ **and** $v \leq_f u^{\textcircled{k}}$

shows $u \cdot v = v \cdot u$

<proof>

lemma *pref-suf-per-fac-comm*:

assumes $v \leq_p u \cdot v$ **and** $v \leq_s v \cdot u$ **and** $u \leq_f v^{\textcircled{k}}$ **shows** $u \cdot v = v \cdot u$

<proof>

lemma *mid-long-pow*: **assumes** *eq*: $y^{\textcircled{m}} = u \cdot x^{\textcircled{k}}(\text{Suc } k) \cdot v$ **and** $|y| \leq |x^{\textcircled{k}}|$

shows $(u \cdot v) \cdot y = y \cdot (u \cdot v)$ **and** $(u \cdot x^{\textcircled{l}} \cdot v) \cdot y = y \cdot (u \cdot x^{\textcircled{l}} \cdot v)$ **and**
 $(u^{-1} \triangleright (y \cdot u)) \cdot x = x \cdot (u^{-1} \triangleright (y \cdot u))$

<proof>

lemma *mid-pow-pref-suf'*: **assumes** $s \cdot w^{\textcircled{k}} (\text{Suc } l) \cdot p \leq f w^{\textcircled{k}}$ **shows** $p \leq p w^{\textcircled{k}}$ **and**
 $s \leq s w^{\textcircled{k}}$
<proof>

lemma *mid-pow-pref-suf*: **assumes** $s \cdot w \cdot p \leq f w^{\textcircled{k}}$ **shows** $p \leq p w^{\textcircled{k}}$ **and** $s \leq s w^{\textcircled{k}}$
<proof>

lemma *fac-marker-pref*: $y \cdot x \leq f y^{\textcircled{k}} \implies x \leq p y \cdot x$
<proof>

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

lemma *fac-two-markers*: **assumes** $u \cdot v \cdot u \leq f u^{\textcircled{k}}$ **shows** $u \cdot v = v \cdot u$
<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*) $q = \varepsilon$
<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-prefE*:
assumes *primitive* r **and** $p \cdot r \leq p r \cdot r$
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 $p = \varepsilon \vee q = \varepsilon$
<proof>

lemma *prim-overlap-sq'*:
assumes *prim*: *primitive* r **and** *pref*: $p \cdot r \leq p r \cdot r$ **and** *len*: $|p| < |r|$
shows $p = \varepsilon$
<proof>

lemma *zxy-fac-xyzx*: **assumes** *nemp*: $y \neq \varepsilon$ $z \neq \varepsilon$ **and** *fac*: $z \cdot x \cdot y \leq f x \cdot y \cdot z \cdot x$
shows \neg *primitive* $(z \cdot x \cdot y)$
<proof>

lemma *prim-overlap-pow*[*elim*]:

assumes *prim*: primitive r **and** *pref*: $u \cdot r \leq_p r^{\textcircled{a}}k$
obtains i **where** $u = r^{\textcircled{a}}i$ **and** $i < k$
 ⟨*proof*⟩

lemma *prim-overlap-pow'*:
assumes *prim*: primitive r **and** *pref*: $u \cdot r \leq_p r^{\textcircled{a}}k$ **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** $\text{Suc } n < |w|$ **and** $w \leq_p \text{drop } (\text{Suc } n) (w \cdot w)$
and primitive w
shows *False*
 ⟨*proof*⟩

lemma *root-suf-conjug*: **assumes** primitive $(s \cdot r)$ **and** $y \leq_p (s \cdot r) \cdot y$ **and** $y \leq_s$
 $y \cdot (r \cdot s)$ **and** $|s \cdot r| \leq |y|$
obtains l **where** $y = (s \cdot r)^{\textcircled{a}}l \cdot s$
 ⟨*proof*⟩

lemma *root-suf-pow-comm*: **assumes** $x \leq_p r \cdot x$ **and** $r \leq_s x^{\textcircled{a}}(\text{Suc } k)$ **shows** $r \cdot$
 $x = x \cdot r$
 ⟨*proof*⟩

lemma *suf-pow-short-suf*: $r \leq_s x^{\textcircled{a}}k \implies |x| \leq |r| \implies x \leq_s r$
 ⟨*proof*⟩

thm *suf-pow-short-suf[reversed]*

lemma *sq-short-per*: **assumes** $|u| \leq |v|$ **and** $v \cdot v \leq_p u \cdot (v \cdot v)$
shows $u \cdot v = v \cdot u$
 ⟨*proof*⟩

lemma *fac-marker*: **assumes** $w \leq_p u \cdot w$ **and** $u \cdot v \cdot u \leq_f w$
shows $u \cdot v = v \cdot u$
 ⟨*proof*⟩

lemma $4 = \text{Suc}(\text{Suc}(\text{Suc}(\text{Suc } 0)))$
 ⟨*proof*⟩

lemma *xyxy-conj-yxy*: **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 $u \leq_p w$ and $v \leq_s w$
and

$|w| + n \leq |u| + |v|$
shows period $w\ n$

<proof>

lemma per-glue-facs: assumes $u \cdot z \leq_f w^{\textcircled{k}}$ and $z \cdot v \leq_f w^{\textcircled{m}}$ and $|w| \leq |z|$
obtains l where $u \cdot z \cdot v \leq_f w^{\textcircled{l}}$

<proof>

lemma per-fac-pow-fac: assumes period $w\ n$ and $v \leq_f w$ and $|v| = n$ and $n \neq 0$
obtains k where $w \leq_f v^{\textcircled{k}}$

<proof>

lemma refine-per: assumes period $w\ n$ and $v \leq_f w$ and $n \leq |v|$ and period $v\ k$
and $k \text{ dvd } n$ and $n \neq 0$

shows period $w\ k$

<proof>

lemma xy-per-comp: assumes $x \cdot y \leq_p q \cdot x \cdot y$

and $q \neq \varepsilon$ and $q \bowtie y$

shows $x \bowtie y$

<proof>

lemma prim-xyxy: $x \cdot y \neq y \cdot x \implies \text{primitive } (x \cdot y \cdot x \cdot y \cdot y)$

<proof>

lemma prim-min-per-suf-eq: assumes primitive x and $\pi x \leq_s x$ shows $\pi x = x$

<proof>

lemma primroot-code[*code*]: $\varrho x = (\text{if } x \neq \varepsilon \text{ then } (\text{if } \pi x \leq_s x \text{ then } \pi x \text{ else } x) \text{ else } \text{Code.abort } (\text{STR } \text{"Empty word has no primitive root."}) (\lambda\cdot. (\varrho x)))$

<proof>

lemma per-lemma-pref-suf: assumes $w <_p p \cdot w$ and $w <_s w \cdot q$ and

fw: $|p| + |q| \leq |w|$

obtains $r\ s\ k\ l\ m$ where $p = (r \cdot s)^{\textcircled{k}}$ and $q = (s \cdot r)^{\textcircled{l}}$ and $w = (r \cdot s)^{\textcircled{m}} \cdot r$
and primitive $(r \cdot s)$

<proof>

lemma fac-two-conjug-primrootE:

assumes *facs:* $w \leq_f p^{\textcircled{k}}$ $w \leq_f q^{\textcircled{l}}$ and *nemps:* $p \neq \varepsilon$ $q \neq \varepsilon$ and *len:* $|p| + |q| \leq |w|$

obtains $r\ s\ m$ where $\varrho p \sim r \cdot s$ and $\varrho q \sim r \cdot s$ and $w = (r \cdot s)^{\textcircled{m}} \cdot r$ and
primitive $(r \cdot s)$

<proof>

corollary *fac-two-conjug-primroot*:

assumes *facts*: $u \leq_f r^{\textcircled{k}} u \leq_f s^{\textcircled{l}}$ **and** *len*: $|r| + |s| \leq |u|$

shows $\varrho r \sim \varrho s$

<proof>

lemma *fac-two-prim-conjug*:

assumes $w \leq_f u^{\textcircled{n}} w \leq_f v^{\textcircled{m}}$ *primitive u primitive v* $|u| + |v| \leq |w|$

shows $u \sim v$

<proof>

lemma *fac-pow-conjug-primroot*: **assumes** $u^{\textcircled{k}} \leq_f v^{\textcircled{l}}$ **and** $|u^{\textcircled{k}}| \geq 2*|v|$ **and** $2 \leq k$ **and** $u \neq \varepsilon$

shows $\varrho u \sim \varrho v$

<proof>

2.25 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* ($[x,y]^{\textcircled{n}} \cdot [x]$)

<proof>

2.26 Factor interpretation

definition *factor-interpretation* :: 'a list \Rightarrow 'a list \Rightarrow 'a list \Rightarrow 'a list list \Rightarrow bool
(- - - $\sim_{\mathcal{I}}$ - [52,52,52,52] 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*: $u \neq \varepsilon \implies p u s \sim_{\mathcal{I}} ws \implies ws \neq \varepsilon$

<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*[*intro*]:

$p <_p hd ws \implies s <_s last ws \implies p \cdot u \cdot s = concat ws \implies p u s \sim_{\mathcal{I}} ws$

<proof>

lemma *fac-fac-interpE*: **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**

$s \cdot \text{concat } ss = su$
 ⟨proof⟩

lemma *pref-interpE*: **assumes** $p \ u \ s \sim_{\mathcal{I}} \ ws \ v \leq_p \ u \ v \neq \varepsilon$
obtains $vs \ s'$ **where** $p \ v \ s' \sim_{\mathcal{I}} \ vs \ vs \leq_p \ ws$
 ⟨proof⟩

lemma *fac-fac-interpE'*: **assumes** $u \leq_f \text{concat } ws$ **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** $w \leq_f \ v^{\textcircled{a}} k$ **and** $|v| \leq |w|$
obtains $m \ v1 \ v2$ **where** $v1 \leq_s \ v \ v2 \leq_p \ v \ w = v1 \cdot v^{\textcircled{a}} m \cdot v2$
 ⟨proof⟩

lemma *fac-interp-inner*: **assumes** $u \neq \varepsilon$ **and** $p \ u \ s \sim_{\mathcal{I}} \ ws$ **and** $1 < |ws|$
shows $p^{-1} \langle \text{hd } ws \rangle \cdot \text{concat}(\text{butlast } (tl \ ws)) \cdot (\text{last } ws)^{<-1} s = u$
 ⟨proof⟩

lemma *fac-interp-inner-len*: **assumes** $u \neq \varepsilon$ **and** $p \ u \ s \sim_{\mathcal{I}} \ ws$
shows $|\text{concat}(\text{butlast } (tl \ ws))| < |u|$
 ⟨proof⟩

lemma *rev-in-set-map-rev-conv*: $\text{rev } u \in \text{set } (\text{map } \text{rev } ws) \longleftrightarrow u \in \text{set } ws$
 ⟨proof⟩

lemma *rev-fac-interp*: **assumes** $p \ u \ s \sim_{\mathcal{I}} \ ws$ **shows** $(\text{rev } s) \ (\text{rev } u) \ (\text{rev } p) \sim_{\mathcal{I}} \ \text{rev } (\text{map } \text{rev } ws)$
 ⟨proof⟩

lemma *rev-fac-interp-iff [reversal-rule]*: $(\text{rev } s) \ (\text{rev } u) \ (\text{rev } p) \sim_{\mathcal{I}} \ \text{rev } (\text{map } \text{rev } ws) \longleftrightarrow p \ u \ s \sim_{\mathcal{I}} \ ws$
 ⟨proof⟩

lemma *fac-interp-mid-fac*: **assumes** $p \ u \ s \sim_{\mathcal{I}} \ ws$
shows $\text{concat } (\text{butlast } (tl \ ws)) \leq_f \ u$
 ⟨proof⟩

definition *disjoint-interpretation* :: 'a list \Rightarrow 'a list list \Rightarrow 'a list \Rightarrow 'a list list \Rightarrow bool (- - - $\sim_{\mathcal{D}}$ - [51,51,51,51] 60)

where $p \ us \ s \sim_{\mathcal{D}} \ ws \equiv p \ (\text{concat } us) \ s \sim_{\mathcal{I}} \ ws \wedge$
 $(\forall \ u \ v. \ u \leq_p \ us \wedge \ v \leq_p \ ws \longrightarrow p \cdot \text{concat } u \neq \text{concat } v)$

lemma *disj-interpI*: $p \ (\text{concat } us) \ s \sim_{\mathcal{I}} \ ws \implies$
 $(\forall \ u \ v. \ u \leq_p \ us \wedge \ v \leq_p \ ws \longrightarrow p \cdot \text{concat } u \neq \text{concat } v) \implies p \ us \ s \sim_{\mathcal{D}} \ ws$
 ⟨proof⟩

lemma *disj-interpI'*[intro]: $p (\text{concat } us) s \sim_{\mathcal{I}} ws \implies$
 $(\bigwedge u v. u \leq_p us \implies v \leq_p ws \implies p \cdot \text{concat } u \neq \text{concat } v) \implies p us s \sim_{\mathcal{D}} ws$
 ⟨proof⟩

lemma *disj-interpD0*: $p us s \sim_{\mathcal{D}} ws \implies p (\text{concat } us) s \sim_{\mathcal{I}} ws$
 ⟨proof⟩

lemmas *disj-interpD = fac-interpD[OF disj-interpD0]*

lemma *disj-interpD1*: **assumes** $p us s \sim_{\mathcal{D}} ws$ **and** $us' \leq_p us$ **and** $ws' \leq_p ws$
shows $p \cdot \text{concat } us' \neq \text{concat } ws'$
 ⟨proof⟩

lemma *disj-interp-nemp*: **assumes** $p us s \sim_{\mathcal{D}} ws$
shows $p \neq \varepsilon$ **and** $s \neq \varepsilon$ **and** $ws \neq \varepsilon$
 ⟨proof⟩

lemma *non-disjoint-interpE*: **assumes** $p (\text{concat } us) s \sim_{\mathcal{I}} ws$ **and**
 $\neg p us s \sim_{\mathcal{D}} ws$
obtains $us1 us2 us1 us2$ **where** $us1 \cdot us2 = us ws1 \cdot ws2 = ws$
 $p \cdot \text{concat } us1 = \text{concat } ws1 \text{ concat } us2 \cdot s = \text{concat } ws2$
 ⟨proof⟩

lemma *emp-disj-interp*: **assumes** $p \neq \varepsilon$ $p <_p u$
shows $p \varepsilon (p^{-1} > u) \sim_{\mathcal{D}} [u]$
 ⟨proof⟩

lemma *emp-fac-interp*: $p \neq \varepsilon \implies s \neq \varepsilon \implies p \varepsilon s \sim_{\mathcal{I}} [p \cdot s]$
 ⟨proof⟩

lemma *emp-disj-interp'*:
assumes $1 < |w|$ **shows** $[hd w] \varepsilon (tl w) \sim_{\mathcal{D}} [w]$
 ⟨proof⟩

lemma *pref-disj-interpE*: **assumes** $p us s \sim_{\mathcal{D}} ws$ $vs \leq_p us$
obtains $ws' s'$ **where** $p vs s' \sim_{\mathcal{D}} ws' ws' \leq_p ws$
 ⟨proof⟩

lemma *rev-disj-interp*: **assumes** $p us s \sim_{\mathcal{D}} ws$ **shows** $(\text{rev } s) (\text{rev } (\text{map rev } us))$
 $(\text{rev } p) \sim_{\mathcal{D}} \text{rev } (\text{map rev } ws)$
 ⟨proof⟩

lemma *rev-disj-interp-iff* [reversal-rule]: $(\text{rev } s) (\text{rev } (\text{map rev } us)) (\text{rev } p) \sim_{\mathcal{D}} \text{rev}$
 $(\text{map rev } ws) \iff p us s \sim_{\mathcal{D}} ws$
 ⟨proof⟩

lemma *disj-interp-split*:
assumes *disj*: $p us s \sim_{\mathcal{D}} ws$ **and** $us = us_1 \cdot us_2$
obtains $ws_1 ws_2 p' s'$ **where** $p us_1 s' \sim_{\mathcal{D}} ws_1 \cdot [p' \cdot s'] p' us_2 s \sim_{\mathcal{D}} [p' \cdot s'] \cdot ws_2$

$ws = ws_1 \cdot [p'.s'] \cdot ws_2$
 ⟨proof⟩

corollary *disj-interp-fac*:

assumes *disj*: $p \ us \ s \sim_{\mathcal{D}} \ ws$ **and** $us = us_1 \cdot us_2 \cdot us_3$

obtains $ws_1 \ ws_2 \ ws_3 \ p' \ s'$ **where** $p' \ us_2 \ s' \sim_{\mathcal{D}} \ ws_2$

and $p \cdot \text{concat } us_1 = \text{concat } ws_1 \cdot p' \ \text{concat } us_3 \cdot s = s' \cdot \text{concat } ws_3$

and $ws = ws_1 \cdot ws_2 \cdot ws_3$

⟨proof⟩

lemma *disj-interp-alternate*:

assumes *disj*: $p \ us \ s \sim_{\mathcal{D}} \ ws$ **and** $1 < |ws|$

obtains $us_1 \ us_2 \ us' \ p' \ s'$ **where** $p' \ (\text{butlast } (\text{tl } ws)) \ s' \sim_{\mathcal{D}} \ us'$

$\text{hd } ws = p \cdot \text{concat } us_1 \cdot p' \ \text{last } ws = s' \cdot \text{concat } us_2 \cdot s \ us_1 \cdot us' \cdot us_2 = us$

⟨proof⟩

lemma *disj-interp-long*: **assumes** $p \ us \ s \sim_{\mathcal{D}} \ ws$

shows $1 < |\text{hd } ws| \ 1 < |\text{last } ws|$

⟨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**

$x \neq \varepsilon \implies x \leq b \ w$ **and**

border-ConsD-neg: $x \neq w$ **and**

border-ConsD-pref: $x \leq p \ w$ **and**

border-ConsD-suf: $x \leq s \ w$

⟨proof⟩

lemma *ext-suf-Cons*:

$\text{Suc } i + |u| = |w| \implies u \leq s \ w \implies (w!i)\#u \leq s \ (w!i)\#w$

⟨proof⟩

lemma *ext-suf-Cons-take-drop*: **assumes** $\text{take } k \ (\text{drop } (\text{Suc } i) \ w) \leq s \ \text{drop } (\text{Suc } i) \ w$ **and** $w ! i = w ! (|w| - \text{Suc } k)$

shows $\text{take } (\text{Suc } k) \ (\text{drop } i \ w) \leq s \ \text{drop } i \ w$

⟨proof⟩

lemma *ext-border-Cons*:

$Suc\ i + |u| = |w| \implies u \leq b\ w \implies (w!i)\#u \leq b\ (w!i)\#w$
<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 \Rightarrow nat list \Rightarrow nat \Rightarrow nat \Rightarrow 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


```
theory Submonoids  
  imports CoWBasic HOL.Hull  
begin
```

Chapter 3

Submonoids of a free monoid

This chapter deals with properties of submonoids of a free monoid, that is, with monoids of words. See more in Chapter 1 of [4].

3.1 Generated monoid (also called hull)

First, we define the hull of a set of words, that is, the monoid generated by them.

definition *word-monoid* **where**

word-monoid $M \equiv (\forall w1\ w2. w1 \in M \longrightarrow w2 \in M \longrightarrow w1 \cdot w2 \in M) \wedge \varepsilon \in M$

lemma *word-monoidI*[*intro*]: **assumes** $\bigwedge w1\ w2. w1 \in M \implies w2 \in M \implies w1 \cdot w2 \in M$
 $\varepsilon \in M$

shows *word-monoid* M

<proof>

inductive-set *hull* :: 'a list set \Rightarrow 'a list set (*<->*)

for G **where**

emp-in[*simp*]: $\varepsilon \in \langle G \rangle$ |

prod-cl: $w1 \in G \implies w2 \in \langle G \rangle \implies w1 \cdot w2 \in \langle G \rangle$

lemmas [*intro*] = *hull.intros*

lemma *hull-closed*[*intro*]: $w1 \in \langle G \rangle \implies w2 \in \langle G \rangle \implies w1 \cdot w2 \in \langle G \rangle$

<proof>

lemma *gen-in* [*intro*]: $w \in G \implies w \in \langle G \rangle$

<proof>

lemma *gen-monoid-monoid*: *word-monoid* $\langle G \rangle$

<proof>

lemma *gen-monoid-hull*: *word-monoid* *hull* $G = \langle G \rangle$

<proof>

lemma *gen-monoid-induct*: **assumes** $x \in \langle G \rangle$ $P \varepsilon \wedge w. w \in G \implies P w$
 $\bigwedge w1 w2. w1 \in \langle G \rangle \implies P w1 \implies w2 \in \langle G \rangle \implies P w2 \implies P (w1 \cdot w2)$ **shows**
 $P x$
<proof>

lemma *genset-sub[simp]*: $G \subseteq \langle G \rangle$
<proof>

lemma *lists-sub*: $ws \in \text{lists } G \implies ws \in \text{lists } \langle G \rangle$
<proof>

lemma *in-lists-conv-set-subset*: $\text{set } ws \subseteq G \longleftrightarrow ws \in \text{lists } G$
<proof>

lemma *concat-in-hull [intro]*:
assumes $\text{set } ws \subseteq G$
shows $\text{concat } ws \in \langle G \rangle$
<proof>

lemma *concat-in-hull' [intro]*:
assumes $ws \in \text{lists } G$
shows $\text{concat } ws \in \langle G \rangle$
<proof>

lemma *hull-concat-lists0*:
 $w \in \langle G \rangle \implies \exists ws \in \text{lists } G. \text{concat } ws = w$ (**is** $w \in \langle G \rangle \implies ?P w$)
<proof>

lemma *hull-concat-listsE[elim]*: **assumes** $w \in \langle G \rangle$
obtains ws **where** $ws \in \text{lists } G$ **and** $\text{concat } ws = w$
<proof>

lemma *hull-concat-lists*: $\langle G \rangle = \text{concat } ' \text{lists } G$
<proof>

lemma *hull-concat-lists'*: $\langle G \rangle = \{\text{concat } xs \mid xs. xs \in \text{lists } G\}$
<proof>

lemma *hull-mono*: $A \subseteq B \implies \langle A \rangle \subseteq \langle B \rangle$
<proof>

lemma *hull-concat-lists-nempE[elim]*: **assumes** $w \in \langle G \rangle$
obtains ws **where** $ws \in \text{lists } (G - \{\varepsilon\})$ **and** $\text{concat } ws = w$
<proof>

lemma *hull-nemp-eq-hull[simp]*: $\langle G - \{\varepsilon\} \rangle = \langle G \rangle$
<proof>

lemma *emp-gen-set*: $\langle \{\} \rangle = \{\varepsilon\}$
<proof>

lemma *concat-lists-minus[simp]*: $\text{concat } \text{' lists } (G - \{\varepsilon\}) = \text{concat } \text{' lists } G$
<proof>

lemma *hull-drop-one*: $\langle G - \{\varepsilon\} \rangle = \langle G \rangle$
<proof>

lemma *sing-gen-power*: $u \in \langle \{x\} \rangle \implies \exists k. u = x^{\textcircled{a}} k$
<proof>

lemma *pow-sing-gen[simp]*: $x^{\textcircled{a}} k \in \langle \{x\} \rangle$
<proof>

lemma *sing-gen-pow-ex-conv*: $u \in \langle \{x\} \rangle \longleftrightarrow (\exists k. u = x^{\textcircled{a}} k)$
<proof>

lemma *sing-gen-primroot-gen*: **assumes** $w \in \langle \{x\} \rangle$
shows $w \in \langle \{\varrho x\} \rangle$
<proof>

lemma *sing-gen-primroot-eq*:
assumes $w \in \langle \{x\} \rangle$ $w \neq \varepsilon$
shows $\varrho w = \varrho x$
<proof>

lemma *sing-gen-pow-conv*: $w \in \langle \{\varrho x\} \rangle \longleftrightarrow w = \varrho x^{\textcircled{a}} e_{\varrho} w$
<proof>

lemma *two-primroots-comm*: **assumes** $w \neq \varepsilon$ **and** $w \in \langle \{\varrho x\} \rangle$ **and** $w \in \langle \{\varrho y\} \rangle$
shows $x \cdot y = y \cdot x$
<proof>

lemma *comm-rootI*: $x \in \langle \{r\} \rangle \implies y \in \langle \{r\} \rangle \implies x \cdot y = y \cdot x$
<proof>

lemma *sing-genE[elim]*:
assumes $u \in \langle \{x\} \rangle$
obtains k **where** $x^{\textcircled{a}} k = u$
<proof>

lemma *lists-gen-to-hull*: $us \in \text{lists } (G - \{\varepsilon\}) \implies us \in \text{lists } (\langle G \rangle - \{\varepsilon\})$

<proof>

lemma *rev-hull*: $\text{rev}'\langle G \rangle = \langle \text{rev}'G \rangle$
<proof>

lemma *power-in[intro]*: $x \in \langle G \rangle \implies x^{\textcircled{a}} k \in \langle G \rangle$
<proof>

lemma *hull-closed-lists*: $us \in \text{lists } \langle G \rangle \implies \text{concat } us \in \langle G \rangle$
<proof>

lemma *hull-I [intro]*:
 $\varepsilon \in H \implies (\bigwedge x y. x \in H \implies y \in H \implies x \cdot y \in H) \implies \langle H \rangle = H$
<proof>

lemma *gen-idemp*: $\langle \langle G \rangle \rangle = \langle G \rangle$
<proof>

lemma *hull-mono'[intro]*: $A \subseteq \langle B \rangle \implies \langle A \rangle \subseteq \langle B \rangle$
<proof>

lemma *hull-conjug [elim]*: $w \in \langle \{r \cdot s, s \cdot r\} \rangle \implies w \in \langle \{r, s\} \rangle$
<proof>

lemma *root-divisor*: **assumes** *period* w k **and** $k \text{ dvd } |w|$ **shows** $w \in \langle \{\text{take } k w\} \rangle$
<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>

theorem *hull-minimal*: $\langle G \rangle = \bigcap \{ S . G \subseteq S \wedge \langle S \rangle = S \}$
<proof>

lemma *all-gen-comm-hull-comm*: **assumes** $\bigwedge x y. x \in G \implies y \in G \implies x \cdot y = y \cdot x$
 $u \in \langle G \rangle \ v \in \langle G \rangle$
shows $u \cdot v = v \cdot u$
<proof>

lemma *bin-comm-hull-comm*: **assumes** $x \cdot y = y \cdot x \ u \in \langle \{x, y\} \rangle \ v \in \langle \{x, y\} \rangle$
shows $u \cdot v = v \cdot u$
<proof>

lemma*[reversal-rule]*: $\text{rev}' \langle \{\text{rev } u, \text{rev } v\} \rangle = \langle \{u, v\} \rangle$
<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 in that case, however, the decomposition need not be unique.

definition *decompose* :: 'a list set \Rightarrow 'a list \Rightarrow '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>

lemma *fac-fac-interpE-dec*: **assumes** $w \in \langle G \rangle$ $u \leq f\ w$ $u \neq \varepsilon$

obtains $p\ s\ ws$ **where** $ws \in \text{lists } (G - \{\varepsilon\})$ $p\ u\ s \sim_{\mathcal{I}} ws$ $ws \leq_f \text{Dec } G\ w$
 ⟨proof⟩

lemma *set-len-mod-concat*: **assumes** $\forall x \in \text{set } us. |x| \bmod n = 0$ $0 < n$
shows $|\text{concat } us| \bmod n = 0$
 ⟨proof⟩

lemma *hull-len-mod-concat*: **assumes** $\forall x \in G. |x| \bmod n = 0$ $0 < n$ $w \in \langle G \rangle$
shows $|w| \bmod n = 0$
 ⟨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 \Rightarrow 'a list list \Rightarrow 'a list list (*Ref* - - [55,56] 56) **where**
 $\text{refine } G\ us = \text{concat}(\text{map } (\text{decompose } G)\ us)$

lemma *ref-morph*: $us \in \text{lists } \langle G \rangle \Longrightarrow vs \in \text{lists } \langle G \rangle \Longrightarrow \text{refine } G\ (us \cdot vs) = \text{refine } G\ us \cdot \text{refine } G\ vs$
 ⟨proof⟩

lemma *ref-morph-plus*: $us \in \text{lists } (\langle G \rangle - \{\varepsilon\}) \Longrightarrow vs \in \text{lists } (\langle G \rangle - \{\varepsilon\}) \Longrightarrow \text{refine } G\ (us \cdot vs) = \text{refine } G\ us \cdot \text{refine } G\ vs$
 ⟨proof⟩

lemma *ref-pref-mono*: $us \in \text{lists } \langle G \rangle \Longrightarrow us \leq_p ws \Longrightarrow \text{Ref } G\ us \leq_p \text{Ref } G\ ws$
 ⟨proof⟩

lemma *ref-suf-mono*: $us \in \text{lists } \langle G \rangle \Longrightarrow us \leq_s ws \Longrightarrow (\text{Ref } G\ us) \leq_s \text{Ref } G\ ws$
 ⟨proof⟩

lemma *ref-fac-mono*: $us \in \text{lists } \langle G \rangle \Longrightarrow us \leq_f ws \Longrightarrow (\text{Ref } G\ us) \leq_f \text{Ref } G\ ws$
 ⟨proof⟩

lemma *ref-pop-hd*: $us \neq \varepsilon \Longrightarrow us \in \text{lists } \langle G \rangle \Longrightarrow \text{refine } G\ us = \text{decompose } G\ (\text{hd } us) \cdot \text{refine } G\ (\text{tl } us)$
 ⟨proof⟩

lemma *ref-in*: $us \in \text{lists } \langle G \rangle \Longrightarrow (\text{Ref } G\ us) \in \text{lists } (G - \{\varepsilon\})$
 ⟨proof⟩

lemma *ref-in'[intro]*: $us \in \text{lists } \langle G \rangle \Longrightarrow (\text{Ref } G\ us) \in \text{lists } G$
 ⟨proof⟩

lemma *concat-ref*: $us \in \text{lists } \langle G \rangle \Longrightarrow \text{concat } (\text{Ref } G\ us) = \text{concat } us$
 ⟨proof⟩

lemma *ref-gen*: $us \in \text{lists } B \Longrightarrow B \subseteq \langle G \rangle \Longrightarrow \text{Ref } G\ us \in \langle \text{decompose } G\ ' B \rangle$

<proof>

lemma *ref-set*: $us \in \text{lists } \langle G \rangle \implies \text{set } (\text{Ref } G \text{ } us) = \bigcup (\text{set}'(\text{decompose } G) \text{'set } us)$
<proof>

lemma *emp-ref*: **assumes** $us \in \text{lists } (\langle G \rangle - \{\varepsilon\})$ **and** $\text{Ref } G \text{ } us = \varepsilon$ **shows** $us = \varepsilon$
<proof>

lemma *sing-ref-sing*:
assumes $us \in \text{lists } (\langle G \rangle - \{\varepsilon\})$ **and** $\text{refine } G \text{ } us = [b]$
shows $us = [b]$
<proof>

lemma *ref-ex*: **assumes** $Q \subseteq \langle G \rangle$ **and** $us \in \text{lists } Q$
shows $\text{Ref } G \text{ } us \in \text{lists } (G - \{\varepsilon\})$ **and** $\text{concat } (\text{Ref } G \text{ } us) = \text{concat } us$
<proof>

lemma *ref-emp [simp]*: $\text{Ref } G \text{ } \varepsilon = \varepsilon$
<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.

Indecomposable element is an element that is not generated by other ones.

definition *ungenerated* :: 'a list \implies 'a list set \implies bool (- $\in U$ - [51,51] 50) **where**
 $\text{ungenerated } b \ G \equiv b \in G \wedge b \notin \langle G - \{b\} \rangle$

lemma *ungen-nemp[simp]*: $b \in U \ G \implies b \neq \varepsilon$
<proof>

lemma *ungen-in[intro]*: $\text{ungenerated } b \ G \implies b \in G$ **and**
ungenD[intro]: $\text{ungenerated } b \ G \implies b \notin \langle G - \{b\} \rangle$
<proof>

An ungenerated element has no nontrivial decomposition

lemma *ungen-dec-triv*: **assumes** $u \in \langle G \rangle \ v \in \langle G \rangle \ u \cdot v \in U \ \langle G \rangle$ **shows** $u = \varepsilon \vee v = \varepsilon$
<proof>

lemma *ungen-dec-triv'*: **assumes** $us \in \text{lists } (\langle G \rangle - \{\varepsilon\})$ $\text{concat } us \in U \ \langle G \rangle$ **shows**
 $|us| = 1$
<proof>

Conversely, any nonempty element that is not ungenerated is a product of at least two shorter elements

lemma *gen-elem-list*: **assumes** $u \in \langle G - \{u\} \rangle$ $u \neq \varepsilon$
obtains us **where** $us \in \text{lists } (G - \{u\} - \{\varepsilon\})$ $\text{concat } us = u$ $1 < |us|$
 $\wedge c. c \in \text{set } us \implies |c| < |u|$
 $\langle \text{proof} \rangle$

lemma *gen-elem-dec*: **assumes** $b \in \langle G - \{b\} \rangle$ $b \neq \varepsilon$
obtains $u v$ **where** $u \in \langle G \rangle$ $u \neq \varepsilon$ $v \in \langle G \rangle$ $v \neq \varepsilon$ $u \cdot v = b$
 $\langle \text{proof} \rangle$

This yields several criteria for being ungenerated

lemma *ungeneratedI*:
assumes $b \in G$ **and** $b \neq \varepsilon$
and all: $\wedge u v. u \neq \varepsilon \implies u \in \langle G \rangle \implies v \neq \varepsilon \implies v \in \langle G \rangle \implies u \cdot v \neq b$
shows $b \in U G$
 $\langle \text{proof} \rangle$

lemma *ungeneratedI'*:
assumes $b \in G$ **and** $b \neq \varepsilon$
and all: $\wedge us. us \in \text{lists } (G - \{b\} - \{\varepsilon\}) \implies \text{concat } us = b \implies |us| \leq 1$
shows $b \in U G$
 $\langle \text{proof} \rangle$

lemma *ungenerated-shortest*:
assumes $b \in G$ **and** $b \neq \varepsilon$
and all: $\wedge c. c \in G - \{\varepsilon\} \implies |b| \leq |c|$
shows $b \in U G$
 $\langle \text{proof} \rangle$

lemma *ungenerated-sing*:
assumes $[a] \in G$
shows $[a] \in U G$
 $\langle \text{proof} \rangle$

lemma *ungen-hull-ungen*: $b \in U \langle G \rangle \iff b \in U G$
 $\langle \text{proof} \rangle$

The *basis* is the set of all ungenerated elements.

definition *basis* :: 'a list set \Rightarrow 'a list set ($\mathfrak{B} - [51]$) **where**
 $\text{basis } G = \{x. x \in U G\}$

lemma *basisD*: $x \in \mathfrak{B} G \implies x \in U G$
 $\langle \text{proof} \rangle$

lemma *basis-in*: $x \in \mathfrak{B} G \implies x \in G$
 $\langle \text{proof} \rangle$

lemma *emp-not-basis*: $x \in \mathfrak{B} G \implies x \neq \varepsilon$
 $\langle \text{proof} \rangle$

lemma *basis-sub-gen*: $\mathfrak{B} G \subseteq G$
<proof>

The basis generates the generating set and therefore also the whole monoid

lemma *gen-sub-basis*: $G \subseteq \langle \mathfrak{B} G \rangle$
<proof>

lemma *basis-concat-listsE*:
assumes $w \in G$
obtains ws **where** $ws \in lists \ \mathfrak{B} G$ **and** $concat \ ws = w$
<proof>

theorem *basis-gen-hull*: $\langle \mathfrak{B} G \rangle = \langle G \rangle$
<proof>

theorem *basis-of-hull*: $\mathfrak{B} \langle G \rangle = \mathfrak{B} G$
<proof>

lemma *basis-gen-hull'*: $\langle \mathfrak{B} \langle G \rangle \rangle = \langle G \rangle$
<proof>

lemma *basis-hull-sub*: $\mathfrak{B} \langle G \rangle \subseteq G$
<proof>

The basis is the smallest generating set.

theorem *gen-basis-sub*: $\langle S \rangle = \langle G \rangle \implies \mathfrak{B} G \subseteq S$
<proof>

lemma *basis-min-gen*: $S \subseteq \mathfrak{B} G \implies \langle S \rangle = G \implies S = \mathfrak{B} G$
<proof>

lemma *basisI*: $(\bigwedge B. \langle B \rangle = \langle C \rangle \implies C \subseteq B) \implies \mathfrak{B} \langle C \rangle = C$
<proof>

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*: $\mathfrak{B} G \subseteq S \implies S \subseteq \langle G \rangle \implies \mathfrak{B} G = \mathfrak{B} S$
<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 *nemp*: $x \in G \implies x \neq \varepsilon$

<proof>

lemma *concat-eq-emp-conv* [*simp*]: $us \in \text{lists } G \implies \text{concat } us = \varepsilon \longleftrightarrow us = \varepsilon$

<proof>

lemma *root-dec-inj-on*: *inj-on* $(\lambda x. [\varrho x]^{\textcircled{e}}(e_{\varrho} x)) G$

<proof>

lemma *concat-len-ruler*: **assumes** $ws \in \text{lists } G \ us \leq_p ws \ vs \leq_p ws \ | \text{concat } us| \leq | \text{concat } vs|$

shows $us \leq_p vs$

<proof>

end

lemma *concat-emp*:

$\varepsilon \notin G \implies us \in \text{lists } G \implies \text{concat } us = \varepsilon \implies us = \varepsilon$

<proof>

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*: $xs \in \text{lists } \mathcal{C} \implies ys \in \text{lists } \mathcal{C} \implies \text{concat } xs = \text{concat } ys \implies xs = ys$

begin

lemma *code-comm-eq*: $x \in \mathcal{C} \implies y \in \mathcal{C} \implies x \cdot y = y \cdot x \implies x = y$

<proof>

lemma *emp-not-in*: $\varepsilon \notin \mathcal{C}$

<proof>

lemma *nemp*: $u \in \mathcal{C} \implies u \neq \varepsilon$

<proof>

sublocale *nemp-words* \mathcal{C}

<proof>

lemma *code-elim-dec*: $us \in \text{lists } \mathcal{C} \implies \text{concat } us = c \implies c \in \mathcal{C} \implies us = [c]$
<proof>

lemma *code-ungen*: **assumes** $c \in \mathcal{C}$ **shows** $c \in U \mathcal{C}$
<proof>

lemma *code-is-basis*: $\mathfrak{B} \mathcal{C} = \mathcal{C}$
<proof>

lemma *code-unique-dec'*: $us \in \text{lists } \mathcal{C} \implies \text{Dec } \mathcal{C} (\text{concat } us) = us$
<proof>

lemma *code-unique-dec [intro!]*: $us \in \text{lists } \mathcal{C} \implies \text{concat } us = u \implies \text{Dec } \mathcal{C} u = us$
<proof>

lemma *triv-refine [intro!]* : $us \in \text{lists } \mathcal{C} \implies \text{concat } us = u \implies \text{Ref } \mathcal{C} [u] = us$
<proof>

lemma *code-unique-ref*: $us \in \text{lists } \langle \mathcal{C} \rangle \implies \text{refine } \mathcal{C} us = \text{decompose } \mathcal{C} (\text{concat } us)$
<proof>

lemma *refI [intro]*: $us \in \text{lists } \langle \mathcal{C} \rangle \implies vs \in \text{lists } \mathcal{C} \implies \text{concat } vs = \text{concat } us \implies \text{Ref } \mathcal{C} us = vs$
<proof>

lemma *code-dec-morph*: **assumes** $x \in \langle \mathcal{C} \rangle$ $y \in \langle \mathcal{C} \rangle$
shows $(\text{Dec } \mathcal{C} x) \cdot (\text{Dec } \mathcal{C} y) = \text{Dec } \mathcal{C} (x \cdot y)$
<proof>

lemma *dec-pow*: $rs \in \langle \mathcal{C} \rangle \implies \text{Dec } \mathcal{C} (rs^{\textcircled{k}}) = (\text{Dec } \mathcal{C} rs)^{\textcircled{k}}$
<proof>

lemma *code-el-dec*: $c \in \mathcal{C} \implies \text{decompose } \mathcal{C} c = [c]$
<proof>

lemma *code-ref-list*: $us \in \text{lists } \mathcal{C} \implies \text{refine } \mathcal{C} us = us$
<proof>

lemma *code-ref-gen*: **assumes** $G \subseteq \langle \mathcal{C} \rangle$ $u \in \langle G \rangle$
shows $\text{Dec } \mathcal{C} u \in \langle \text{decompose } \mathcal{C} \text{ ' } G \rangle$
<proof>

find-theorems $\rho \text{ ?}x^{\textcircled{k}} \text{ ?}k = \text{?}x \ 0 < \text{?}k$

lemma *code-rev-code*: $\text{code } (\text{rev ' } \mathcal{C})$
<proof>

lemma *dec-rev* [*simp, reversal-rule*]:

$u \in \langle \mathcal{C} \rangle \implies \text{Dec rev } \langle \mathcal{C} \rangle (\text{rev } u) = \text{rev } (\text{map rev } (\text{Dec } \mathcal{C} \ u))$

<proof>

lemma *elem-comm-sing-set*: **assumes** $ws \in \text{lists } \mathcal{C}$ **and** $ws \neq \varepsilon$ **and** $u \in \mathcal{C}$ **and**
concat $ws \cdot u = u \cdot \text{concat } ws$

shows $\text{set } ws = \{u\}$

<proof>

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* ($\text{Dec } \mathcal{C} \ w$)

shows *primitive* w

<proof>

lemma *inj-on-dec*: *inj-on* (*decompose* \mathcal{C}) $\langle \mathcal{C} \rangle$

<proof>

lemma *ref-disj-interp*: **assumes** $vs \in \text{lists } \langle \mathcal{C} \rangle$ *p* *Ref* $\mathcal{C} \ vs \ s \sim_{\mathcal{D}} \ ws$

shows $p \ vs \ s \sim_{\mathcal{D}} \ ws$

<proof>

end — end context code

lemma *emp-is-code*: *code* $\{\}$

<proof>

lemma *code-rev-code-iff* [*reversal-rule*]: *code* ($\text{rev } \langle \mathcal{C} \rangle$) \longleftrightarrow *code* \mathcal{C}

<proof>

lemma *code-induct-hd*: **assumes** $\varepsilon \notin \mathcal{C}$ **and**

$\bigwedge \ xs \ ys. \ xs \in \text{lists } \mathcal{C} \implies \ ys \in \text{lists } \mathcal{C} \implies \text{concat } xs = \text{concat } ys \implies \text{hd } xs = \text{hd } ys$

shows *code* \mathcal{C}

<proof>

lemma *ref-set-primroot*: **assumes** $ws \in \text{lists } (G - \{\varepsilon\})$ **and** *code* ($\varrho \langle G \rangle$)

shows $\text{set } (\text{Ref } \varrho \langle G \rangle \ ws) = \varrho \langle \text{set } ws \rangle$

<proof>

3.5 Prefix code

locale *prefix-code* =

fixes \mathcal{C}

assumes

emp-not-in: $\varepsilon \notin \mathcal{C}$ **and**

pref-free: $u \in \mathcal{C} \implies v \in \mathcal{C} \implies u \leq_p v \implies u = v$

begin

lemma *nemp*: $u \in \mathcal{C} \implies u \neq \varepsilon$
<proof>

lemma *concat-pref-concat*:
assumes $us \in \text{lists } \mathcal{C} \text{ } vs \in \text{lists } \mathcal{C} \text{ } \text{concat } us \leq_p \text{concat } vs$
shows $us \leq_p vs$
<proof>

lemma *concat-pref-concat-conv*:
assumes $us \in \text{lists } \mathcal{C} \text{ } vs \in \text{lists } \mathcal{C}$
shows $\text{concat } us \leq_p \text{concat } vs \longleftrightarrow us \leq_p vs$
<proof>

sublocale *code*
<proof>

lemmas *is-code = is-code* **and**
code = code-axioms

lemma *dec-pref-unique*:
 $w \in \langle \mathcal{C} \rangle \implies p \in \langle \mathcal{C} \rangle \implies p \leq_p w \implies \text{Dec } \mathcal{C} \text{ } p \leq_p \text{Dec } \mathcal{C} \text{ } w$
<proof>

lemma *concat-suf-eq*: **assumes**
 $us \in \text{lists } \mathcal{C} \text{ } ws \in \text{lists } \mathcal{C}$ **and**
 $\text{concat } us \cdot s = \text{concat } ws$ **and** $s <_s \text{last } ws$
shows $us = ws$ **and** $s = \varepsilon$
<proof>

end

thm *prefix-code.concat-suf-eq[reversed]*

3.5.1 Suffix code

locale *suffix-code = prefix-code (rev ' C) for 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]

lemma *concat-pref-eq*: **assumes**
 $us \in \text{lists } \mathcal{C}$ $ws \in \text{lists } \mathcal{C}$ **and**
 $p \cdot \text{concat } us = \text{concat } ws$ **and** $p <_p \text{hd } ws$
shows $us = ws$ **and** $p = \varepsilon$
<proof>

thm *is-code*
code-axioms
code

end

3.5.2 Bifix code

locale *bifix-code* = *prefix-code* + *suf*: *suffix-code*
begin

lemma *joint-interp-triv*: **assumes**
 $us \in \text{lists } \mathcal{C}$ $ws \in \text{lists } \mathcal{C}$ **and**
interp: $p (\text{concat } us) s \sim_{\mathcal{I}} ws$ **and**
joint: $\neg p \text{ us } s \sim_{\mathcal{D}} ws$
shows $p = \varepsilon$ **and** $s = \varepsilon$ **and** $us = ws$
<proof>

end

3.6 Marked code

locale *marked-code* =
fixes \mathcal{C}
assumes
emp-not-in: $\varepsilon \notin \mathcal{C}$ **and**
marked: $u \in \mathcal{C} \implies v \in \mathcal{C} \implies \text{hd } u = \text{hd } v \implies u = v$

begin

lemma *nemp*: $u \in \mathcal{C} \implies u \neq \varepsilon$
<proof>

sublocale *prefix-code*
<proof>

lemma *marked-concat-lcp*: $us \in \text{lists } \mathcal{C} \implies vs \in \text{lists } \mathcal{C} \implies \text{concat } (us \wedge_p vs) =$
 $(\text{concat } us) \wedge_p (\text{concat } vs)$
<proof>

lemma *hd-concat-hd*: **assumes** $xs \in \text{lists } \mathcal{C}$ **and** $ys \in \text{lists } \mathcal{C}$ **and** $xs \neq \varepsilon$ **and** ys

$\neq \varepsilon$ **and**
 $hd (concat\ xs) = hd (concat\ ys)$
shows $hd\ xs = hd\ ys$
 $\langle proof \rangle$
end

3.7 Non-overlapping code

locale *non-overlapping* =
fixes \mathcal{C}
assumes
 $emp-not-in: \varepsilon \notin \mathcal{C}$ **and**
 $no-overlap: u \in \mathcal{C} \implies v \in \mathcal{C} \implies z \leq_p u \implies z \leq_s v \implies z \neq \varepsilon \implies u = v$ **and**
 $no-fac: u \in \mathcal{C} \implies v \in \mathcal{C} \implies u \leq_f v \implies u = v$
begin

lemma *nemp*: $u \in \mathcal{C} \implies u \neq \varepsilon$
 $\langle proof \rangle$

sublocale *prefix-code*
 $\langle proof \rangle$

lemma *rev-non-overlapping*: *non-overlapping* (*rev* ‘ \mathcal{C})
 $\langle proof \rangle$

sublocale *suf*: *suffix-code* \mathcal{C}
 $\langle proof \rangle$

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 $u = last\ vs$
 $\langle proof \rangle$

lemma *overlap-concat-hd*: **assumes** $u \in \mathcal{C}$ **and** $vs \in lists\ \mathcal{C}$ **and** $vs \neq \varepsilon$ **and** $r \neq$
 ε **and** $r \leq_s u$ **and** $r \leq_p concat\ vs \cdot s$
shows $u = hd\ vs$
 $\langle proof \rangle$

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** $ps \cdot us \cdot ss = vs$
 $\langle proof \rangle$

theorem *prim-morph*:
assumes $ws \in lists\ \mathcal{C}$
and $|ws| \neq 1$

and *primitive ws*
shows *primitive (concat ws)*
 \langle *proof* \rangle

lemma *prim-concat-conv*:
assumes *ws* \in *lists C*
and $|ws| \neq 1$
shows *primitive (concat ws)* \longleftrightarrow *primitive ws*
 \langle *proof* \rangle

end

3.8 Binary code

We pay a special attention to two element codes. In particular, we show that two words form a code if and only if they do not commute. This means that two words either commute, or do not satisfy any nontrivial relation.

definition *bin-lcp* **where** $bin-lcp\ x\ y = x \cdot y \wedge_p y \cdot x$

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*: $bin-lcs\ x\ y = rev\ (bin-lcp\ (rev\ x)\ (rev\ y))$
 \langle *proof* \rangle

lemma *bin-lcp-sym*: $bin-lcp\ x\ y = bin-lcp\ y\ x$
 \langle *proof* \rangle

lemma *bin-mismatch-comm*: $(bin-mismatch\ x\ y = bin-mismatch\ y\ x) \longleftrightarrow (x \cdot y = y \cdot x)$
 \langle *proof* \rangle

lemma *bin-lcp-pref-fst-snd*: $bin-lcp\ x\ y \leq_p x \cdot y$
 \langle *proof* \rangle

lemma *bin-lcp-pref-snd-fst*: $bin-lcp\ x\ y \leq_p y \cdot x$
 \langle *proof* \rangle

lemma *bin-lcp-bin-lcs [reversal-rule]*: $bin-lcp\ (rev\ x)\ (rev\ y) = rev\ (bin-lcs\ x\ y)$
 \langle *proof* \rangle

lemmas *bin-lcs-sym = bin-lcp-sym[reversed]*

lemma *bin-lcp-len*: $x \cdot y \neq y \cdot x \implies |bin-lcp\ x\ y| < |x \cdot y|$

<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 · *u*₁ ≠ *u*₁ · *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* (\neg *a*))

<proof>

lemma *bin-code-neq*: *u*₀ ≠ *u*₁

<proof>

lemma *bin-code-neq-bool*: *uu a* ≠ *uu* (\neg *a*)

<proof>

lemma *bin-fst-nemp*: *u*₀ ≠ ε **and** *bin-snd-nemp*: *u*₁ ≠ ε **and** *bin-nemp-bool*: *uu a* ≠ ε

<proof>

lemma *bin-not-comp*: \neg *u*₀ · *u*₁ \bowtie *u*₁ · *u*₀

<proof>

lemma *bin-not-comp-bool*: \neg (*uu a* · *uu* (\neg *a*) \bowtie *uu* (\neg *a*) · *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-neg*: $bin_mismatch\ u_0\ u_1 \neq bin_mismatch\ u_1\ u_0$
 ⟨*proof*⟩

abbreviation *bin-code-lcp* (α) **where** $bin_code_lcp \equiv bin_lcp\ u_0\ u_1$

abbreviation *bin-code-lcs* **where** $bin_code_lcs \equiv bin_lcs\ u_0\ u_1$

abbreviation *bin-code-mismatch-fst* (c_0) **where** $bin_code_mismatch_fst \equiv bin_mismatch\ u_0\ u_1$

abbreviation *bin-code-mismatch-snd* (c_1) **where** $bin_code_mismatch_snd \equiv bin_mismatch\ u_1\ u_0$

definition *cc* **where** $cc\ a = (if\ a\ then\ c_0\ else\ c_1)$

lemmas *bin-lcp-swap* = $bin_lcp_sym[of\ u_0\ u_1,\ symmetric]$ **and**
 $bin_lcp_pref = bin_lcp_pref_fst_snd[of\ u_0\ u_1]$ **and**
 $bin_lcp_pref' = bin_lcp_pref_snd_fst[of\ u_0\ u_1]$ **and**
 $bin_lcp_short = bin_lcp_len[OF\ non_comm,\ unfolded\ lenmorph]$

lemmas *bin-code-simps* = $cc_def\ uu_def\ if_True\ if_False\ bool_simps$

lemma *bin-lcp-bool*: $bin_lcp\ (uu\ a)\ (uu\ (\neg\ a)) = bin_code_lcp$
 ⟨*proof*⟩

lemma *bin-lcp-spref*: $\alpha <_p\ u_0 \cdot u_1$
 ⟨*proof*⟩

lemma *bin-lcp-spref'*: $\alpha <_p\ u_1 \cdot u_0$
 ⟨*proof*⟩

lemma *bin-lcp-spref-bool*: $\alpha <_p\ uu\ a \cdot uu\ (\neg\ a)$
 ⟨*proof*⟩

lemma *bin-mismatch-bool'*: $\alpha \cdot [cc\ a] \leq_p\ uu\ a \cdot uu\ (\neg\ a)$
 ⟨*proof*⟩

lemma *bin-mismatch-bool*: $\alpha \cdot [cc\ a] \leq_p\ uu\ a \cdot \alpha$
 ⟨*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 $bin-lcp-pref-all$: $xs \in lists\ \{u_0, u_1\} \implies \alpha \leq_p concat\ xs \cdot \alpha$
 ⟨proof⟩

lemma $bin-lcp-pref-all-hull$: $w \in \langle\{u_0, u_1\}\rangle \implies \alpha \leq_p w \cdot \alpha$
 ⟨proof⟩

lemma $bin-lcp-mismatch-pref-all-bool$: **assumes** $q \leq_p w$ **and** $w \in \langle\{uu\ b, uu\ (\neg b)\}\rangle$ **and** $|\alpha| < |uu\ a \cdot q|$
shows $\alpha \cdot [cc\ a] \leq_p uu\ a \cdot q$
 ⟨proof⟩

lemmas $bin-lcp-mismatch-pref-all-fst = bin-lcp-mismatch-pref-all-bool[of\ -\ -\ True\ True, unfolded\ bin-code-simps]$ **and**
 $bin-lcp-mismatch-pref-all-snd = bin-lcp-mismatch-pref-all-bool[of\ -\ -\ True\ False, unfolded\ bin-code-simps]$

lemma $bin-lcp-pref-all-len$: **assumes** $q \leq_p w$ **and** $w \in \langle\{u_0, u_1\}\rangle$ **and** $|\alpha| \leq |q|$
shows $\alpha \leq_p q$
 ⟨proof⟩

lemma $bin-mismatch-all-bool$: **assumes** $xs \in lists\ \{uu\ b, uu\ (\neg b)\}$ **shows** $\alpha \cdot [cc\ a] \leq_p (uu\ a) \cdot concat\ xs \cdot \alpha$
 ⟨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** $\alpha \cdot [cc\ a] \leq_p uu\ a \cdot w \cdot \alpha$
 ⟨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** $q \leq_p uu\ a \cdot w$ **and** $w \in \langle\{uu\ b, uu\ (\neg b)\}\rangle$ **and** $|\alpha| < |q|$
shows $\alpha \cdot [cc\ a] \leq_p q$
 ⟨proof⟩

lemmas $bin-fst-mismatch-all-len = bin-mismatch-all-len-bool[of\ -\ True\ -\ True, unfolded\ bin-code-simps]$ **and**
 $bin-snd-mismatch-all-len = bin-mismatch-all-len-bool[of\ -\ False\ -\ True, unfolded\ bin-code-simps]$

lemma *bin-code-delay*: **assumes** $|\alpha| \leq |q_0|$ **and** $|\alpha| \leq |q_1|$ **and**
 $q_0 \leq_p u_0 \cdot w_0$ **and** $q_1 \leq_p u_1 \cdot w_1$ **and**
 $w_0 \in \langle \{u_0, u_1\} \rangle$ **and** $w_1 \in \langle \{u_0, u_1\} \rangle$
shows $q_0 \wedge_p q_1 = \alpha$
 $\langle \text{proof} \rangle$

lemma *hd-lq-mismatch-fst*: $hd (\alpha^{-1}\rangle (u_0 \cdot \alpha)) = c_0$
 $\langle \text{proof} \rangle$

lemma *hd-lq-mismatch-snd*: $hd (\alpha^{-1}\rangle (u_1 \cdot \alpha)) = c_1$
 $\langle \text{proof} \rangle$

lemma *hds-bin-mismatch-neq*: $hd (\alpha^{-1}\rangle (u_0 \cdot \alpha)) \neq hd (\alpha^{-1}\rangle (u_1 \cdot \alpha))$
 $\langle \text{proof} \rangle$

lemma *bin-lcp-fst-pow-pref*: **assumes** $0 < k$ **shows** $\alpha \cdot [c_0] \leq_p u_0^{\textcircled{k}} \cdot u_1 \cdot z$
 $\langle \text{proof} \rangle$

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*: $\alpha \leq_p u_1 \cdot \alpha$
 $\langle \text{proof} \rangle$

lemma *bin-lcp-pref-all-set*: **assumes** $set\ ws = \{u_0, u_1\}$
shows $\alpha \leq_p concat\ ws$
 $\langle \text{proof} \rangle$

lemma *bin-lcp-conjug-morph*:
assumes $u \in \langle \{u_0, u_1\} \rangle$ **and** $v \in \langle \{u_0, u_1\} \rangle$
shows $\alpha^{-1}\rangle (u \cdot \alpha) \cdot \alpha^{-1}\rangle (v \cdot \alpha) = \alpha^{-1}\rangle ((u \cdot v) \cdot \alpha)$
 $\langle \text{proof} \rangle$

lemma *lcp-bin-conjug-prim-iff*:
 $set\ ws = \{u_0, u_1\} \implies primitive (\alpha^{-1}\rangle (concat\ ws) \cdot \alpha) \iff primitive (concat\ ws)$
 $\langle \text{proof} \rangle$

lemma *bin-lcp-conjug-inj-on*: $inj\text{-on } (\lambda u. \alpha^{-1}\rangle (u \cdot \alpha)) \langle \{u_0, u_1\} \rangle$
 $\langle \text{proof} \rangle$

lemma *bin-code-lcp-marked*: **assumes** $us \in lists\ \{u_0, u_1\}$ **and** $vs \in lists\ \{u_0, u_1\}$
and $hd\ us \neq hd\ vs$
shows $concat\ us \cdot \alpha \wedge_p concat\ vs \cdot \alpha = \alpha$
 $\langle \text{proof} \rangle$

lemma **assumes** $us \in lists\ \{u_0, u_1\}$ **and** $vs \in lists\ \{u_0, u_1\}$ **and** $hd\ us \neq hd\ vs$
shows $concat\ us \cdot \alpha \wedge_p concat\ vs \cdot \alpha = \alpha$
 $\langle \text{proof} \rangle$

lemma bin-code-lcp-concat: **assumes** $us \in \text{lists } \{u_0, u_1\}$ **and** $vs \in \text{lists } \{u_0, u_1\}$
and $\neg us \bowtie vs$
shows $\text{concat } us \cdot \alpha \wedge_p \text{concat } vs \cdot \alpha = \text{concat } (us \wedge_p vs) \cdot \alpha$
 $\langle \text{proof} \rangle$

lemma bin-code-lcp-concat': **assumes** $us \in \text{lists } \{u_0, u_1\}$ **and** $vs \in \text{lists } \{u_0, u_1\}$
and $\neg \text{concat } us \bowtie \text{concat } vs$
shows $\text{concat } us \wedge_p \text{concat } vs = \text{concat } (us \wedge_p vs) \cdot \alpha$
 $\langle \text{proof} \rangle$

lemma bin-lcp-pows: $0 < k \implies 0 < l \implies u_0^{\textcircled{k}} \cdot u_1 \cdot z \wedge_p u_1^{\textcircled{l}} \cdot u_0 \cdot z' = \alpha$
 $\langle \text{proof} \rangle$

theorem bin-code: **assumes** $us \in \text{lists } \{u_0, u_1\}$ **and** $vs \in \text{lists } \{u_0, u_1\}$ **and** $\text{concat } us = \text{concat } vs$
shows $us = vs$
 $\langle \text{proof} \rangle$

lemma code-bin-roots: $\text{binary-code } (\varrho u_0) (\varrho u_1)$
 $\langle \text{proof} \rangle$

sublocale $\text{code } \{u_0, u_1\}$
 $\langle \text{proof} \rangle$

lemma primroot-dec: $(\text{Dec } \{\varrho u_0, \varrho u_1\} u_0) = [\varrho u_0]^{\textcircled{e}} e_{\varrho} u_0 (\text{Dec } \{\varrho u_0, \varrho u_1\} u_1)$
 $= [\varrho u_1]^{\textcircled{e}} e_{\varrho} u_1$
 $\langle \text{proof} \rangle$

lemma bin-code-prefs: **assumes** $w \in \langle \{u_0, u_1\} \rangle$ **and** $p \leq_p w$ $w' \in \langle \{u_0, u_1\} \rangle$ **and**
 $|u_1| \leq |p|$
shows $\neg u_0 \cdot p \leq_p u_1 \cdot w'$
 $\langle \text{proof} \rangle$

lemma bin-code-rev: $\text{binary-code } (\text{rev } u_0) (\text{rev } u_1)$
 $\langle \text{proof} \rangle$

lemma bin-lcp-pows-lcp: $0 < k \implies 0 < l \implies u_0^{\textcircled{k}} \cdot u_1^{\textcircled{l}} \wedge_p u_1^{\textcircled{l}} \cdot u_0^{\textcircled{k}} = u_0$
 $\cdot u_1 \wedge_p u_1 \cdot u_0$
 $\langle \text{proof} \rangle$

lemma bin-mismatch: $u_0 \cdot \alpha \wedge_p u_1 \cdot \alpha = \alpha$
 $\langle \text{proof} \rangle$

lemma not-comp-bin-fst-snd: $\neg u_0 \cdot \alpha \bowtie u_1 \cdot \alpha$
 $\langle \text{proof} \rangle$

theorem bin-bounded-delay: **assumes** $z \leq_p u_0 \cdot w_0$ **and** $z \leq_p u_1 \cdot w_1$
and $w_0 \in \langle \{u_0, u_1\} \rangle$ **and** $w_1 \in \langle \{u_0, u_1\} \rangle$

shows $|z| \leq |\alpha|$
 ⟨proof⟩

thm *binary-code.bin-lcp-pows-lcp*

lemma *prim-roots-lcp: bin-lcp* $(\varrho u_0) (\varrho u_1) = \alpha$
 ⟨proof⟩

lemma *bin-roots-decompose:*

Dec $\{\varrho u_0, u_1\} u_0 = [\varrho u_0]^{\textcircled{a}} e_{\varrho} u_0$

Dec $\{\varrho u_0, u_1\} u_1 = [u_1]$

Dec $\{u_0, \varrho u_1\} u_1 = [\varrho u_1]^{\textcircled{a}} e_{\varrho} u_1$

Dec $\{u_0, \varrho u_1\} u_0 = [u_0]$

Dec $\{u_0, u_1\} u_0 = [u_0]$

Dec $\{u_0, u_1\} u_1 = [u_1]$

⟨proof⟩

lemma *ref-fst-sq:* *Ref* $\{\varrho u_0, u_1\}[u_0, u_0] = [\varrho u_0]^{\textcircled{a}} (e_{\varrho} u_0 * \varrho)$
 ⟨proof⟩

lemma *ref-fst-pow:* *Ref* $\{\varrho u_0, u_1\}[u_0]^{\textcircled{a}} k = [\varrho u_0]^{\textcircled{a}} (e_{\varrho} u_0 * k)$
 ⟨proof⟩

lemma *bin-code-concat-len:* **assumes** $ws \in \text{lists } \{u_0, u_1\}$

shows $|\text{concat } ws| = \text{count-list } ws u_0 * |u_0| + \text{count-list } ws u_1 * |u_1|$

⟨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 $u_1 \cdot q \wedge_p r \cdot u_1 \cdot q = \text{take } |u_1 \cdot q| \alpha$

⟨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 $u_1 \cdot q \wedge_p r \cdot u_1 \cdot q = \text{take } |u_1 \cdot q| \alpha$

⟨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:* $u_0 \cdot u_1 <_p r \cdot u_0 \cdot u_1 \implies \alpha \cdot [c_0] \leq_p r \cdot \alpha$
 ⟨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** $|\alpha| < |u_1 \cdot q|$

$q' \leq p \ w' \text{ and } w' \in \langle \{u_0, u_1\} \rangle \text{ and } |\alpha| \leq |q'|$

shows $u_1 \cdot q \wedge_p r \cdot q' = \alpha$
 $\langle \text{proof} \rangle$

end

lemmas allowing to translate properties of binary code to its roots

named-theorems *bin-code-primroots*

lemma *bin-lcp-eq-primroots* [*bin-code-primroots*]: **assumes** $x \cdot y \neq y \cdot x$
shows $\text{bin-lcp } (\varrho x) (\varrho y) = \text{bin-lcp } x y$
 $\langle \text{proof} \rangle$

lemma *bin-lcs-eq-primroots* [*bin-code-primroots*]: **assumes** $x \cdot y \neq y \cdot x$
shows $\text{bin-lcs } (\varrho x) (\varrho y) = \text{bin-lcs } x y$
 $\langle \text{proof} \rangle$

lemma *bin-mismatch-fst-eq-primroots* [*bin-code-primroots*]: **assumes** $x \cdot y \neq y \cdot x$
shows $\text{bin-mismatch } (\varrho x) (\varrho y) = \text{bin-mismatch } x y$
 $\langle \text{proof} \rangle$

lemmas *bin-mismatch-snd-eq-primroots*[*bin-code-primroots*] = *bin-mismatch-fst-eq-primroots*[*OF not-sym*] **and**
bin-mismatch-suf-fst-eq-primroots[*bin-code-primroots*] = *bin-mismatch-fst-eq-primroots*[*reversed*]
and
bin-mismatch-suf-snd-primroots[*bin-code-primroots*] = *bin-mismatch-fst-eq-primroots*[*reversed, OF not-sym*]

lemma *bin-lcp-eq-primroots'* [*bin-code-primroots*]: $x \cdot y \neq y \cdot x \implies \varrho x \cdot \varrho y \wedge_p \varrho y \cdot \varrho x = x \cdot y \wedge_p y \cdot x$
 $\langle \text{proof} \rangle$

lemmas *no-comm-bin-code* = *binary-code.bin-code*[*unfolded binary-code-def*]

theorem *bin-code-code*[*intro*]: **assumes** $u \cdot v \neq v \cdot u$ **shows** $\text{code } \{u, v\}$
 $\langle \text{proof} \rangle$

lemma *code-bin-code*: $u \neq v \implies \text{code } \{u, v\} \implies u \cdot v \neq v \cdot u$
 $\langle \text{proof} \rangle$

lemma *lcp-roots-lcp*: $x \cdot y \neq y \cdot x \implies x \cdot y \wedge_p y \cdot x = \varrho x \cdot \varrho y \wedge_p \varrho y \cdot \varrho x$
 $\langle \text{proof} \rangle$

lemma *sing-gen-primroot* [*simp*]: $u \in \langle \{\varrho u\} \rangle$
 $\langle \text{proof} \rangle$

lemma *sing-gen-pref-cancel* [*elim*]: $u \cdot v \in \langle \{r\} \rangle \implies u \in \langle \{r\} \rangle \implies v \in \langle \{r\} \rangle$
 ⟨*proof*⟩

lemma *sing-gen-suf-cancel* [*elim*]: $u \cdot v \in \langle \{r\} \rangle \implies v \in \langle \{r\} \rangle \implies u \in \langle \{r\} \rangle$
 ⟨*proof*⟩

lemma *prim-comm-root*[*elim*]: **assumes** *primitive r* **and** $u \cdot r = r \cdot u$ **shows** $u \in \langle \{r\} \rangle$
 ⟨*proof*⟩

lemma *prim-root-drop-exp*[*elim*]: **assumes** $u^{\textcircled{k}} \in \langle \{r\} \rangle$ **and** $0 < k$ **and** *primitive r*
shows $u \in \langle \{r\} \rangle$
 ⟨*proof*⟩

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

lemma *per-root-trans'*[*intro*]: $w \leq_p u \cdot w \implies u \in \langle \{r\} \rangle \implies u \neq \varepsilon \implies w \leq_p r \cdot w$
 ⟨*proof*⟩

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

lemma *per-root-pref*: $w \neq \varepsilon \implies w \in \langle \{r\} \rangle \implies r \leq_p w$
 ⟨*proof*⟩

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

lemma *comm-primrootE*: **assumes** $x \cdot y = y \cdot x$
obtains t **where** $x \in \langle \{t\} \rangle$ **and** $y \in \langle \{t\} \rangle$ **and** *primitive t*
 ⟨*proof*⟩

lemma *root-trans*[*trans*]: $\llbracket v \in \langle \{u\} \rangle; u \in \langle \{t\} \rangle \rrbracket \implies v \in \langle \{t\} \rangle$
 ⟨*proof*⟩

lemma *root-rev-iff*[*reversal-rule*]: $((\text{rev } u) \in \langle \{\text{rev } t\} \rangle) \longleftrightarrow (u \in \langle \{t\} \rangle)$
 ⟨*proof*⟩

3.9 Two words hull (not necessarily a code)

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

lemma *two-elem-first-block*: **assumes** $w \in \langle \{u, v\} \rangle$

obtains m **where** $u^{\textcircled{m}} \cdot v \leq_p w \vee w = u^{\textcircled{m}}$
 ⟨proof⟩

lemmas $two\text{-}elem\text{-}last\text{-}block = two\text{-}elem\text{-}first\text{-}block[reversed]$

lemma $two\text{-}elem\text{-}pref$: **assumes** $v \leq_p u \cdot p$ **and** $p \in \langle \{u, v\} \rangle$
shows $v \leq_p u \cdot v$
 ⟨proof⟩

lemmas $two\text{-}elem\text{-}suf = two\text{-}elem\text{-}pref[reversed]$

lemma $gen\text{-}drop\text{-}exp$: **assumes** $p \in \langle \{u, v^{\textcircled{k}}(Suc\ k)\} \rangle$ **shows** $p \in \langle \{u, v\} \rangle$
 ⟨proof⟩

lemma $gen\text{-}drop\text{-}exp\text{-}pos$: **assumes** $p \in \langle \{u, v^{\textcircled{k}}\} \rangle$ $0 < k$ **shows** $p \in \langle \{u, v\} \rangle$
 ⟨proof⟩

lemma $gen\text{-}prim$: $p \in \langle \{u, v\} \rangle \implies p \in \langle \{u, \varrho\ v\} \rangle$
 ⟨proof⟩

lemma $roots\text{-}hull$: **assumes** $w \in \langle \{u^{\textcircled{k}}, v^{\textcircled{m}}\} \rangle$ **shows** $w \in \langle \{u, v\} \rangle$
 ⟨proof⟩

lemma $in\text{-}hull\text{-}primroots$: $w \in \langle \{x, y\} \rangle \implies w \in \langle \{\varrho\ x, \varrho\ y\} \rangle$
 ⟨proof⟩

lemma $roots\text{-}hull\text{-}sub$: $\langle \{u^{\textcircled{k}}, v^{\textcircled{m}}\} \rangle \subseteq \langle \{u, v\} \rangle$
 ⟨proof⟩

lemma $primroot\text{-}gen[simp, intro]$: $v \in \langle \{u, \varrho\ v\} \rangle$
 ⟨proof⟩

lemma $primroot\text{-}gen'[simp, intro]$: $u \in \langle \{\varrho\ u, v\} \rangle$
 ⟨proof⟩

lemma $lists\text{-}lists\text{-}gen\text{-}primroots [intro]$: $u \in lists\ \{x, y\} \implies u \in lists\ \langle \{\varrho\ x, \varrho\ y\} \rangle$
 ⟨proof⟩

lemma $dec\text{-}primroot\text{-}bin\text{-}sing [intro]$: **assumes** $a \in \{x, y\}$ $x \cdot y \neq y \cdot x$
shows $Dec\ \{\varrho\ x, \varrho\ y\}\ a = [\varrho\ a]^{\textcircled{e_\varrho}}\ a$
 ⟨proof⟩

lemma $ref\text{-}primroot\text{-}bin\text{-}sing$: **assumes** $a \in \{x, y\}$ $x \cdot y \neq y \cdot x$
shows $Ref\ \{\varrho\ x, \varrho\ y\}\ [a] = [\varrho\ a]^{\textcircled{e_\varrho}}\ a$
 ⟨proof⟩

lemma $lists\text{-}sub\text{-}mono\text{-}gen$: $ws \in lists\ S \implies S \subseteq \langle G \rangle \implies ws \in lists\ \langle G \rangle$
 ⟨proof⟩

3.9.1 Binary Mismatch tools

definition *bin-mismatch-hard* :: 'a list \Rightarrow 'a list \Rightarrow 'a list \Rightarrow bool **where**
bin-mismatch-hard $x\ y\ w \equiv \exists k. \varrho\ x \cdot (\varrho\ x)^{\textcircled{k}} \cdot \varrho\ y \leq_p w$

lemma *bin-mismatch-hard-def'*: **assumes** $x \cdot y \neq y \cdot x$ *bin-mismatch-hard* $x\ y\ w$
shows *bin-lcp* $x\ y \cdot [\textit{bin-mismatch}\ x\ y] \leq_p w$
<proof>

definition *bin-mismatch-pref* :: 'a list \Rightarrow 'a list \Rightarrow 'a list \Rightarrow bool **where**
bin-mismatch-pref $x\ y\ w \equiv \exists k. \varrho\ x^{\textcircled{k}} \cdot \varrho\ y \leq_p w$

lemma *bm-pref-letter*: **assumes** $x \cdot y \neq y \cdot x$ **and** *bin-mismatch-pref* $x\ y\ (w \cdot \varrho\ y)$
shows *bin-lcp* $x\ y \cdot [\textit{bin-mismatch}\ x\ y] \leq_p x \cdot w \cdot \textit{bin-lcp}\ x\ y$ (**is** $?\alpha \cdot [?c] \leq_p x \cdot w \cdot ?\alpha$)
<proof>

named-theorems *bm-elim*s

lemma *bm-eq1* [*bm-elim*s]: **assumes** $x \cdot w1 = y \cdot w2$ **and** *bin-mismatch-pref* $x\ y\ (w1 \cdot \varrho\ y)$ **and** *bin-mismatch-pref* $y\ x\ (w2 \cdot \varrho\ x)$
shows $x \cdot y = y \cdot x$
<proof>

lemma *bm-eq2* [*bm-elim*s]: **assumes** $\varrho\ x \cdot w1 = y \cdot w2$ **and** *bin-mismatch-pref* $x\ y\ (w1 \cdot \varrho\ y)$ **and** *bin-mismatch-pref* $y\ x\ (w2 \cdot \varrho\ x)$
shows $x \cdot y = y \cdot x$
<proof>

lemma *bm-eq3* [*bm-elim*s]: **assumes** $x \cdot w1 = \varrho\ y \cdot w2$ **and** *bin-mismatch-pref* $x\ y\ (w1 \cdot \varrho\ y)$ **and** *bin-mismatch-pref* $y\ x\ (w2 \cdot \varrho\ x)$
shows $x \cdot y = y \cdot x$
<proof>

lemma *bm-eq4* [*bm-elim*s]: **assumes** $\varrho\ x \cdot w1 = \varrho\ y \cdot w2$ **and** *bin-mismatch-pref* $x\ y\ (w1 \cdot \varrho\ y)$ **and** *bin-mismatch-pref* $y\ x\ (w2 \cdot \varrho\ x)$
shows $x \cdot y = y \cdot x$
<proof>

lemma *bm-hard-lcp* [*bm-elim*s]: **assumes** $x \cdot y \neq y \cdot x$ **and** *bin-mismatch-hard* $x\ y\ w1$ **and** *bin-mismatch-hard* $y\ x\ w2$
shows $w1 \wedge_p w2 = x \cdot y \wedge_p y \cdot x$
<proof>

lemma *bin-mismatch-pref-ext*: *bin-mismatch-pref* $x\ y\ w1 \implies \textit{bin-mismatch-pref}\ x\ y\ (w1 \cdot z)$
<proof>

lemma *bm-pref1* [*bm-elim*s]: **assumes** $x \cdot w1 \leq_p y \cdot w2$ **and** *bin-mismatch-pref* $x\ y\ w1$

and *bin-mismatch-pref* $y\ x\ (w2 \cdot \rho\ x)$
shows $x \cdot y = y \cdot x$
 \langle *proof* \rangle

lemma *bm-pref2* [*bm-elim*]: **assumes** $\rho\ x \cdot w1 \leq p\ y \cdot w2$ **and** *bin-mismatch-pref* $x\ y\ w1$
and *bin-mismatch-pref* $y\ x\ (w2 \cdot \rho\ x)$
shows $x \cdot y = y \cdot x$
 \langle *proof* \rangle

lemma *bm-pref3* [*bm-elim*]: **assumes** $x \cdot w1 \leq p\ \rho\ y \cdot w2$ **and** *bin-mismatch-pref* $x\ y\ w1$
and *bin-mismatch-pref* $y\ x\ (w2 \cdot \rho\ x)$
shows $x \cdot y = y \cdot x$
 \langle *proof* \rangle

lemma *bm-pref4* [*bm-elim*]: **assumes** $\rho\ x \cdot w1 \leq p\ \rho\ y \cdot w2$ **and** *bin-mismatch-pref* $x\ y\ w1$
and *bin-mismatch-pref* $y\ x\ (w2 \cdot \rho\ x)$
shows $x \cdot y = y \cdot x$
 \langle *proof* \rangle

lemmas [*bm-elim*] = *bm-elim*[*symmetric*]

— Binary mismatch predicate evaluation

named-theorems *bm-simps*

lemma *bm-mismatch-rhoI1* [*bm-simps*]: **assumes** *bin-mismatch-pref* $x\ y\ w$ **shows** *bin-mismatch-hard* $x\ y\ (\rho\ x \cdot w)$
 \langle *proof* \rangle

lemma *bm-mismatchI1* [*bm-simps*]: **assumes** *bin-mismatch-pref* $x\ y\ w$ **shows** *bin-mismatch-hard* $x\ y\ (x \cdot w)$
 \langle *proof* \rangle

lemma *bm-mismatch-rhoI2'* [*bm-simps*]: **assumes** *bin-mismatch-pref* $x\ y\ w$ **shows** *bin-mismatch-pref* $x\ y\ (\rho\ x^{\textcircled{k}} \cdot w)$
 \langle *proof* \rangle

lemma *bm-mismatch-rhoI2* [*bm-simps*]: **assumes** *bin-mismatch-pref* $x\ y\ w$ **shows** *bin-mismatch-pref* $x\ y\ (\rho\ x \cdot w)$
 \langle *proof* \rangle

lemma *bm-mismatchI2* [*bm-simps*]: **assumes** *bin-mismatch-pref* $x\ y\ w$ **shows** *bin-mismatch-pref* $x\ y\ (x \cdot w)$
 \langle *proof* \rangle

lemma *bm-mismatchI2'* [*bm-simps*]: **assumes** *bin-mismatch-pref* $x\ y\ w$ **shows** *bin-mismatch-pref* $x\ y\ (x^{\textcircled{k}} \cdot w)$

<proof>

lemma [bm-simps]: *bin-mismatch-pref* $x\ y\ (y \cdot v)$
<proof>

lemma [bm-simps]: *bin-mismatch-pref* $x\ y\ y$
<proof>

lemma [bm-simps]: **assumes** $w1 \in \langle\{x,y\}\rangle$ *bin-mismatch-pref* $x\ y\ w$
shows *bin-mismatch-pref* $x\ y\ (w1 \cdot w)$
<proof>

lemma *bm-pref-triv-rho* [bm-simps]: *bin-mismatch-pref* $x\ y\ (\rho\ y \cdot w)$
<proof>

lemma *bm-pref-fst-rho* [bm-simps]: *bin-mismatch-pref* $x\ y\ (\rho\ y)$
<proof>

lemma[bm-simps]: $x \in \langle\{x,y\}\rangle$
<proof>

lemma[bm-simps]: $y \in \langle\{x,y\}\rangle$
<proof>

lemma[bm-simps]: $w \in \langle\{x,y\}\rangle \longleftrightarrow w \in \langle\{y,x\}\rangle$
<proof>

lemmas[bm-simps] = *hull-closed power-in*

lemmas [bm-simps] = *lcp-ext-left*

— the method setup

method *mismatch0* =
(*insert method-facts, use nothing in*
 <(*(simp only: shifts bm-simps)?,*
 (elim bm-elim);(simp-all only: shifts bm-simps)
)>
)

lemmas *bm-simps-rev* = *bm-simps[reversed]*

lemmas *bm-elim-rev* = *bm-elim[reversed]*

method *mismatch-rev* =
(*insert method-facts, use nothing in*
 <(*(simp only: shifts-rev bm-simps-rev)?,*
 (elim bm-elim-rev);(simp-all only: shifts-rev bm-simps-rev)
)>
)

method *mismatch* =
 (insert method-facts, use nothing in
 ⟨(mismatch0;fail|mismatch-rev)⟩
)

find-theorems name: *bm-elim* ? $x \wedge_p$? y

Mismatch method demonstrations

lemma assumes $x \cdot y \cdot z = y \cdot y \cdot x \cdot v$ shows $x \cdot y = y \cdot x$
 ⟨proof⟩

lemma assumes $y \cdot x \cdot x \cdot y \cdot z = (y \cdot x \cdot y) \cdot y \cdot x \cdot v$ shows $x \cdot y = y \cdot x$ —
 cancel
 ⟨proof⟩

lemma $y \cdot y \cdot x \cdot v = x \cdot x \cdot y \cdot z \implies x \cdot y = y \cdot x$ — swap
 ⟨proof⟩

lemma $\varrho x \cdot \varrho x^{\textcircled{k}} \cdot y = y \cdot w \cdot \varrho x \implies w \in \langle\{x,y\}\rangle \implies x \cdot y = y \cdot x$ — primitive
 root and hull
 ⟨proof⟩

lemma $(r \cdot q) \cdot (r \cdot q)^{\textcircled{k}} \cdot t \cdot r \cdot (q \cdot r \cdot r \cdot q)^{\textcircled{k}} \cdot k = ((r \cdot q)^{\textcircled{k}} \cdot t \cdot r \cdot (q \cdot r \cdot r \cdot q)^{\textcircled{k}} \cdot k) \cdot r \cdot q \implies$
 $0 < k \implies r \cdot q = q \cdot r$ — power
 ⟨proof⟩

lemma $((u \cdot v)^{\textcircled{k}} \cdot m \cdot u) \cdot v \cdot (u \cdot v)^{\textcircled{k}} \cdot k = (v \cdot (u \cdot v)^{\textcircled{k}} \cdot k) \cdot (u \cdot v)^{\textcircled{k}} \cdot m \cdot u \implies$
 $u \cdot v = v \cdot u$
 ⟨proof⟩

lemma $w1 \in \langle\{x,y\}\rangle \implies y \cdot x \cdot w2 \cdot z = x \cdot w1 \implies x \cdot y = y \cdot x$
 ⟨proof⟩

lemma $x \cdot y \cdot u \cdot x \cdot y = y \cdot v \cdot y \cdot x \implies x \cdot y = y \cdot x$ — reverse mismatch
 ⟨proof⟩

lemma $z \cdot x \cdot y \cdot x \cdot x = v \cdot x \cdot y \cdot y \implies y \cdot x = x \cdot y$ — reverse mismatch
 ⟨proof⟩

lemma $k \neq 0 \implies j \neq 0 \implies (x^{\textcircled{j}} \cdot y^{\textcircled{k}}) \cdot y = y^{\textcircled{k}} \cdot x^{\textcircled{j}} \cdot y^{\textcircled{k-1}} \implies$
 $x \cdot y = y \cdot x$
 ⟨proof⟩

lemma $\varrho x \cdot \varrho x^{\textcircled{k}} \cdot y \leq_p y \cdot w \cdot \varrho x \implies w \in \langle\{x,y\}\rangle \implies x \cdot y = y \cdot x$
 ⟨proof⟩

lemma $y \cdot x \leq_p x^{\textcircled{k}} \cdot x \cdot y \cdot w \implies x \cdot y = y \cdot x$
 ⟨proof⟩

lemma $0 < k \implies y \cdot x \leq_p x^{\textcircled{k}} \cdot y \cdot w \implies x \cdot y = y \cdot x$
 ⟨proof⟩

lemma $0 < k \implies 0 < l \implies y^{\textcircled{l}} \cdot x \leq_p x^{\textcircled{k}} \cdot y \cdot w \implies x \cdot y = y \cdot x$
 ⟨proof⟩

lemma $0 < k \implies 1 < l \implies y^{\textcircled{l}} \cdot x \leq_p y \cdot x^{\textcircled{k}} \cdot y \cdot w \implies x \cdot y = y \cdot x$
 ⟨proof⟩

lemma $0 < k \implies m < l \implies y^{\textcircled{l}} \cdot x \leq_p y^{\textcircled{m}} \cdot x^{\textcircled{k}} \cdot y \cdot w \implies x \cdot y = y \cdot x$
 ⟨proof⟩

lemma **assumes** $0 < k \ m < l \ y^{\textcircled{l}} \cdot x \leq_p y^{\textcircled{m}} \cdot x^{\textcircled{k}} \cdot y \cdot w$ **shows** $x \cdot y = y \cdot x$
 ⟨proof⟩

lemma $w1 \in \langle \{x, y\} \rangle \implies w2 \in \langle \{x, y\} \rangle \implies x \cdot w2 \cdot y \cdot z = y \cdot w1 \cdot x \cdot v \implies x \cdot y = y \cdot x$
 ⟨proof⟩

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** $x \cdot y = y \cdot x$
 ⟨proof⟩

lemma $u \cdot v^{\textcircled{k}} \cdot v \leq_s v \cdot w \cdot u \implies w \in \langle \{u, v\} \rangle \implies u \cdot v = v \cdot u$
 ⟨proof⟩

lemma $u \cdot \rho v^{\textcircled{k}} \cdot \rho v \leq_s \rho v \cdot w \cdot u \implies w \in \langle \{u, v\} \rangle \implies u \cdot v = v \cdot u$
 ⟨proof⟩

lemma $u \cdot v^{\textcircled{k}} \cdot \rho v \leq_s \rho v \cdot w \cdot u \implies w \in \langle \{u, v\} \rangle \implies u \cdot v = v \cdot u$
 ⟨proof⟩

lemma $2 \leq j \implies q \cdot p \cdot q \leq_s (q \cdot p)^{\textcircled{2}} \cdot q^{\textcircled{j}} \implies p \cdot q = q \cdot p$
 ⟨proof⟩

lemma $w1 \in \langle \{x, y\} \rangle \implies w2 \in \langle \{x, y\} \rangle \implies x \cdot y \cdot w2 \cdot x \leq_s x \cdot w1 \cdot y \implies x \cdot y = y \cdot x$
 ⟨proof⟩

lemma $w \in \langle \{x, y\} \rangle \implies w' \in \langle \{x, y\} \rangle \implies \text{bin-mismatch-pref } x \ y \ (w \cdot w \cdot y)$
 ⟨proof⟩

lemma $x \cdot y = y^{\textcircled{k}} \implies 0 < k \implies x \cdot y = y \cdot x$
 ⟨proof⟩

lemma **assumes** $x \cdot y \neq y \cdot x$
shows $x \cdot x \cdot y \wedge_p y \cdot y \cdot x = (x \cdot y \wedge_p y \cdot x)$
 ⟨proof⟩

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⟩

lemma assumes $x \cdot y \neq y \cdot x$
shows $y \cdot y \cdot x \wedge_p x \cdot x \cdot y = (x \cdot y \wedge_p y \cdot x)$
 ⟨proof⟩

lemma assumes $x \cdot y \neq y \cdot x$
shows $x \cdot x \cdot y \wedge_s y \cdot y \cdot x = (x \cdot y \wedge_s y \cdot x)$
 ⟨proof⟩

lemma assumes $x \cdot y \neq y \cdot x$
shows $x \cdot x \cdot y \cdot z \cdot y \wedge_s y \cdot y \cdot x \cdot z \cdot y = (x \cdot y \wedge_s y \cdot x) \cdot (z \cdot y)$
 ⟨proof⟩

3.9.2 Applied mismatch

lemma assumes $x^{\textcircled{a}j} = y^{\textcircled{a}k} \ 0 < j$ **shows** $x \cdot y = y \cdot x$
 ⟨proof⟩

lemma *pows-comm-comm*: **assumes** $u^{\textcircled{a}k} \cdot v^{\textcircled{a}m} = u^{\textcircled{a}l} \cdot v^{\textcircled{a}n} \ k \neq l$ **shows** $u \cdot v = v \cdot u$
 ⟨proof⟩

lemma *sq-not-pref-mesosome-cover*: **assumes** *eq*: $x \cdot x \cdot \text{concat } ws = p \cdot x \cdot y^{\textcircled{a}k}$
 $\cdot x \cdot s$ **and** $ws \in \text{lists } \{x, y\} \ p <_p \ x \ p \neq \varepsilon \ \langle 1 < k \rangle$
shows $x \cdot y = y \cdot x$
 ⟨proof⟩

lemma (in *binary-code*) *bin-mismatch-pows*: $u_0^{\textcircled{a} \text{Suc } k} \cdot u_1 \cdot z \neq u_1^{\textcircled{a} \text{Suc } l} \cdot u_0 \cdot z'$
 ⟨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 \Rightarrow 'a list set ($\langle _ \rangle_F$)
for G **where**
 $\varepsilon \in \langle G \rangle_F$
 | *free-gen-in*: $w \in G \implies w \in \langle G \rangle_F$
 | $w1 \in \langle G \rangle_F \implies w2 \in \langle G \rangle_F \implies w1 \cdot w2 \in \langle G \rangle_F$
 | *stability*: $p \in \langle G \rangle_F \implies q \in \langle G \rangle_F \implies p \cdot w \in \langle G \rangle_F \implies w \cdot q \in \langle G \rangle_F \implies w \in \langle G \rangle_F$ — 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. (See the AFP theory Combinatorics-Words-Graph-Lemma.Graph-Lemma)

definition *free-basis* :: 'a list set \Rightarrow '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:

- inductive point of view: to satisfy the stability condition;
- being generated by a code.

We now show their equivalence

First, basis of a free hull is a code.

lemma *free-basis-code[simp]*: *code* ($\mathfrak{B}_F G$)
 ⟨proof⟩

lemma *gen-in-free-hull*: $x \in G \implies x \in \langle \mathfrak{B}_F G \rangle$
 ⟨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*: $p \in \langle \mathcal{C} \rangle \implies q \in \langle \mathcal{C} \rangle \implies p \cdot w \in \langle \mathcal{C} \rangle \implies w \cdot q \in \langle \mathcal{C} \rangle$
 $\implies w \in \langle \mathcal{C} \rangle$
 ⟨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*: **assumes** $G \subseteq H$ **shows** $\langle G \rangle_F \subseteq \langle H \rangle_F$
<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*: $u \in \langle G \rangle_F \implies v \in \langle G \rangle_F \implies$
 $Dec (\mathfrak{B}_F G) (u \cdot v) = (Dec (\mathfrak{B}_F G) u) \cdot (Dec (\mathfrak{B}_F G) v)$
<proof>

3.11 Reversing hulls and decompositions

lemma *basis-rev-commute*[*reversal-rule*]: $\mathfrak{B} (\text{rev } \text{' } G) = \text{rev } \text{' } (\mathfrak{B} G)$
<proof>

lemma *rev-free-hull-comm*: $\langle \text{rev } \text{' } X \rangle_F = \text{rev } \text{' } \langle X \rangle_F$
<proof>

lemma *free-basis-rev-commute* [*reversal-rule*]: $\mathfrak{B}_F \text{rev } \text{' } X = \text{rev } \text{' } (\mathfrak{B}_F X)$
<proof>

lemma *rev-dec*[*reversal-rule*]: **assumes** $x \in \langle X \rangle_F$ **shows** $Dec \text{rev } \text{' } (\mathfrak{B}_F X) (\text{rev } x) = \text{map } \text{rev } (\text{rev } (Dec (\mathfrak{B}_F X) x))$
<proof>

lemma *rev-hd-dec-last-eq*[*reversal-rule*]: **assumes** $x \in X$ **and** $x \neq \varepsilon$ **shows**
 $\text{rev } (\text{hd } (Dec (\text{rev } \text{' } (\mathfrak{B}_F X)) (\text{rev } x))) = \text{last } (Dec \mathfrak{B}_F X x)$
<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** $sings\ B \equiv \{[b] \mid b. b \in B\}$

term *Set.filter* $P\ A$

lemma *sings-image*: $sings\ B = (\lambda\ x. [x])\ ' B$
 ⟨proof⟩

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

lemma *code-sings*: $code\ (sings\ B)$
 ⟨proof⟩

lemma *sings-gen-lists*: $\langle sings\ B \rangle = lists\ B$
 ⟨proof⟩

lemma *sing-gen-lists*: $lists\ \{x\} = \langle \{[x]\} \rangle$
 ⟨proof⟩

lemma *bin-gen-lists*: $lists\ \{x, y\} = \langle \{[x], [y]\} \rangle$
 ⟨proof⟩

lemma $sings\ B = \mathfrak{B}_F\ (lists\ B)$
 ⟨proof⟩

lemma *map-sings*: $xs \in lists\ B \implies map\ (\lambda x. x \# \varepsilon)\ xs \in lists\ (sings\ B)$
 ⟨proof⟩

lemma *dec-sings*: $xs \in lists\ B \implies Dec\ (sings\ B)\ xs = map\ (\lambda\ x. [x])\ xs$
 ⟨proof⟩

lemma *sing-lists-exp*: **assumes** $ws \in lists\ \{a\}$
obtains k **where** $ws = [a]^{\textcircled{k}}$
 ⟨proof⟩

lemma *sing-lists-exp-len*: $ws \in lists\ \{a\} \longleftrightarrow [a]^{\textcircled{|ws|}} = ws$
 ⟨proof⟩

lemma *sing-lists-exp-count*: $ws \in lists\ \{a\} \longleftrightarrow [a]^{\textcircled{(count-list\ ws\ a)}} = ws$

<proof>

lemma *sing-set-pow-count-list*: $set\ ws \subseteq \{a\} \longleftrightarrow [a]^{\textcircled{a}}(\text{count-list}\ ws\ a) = ws$
<proof>

lemma *count-le-length-iff*: $set\ ws \subseteq \{a\} \longleftrightarrow \text{count-list}\ ws\ a = |ws|$
<proof>

lemma *sing-set-pow*: $set\ ws \subseteq \{a\} \longleftrightarrow [a]^{\textcircled{a}}|ws| = ws$
<proof>

lemma *count-sing-exp[simp]*: $\text{count-list}\ ([a]^{\textcircled{k}}) a = k$
<proof>

lemma *count-sing-exp'[simp]*: $\text{count-list}\ ([a]) a = 1$
<proof>

lemma *count-sing-distinct[simp]*: $a \neq b \implies \text{count-list}\ ([a]^{\textcircled{k}}) b = 0$
<proof>

lemma *count-sing-distinct'[simp]*: $a \neq b \implies \text{count-list}\ ([a]) b = 0$
<proof>

lemma *sing-letter-imp-prim*: **assumes** $\text{count-list}\ w\ a = 1$ **shows** *primitive w*
<proof>

lemma *prim-abk*: $a \neq b \implies \text{primitive}\ ([a] \cdot [b]^{\textcircled{k}})$
<proof>

lemma *sing-code*: $x \neq \varepsilon \implies \text{code}\ \{x\}$
<proof>

lemma *sings-card*: $\text{card}\ A = \text{card}\ (\text{sings}\ A)$
<proof>

lemma *sings-finite*: $\text{finite}\ A = \text{finite}\ (\text{sings}\ A)$
<proof>

lemma *sings-conv*: $A = B \longleftrightarrow \text{sings}\ A = \text{sings}\ B$
<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** $\text{code}\ \{\varrho\ x, \varrho\ y\}$
<proof>

lemma *primroot-in-set-dec*: **assumes** $x \neq \varepsilon$ **and** $y \neq \varepsilon$ **shows** $\varrho\ x \in \text{set}\ (\text{Dec}\ \{\varrho\$

$x, \varrho y\} x)$
 $\langle proof \rangle$

lemma *primroot-dec*: **assumes** $x \cdot y \neq y \cdot x$
shows $(Dec \{\varrho x, \varrho y\} x) = [\varrho x]^{\textcircled{\varrho}} e_{\varrho} x (Dec \{\varrho x, \varrho y\} y) = [\varrho y]^{\textcircled{\varrho}} e_{\varrho} y$
 $\langle proof \rangle$

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** $[x]^{\textcircled{\varrho}} k \cdot [y] \cdot [x]^{\textcircled{\varrho}} m = ws$
 $\langle proof \rangle$

lemma *bin-count-one-conjug*: **assumes** $ws \in lists \{x,y\}$ **and** $x \neq y$ **and** *count-list*
 $ws \ y = 1$
shows $ws \sim [x]^{\textcircled{\varrho}} (count-list \ ws \ x) \cdot [y]$
 $\langle proof \rangle$

lemma *bin-prim-long-set*: **assumes** $ws \in lists \{x,y\}$ **and** *primitive* ws **and** $2 \leq$
 $|ws|$
shows $set \ ws = \{x,y\}$
 $\langle proof \rangle$

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** $[x,y] \leq_p ws'$
 $\langle proof \rangle$

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 ⇒ 'b list) ⇒ ('a list ⇒ 'b list) (-^ℒ [1000] 1000)
where $t^{\mathcal{L}} \equiv (\lambda x. \text{concat } (\text{map } t \ x))$

definition *morphism-core* :: ('a list ⇒ 'b list) ⇒ ('a ⇒ 'b list) (-^ℒ [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*: $\text{range } (\text{map } f^{\mathcal{C}}) = \text{lists } (\text{range } f^{\mathcal{C}})$
⟨*proof*⟩

lemma *map-core-lists[simp]*: $(\text{map } f^{\mathcal{C}} \ w) \in \text{lists } (\text{range } f^{\mathcal{C}})$
⟨*proof*⟩

lemma *comp-core*: $(f \circ g)^{\mathcal{C}} = f \circ g^{\mathcal{C}}$
⟨*proof*⟩

lemma *ext-core-id [simp]*: $f^{\mathcal{L}\mathcal{C}} = f$
⟨*proof*⟩

locale *morphism-on* =

fixes $f :: 'a \ \text{list} \Rightarrow 'b \ \text{list}$ and $A :: 'a \ \text{list set}$

assumes *morph-on*: $u \in \langle A \rangle \Longrightarrow v \in \langle A \rangle \Longrightarrow f \ (u \cdot v) = f \ u \cdot f \ v$

begin

lemma *emp-to-emp-on[simp]*: $f \ \varepsilon = \varepsilon$
⟨*proof*⟩

lemma *emp-to-emp'*: $w = \varepsilon \implies f w = \varepsilon$
 ⟨proof⟩

lemma *morph-concat-concat-map-on*: $ws \in \text{lists } \langle A \rangle \implies f (\text{concat } ws) = \text{concat } (map f ws)$
 ⟨proof⟩

lemma *hull-im-hull*:
shows $\langle f ' A \rangle = f ' \langle A \rangle$
 ⟨proof⟩

lemma *inj-basis-to-basis*: **assumes** *inj-on* $f \langle A \rangle$
shows $f ' (\mathfrak{B} \langle A \rangle) = \mathfrak{B} (f' \langle A \rangle)$
 ⟨proof⟩

lemma *inj-code-to-code*: **assumes** *inj-on* $f \langle A \rangle$ **and** *code* A
shows *code* $(f ' A)$
 ⟨proof⟩

end

locale *morphism* =
fixes $f :: 'a \text{ list} \Rightarrow 'b \text{ list}$
assumes *morph*: $f (u \cdot v) = f u \cdot f v$
begin

sublocale *morphism-on* $f UNIV$
 ⟨proof⟩

lemmas *emp-to-emp* = *emp-to-emp-on*

lemma *pow-morph*: $f (x^{\textcircled{k}}) = (f x)^{\textcircled{k}}$
 ⟨proof⟩

lemma *rev-map-pow*: $(rev\text{-map } f) (w^{\textcircled{n}}) = rev ((f (rev w))^{\textcircled{n}})$
 ⟨proof⟩

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

lemma *pop-hd-nemp*: $u \neq \varepsilon \implies f (u) = f [hd u] \cdot f (tl u)$
 ⟨proof⟩

lemma *pop-last-nemp*: $u \neq \varepsilon \implies f (u) = f (butlast u) \cdot f [last u]$
 ⟨proof⟩

lemma *pref-mono*: $u \leq_p v \implies f u \leq_p f v$
 ⟨proof⟩

lemma *suf-mono*: $u \leq_s v \implies f u \leq_s f v$
<proof>

lemma *morph-concat-map*: $\text{concat} (\text{map } f^{\mathcal{C}} x) = f x$
<proof>

lemma *morph-concat-map'*: $(\lambda x. \text{concat} (\text{map } f^{\mathcal{C}} x)) = f$
<proof>

lemma *morph-to-concat*:
obtains xs **where** $xs \in \text{lists} (\text{range } f^{\mathcal{C}})$ **and** $f x = \text{concat } xs$
<proof>

lemma *range-hull*: $\text{range } f = \langle\langle \text{range } f^{\mathcal{C}} \rangle\rangle$
<proof>

lemma *im-in-hull*: $f w \in \langle\langle \text{range } f^{\mathcal{C}} \rangle\rangle$
<proof>

lemma *core-ext-id*: $f^{\mathcal{C}\mathcal{L}} = f$
<proof>

lemma *rev-map-morph*: *morphism* $(\text{rev-map } f)$
<proof>

lemma *morph-rev-len*: $|f (\text{rev } u)| = |f u|$
<proof>

lemma *rev-map-len*: $|\text{rev-map } f u| = |f u|$
<proof>

lemma *in-set-morph-len*: **assumes** $a \in \text{set } w$ **shows** $|f [a]| \leq |f w|$
<proof>

lemma *morph-lq-comm*: $u \leq_p v \implies f (u^{-1} > v) = (f u)^{-1} > (f v)$
<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** $c: \text{code } (f^{\mathcal{C}} \text{ `} (\text{set } (u \cdot v)))$ **and** $i: \text{inj-on } f^{\mathcal{C}} (\text{set } (u \cdot v))$
and $f u = f v$
shows $u = v$
<proof>

lemma *morph-concat-concat-map*: $f (\text{concat } ws) = \text{concat} (\text{map } f ws)$

<proof>

lemma *morph-on-all: 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: $u \leq_f w \implies f u \leq_f f w$*
<proof>

lemma *set-core-set: $set (f w) = \bigcup (set ' f^c ' (set w))$*
<proof>

end

lemma *morph-map: morphism (map f)*
<proof>

lemma *list-ext-morph: morphism t^c*
<proof>

lemma *ext-def-on-set[intro]: $(\bigwedge a. a \in set u \implies g a = f a) \implies g^c u = f^c u$*
<proof>

lemma *morph-def-on-set: morphism f \implies morphism g $\implies (\bigwedge a. a \in set u \implies g^c a = f^c a) \implies g u = f u$*
<proof>

lemma *morph-compose: morphism f \implies morphism g \implies 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:*
 $\exists r. \forall u. \exists n. f u = r^{\textcircled{n}} \wedge \text{primitive } r$
<proof>

definition *mroot where mroot \equiv (SOME r. $(\forall u. \exists n. f u = r^{\textcircled{n}}) \wedge \text{primitive } r$)*

definition *im-exp :: 'a list \Rightarrow nat where im-exp u \equiv (SOME n. $f u = mroot^{\textcircled{n}}$)*

lemma *per-morph-rootI: $\forall u. \exists n. f u = mroot^{\textcircled{n}}$ and*
per-morph-root-prim: primitive mroot
<proof>

lemma *per-morph-im-exp*:

$f u = mroot^{\circ} im-exp u$
<proof>

lemma *mroot-primroot*: **assumes** $f u \neq \varepsilon$

shows $mroot = \varrho (f u)$
<proof>

interpretation *mirror*: *periodic-morphism rev-map* f

<proof>

lemma *per-morph-rev-map*: $rev-map f u = rev (f u)$

<proof>

lemma *mroot-rev*: $mirror.mroot = rev mroot$

<proof>

end

4.1.3 Non-erasing morphism

locale *nonerasing-morphism* = *morphism* +

assumes *nonerasing*: $f w = \varepsilon \implies w = \varepsilon$

begin

lemma *core-nemp*: $f^c a \neq \varepsilon$

<proof>

lemma *nemp-to-nemp*: $w \neq \varepsilon \implies f w \neq \varepsilon$

<proof>

lemma *sing-to-nemp*: $f [a] \neq \varepsilon$

<proof>

lemma *pref-morph-pref-eq*: $u \leq_p v \implies f v \leq_p f u \implies u = v$

<proof>

lemma *comm-eq-im-eq*:

$u \cdot v = v \cdot u \implies f u = f v \implies u = v$

<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*: $u <_s v \implies f u <_s f v$
 ⟨proof⟩

lemma *im-len-le*: $|u| \leq |f u|$
 ⟨proof⟩

lemma *im-len-eq-iff*: $|u| = |f u| \iff (\forall c. c \in set u \longrightarrow |f [c]| = 1)$
 ⟨proof⟩

lemma *im-len-less*: $a \in set u \implies |f [a]| \neq 1 \implies |u| < |f u|$
 ⟨proof⟩

end

lemma (**in morphism**) *nonerI[intro]*: **assumes** $(\bigwedge a. f^C a \neq \varepsilon)$
shows *nonerasing-morphism* f
 ⟨proof⟩

lemma (**in morphism**) *prim-morph-non*:
assumes *prim-morph*: $\bigwedge u. 2 \leq |u| \implies primitive\ u \implies primitive\ (f u)$
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^C)*, since the core can be not injective.

lemma (**in morphism**) *code-core-range-inj*: $inj\ f \iff code\ (range\ f^C) \wedge inj\ f^C$
 ⟨proof⟩

locale *code-morphism = morphism* f **for** f +
assumes *code-morph*: $inj\ f$

begin

lemma *inj-core*: $inj\ f^C$
 ⟨proof⟩

lemma *sing-im-core*: $f [a] \in (\text{range } f^{\mathcal{C}})$
<proof>

lemma *code-im*: *code* ($\text{range } f^{\mathcal{C}}$)
<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* $\langle (\text{range } f^{\mathcal{C}}) \rangle$
<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* $f \implies$ *code-morphism* f
<proof>

lemma (**in** *nonerasing-morphism*) *code-morphismI'* :
assumes *comm*: $\bigwedge u v. f u = f v \implies u \cdot v = v \cdot u$
shows *code-morphism* f
<proof>

4.1.5 Prefix code morphism

locale *prefix-code-morphism* = *nonerasing-morphism* +
assumes
pref-free: $f^{\mathcal{C}} a \leq_p f^{\mathcal{C}} b \implies a = b$

begin

interpretation *prefrange*: *prefix-code* ($\text{range } f^{\mathcal{C}}$)
<proof>

lemma *inj-core*: *inj* $f^{\mathcal{C}}$
<proof>

sublocale *code-morphism*
⟨*proof*⟩

lemma *pref-free-morph*: **assumes** $f r \leq_p f s$ **shows** $r \leq_p s$
⟨*proof*⟩

end

4.1.6 Marked morphism

locale *marked-morphism* = *nonerasing-morphism* +
assumes
 marked-core: $hd (f^C a) = hd (f^C b) \implies a = b$

begin

lemma *marked-im*: *marked-code* (range f^C)
⟨*proof*⟩

interpretation *marked-code* (range f^C)
⟨*proof*⟩

lemmas *marked-morph* = *marked-core*[*unfolded core-sing*]

sublocale *prefix-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*: $inj e \implies \text{marked-morphism } ((map e) \circ f)$
⟨*proof*⟩

end

thm *morphism.nonerI*

lemma (**in** *morphism*) *marked-morphismI*:
 $(\bigwedge a. f[a] \neq \varepsilon) \implies (\bigwedge a b. a \neq b) \implies hd (f[a]) \neq hd (f[b]) \implies \text{marked-morphism } f$
⟨*proof*⟩

4.1.7 Image length

definition *max-image-length*:: (*'a list* \Rightarrow *'b list*) \Rightarrow *nat* ($[_]$)
where *max-image-length* $f = \text{Max } (\text{length } (range f^C))$

definition *min-image-length*::('a list \Rightarrow 'b list) \Rightarrow nat ([_])
where *min-image-length* f = Min (length'(range f^C))

lemma *max-im-len-id*: [id::('a list \Rightarrow 'a list)] = 1 **and** *min-im-len-id*: [id::('a list \Rightarrow 'a list)] = 1
 <proof>

context *morphism*
begin

lemma *max-im-len-le*: finite (length'range f^C) \Longrightarrow |f z| \leq |z|*[f]
 <proof>

lemma *max-im-len-le-sing*: **assumes** finite (length'range f^C)
shows |f [a]| \leq [f]
 <proof>

lemma *min-im-len-ge*: finite (length'range f^C) \Longrightarrow |z| * [f] \leq |f z|
 <proof>

lemma *max-im-len-comp-le*: **assumes** *finite-f*: finite (length'range f^C) **and**
finite-g: finite (length'range g^C) **and** *morphism g*
shows finite (length ' range (g \circ f)^C) [g \circ f] \leq [f]*[g]
 <proof>

lemma *max-im-len-emp*: **assumes** finite (length ' range f^C)
shows [f] = 0 \longleftrightarrow (f = ($\lambda w. \varepsilon$))
 <proof>

lemmas *max-im-len-le-dom* = *max-im-len-le*[OF *finite-imageI*, OF *finite-imageI*]
and
min-im-len-le-sing-dom = *min-im-len-le-sing*[OF *finite-imageI*, OF *finite-imageI*]
and
min-im-len-ge-dom = *min-im-len-ge*[OF *finite-imageI*, OF *finite-imageI*] **and**
max-im-len-comp-le-dom = *max-im-len-comp-le*[OF *finite-imageI*, OF *finite-imageI*]
and
max-im-len-emp-dom = *max-im-len-emp*[OF *finite-imageI*, OF *finite-imageI*]

lemma *Cons-im*: f (x#xs) = f^C x \cdot f xs
 <proof>

end

4.1.8 Endomorphism

locale *endomorphism* = *morphism f* **for** f:: 'a list \Rightarrow 'a list
begin

lemma *pow-endomorphism: endomorphism* $(f \sim k)$
<proof>

interpretation *pow-endm: endomorphism* $(f \sim k)$
<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* $w = \text{set } v \implies \text{set } ((f \sim k) w) = \text{set } ((f \sim k) v)$
<proof>

lemma *fin-len-ran-pow: finite* $(\text{length } \text{'range } f^c) \implies \text{finite } (\text{length } \text{'range } (f \sim k)^c)$
<proof>

lemma *max-im-len-pow-le: assumes* $\text{finite } (\text{length } \text{'range } f^c)$ **shows** $[f \sim k] \leq [f] \wedge k$
<proof>

lemma *max-im-len-pow-le': finite* $(\text{length } \text{'range } f^c) \implies |(f \sim k) w| \leq |w| * [f] \wedge k$
<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* $(f \sim k)$
<proof>

lemma *im-len-pow-mono: assumes* *nonerasing-morphism* f $i \leq j$
shows $|(f \sim i) w| \leq |(f \sim j) w|$
<proof>

lemma *fac-mono-pow: u* $\leq f (f \sim k) w \implies (f \sim l) u \leq f (f \sim (l+k)) w$
<proof>

lemma *rev-map-endomorph: endomorphism* $(\text{rev-map } f)$
<proof>

end

4.1.9 Decomposition into primitive roots

definition *root-refine* (*Ref_ρ* - [55] 56) where *root-refine* = $(\lambda x. [\rho x]^{\textcircled{e}} e_{\rho} x)^{\mathcal{L}}$

lemma *root-ref-morph*: *morphism root-refine*
<proof>

thm *morphism.morph-concat-concat-map*

interpretation *rr-morph*: *morphism root-refine*
<proof>

lemma *root-ref-def*: $\text{Ref}_{\rho} us = \text{concat} (\text{map} (\lambda x. [\rho x]^{\textcircled{e}} e_{\rho} x) us)$
<proof>

lemma *root-ref-refines*: $\text{concat} (\text{Ref}_{\rho} us) = \text{concat} us$
<proof>

lemmas *root-ref-emp* = *rr-morph.emp-to-emp*

lemma *Ref_ρ [ε,ε] = ε*
<proof>

lemma *root-ref-empty-conv*: $\text{Ref}_{\rho} us = \varepsilon \longleftrightarrow \text{set } us \subseteq \{\varepsilon\}$
<proof>

lemma *root-ref-hd*: **assumes** $x \neq \varepsilon$
shows $\text{hd} (\text{Ref}_{\rho} (x\#xs)) = \rho x$
<proof>

lemma *root-ref-injective*:
assumes
emp-not-in: $\varepsilon \notin \mathcal{C}$ **and**
comm-eq: $\bigwedge x y. x \in \mathcal{C} \implies y \in \mathcal{C} \implies x \cdot y = y \cdot x \implies x = y$ **and**
 $us \in \text{lists } \mathcal{C} \text{ vs} \in \text{lists } \mathcal{C}$
shows $\text{Ref}_{\rho} us = \text{Ref}_{\rho} vs \implies us = vs$
<proof>

lemma (*in code*) *code-root-ref-injective*:
assumes $us \in \text{lists } \mathcal{C} \text{ vs} \in \text{lists } \mathcal{C}$
shows $\text{Ref}_{\rho} us = \text{Ref}_{\rho} vs \implies us = vs$
<proof>

find-theorems *set ?w = {?a}*

lemma (*in code*) *code-roots-non-overlapping*: *non-overlapping* $((\lambda x. [\rho x]^{\textcircled{e}} (e_{\rho} x))$
 $'\mathcal{C})$
<proof>

lemma (in *binary-code*) *bin-roots-sings-code: non-overlapping* {*Dec* { ϱ u_0 , ϱ u_1 }
 u_0 , *Dec* { ϱ u_0 , ϱ u_1 } u_1 }
 ⟨*proof*⟩

theorem (in *code*) *roots-prim-morph*:
 assumes $ws \in \text{lists } \mathcal{C}$
 and $|ws| \neq 1$
 and *primitive* ws
 shows *primitive* (*Ref* _{ϱ} ws)
 ⟨*proof*⟩

4.2 Primitivity preserving morphisms

locale *primitivity-preserving-morphism = nonerasing-morphism +*
 assumes *prim-morph* : $2 \leq |u| \implies \text{primitive } u \implies \text{primitive } (f \ u)$
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* \implies ('*a list* \implies '*b list*) \implies ('*a list* \implies '*b list*) \implies
bool

($- \in - =_M - [80,80,80] \ 51$)

where *min-sol-def*: *minimal-solution* $s \ g \ h \equiv s \neq \varepsilon \wedge g \ s = h \ s$
 $\wedge (\forall \ s'. \ s' \neq \varepsilon \wedge s' \leq_p s \wedge g \ s' = h \ s' \implies s' = s)$

lemma *min-solD*: $s \in g =_M h \implies g \ s = h \ s$
 ⟨*proof*⟩

lemma *min-solD'*: $s \in g =_M h \implies s \neq \varepsilon$
 ⟨*proof*⟩

lemma *min-solD-min*: $s \in g =_M h \implies p \neq \varepsilon \implies p \leq_p s \implies g \ p = h \ p \implies p =$
 s
 ⟨*proof*⟩

lemma *min-solI[intro]*: $s \neq \varepsilon \implies g \ s = h \ s \implies (\bigwedge \ s'. \ s' \leq_p s \implies s' \neq \varepsilon \implies g$
 $s' = h \ s' \implies s' = s) \implies s \in g =_M h$

<proof>

lemma *min-sol-sym-iff*: $s \in g =_M h \longleftrightarrow s \in h =_M g$
<proof>

lemma *min-sol-sym[sym]*: $s \in g =_M h \implies s \in h =_M g$
<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* :: $('a \text{ list} \Rightarrow 'b \text{ list}) \Rightarrow ('a \text{ list} \Rightarrow 'b \text{ list}) \Rightarrow ('a \text{ list} \times 'a \text{ list}) \text{ set}$ (\mathfrak{C})
where *coincidence-set* $g h \equiv \{(r,s). g r = h s\}$

lemma *coin-set-eq*: $(g \circ \text{fst})'(\mathfrak{C} g h) = (h \circ \text{snd})'(\mathfrak{C} g h)$
<proof>

lemma *coin-setD*: $\text{pair} \in \mathfrak{C} g h \implies g (\text{fst pair}) = h (\text{snd pair})$
<proof>

lemma *coin-setD-iff*: $\text{pair} \in \mathfrak{C} g h \longleftrightarrow g (\text{fst pair}) = h (\text{snd pair})$
<proof>

lemma *coin-set-sym*: $\text{fst}'(\mathfrak{C} g h) = \text{snd}'(\mathfrak{C} h g)$
<proof>

lemma *coin-set-inter-fst*: $(g \circ \text{fst})'(\mathfrak{C} g h) = \text{range } g \cap \text{range } h$
<proof>

lemmas *coin-set-inter-snd = coin-set-inter-fst[unfolded coin-set-eq]*

definition *minimal-coincidence* :: $('a \text{ list} \Rightarrow 'b \text{ list}) \Rightarrow 'a \text{ list} \Rightarrow ('a \text{ list} \Rightarrow 'b \text{ list}) \Rightarrow 'a \text{ list} \Rightarrow \text{bool}$ $((- \ -) =_m (- \ -) [80,81,80,81] 51)$
where *min-coin-def*: *minimal-coincidence* $g r h s \equiv r \neq \varepsilon \wedge s \neq \varepsilon \wedge g r = h s \wedge (\forall r' s'. r' \leq_p r \wedge r' \neq \varepsilon \wedge s' \leq_p s \wedge s' \neq \varepsilon \wedge g r' = h s' \longrightarrow r' = r \wedge s' = s)$

definition *min-coincidence-set* :: $('a \text{ list} \Rightarrow 'b \text{ list}) \Rightarrow ('a \text{ list} \Rightarrow 'b \text{ list}) \Rightarrow ('a \text{ list} \times 'a \text{ list}) \text{ set}$ (\mathfrak{C}_m)
where *min-coincidence-set* $g h \equiv (\{(r,s) . g r =_m h s\})$

lemma *min-coin-minD*: $g r =_m h s \implies r' \leq_p r \implies r' \neq \varepsilon \implies s' \leq_p s \implies s' \neq \varepsilon \implies g r' = h s' \implies r' = r \wedge s' = s$
<proof>

lemma *min-coin-setD*: $p \in \mathfrak{C}_m g h \implies g (fst p) =_m h (snd p)$
<proof>

lemma *min-coinD*: $g r =_m h s \implies g r = h s$
<proof>

lemma *min-coinD'*: $g r =_m h s \implies r \neq \varepsilon \wedge s \neq \varepsilon$
<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**
 $(\bigwedge r' s'. r' \leq_p r \implies r' \neq \varepsilon \implies s' \leq_p s \implies s' \neq \varepsilon \implies g r' = h s' \implies r' = r \wedge s' = s)$
shows $g r =_m h s$
<proof>

lemma *min-coin-sym[sym]*: $g r =_m h s \implies h s =_m g r$
<proof>

lemma *min-coin-sym-iff*: $g r =_m h s \iff h s =_m g r$
<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 $\bigwedge a. a \in set u \implies g [a] = h [a]$
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 $u \leq_p u'$ **and** $v \leq_p v'$ **and** $g u' \bowtie h v'$ **shows** $g u \bowtie h v$
<proof>

lemma *ims-sufs-comp*:

assumes $u \leq_s u'$ **and** $v \leq_s v'$ **and** $g u' \bowtie_s h v'$ **shows** $g u \bowtie_s h v$
 ⟨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 $(\bigwedge a. a \in set\ s \implies |g\ [a]| \leq |h\ [a]|)$
shows $|g\ s| \leq |h\ s|$
 ⟨proof⟩

lemma *len-im-less*:

assumes $\bigwedge a. a \in set\ s \implies |g\ [a]| \leq |h\ [a]|$ **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** $\bigwedge a. a \in set\ s \implies |g\ [a]| = |h\ [a]|$
shows $\bigwedge a. a \in set\ s \implies g\ [a] = h\ [a]$
 ⟨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 $(rev\ s) \in (rev\ map\ g) =_M (rev\ map\ h)$
 ⟨proof⟩

lemma *coin-set-lists-concat*: $ps \in lists\ (\mathfrak{C}\ g\ h) \implies g\ (concat\ (map\ fst\ ps)) = h\ (concat\ (map\ snd\ ps))$

⟨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 Factor intepretation of morphic images

context *morphism*
begin

lemma *image-fac-interp'*: **assumes** $w \leq_f f z w \neq \varepsilon$
obtains $p w\text{-pred } s$ **where** $w\text{-pred} \leq_f z p w s \sim_{\mathcal{I}} (\text{map } f^c w\text{-pred})$
 ⟨*proof*⟩

lemma *image-fac-interp*: **assumes** $u \cdot w \cdot v = f z w \neq \varepsilon$
obtains $p w\text{-pred } s u\text{-pred } v\text{-pred}$ **where**
 $u\text{-pred} \cdot w\text{-pred} \cdot v\text{-pred} = z p w s \sim_{\mathcal{I}} (\text{map } f^c w\text{-pred})$
 $u = (f u\text{-pred}) \cdot p$ $v = s \cdot (f v\text{-pred})$
 ⟨*proof*⟩

lemma *image-fac-interp-mid*: **assumes** $p w s \sim_{\mathcal{I}} \text{map } f^c w\text{-pred}$ $2 \leq |w\text{-pred}|$
obtains $pw sw$ **where**
 $w = pw \cdot (f (\text{butlast } (tl w\text{-pred}))) \cdot sw$ $p \cdot pw = f [hd w\text{-pred}] sw \cdot s = f [last w\text{-pred}]$
 ⟨*proof*⟩

end

4.3.5 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 \implies u = $\varepsilon \longleftrightarrow$ v = ε*
<proof>

lemma *min-coin-rev:*
assumes $g r =_m h s$
shows $(\text{rev-map } g) (\text{rev } r) =_m (\text{rev-map } h) (\text{rev } s)$
<proof>

lemma *min-coin-pref-eq:*
assumes $g e =_m h f$ and $g e' = h f'$ and $e' \leq_p e$ and $e' \neq \varepsilon$ 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 $e \leq_p r$ and $f \leq_p s$ and $\text{hd } e = \text{hd } r$
<proof>

lemma *min-coin-dec: assumes g e = h f*
obtains ps where $\text{concat } (\text{map } \text{fst } ps) = e$ and $\text{concat } (\text{map } \text{snd } ps) = f$ and
 $\bigwedge p. p \in \text{set } ps \implies g (\text{fst } p) =_m h (\text{snd } p)$
<proof>

lemma *min-coin-code:*
assumes $xs \in \text{lists } (\mathfrak{C}_m g h)$ and $ys \in \text{lists } (\mathfrak{C}_m g h)$ and
 $\text{concat } (\text{map } \text{fst } xs) = \text{concat } (\text{map } \text{fst } ys)$ and
 $\text{concat } (\text{map } \text{snd } xs) = \text{concat } (\text{map } \text{snd } ys)$
shows $xs = ys$
<proof>

lemma *coin-closed: ps \in lists $(\mathfrak{C} g h) \implies (\text{concat } (\text{map } \text{fst } ps), \text{concat } (\text{map } \text{snd } ps)) \in \mathfrak{C} g h$*
<proof>

lemma *min-coin-gen-snd: $\langle \text{snd } ' (\mathfrak{C}_m g h) \rangle = \text{snd } ' (\mathfrak{C} g h)$*
<proof>

lemma *min-coin-gen-fst: $\langle \text{fst } ' (\mathfrak{C}_m g h) \rangle = \text{fst } ' (\mathfrak{C} g h)$*
<proof>

lemma *min-coin-code-snd:*

assumes *code-morphism g* **shows** $\text{code } (\text{snd } \text{' } (\mathfrak{C}_m \text{ } g \text{ } h))$
 ⟨*proof*⟩

lemma *min-coin-code-fst*:
code-morphism h $\implies \text{code } (\text{fst } \text{' } (\mathfrak{C}_m \text{ } g \text{ } h))$
 ⟨*proof*⟩

lemma *min-coin-basis-snd*:
assumes *code-morphism g*
shows $\mathfrak{B} (\text{snd } \text{' } (\mathfrak{C} \text{ } g \text{ } h)) = \text{snd } \text{' } (\mathfrak{C}_m \text{ } g \text{ } h)$
 ⟨*proof*⟩

lemma *min-coin-basis-fst*:
code-morphism h $\implies \mathfrak{B} (\text{fst } \text{' } (\mathfrak{C} \text{ } g \text{ } h)) = \text{fst } \text{' } (\mathfrak{C}_m \text{ } g \text{ } h)$
 ⟨*proof*⟩

lemma *sol-im-len-less*: **assumes** $g \text{ } u = h \text{ } u$ **and** $g \neq h$ **and** *set u = UNIV*
shows $|u| < |g \text{ } u|$
 ⟨*proof*⟩

end

locale *two-code-morphisms* = *g: code-morphism g + h: code-morphism h*
for $g \text{ } h :: \text{' } a \text{ list} \Rightarrow \text{' } b \text{ list}$

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*:
 $\mathfrak{B} (h \text{' } (\text{snd } \text{' } (\mathfrak{C} \text{ } g \text{ } h))) = h \text{' } \text{snd } \text{' } (\mathfrak{C}_m \text{ } g \text{ } h)$
 $\mathfrak{B} (g \text{' } (\text{fst } \text{' } (\mathfrak{C} \text{ } g \text{ } h))) = g \text{' } \text{fst } \text{' } (\mathfrak{C}_m \text{ } g \text{ } h)$
 ⟨*proof*⟩

lemma *range-inter-basis-snd*:
shows $\mathfrak{B} (\text{range } g \cap \text{range } h) = h \text{' } (\text{snd } \text{' } \mathfrak{C}_m \text{ } g \text{ } h)$
 $\mathfrak{B} (\text{range } g \cap \text{range } h) = g \text{' } (\text{fst } \text{' } \mathfrak{C}_m \text{ } g \text{ } h)$
 ⟨*proof*⟩

lemma *range-inter-code*:
shows *code* $\mathfrak{B} (\text{range } g \cap \text{range } h)$
 ⟨*proof*⟩

end

4.3.6 Two marked morphisms

locale *two-marked-morphisms* = *two-nonerasing-morphisms* +
 g: marked-morphism *g* + *h*: marked-morphism *h*

begin

sublocale *revs*: *two-code-morphisms g h*
 ⟨*proof*⟩

lemmas *ne-g* = *g.nonerasing* and
 ne-h = *h.nonerasing*

lemma *unique-continuation*:

$z \cdot g \ r = z' \cdot h \ s \implies z \cdot g \ r' = z' \cdot h \ s' \implies z \cdot g \ (r \wedge_p r') = z' \cdot h \ (s \wedge_p s')$
 ⟨*proof*⟩

lemmas *empty-sol* = *noner-eq-emp-iff*

lemma *comparable-min-sol-eq*: **assumes** $r \leq_p r'$ 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 $e \leq_p e'$ and $f \leq_p f'$
 ⟨*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 $g \ (fst \ (pre\text{-}block \ a)) =_m \ h \ (snd \ (pre\text{-}block \ a)) \wedge$ $hd \ (fst \ (pre\text{-}block \ a)) = a$

 — Predicate: the *pre-block* of the letter *a* is indeed a block

lemma *pre-blockI*: $g\ u =_m\ h\ v \implies \text{pre-block}\ (hd\ u) = (u, v)$
 ⟨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*: $g^C\ (hd\ w) \cdot h^C\ (hd\ w) = h^C\ (hd\ w) \cdot g^C\ (hd\ w)$ **and** *len*: $|g^C\ (hd\ w)| \leq |h^C\ (hd\ w)|$
shows $g^C\ (hd\ w) = h^C\ (hd\ w)$
 ⟨proof⟩

lemma *hd-im-comm-eq*:
assumes $g\ w = h\ w$ **and** $w \neq \varepsilon$ **and** *comm*: $g^C\ (hd\ w) \cdot h^C\ (hd\ w) = h^C\ (hd\ w) \cdot g^C\ (hd\ w)$
shows $g^C\ (hd\ w) = h^C\ (hd\ w)$
 ⟨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** $suc-fst \equiv \lambda\ x.\ \text{concat}(\text{map}\ (\lambda\ y.\ \text{fst}\ (\text{pre-block}\ y))\ x)$
definition *suc-snd* **where** $suc-snd \equiv \lambda\ x.\ \text{concat}(\text{map}\ (\lambda\ y.\ \text{snd}\ (\text{pre-block}\ y))\ x)$

lemma *blockP-D*: $\text{blockP}\ a \implies g\ (\text{suc-fst}\ [a]) =_m\ h\ (\text{suc-snd}\ [a])$
 ⟨proof⟩

lemma *blockP-D-hd*: $\text{blockP}\ a \implies hd\ (\text{suc-fst}\ [a]) = a$
 ⟨proof⟩

abbreviation *blocks* $\tau \equiv (\forall\ a.\ a \in \text{set}\ \tau \longrightarrow \text{blockP}\ a)$

sublocale *sucs*: *two-morphisms* *suc-fst* *suc-snd*
 ⟨proof⟩

lemma *blockP-D-hd-hd*: **assumes** *blockP* a
shows $hd\ (h^C\ (hd\ (\text{suc-snd}\ [a]))) = hd\ (g^C\ a)$
 ⟨proof⟩

lemma *suc-morph-sings*: **assumes** $g\ e =_m\ h\ f$
shows $\text{suc-fst}\ [hd\ e] = e$ **and** $\text{suc-snd}\ [hd\ e] = f$
 ⟨proof⟩

lemma *blocks-eq*: $\text{blocks } \tau \implies g (\text{suc-fst } \tau) = h (\text{suc-snd } \tau)$
(proof)

lemma *suc-eq'*: **assumes** $\bigwedge a. \text{blockP } a$ **shows** $g(\text{suc-fst } w) = h(\text{suc-snd } w)$
(proof)

lemma *suc-eq*: **assumes** *all-blocks*: $\bigwedge a. \text{blockP } a$ **shows** $g \circ \text{suc-fst} = h \circ \text{suc-snd}$
(proof)

lemma *block-eq*: $\text{blockP } a \implies g (\text{suc-fst } [a]) = h (\text{suc-snd } [a])$
(proof)

lemma *blocks-inj-suc*: **assumes** *blocks* τ **shows** *inj-on* $\text{suc-fst}^C (\text{set } \tau)$
(proof)

lemma *blocks-inj-suc'*: **assumes** *blocks* τ **shows** *inj-on* $\text{suc-snd}^C (\text{set } \tau)$
(proof)

lemma *blocks-marked-code*: **assumes** *blocks* τ
shows *marked-code* $(\text{suc-fst}^C (\text{set } \tau))$
(proof)

lemma *blocks-marked-code'*: **assumes** *all-blocks*: $\bigwedge a. a \in \text{set } \tau \implies \text{blockP } a$
shows *marked-code* $(\text{suc-snd}^C (\text{set } \tau))$
(proof)

lemma *sucs-marked-morphs*: **assumes** *all-blocks*: $\bigwedge a. \text{blockP } a$
shows *two-marked-morphisms* suc-fst suc-snd
(proof)

lemma *pre-blocks-range*: $\{(e,f). g e =_m h f\} \subseteq \text{range } \text{pre-block}$
(proof)

corollary *card-blocks*: **assumes** *finite* $(UNIV :: 'a \text{ set})$ **shows** $\text{card } \{(e,f). g e =_m h f\} \leq \text{card } (UNIV :: 'a \text{ set})$
(proof)

lemma *block-decomposition*: **assumes** $g e = h f$
obtains τ **where** $\text{suc-fst } \tau = e$ **and** $\text{suc-snd } \tau = f$ **and** *blocks* τ
(proof)

lemma *block-decomposition-unique*: **assumes** $g e = h f$ **and**
 $\text{suc-fst } \tau = e$ **and** $\text{suc-fst } \tau' = e$ **and** *blocks* τ **and** *blocks* τ' **shows** $\tau = \tau'$
(proof)

lemma *block-decomposition-unique'*: **assumes** $g e = h f$ **and**
 $\text{suc-snd } \tau = f$ **and** $\text{suc-snd } \tau' = f$ **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** $\text{suc-fst } [a] = [a]^{\textcircled{a}} \text{Suc } m$ **and** $\text{suc-snd } [a] = [b]^{\textcircled{a}} \text{Suc } n$
<proof>

definition *sucs-encoding* **where** $\text{sucs-encoding} = (\lambda a. \text{hd } (g [a]))$

definition *sucs-decoding* **where** $\text{sucs-decoding} = (\lambda a. \text{SOME } c. \text{hd } (g[c]) = a)$

lemma *sucs-encoding-inv*: $\text{sucs-decoding} \circ \text{sucs-encoding} = \text{id}$
<proof>

lemma *encoding-inj*: $\text{inj } \text{sucs-encoding}$
<proof>

lemma *map-encoding-inj*: $\text{inj } (\text{map } \text{sucs-encoding})$
<proof>

definition *suc-fst'* **where** $\text{suc-fst}' = (\text{map } \text{sucs-encoding}) \circ \text{suc-fst}$

definition *suc-snd'* **where** $\text{suc-snd}' = (\text{map } \text{sucs-encoding}) \circ \text{suc-snd}$

lemma *encoded-sucs-eq-conv*: $\text{suc-fst } w = \text{suc-snd } w' \longleftrightarrow \text{suc-fst}' w = \text{suc-snd}' w'$
<proof>

lemma *encoded-sucs-eq-conv'*: $\text{suc-fst} = \text{suc-snd} \longleftrightarrow \text{suc-fst}' = \text{suc-snd}'$
<proof>

lemma *encoded-sucs*: **assumes** $\bigwedge c. \text{blockP } c$ **shows** *two-marked-morphisms* $\text{suc-fst}'$
 $\text{suc-snd}'$
<proof>

lemma *encoded-sucs-len*: $|\text{suc-fst } w| = |\text{suc-fst}' w|$ **and** $|\text{suc-snd } w| = |\text{suc-snd}' w|$
<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 theory *Combinatorics-Words.CoWBasic* as *two-pers*: $\llbracket w \leq_p u \cdot w; w \leq_p v \cdot w; |u| + |v| \leq |w| \rrbracket \implies u \cdot v = v \cdot u$.

theorem *per-lemma-relaxed*:

assumes *period* $w p$ **and** *period* $w q$ **and** $p + q \leq |w|$

shows $(\text{take } p w) \cdot (\text{take } q w) = (\text{take } q w) \cdot (\text{take } p w)$

<proof>

5.1 Main claim

We first formulate the claim of the periodicity lemma in terms of commutation of two periodic roots. For trivial reasons we can also drop the requirement that the roots are nonempty.

The proof is by induction which mimics the Euclidean algorithm. The step is given by the following lemma:

lemma *per-lemma-step*:

fixes $u v w :: 'a \text{ list}$

defines $w' \equiv u \cdot w$ **and** $v' \equiv u \cdot v$

shows $w' \leq_p u \cdot w' \wedge w' \leq_p v' \cdot w' \iff w \leq_p u \cdot w \wedge w \leq_p v \cdot w$

<proof>

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>

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 ≤ |w|*
shows *period w (gcd p q)*
<proof>

5.2 Optimality of the bound by construction of the Fine and Wilf word.

FW-word (where FW stands for Fine and Wilf) yields a word which shows the optimality of the bound in the periodicity lemma.

definition *fw-per k d* $\equiv [0..<d]^{\textcircled{k}} \cdot [0..<(d-1)]$

definition *fw-base k d* $\equiv \text{fw-per } k \ d \cdot [d] \cdot \text{fw-per } k \ d$

fun *FW-word* :: *nat* \Rightarrow *nat* \Rightarrow *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 $p \text{ dvd } q$ then ε else
— base case if $\text{gcd } p \ q = q - p$ then *fw-base (p div (gcd p q) - 1) (gcd p q)*
— step else (take p (*FW-word p (q-p)*)) \cdot *FW-word p (q-p)*)

lemma *FW-sym*: *FW-word p q* = *FW-word q p*
<proof>

lemma *FW-dvd*: **assumes** $p \text{ dvd } q$ **shows** *FW-word p q* = ε
<proof>

lemma *upt-minus-pref*: $[i..<j-d] \leq p [i..<j]$
<proof>

lemma *fw-per-pref*: $\text{fw-per } k \ d \leq p \text{ take } d \ (\text{fw-per } k \ d) \cdot (\text{fw-per } k \ d)$
<proof>

lemma *fw-per-len*: **shows** $|\text{fw-per } k \ d| = (k+1)*d - 1$
<proof>

lemma *fw-base-len*: **assumes** $0 < d$ **shows** $|\text{fw-base } k \ d| = (k+1)*d + (k+1)*d - 1$
<proof>

lemma *fw-base-per1*: **assumes** $0 < d$
shows *period (fw-base k d) ((k+1)*d)*
<proof>

lemma *fw-base-per2*: **assumes** $0 < d$
shows $(fw\text{-base } k\ d) <_p \text{ take } ((k+1)*d + d) (fw\text{-base } k\ d) \cdot (fw\text{-base } k\ d)$
 $\langle proof \rangle$

lemma *fw-base-not-per*: **assumes** $0 < d\ 0 < k$
shows $\neg \text{period } (fw\text{-base } k\ d)\ d$ (**is** $\neg \text{period } ?w\ d$)
 $\langle proof \rangle$

lemma *fw-per-per*: **assumes** $fw\text{-per } k\ d \neq \varepsilon$
shows $(fw\text{-per } k\ d) <_p \text{ take } d (fw\text{-per } k\ d) \cdot (fw\text{-per } k\ d)$
 $\langle proof \rangle$

lemma *fw-per-nth*: **assumes** $i < |fw\text{-per } k\ d|$
shows $(fw\text{-per } k\ d)!i = i \bmod d$
 $\langle proof \rangle$

lemma *fw-base-nth*: **assumes** $i < |fw\text{-base } k\ d|$
shows $(fw\text{-base } k\ d)!i = (\text{if } i = |fw\text{-per } k\ d| \text{ then } d \text{ else } i \bmod d)$
 $\langle proof \rangle$

lemma *fw-base-match*: **assumes** $i < |fw\text{-base } k\ d|\ j < |fw\text{-base } k\ d|\ i \neq j$ **and**
 $eq: (fw\text{-base } k\ d)!i = (fw\text{-base } k\ d)!j$
shows $i \bmod d = j \bmod d$ **and** $i \neq |fw\text{-per } k\ d|$ **and** $j \neq |fw\text{-per } k\ d|$
 $\langle proof \rangle$

lemma *ext-per-sum*: **assumes** $\text{period } w\ p$ **and** $\text{period } w\ q$ **and** $p \leq |w|$
shows $\text{period } ((\text{take } p\ w) \cdot w) (p+q)$
 $\langle proof \rangle$

lemma *drop-per-diff*: **assumes** $\text{period } w\ p$ **and** $\text{period } w\ q$ **and** $p < q$ **and** $p < |w|$
shows $\text{period } (\text{drop } p\ w) (q-p)$
 $\langle proof \rangle$

theorem *fw-word*: **assumes** $\neg p\ \text{dvd } q\ \neg q\ \text{dvd } p$
shows $|FW\text{-word } p\ q| = p + q - \text{gcd } p\ q - 1$
 $\text{period } (FW\text{-word } p\ q)\ p$ **and** $\text{period } (FW\text{-word } p\ q)\ q$
 $\neg \text{period } (FW\text{-word } p\ q) (\text{gcd } p\ q)$
 $\langle proof \rangle$

Calculation examples

value $FW\text{-word } 0\ 4$
value $FW\text{-word } 3\ 4$
value $FW\text{-word } 4\ 7$
value $FW\text{-word } 3\ 7$
value $FW\text{-word } 5\ 7$
value $FW\text{-word } 5\ 13$
value $FW\text{-word } 4\ 6$
value $FW\text{-word } 12\ 18$
value $FW\text{-word } 6\ 10$

value *FW-word* 10 16

5.3 Optimality of the Fine and Wilf word.

The *FW-word* is the most general word having the two desired properties. That is, each equality of letters is forced by periods.

lemma *fw-base-mod-aux*: **assumes** $(i :: nat) < p + p - 1$ $i \neq p - 1$
shows $i \bmod p < p - 1$
<proof>

theorem *fw-minimal*: **assumes** *period w p* **and** *period w q* **and** $|w| = |\text{FW-word } p \ q|$ **and**
 $i < |w|$ **and** $j < |w|$ **and** $(\text{FW-word } p \ q)!i = (\text{FW-word } p \ q)!j$
shows $w!i = w!j$
<proof>

theorem *fw-minimal-set*: **assumes** *period w p* **and** *period w q* **and** $|w| = |\text{FW-word } p \ q|$
shows $\text{card}(\text{set } w) \leq \text{card}(\text{set } (\text{FW-word } p \ q))$
<proof>

5.4 Other variants of the periodicity lemma

periodicity lemma is one of the most frequent tools in Combinatorics on words. Here are some useful variants.

Note that the following lemmas are stronger versions of $\llbracket ?w <_p ?p \cdot ?w; <_s ?w \ (\ ?w \cdot ?q); |?p| + |?q| \leq |?w|; \bigwedge r \ s \ k \ l \ m. \llbracket ?p = (r \cdot s)^\text{@} k; ?q = (s \cdot r)^\text{@} l; ?w = (r \cdot s)^\text{@} m \cdot r; \text{primitive } (r \cdot s) \rrbracket \implies ?thesis \rrbracket \implies ?thesis$
 $\llbracket \leq_f ?u \ (\ ?r^\text{@} ?k); \leq_f ?u \ (\ ?s^\text{@} ?l); |?r| + |?s| \leq |?u| \rrbracket \implies \varrho \ ?r \sim \varrho \ ?s$
 $\llbracket \leq_f ?u \ (\ ?r^\text{@} ?k); \leq_f ?u \ (\ ?s^\text{@} ?l); |?r| + |?s| \leq |?u| \rrbracket \implies \varrho \ ?r \sim \varrho \ ?s$
 $\llbracket \leq_f ?w \ (\ ?u^\text{@} ?n); \leq_f ?w \ (\ ?v^\text{@} ?m); \text{primitive } ?u; \text{primitive } ?v; |?u| + |?v| \leq |?w| \rrbracket \implies ?u \sim ?v$ that have a relaxed length assumption $|p| + |q| \leq |w|$ instead of $|p| + |q| - \text{gcd } |p| \ |q| \leq |w|$ (and which follow from the relaxed version of periodicity lemma $\llbracket ?w \leq_p ?u \cdot ?w; ?w \leq_p ?v \cdot ?w; |?u| + |?v| \leq |?w| \rrbracket \implies ?u \cdot ?v = ?v \cdot ?u$.

lemma *per-lemma-comm-pref*:
assumes $u \leq_p r^\text{@} k$ $u \leq_p s^\text{@} l$
and *len*: $|r| + |s| - \text{gcd } (|r|) \ (|s|) \leq |u|$
shows $r \cdot s = s \cdot r$
<proof>

lemma *per-lemma-pref-suf-gcd*: **assumes** $w <_p p \cdot w$ **and** $w <_s w \cdot q$ **and**
fw: $|p| + |q| - (\text{gcd } |p| \ |q|) \leq |w|$

obtains $r s k l m$ **where** $p = (r \cdot s)^{\textcircled{k}}$ **and** $q = (s \cdot r)^{\textcircled{l}}$ **and** $w = (r \cdot s)^{\textcircled{m}} \cdot r$
and *primitive* $(r \cdot s)$

<proof>

lemma *fac-two-conjug-primroot-gcd*:

assumes *facts*: $w \leq_f p^{\textcircled{k}}$ $w \leq_f q^{\textcircled{l}}$ **and** *nemps*: $p \neq \varepsilon$ $q \neq \varepsilon$ **and** *len*: $|p| + |q| - \text{gcd}(|p|)(|q|) \leq |w|$

obtains $r s m$ **where** $\varrho p \sim r \cdot s$ **and** $\varrho q \sim r \cdot s$ **and** $w = (r \cdot s)^{\textcircled{m}} \cdot r$ **and**
primitive $(r \cdot s)$

<proof>

corollary *fac-two-conjug-primroot'-gcd*:

assumes *facts*: $u \leq_f r^{\textcircled{k}}$ $u \leq_f s^{\textcircled{l}}$ **and** *nemps*: $r \neq \varepsilon$ $s \neq \varepsilon$ **and** *len*: $|r| + |s| - \text{gcd}(|r|)(|s|) \leq |u|$

shows $\varrho r \sim \varrho s$

<proof>

lemma *fac-two-conjug-primroot''-gcd*:

assumes *facts*: $u \leq_f r^{\textcircled{k}}$ $u \leq_f s^{\textcircled{l}}$ **and** $u \neq \varepsilon$ **and** *len*: $|r| + |s| - \text{gcd}(|r|)(|s|) \leq |u|$

shows $\varrho r \sim \varrho s$

<proof>

lemma *fac-two-prim-conjug-gcd*:

assumes $w \leq_f u^{\textcircled{n}}$ $w \leq_f v^{\textcircled{m}}$ *primitive* u *primitive* v $|u| + |v| - \text{gcd}(|u|)(|v|) \leq |w|$

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 $x^{\textcircled{a}} \cdot y^{\textcircled{b}} = z^{\textcircled{c}}$.

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 with $2 \leq a, b, c$ 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: $x^{\textcircled{a}} \cdot y^{\textcircled{b}} = z^{\textcircled{c}}$ **and** $0 < a$ **and** $0 < b$ **and** $|z| + |x| - 1 \leq |x^{\textcircled{a}}|$

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: $x^{\textcircled{a}}a \cdot y^{\textcircled{b}}b = z^{\textcircled{c}}c$ **and** $0 < a$ **and** $0 < b$ **and** $|z| + |x| \leq |x^{\textcircled{a}}a|$
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: $x^{\textcircled{a}}a \cdot y^{\textcircled{b}}b = z^{\textcircled{c}}c$ **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'*:

$\llbracket x^{\textcircled{a}}a \cdot y^{\textcircled{b}}b = z^{\textcircled{c}}c; 2 \leq a; 2 \leq b; 2 \leq c \rrbracket$
 $\implies x \cdot y = y \cdot x$
<proof>

theorem *Lyndon-Schutzenberger*:

assumes $x^{\textcircled{a}}a \cdot y^{\textcircled{b}}b = z^{\textcircled{c}}c$ **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** $x^{\textcircled{a}}c = r^{\textcircled{a}}k \cdot u^{\textcircled{b}}b \cdot r^{\textcircled{a}}k'$
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 $x^{\textcircled{j}} \cdot y^{\textcircled{k}} = z^{\textcircled{l}}$

6.3.1 Auxiliary lemmas

lemma *xjy-imprim-len*: **assumes** $x \cdot y \neq y \cdot x$ **and** $eq: x^{\textcircled{j}} \cdot y = z^{\textcircled{l}}$ **and** $2 \leq j$ **and** $2 \leq l$

shows $|x^{\textcircled{j}}| < |y| + 2 \cdot |x|$ **and** $|z| < |x| + |y|$ **and** $|x| < |z|$ **and** $|x^{\textcircled{j}}| < |z| + |x|$

<proof>

lemma *case-j1k1*: **assumes**

eq: $x \cdot y = z^{\textcircled{l}}$ **and**

non-comm: $x \cdot y \neq y \cdot x$ **and**

l-min: $2 \leq l$

obtains $r \ q \ m \ n$ **where**

$x = (r \cdot q)^{\textcircled{m}} \cdot r$ **and**

$y = q \cdot (r \cdot q)^{\textcircled{n}}$ **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*: $x^{\textcircled{j}} \cdot y^{\textcircled{k}} = z^{\textcircled{l}}$

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**

$(r \cdot q)^{\textcircled{t}} \cdot r = x$ **and**

$q \cdot r \cdot r \cdot q = y$ **and**

$(r \cdot q)^{\textcircled{t}} \cdot r \cdot r \cdot q = z$ **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** $j = 1 \ 2 \leq k \ z \leq_s y^{\textcircled{k}}$
obtains $r \ q \ t$ **where**
 $x = ((q \cdot r) \cdot (r \cdot (q \cdot r)^{\textcircled{t}})^{\textcircled{k-1}})^{\textcircled{l-2}} \cdot$
 $((q \cdot r) \cdot (r \cdot (q \cdot r)^{\textcircled{t}})^{\textcircled{k-2}}) \cdot r \cdot q$ **and**
 $y = r \cdot (q \cdot r)^{\textcircled{t}}$ **and**
 $z = (q \cdot r) \cdot (r \cdot (q \cdot r)^{\textcircled{t}})^{\textcircled{k-1}}$ **and** $\langle 0 < t \rangle$ **and** $r \cdot q \neq q \cdot r$
 ⟨*proof*⟩

lemma *case-j1k2-b*: **assumes** $j = 1 \ 2 \leq k \ y^{\textcircled{k}} <_s z$
obtains q **where**
 $x = (q \cdot y^{\textcircled{k}})^{\textcircled{l-1}} \cdot q$ **and**
 $z = q \cdot y^{\textcircled{k}}$ **and**
 $q \cdot y \neq y \cdot q$
 ⟨*proof*⟩

6.3.3 Putting things together

lemma *solution-cases*: **obtains**
 $j = 2 \ k = 1 \ |$
 $j = 1 \ 2 \leq k \ z <_s y^{\textcircled{k}} \ |$
 $j = 1 \ 2 \leq k \ y^{\textcircled{k}} <_s z \ |$
 $j = 1 \ k = 1$
 ⟨*proof*⟩

theorem *parametric-solutionE*: **obtains**
 — case $x \cdot y$
 $r \ q \ m \ n$ **where**
 $x = (r \cdot q)^{\textcircled{m}} \cdot r$ **and**
 $y = q \cdot (r \cdot q)^{\textcircled{n}}$ **and**
 $z = r \cdot q$ **and**
 $l = m + n + 1$ **and** $r \cdot q \neq q \cdot r$
 |
 — case $x \cdot y^{\textcircled{k}}$ with $2 \leq k$ and $<_s z (y^{\textcircled{k}})$
 $r \ q \ t$ **where**
 $x = ((q \cdot r) \cdot (r \cdot (q \cdot r)^{\textcircled{t}})^{\textcircled{k-1}})^{\textcircled{l-2}} \cdot$
 $((q \cdot r) \cdot (r \cdot (q \cdot r)^{\textcircled{t}})^{\textcircled{k-2}}) \cdot r \cdot q$ **and**
 $y = r \cdot (q \cdot r)^{\textcircled{t}}$ **and**
 $z = (q \cdot r) \cdot (r \cdot (q \cdot r)^{\textcircled{t}})^{\textcircled{k-1}}$ **and**
 $0 < t$ **and** $r \cdot q \neq q \cdot r$
 |
 — case $x \cdot y^{\textcircled{k}}$ with $2 \leq k$ and $<_s (y^{\textcircled{k}}) z$
 q **where**
 $x = (q \cdot y^{\textcircled{k}})^{\textcircled{l-1}} \cdot q$ **and**
 $z = q \cdot y^{\textcircled{k}}$ **and**

$q \cdot y \neq y \cdot q$
|
— case $x^{\textcircled{j}} \cdot y$ with $2 \leq j$
 $r \ q \ t$ **where**
 $x = (r \cdot q)^{\textcircled{t}} \cdot r$ **and**
 $y = q \cdot r \cdot r \cdot q$ **and**
 $z = (r \cdot q)^{\textcircled{t}} \cdot r \cdot r \cdot q$ **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:

$$x^i y^j = z^l$$

theorem *LS-parametric-solution*:

assumes $y \text{-le-} x$: $|y| \leq |x|$
and $j \text{-min}$: $1 \leq j$ **and** $k \text{-min}$: $1 \leq k$ **and** $l \text{-min}$: $2 \leq l$

shows

$$x^{\textcircled{j}} \cdot y^{\textcircled{k}} = z^{\textcircled{l}}$$

\longleftrightarrow

$(\exists r \ m \ n \ t.$

$$x = r^{\textcircled{m}} \wedge y = r^{\textcircled{n}} \wedge z = r^{\textcircled{t}} \wedge m * j + n * k = t * l) \text{ — Case A: } x, y \text{ is not a}$$

code

$$\vee (j = 1 \wedge k = 1) \wedge$$

$(\exists r \ q \ m \ n.$

$$x = (r \cdot q)^{\textcircled{m}} \cdot r \wedge y = q \cdot (r \cdot q)^{\textcircled{n}} \wedge z = r \cdot q \wedge m + n + 1 = l \wedge r \cdot q \neq q \cdot r)$$

— Case B

$$\vee (j = 1 \wedge 2 \leq k) \wedge$$

$(\exists r \ q.$

$$x = (q \cdot r^{\textcircled{k}})^{\textcircled{l-1}} \cdot q \wedge y = r \wedge z = q \cdot r^{\textcircled{k}} \wedge r \cdot q \neq q \cdot r) \text{ — Case C}$$

$$\vee (j = 1 \wedge 2 \leq k) \wedge$$

$(\exists r \ q \ t. 0 < t \wedge$

$$x = ((q \cdot r) \cdot (r \cdot (q \cdot r)^{\textcircled{t}})^{\textcircled{k-1}})^{\textcircled{l-2}} \cdot ((q \cdot r) \cdot (r \cdot (q \cdot r)^{\textcircled{t}})^{\textcircled{k-2}}) \cdot r \cdot q$$

$$\wedge y = r \cdot (q \cdot r)^{\textcircled{t}}$$

$$\wedge z = (q \cdot r) \cdot (r \cdot (q \cdot r)^{\textcircled{t}})^{\textcircled{k-1}}$$

$$\wedge r \cdot q \neq q \cdot r) \text{ — Case D}$$

$$\vee (j = 2 \wedge k = 1 \wedge l = 2) \wedge$$

$(\exists r \ q \ t. 2 \leq t \wedge$

$$x = (r \cdot q)^{\textcircled{t}} \cdot r \wedge y = q \cdot r \cdot r \cdot q$$

$$\wedge z = (r \cdot q)^{\textcircled{t}} \cdot r \cdot r \cdot q \wedge r \cdot q \neq q \cdot r) \text{ — Case E}$$

(is $?eq =$

$$(?sol\text{-}per \vee (?cond\text{-}j1k1 \wedge ?sol\text{-}j1k1) \vee$$

$$(?cond\text{-}j1k2 \wedge ?sol\text{-}j1k2\text{-}b) \vee$$

($?cond-j1k2 \wedge ?sol-j1k2-a$) \vee
($?cond-j2k1l2 \wedge ?sol-j2k1l2$))
<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 $x^{\textcircled{a}j} \cdot y^{\textcircled{a}k}$ and $x^{\textcircled{a}j'} \cdot y^{\textcircled{a}k'}$ 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 $0 < j$ **and** $0 < k$ **and** $\neg \text{primitive}(x^{\textcircled{a}j} \cdot y^{\textcircled{a}k})$
and $0 < k'$ **and** $\neg \text{primitive}(x^{\textcircled{a}j'} \cdot y^{\textcircled{a}k'})$
shows $k = k'$
<proof>

lemma *LS-unique-distinct-le*: **assumes** $x \cdot y \neq y \cdot x$
and $2 \leq j$ **and** $\neg \text{primitive}(x^{\textcircled{a}j} \cdot y)$
and $2 \leq k$ **and** $\neg \text{primitive}(x \cdot y^{\textcircled{a}k})$
and $|y| \leq |x|$
shows *False*
<proof>

lemma *LS-unique-distinct*: **assumes** $x \cdot y \neq y \cdot x$
and $2 \leq j$ **and** $\neg \text{primitive}(x^{\textcircled{a}j} \cdot y)$
and $2 \leq k$ **and** $\neg \text{primitive}(x \cdot y^{\textcircled{a}k})$
shows *False*
<proof>

lemma *LS-unique'*: **assumes** $x \cdot y \neq y \cdot x$
and $0 < j$ **and** $0 < k$ **and** $\neg \text{primitive}(x^{\textcircled{a}j} \cdot y^{\textcircled{a}k})$
and $0 < j'$ **and** $0 < k'$ **and** $\neg \text{primitive}(x^{\textcircled{a}j'} \cdot y^{\textcircled{a}k'})$
shows $k = k'$
<proof>

lemma *LS-unique*: **assumes** $x \cdot y \neq y \cdot x$
and $0 < j$ **and** $0 < k$ **and** $\neg \text{primitive}(x^{\textcircled{a}j} \cdot y^{\textcircled{a}k})$
and $0 < j'$ **and** $0 < k'$ **and** $\neg \text{primitive}(x^{\textcircled{a}j'} \cdot y^{\textcircled{a}k'})$
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*: $x \cdot y^{\textcircled{a}} k = z^{\textcircled{a}} l$

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 $x \ y \ k :: 'a :: \text{linordered-field}$

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.a₁* (*binA*)

notation *finite-2.a₂* (*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-a₂-iff*

lemmas *bin-eq-neq-iff'* = *Enum.neq-finite-2-a₁-iff*

abbreviation *bin-word-a* :: *binA list* (**a**) **where**

bin-word-a ≡ [*binA*]

abbreviation *bin-word-b* :: *binA list* (**b**) **where**

bin-word-b ≡ [*binb*]

abbreviation *binUNIV* :: *binA set* **where** *binUNIV* ≡ *UNIV*

lemma *binUNIV-I* [*simp, intro*]: *binA* ∈ *A* ⇒ *binb* ∈ *A* ⇒ *A* = *UNIV*

⟨*proof*⟩

lemma *bin-basis-code*: *code* {**a,b**}

⟨*proof*⟩

lemma *bin-num*: *binA* = 0 *binb* = 1

⟨*proof*⟩

lemma *binA-simps* [simp]: $\text{bina} - \text{binb} = \text{binb} \text{ binb} - \text{bina} = \text{binb } 1 - \text{bina} = \text{binb } 1 - \text{binb} = \text{bina } a - a = \text{bina } 1 - (1 - a) = a$

<proof>

definition *bin-swap* :: $\text{bin}A \Rightarrow \text{bin}A$ **where** *bin-swap* $x \equiv 1 - x$

lemma *bin-swap-if-then*: $1 - x = (\text{if } x = \text{bina} \text{ then } \text{binb} \text{ else } \text{bina})$

<proof>

definition *bin-swap-morph* **where** *bin-swap-morph* $\equiv \text{map } \text{bin-swap}$

lemma *alphabet-or*[simp]: $a = \text{bina} \vee a = \text{binb}$

<proof>

lemma *bin-im-or*: $f [a] = f \mathbf{a} \vee f [a] = f \mathbf{b}$

<proof>

thm *triv-forall-equality*

lemma *binUNIV-card*: $\text{card } \text{binUNIV} = 2$

<proof>

lemma *other-letter*: **obtains** b **where** $b \neq (a :: \text{bin}A)$

<proof>

lemma *alphabet-or-neg*: $x \neq y \implies x = (a :: \text{bin}A) \vee y = a$

<proof>

lemma *binA-neg-cases*: **assumes** $\text{neg}: a \neq b$

obtains $a = \text{bina}$ **and** $b = \text{binb} \mid a = \text{binb}$ **and** $b = \text{bina}$

<proof>

lemma *bin-neg-sym-pred*: **assumes** $a \neq b$ **and** $P \text{ bina } \text{binb}$ **and** $P \text{ binb } \text{bina}$ **shows** $P a b$

<proof>

lemma *no-third*: $(c :: \text{bin}A) \neq a \implies b \neq a \implies b = c$

<proof>

lemma *two-in-bin-UNIV*: **assumes** $a \neq b$ **and** $a \in S$ **and** $b \in S$ **shows** $S = \text{binUNIV}$

<proof>

lemmas *two-in-bin-set* = *two-in-bin-UNIV*[*unfolded bin-UNIV*]

lemma *bin-not-comp-set-UNIV*: **assumes** $\neg u \bowtie v$ **shows** $\text{set } (u \cdot v) = \text{binUNIV}$

<proof>

lemma *bin-basis-singletons*: $\{[q] \mid q. q \in \{\text{bina}, \text{binb}\}\} = \{\mathbf{a}, \mathbf{b}\}$

<proof>

lemma *bin-basis-generates*: $\langle \{\mathbf{a}, \mathbf{b}\} \rangle = UNIV$
<proof>

lemma *a-in-bin-basis*: $[a] \in \{\mathbf{a}, \mathbf{b}\}$
<proof>

lemma *lcp-zero-one-emp*: $\mathbf{a} \wedge_p \mathbf{b} = \varepsilon$ **and** *lcp-one-zero-emp*: $\mathbf{b} \wedge_p \mathbf{a} = \varepsilon$
<proof>

lemma *bin-neq-induct*: $(a :: binA) \neq b \implies P a \implies P b \implies P c$
<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'*: $P (a :: binA) \implies P (1 - a) \implies (\bigwedge c. P c)$
<proof>

lemma *swap-UNIV*: $\{a, 1 - a\} = binUNIV$ (**is** $?P a$)
<proof>

lemma *bin-neq-swap'*[*intro*]: $a \neq b \implies 1 - b = (a :: binA)$
<proof>

lemma *bin-neq-swap*[*intro*]: $a \neq b \implies 1 - a = (b :: binA)$
<proof>

lemma *bin-neq-swap''*[*intro*]: $a \neq b \implies b = 1 - (a :: binA)$
<proof>

lemma *bin-neq-swap'''*[*intro*]: $a \neq b \implies a = 1 - (b :: binA)$
<proof>

lemma *bin-neq-iff*: $c \neq d \iff 1 - d = (c :: binA)$
<proof>

lemma *bin-neq-iff'*: $c \neq d \iff 1 - c = (d :: binA)$

<proof>

lemma *binA-neg-cases-swap*: **assumes** $neg: a \neq (b :: binA)$
obtains $a = c$ **and** $b = 1 - c \mid a = 1 - c$ **and** $b = c$
<proof>

lemma *im-swap-neg*: $f a = f b \implies f bina \neq f binb \implies a = b$
<proof>

lemma *bin-without-letter*: **assumes** $(a :: binA) \notin set w$
obtains k **where** $w = [1-a]^{\circledast k}$
<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*: $a \in S \implies 1-a \in S \implies S = binUNIV$
<proof>

lemma *bin-sing-iff*: $A = \{a :: binA\} \longleftrightarrow a \in A \wedge 1-a \notin A$
<proof>

lemma *set-binUNIV-iff*: $\neg set w \subseteq \{hd w\} \longleftrightarrow set w = binUNIV$
<proof>

lemma *bin-set-cases*: **obtains** $S = \{\} \mid S = \{a\} \mid S = \{1-a\} \mid S = binUNIV$
<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*: $set w \subseteq \{hd w\} \longleftrightarrow set w \neq binUNIV$
<proof>

lemma *not-swap-eq*: $P a b \implies (\bigwedge (c :: binA). \neg P c (1-c)) \implies a = b$
<proof>

lemma *bin-distinct-letter*: **assumes** $set w = binUNIV$
obtains $k w'$ **where** $[hd w]^{\circledast Suc k} \cdot [1-hd w] \cdot w' = w$

<proof>

lemma $P a \longleftrightarrow P (1-a) \implies P a \implies (\bigwedge (b :: \text{bin}A). P b)$
<proof>

lemma *bin-sym-all*: $P (a :: \text{bin}A) \longleftrightarrow P (1-a) \implies P a \implies P x$
<proof>

lemma *bin-sym-all-comm*:
 $f [a] \cdot f [1-a] \neq f [1-a] \cdot f [a] \implies f [b] \cdot f [1-b] \neq f [1-b] \cdot f [(b :: \text{bin}A)]$ (**is**
 $?P a \implies ?P b$)
<proof>

lemma *bin-sym-all-neg*:
 $f [(a :: \text{bin}A)] \neq f [1-a] \implies f [b] \neq f [1-b]$ (**is** $?P a \implies ?P b$)
<proof>

lemma *bin-len-count*:
fixes $w :: \text{bin}A \text{ list}$
shows $|w| = \text{count-list } w \ a \ + \ \text{count-list } w \ (1-a)$
<proof>

lemma *bin-len-count'*:
fixes $w :: \text{bin}A \text{ list}$
shows $|w| = \text{count-list } w \ \text{bina} \ + \ \text{count-list } w \ \text{binb}$
<proof>

7.2 Binary morphisms

lemma *bin-map-core-lists*: $(\text{map } f^C \ w) \in \text{lists } \{f \ \mathbf{a}, f \ \mathbf{b}\}$
<proof>

lemma *bin-range*: $\text{range } f = \{f \ \text{bina}, f \ \text{binb}\}$
<proof>

lemma *bin-core-range*: $\text{range } f^C = \{f \ \mathbf{a}, f \ \mathbf{b}\}$
<proof>

lemma *bin-core-range-swap*: $\text{range } f^C = \{f [(a :: \text{bin}A)], f [1-a]\}$ (**is** $?P a$)
<proof>

lemma *bin-map-core-lists-swap*: $(\text{map } f^C \ w) \in \text{lists } \{f [(a :: \text{bin}A)], f [1-a]\}$
<proof>

locale *binary-morphism = morphism f*
for $f :: \text{bin}A \text{ list} \Rightarrow 'a \text{ list}$
begin

lemma *bin-len-count-im*:

fixes $a :: \text{bin}A$
shows $|f w| = \text{count-list } w \ a * |f [a]| + \text{count-list } w \ (1-a) * |f [1-a]|$
 $\langle \text{proof} \rangle$

lemma *bin-len-count-im'*:
shows $|f w| = \text{count-list } w \ \mathbf{bina} * |f \mathbf{a}| + \text{count-list } w \ \mathbf{binb} * |f \mathbf{b}|$
 $\langle \text{proof} \rangle$

lemma *bin-neq-inj-core*: **assumes** $f [a] \neq f [1-a]$ **shows** *inj* f^C
 $\langle \text{proof} \rangle$

lemma *bin-code-morphism-inj*: **assumes** $f [a] \cdot f [1-a] \neq f [1-a] \cdot f [a]$
shows *inj* f
 $\langle \text{proof} \rangle$

lemma *bin-code-morphismI*: $f [a] \cdot f [1-a] \neq f [1-a] \cdot f [a] \implies \text{code-morphism } f$
 $\langle \text{proof} \rangle$

end

7.2.1 Binary periodic morphisms

locale *binary-periodic-morphism* = *periodic-morphism* f
for $f :: \text{bin}A \ \text{list} \Rightarrow 'a \ \text{list}$
begin

sublocale *binary-morphism*
 $\langle \text{proof} \rangle$

definition *fn0* **where** $\text{fn0} \equiv (\text{SOME } n. f \ \mathbf{a} = \text{mroot}^{\textcircled{n}})$

definition *fn1* **where** $\text{fn1} \equiv (\text{SOME } n. f \ \mathbf{b} = \text{mroot}^{\textcircled{n}})$

lemma *bin0-im*: $f \ \mathbf{a} = \text{mroot}^{\textcircled{\text{fn0}}}$
 $\langle \text{proof} \rangle$

lemma *bin1-im*: $f \ \mathbf{b} = \text{mroot}^{\textcircled{\text{fn1}}}$
 $\langle \text{proof} \rangle$

lemma *sorted-image* : $f w = (f [a])^{\textcircled{\text{count-list } w \ a}} \cdot (f [1-a])^{\textcircled{\text{count-list } w \ (1-a)}}$
 $\langle \text{proof} \rangle$

lemma *bin-per-morph-expI*: $f u = \text{mroot}^{\textcircled{(\text{im-exp } \mathbf{a}) * (\text{count-list } u \ \mathbf{bina}) + \text{im-exp } \mathbf{b} * (\text{count-list } u \ \mathbf{binb})}}$
 $\langle \text{proof} \rangle$

end

7.3 From two words to a binary morphism

definition *bin-morph-of'*: 'a list \Rightarrow 'a list \Rightarrow binA list \Rightarrow 'a list **where** *bin-morph-of'*
 $x\ y\ u = \text{concat} (\text{map} (\lambda\ a.\ (\text{case}\ a\ \text{of}\ \text{bina} \Rightarrow x \mid \text{binb} \Rightarrow y))\ u)$

definition *bin-morph-of*: 'a list \Rightarrow 'a list \Rightarrow binA list \Rightarrow 'a list **where** *bin-morph-of*
 $x\ y\ u = \text{concat} (\text{map} (\lambda\ a.\ \text{if}\ a = \text{bina}\ \text{then}\ x\ \text{else}\ y)\ u)$

lemma *case-finite-2-if-else*: *case-finite-2* $x\ y = (\lambda\ a.\ \text{if}\ a = \text{bina}\ \text{then}\ x\ \text{else}\ y)$
 $\langle \text{proof} \rangle$

lemma *bin-morph-of-case-def*: *bin-morph-of* $x\ y\ u = \text{concat} (\text{map} (\lambda\ a.\ (\text{case}\ a\ \text{of}\ \text{bina} \Rightarrow x \mid \text{binb} \Rightarrow y))\ u)$
 $\langle \text{proof} \rangle$

lemma *case-finiteD*: *case-finite-2* $(f\ \mathbf{a})\ (f\ \mathbf{b}) = f^{\mathbf{C}}$
 $\langle \text{proof} \rangle$

lemma *case-finiteD'*: *case-finite-2* $(f\ \mathbf{a})\ (f\ \mathbf{b})\ u = f^{\mathbf{C}}\ u$
 $\langle \text{proof} \rangle$

lemma *bin-morph-of-maps*: *bin-morph-of* $x\ y = \text{List.maps} (\text{case-finite-2}\ x\ y)$
 $\langle \text{proof} \rangle$

lemma *bin-morph-ofD*: $(\text{bin-morph-of}\ x\ y)\ \mathbf{a} = x\ (\text{bin-morph-of}\ x\ y)\ \mathbf{b} = y$
 $\langle \text{proof} \rangle$

lemma *bin-range-swap*: $\text{range}\ f = \{f\ (a::\text{binA}), f\ (1-a)\}$ (**is** $?P\ a$)
 $\langle \text{proof} \rangle$

lemma *bin-morph-of-core-range*: $\text{range}\ (\text{bin-morph-of}\ x\ y)^{\mathbf{C}} = \{x, y\}$
 $\langle \text{proof} \rangle$

lemma *bin-morph-of-morph*: *morphism* $(\text{bin-morph-of}\ x\ y)$
 $\langle \text{proof} \rangle$

lemma *bin-morph-of-bin-morph*: *binary-morphism* $(\text{bin-morph-of}\ x\ y)$
 $\langle \text{proof} \rangle$

lemma *bin-morph-of-range*: $\text{range}\ (\text{bin-morph-of}\ x\ y) = \langle \{x, y\} \rangle$
 $\langle \text{proof} \rangle$

context *binary-code*
begin

lemma *code-morph-of*: *code-morphism* $(\text{bin-morph-of}\ u_0\ u_1)$
 $\langle \text{proof} \rangle$

lemma *inj-morph-of*: *inj* $(\text{bin-morph-of}\ u_0\ u_1)$

<proof>

end

7.4 Two binary morphism

locale *two-binary-morphisms = two-morphisms* $g\ h$
for $g\ h :: \text{binA list} \Rightarrow 'a\ \text{list}$

begin

lemma *eq-on-letters-eq*: $g\ \mathbf{a} = h\ \mathbf{a} \implies g\ \mathbf{b} = h\ \mathbf{b} \implies g = h$
<proof>

sublocale *g: binary-morphism* g
<proof>

sublocale *h: binary-morphism* h
<proof>

lemma *rev-morphs: two-binary-morphisms* $(\text{rev-map } g)\ (\text{rev-map } h)$
<proof>

lemma *solution-UNIV*:
assumes $s \neq \varepsilon$ and $g\ s = h\ s$ and $\bigwedge a. g\ [a] \neq h\ [a]$
shows $\text{set } s = \text{UNIV}$
<proof>

lemma *solution-len-im-sing-less*:
assumes *sol*: $g\ s = h\ s$ and *set*: $a \in \text{set } s$ 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*: $\text{set } s = \text{UNIV}$ and *less*: $|g\ [a]| \leq |h\ [a]|$
shows $|h\ [1-a]| \leq |g\ [1-a]|$
<proof>

lemma *solution-sing-len-cases*:
assumes *set*: $\text{set } s = \text{UNIV}$ 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 $g\ s = h\ s$ and $g \neq h$ and $\text{set } s = \text{binUNIV}$
shows $|g\ [c]| \neq |h\ [c]|$
<proof>

end

lemma *two-binary-morphismsI*: *binary-morphism* $g \implies \text{binary-morphism } h \implies$
two-binary-morphisms $g \ h$
 ⟨*proof*⟩

7.5 Binary code morphism

7.5.1 Locale - binary code morphism

locale *binary-code-morphism* = *code-morphism* $f :: \text{binA list} \Rightarrow 'a \text{ list}$ **for** f

begin

lemma *morph-bin-morph-of*: $f = \text{bin-morph-of } (f \ \mathbf{a}) \ (f \ \mathbf{b})$
 ⟨*proof*⟩

lemma *non-comm-morph* [*simp*]: $f \ [a] \cdot f \ [1-a] \neq f \ [1-a] \cdot f \ [a]$
 ⟨*proof*⟩

lemma *non-comp-morph*: $\neg f \ [a] \cdot f \ [1-a] \bowtie f \ [1-a] \cdot f \ [a]$
 ⟨*proof*⟩

lemma *swap-non-comm-morph* [*simp*, *intro*]: $a \neq b \implies f \ [a] \cdot f \ [b] \neq f \ [b] \cdot f \ [a]$
 ⟨*proof*⟩

thm *bin-core-range*[*of* f]

lemma *bin-code-morph-rev-map*: *binary-code-morphism* (*rev-map* f)
 ⟨*proof*⟩

sublocale *swap*: *binary-code* $f \ \mathbf{b} \ f \ \mathbf{a}$
 ⟨*proof*⟩

sublocale *binary-code* $f \ \mathbf{a} \ f \ \mathbf{b}$
 ⟨*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* ($f \ \mathbf{a}$) ($f \ \mathbf{b}$)

abbreviation *bin-morph-mismatch* (\mathbf{c})

where *bin-morph-mismatch* $a \equiv \text{bin-mismatch } (f[a]) \ (f[1-a])$

abbreviation *bin-morph-mismatch-suf* (\mathfrak{d})

where *bin-morph-mismatch-suf* $a \equiv \text{bin-mismatch-suf } (f[1-a]) \ (f[a])$

lemma *bin-lcp-def'*: $\alpha = f \ ([a] \cdot [1-a]) \wedge_p f \ ([1-a] \cdot [a])$
 ⟨*proof*⟩

lemma *bin-lcp-neq*: $a \neq b \implies \alpha = f \ ([a] \cdot [b]) \wedge_p f \ ([b] \cdot [a])$

<proof>

lemma *sing-in*: $f [a] \in \{f \mathbf{a}, f \mathbf{b}\}$
<proof>

lemma *bin-mismatch-inj*: *inj c*
<proof>

lemma *map-in-lists*: $\text{map } (\lambda x. f [x]) w \in \text{lists } \{f \mathbf{a}, f \mathbf{b}\}$
<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 d*
<proof>

lemma *bin-lcp-sing*: $\text{bin-lcp } (f [a]) (f [1-a]) = \alpha$
<proof>

lemma *bin-lcs-sing*: $\text{bin-lcs } (f [a]) (f [1-a]) = \beta$
<proof>

lemma *bin-code-morph-sing*: $\text{binary-code } (f [a]) (f [1-a])$
<proof>

lemma *bin-mismatch-swap-neq*: $\mathbf{c} a \neq \mathbf{c} (1-a)$
<proof>

lemma *set w* $\subseteq \{\text{hd } w\} \longleftrightarrow (\exists k. w = [\text{hd } w]^{\textcircled{k}})$
<proof>

lemma *long-bin-lcp-hd*: **assumes** $|f w| \leq |\alpha|$
shows $\text{set } w \subseteq \{\text{hd } w\}$
<proof>

thm *nonerasing*
nonerasing-morphism.nonerasing
lemmas *nonerasing* = *nonerasing*
thm *nonerasing-morphism.nonerasing*
binary-code-morphism.nonerasing

lemma *bin-morph-lcp-mismatch-pref*:

$\alpha \cdot [\mathbf{c} \ a] \leq_p f \ [a] \cdot \alpha$
<proof>

lemma $[\mathfrak{d} \ a] \cdot \beta \leq_s \beta \cdot f \ [a]$ *<proof>*

lemma *bin-lcp-pref-all*: $\alpha \leq_p f \ w \cdot \alpha$
<proof>

lemma *bin-lcp-spref-all*: $w \neq \varepsilon \implies \alpha <_p f \ w \cdot \alpha$
<proof>

lemma *pref-mono-lcp*: **assumes** $w \leq_p w'$ **shows** $f \ w \cdot \alpha \leq_p f \ w' \cdot \alpha$
<proof>

lemma *long-bin-lcp*: **assumes** $|f \ w| \leq |\alpha|$
shows $set \ w \subseteq \{hd \ w\}$
<proof>

thm *sing-to-nemp*
nonerasing

lemma *bin-mismatch-code-morph*: $c_0 = \mathbf{c} \ 0 \ c_1 = \mathbf{c} \ 1$
<proof>

lemma *bin-lcp-mismatch-pref-all*: $\alpha \cdot [\mathbf{c} \ a] \leq_p f \ [a] \cdot f \ w \cdot \alpha$
<proof>

lemma *bin-fst-mismatch-all*: $\alpha \cdot [c_0] \leq_p f \ \mathbf{a} \cdot f \ w \cdot \alpha$
<proof>

lemma *bin-snd-mismatch-all*: $\alpha \cdot [c_1] \leq_p f \ \mathbf{b} \cdot f \ w \cdot \alpha$
<proof>

lemma *bin-long-mismatch*: **assumes** $|\alpha| < |f \ w|$ **shows** $\alpha \cdot [\mathbf{c} \ (hd \ w)] \leq_p f \ w$
<proof>

lemma *sing-pow-mismatch*: **assumes** $f \ [a] = [b]^{\textcircled{q}} Suc \ n$ **shows** $\mathbf{c} \ a = b$
<proof>

lemma *sing-pow-mismatch-suf*: $f \ [a] = [b]^{\textcircled{q}} Suc \ n \implies \mathfrak{d} \ a = b$
<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*: $a \neq b \implies f \ [a] \cdot f \ w \cdot \alpha \wedge_p f \ [b] \cdot f \ w' \cdot \alpha = \alpha$
<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** *set w = binUNIV*
shows $\alpha \leq_p (f w)$
 ⟨proof⟩

lemma *bin-lcp-pref''*: $[a] \leq f w \implies [1-a] \leq f w \implies \alpha \leq_p (f w)$
 ⟨proof⟩

lemma *bin-lcp-pref'*: $\mathbf{a} \leq f w \implies \mathbf{b} \leq f w \implies \alpha \leq_p (f w)$
 ⟨proof⟩

lemma *bin-lcp-mismatch-pref-all-set*: **assumes** $1-a \in \text{set } w$
shows $\alpha \cdot [c a] \leq_p f [a] \cdot f w$
 ⟨proof⟩

lemma *bin-lcp-comp-hd*: $\alpha \bowtie f (\mathbf{a} \cdot w0) \wedge_p f (\mathbf{b} \cdot w1)$
 ⟨proof⟩

lemma *sing-mismatch*: **assumes** $\text{set } (f \mathbf{a}) \subseteq \{a\}$ **shows** $c_0 = a$
 ⟨proof⟩

lemma *sing-mismatch'*: **assumes** $\text{set } (f \mathbf{b}) \subseteq \{a\}$ **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** $\text{set } w = \text{UNIV}$ **shows** $|f [a]| + |f [1-a]|$
 $\leq |f w|$
 ⟨proof⟩

lemma *set-UNIV-bin-lcp-pref*: **assumes** $\text{set } w = \text{UNIV}$ **shows** $\alpha \cdot [c (hd w)] \leq_p$
 $f w$
 ⟨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*: $f w \cdot \alpha = \alpha \cdot f w \implies (\forall a. a \in \text{set } w \longrightarrow f [a] \cdot \alpha = \alpha \cdot f [a])$
 ⟨*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** $w = [\text{hd } w]^{\textcircled{k}}$ **and** $0 < k$
 ⟨*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* $f \longleftrightarrow \text{morphism } f \wedge f [a] \cdot f [1-a] \neq f [1-a] \cdot f [a]$
 ⟨*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-nonzer-morph-iff*: *nonerasing-morphism* (*bin-morph-of* $x y$) $\longleftrightarrow x \neq \varepsilon \wedge y \neq \varepsilon$
 ⟨*proof*⟩

lemma *morph-bin-morph-of*: *morphism* $f \longleftrightarrow \text{bin-morph-of } (f \mathbf{a}) (f \mathbf{b}) = f$
 ⟨*proof*⟩

lemma *two-bin-code-morphs-nonerasing-morphs*: *binary-code-morphism* $g \implies$ *binary-code-morphism* $h \implies$ *two-nonerasing-morphisms* $g\ h$
 ⟨*proof*⟩

7.6 Marked binary morphism

lemma *marked-binary-morphI*: **assumes** *morphism* f **and** $f\ [a :: \text{bin}A] \neq \varepsilon$ **and** $f\ [1-a] \neq \varepsilon$ **and** $\text{hd}\ (f\ [a]) \neq \text{hd}\ (f\ [1-a])$
shows *marked-morphism* f
 ⟨*proof*⟩

locale *marked-binary-morphism* = *marked-morphism* $f :: \text{bin}A\ \text{list} \Rightarrow 'a\ \text{list}$ **for** f
begin

lemma *bin-marked*: $\text{hd}\ (f\ \mathbf{a}) \neq \text{hd}\ (f\ \mathbf{b})$
 ⟨*proof*⟩

lemma *bin-marked-sing*: $\text{hd}\ (f\ [a]) \neq \text{hd}\ (f\ [1-a])$
 ⟨*proof*⟩

sublocale *binary-code-morphism*
 ⟨*proof*⟩

lemma *marked-lcp-emp*: $\alpha = \varepsilon$
 ⟨*proof*⟩

lemma *bin-marked'*: $(f\ \mathbf{a})!0 \neq (f\ \mathbf{b})!0$
 ⟨*proof*⟩

lemma *marked-bin-morph-prefix-code*: $r \bowtie s \vee f\ (r \cdot z1) \wedge_p f\ (s \cdot z2) = f\ (r \wedge_p s)$
 ⟨*proof*⟩

end

lemma *bin-marked-preimg-hd*:
assumes *marked-binary-morphism* $(f :: \text{bin}A\ \text{list} \Rightarrow \text{bin}A\ \text{list})$

obtains c **where** $hd (f [c]) = a$
<proof>

7.7 Marked version

context *binary-code-morphism*

begin

definition *marked-version* (f_m) **where** $f_m = (\lambda w. \alpha^{-1} > (f w \cdot \alpha))$

lemma *marked-version-conjugates*: $\alpha \cdot f_m w = f w \cdot \alpha$
<proof>

lemma *marked-eq-conv*: $f w = f w' \iff f_m w = f_m w'$
<proof>

lemma *marked-marked*: **assumes** *marked-morphism* f **shows** $f_m = f$
<proof>

lemma *marked-version-all-nemp*: $w \neq \varepsilon \implies f_m w \neq \varepsilon$
<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*: $bin\text{-}lcs (f_m \mathbf{a}) (f_m \mathbf{b}) = \beta \cdot \alpha$
<proof>

lemma *bin-lcp-shift*: **assumes** $|\alpha| < |f w|$ **shows** $(f w)!|\alpha| = hd (f_m w)$
<proof>

lemma *mismatch-fst*: $hd (f_m \mathbf{a}) = c_0$
<proof>

lemma *mismatch-snd*: $hd (f_m \mathbf{b}) = c_1$
<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*: $f u \cdot \alpha \leq_p f w \cdot \alpha \implies u \leq_p w$
 ⟨proof⟩

lemma *mismatch-pref0*: $[c_0] \leq_p f_m \mathbf{a}$
 ⟨proof⟩

lemma *mismatch-pref1*: $[c_1] \leq_p f_m \mathbf{b}$
 ⟨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*: $f u \cdot \alpha \leq_p f v \cdot \alpha \iff u \leq_p v$
 ⟨proof⟩

lemma *bin-morph-compare-conv*: $f u \cdot \alpha \bowtie f v \cdot \alpha \iff u \bowtie v$
 ⟨proof⟩

lemma *code-lcp'*: $\neg r \bowtie s \implies \alpha \leq_p f z \implies \alpha \leq_p f z' \implies f (r \cdot z) \wedge_p f (s \cdot z')$
 $= f (r \wedge_p s) \cdot \alpha$
 ⟨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*: $\alpha' \leq_p \alpha$

begin

definition *shifted-f* **where** $\text{shifted-}f = (\lambda w. \alpha'^{-1} \triangleright (f w \cdot \alpha'))$

lemma *shift-pref-all*: $\alpha' \leq_p f w \cdot \alpha'$
 ⟨proof⟩

sublocale *shifted: binary-code-morphism shifted-f*
<proof>

lemma *shifted-lcp: $\alpha' \cdot \text{shifted.bin-code-lcp} = \alpha$*
<proof>

lemma *$\alpha' = \alpha \implies \text{shifted-f} = f_m$*
<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 \Rightarrow '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** $\neg r \boxtimes s$

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* $s = \text{binUNIV}$

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*: $g r = h s \iff \alpha \cdot g_m r = h_m s \cdot \alpha$

<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** $\alpha' = (\text{if } |\alpha_g| \leq |\alpha_h| \text{ then } \alpha_g \text{ else } \alpha_h)$

definition g' **where** $g' = (\text{if } |\alpha_g| \leq |\alpha_h| \text{ then } (\lambda w. \alpha'^{-1} \langle g w \cdot \alpha' \rangle) \text{ else } (\lambda w. \alpha'^{-1} \langle h w \cdot \alpha' \rangle))$

definition h' **where** $h' = (\text{if } |\alpha_g| \leq |\alpha_h| \text{ then } (\lambda w. \alpha'^{-1} \langle h w \cdot \alpha' \rangle) \text{ else } (\lambda w. \alpha'^{-1} \langle g w \cdot \alpha' \rangle))$

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* $(\lambda w. \alpha'^{-1} \langle g w \cdot \alpha' \rangle)$
<proof>

interpretation *shifted-h*: *binary-code-morphism* $(\lambda w. \alpha'^{-1} \langle h w \cdot \alpha' \rangle)$
<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* = ε
<proof>

lemma *shifted-alphas*: **assumes** le : $|\alpha_g| \leq |\alpha_h|$
shows $\alpha' \cdot \text{shifted.g.bin-code-lcp} = \alpha_g$ **and** $\alpha' \cdot \text{shifted.h.bin-code-lcp} = \alpha_h$
 $\langle \text{proof} \rangle$

interpretation *swapped*: *binary-code-coincidence-sym* h g
 $\langle \text{proof} \rangle$

lemma *eq-len-eq-conv*: $\alpha_g = \alpha_h \longleftrightarrow |\alpha_g| = |\alpha_h|$
 $\langle \text{proof} \rangle$

lemma *shift-swapped*: $\text{swapped}.\alpha' = \alpha'$
 $\langle \text{proof} \rangle$

lemma *morphs-swapped*: **assumes** $|\alpha_g| \neq |\alpha_h|$ **shows** $\text{swapped}.g' = g'$ $\text{swapped}.h'$
 $= h'$
 $\langle \text{proof} \rangle$

lemma *morphs-swapped'*: **assumes** $|\alpha_g| = |\alpha_h|$ **shows** $\text{swapped}.g' = h'$ $\text{swapped}.h'$
 $= g'$
 $\langle \text{proof} \rangle$

lemma *shifted-lcp-len-eq*: $|\text{shifted.g.bin-code-lcp}| = |\text{shifted.h.bin-code-lcp}| \longleftrightarrow |\alpha_g|$
 $= |\alpha_h|$ **and**
shifted-lcp-len-le: $|\text{shifted.g.bin-code-lcp}| \leq |\text{shifted.h.bin-code-lcp}|$
 $\langle \text{proof} \rangle$

end

locale *two-marked-binary-morphisms* = *two-marked-morphisms* g h
for g h :: *binA list* \Rightarrow *'a list*
begin

sublocale *two-binary-code-morphisms* g h $\langle \text{proof} \rangle$

lemma *not-comm-im*: **assumes** $g \neq h$ **and** g $s = h$ s **and** $s \neq \varepsilon$
and hd $s = a$ **and** set $s = \text{binUNIV}$
shows $g[a] \cdot h$ $[a] \neq h[a] \cdot g[a]$
 $\langle \text{proof} \rangle$

lemma *sol-set-not-com-hd*:
assumes

morphs-neg: $g \neq h$ **and**
sol: $g s = h s$ **and**
sol-set: $set s = binUNIV$
shows $g ([hd s]) \cdot h ([hd s]) \neq h ([hd s]) \cdot g ([hd s])$
 ⟨*proof*⟩

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*: $\bigwedge a. blockP a$

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*: $[b] \leq_f suc-fst ([a] \cdot [1-a])$
 ⟨*proof*⟩

lemma *blocks-all-letters-snd*: $[b] \leq_f suc-snd ([a] \cdot [1-a])$
 ⟨*proof*⟩

lemma *lcs-suf-blocks-fst*: $g.bin-code-lcs \leq_s g (suc-fst ([a] \cdot [1-a]))$
 ⟨*proof*⟩

lemma *lcs-suf-blocks-snd*: $h.bin-code-lcs \leq_s h (suc-snd ([a] \cdot [1-a]))$
 ⟨*proof*⟩

lemma *lcs-fst-suf-snd*: $g.bin-code-lcs \leq_s h.bin-code-lcs \cdot h \text{ sucs}.h.bin-code-lcs$
 ⟨*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 \Rightarrow 'a list \Rightarrow bool

where *bin-prim* $x\ y \iff$ *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 list concatenation

lemma *bin-concat-prim-pres-noner1*:

assumes $x \neq y$

and *prim-pres*: $\bigwedge ws. ws \in lists\ \{x,y\} \implies 2 \leq |ws| \implies primitive\ ws \implies primitive\ (concat\ ws)$

shows $x \neq \varepsilon$

<proof>

lemma *bin-concat-prim-pres-noner*:

assumes $x \neq y$

and *prim-pres*: $\bigwedge ws. ws \in lists\ \{x,y\} \implies 2 \leq |ws| \implies primitive\ ws \implies primitive\ (concat\ ws)$

shows *nonerasing-morphism* (*bin-morph-of* $x\ y$)

<proof>

lemma *bin-prim-concat-prim-pres-conv*:

assumes $x \neq y$

shows *bin-prim* $x\ y \iff (\forall ws \in lists\ \{x,y\}. 2 \leq |ws| \implies primitive\ ws \implies primitive\ (concat\ ws))$

(is \iff ?condition)

<proof>

lemma *bin-prim-concat-prim-pres*:

assumes *bin-prim* $x\ y$

shows $ws \in lists\ \{x, y\} \implies 2 \leq |ws| \implies primitive\ ws \implies primitive\ (concat\ ws)$

<proof>

lemma *bin-prim-altdef1*:

bin-prim $x\ y \iff$

$(x \neq y) \wedge (\forall ws \in lists\ \{x,y\}. 2 \leq |ws| \implies primitive\ ws \implies primitive\ (concat\ ws))$

<proof>

lemma *bin-prim-altdef2*:

bin-prim $x\ y \longleftrightarrow$
 $(x \cdot y \neq y \cdot x) \wedge (\forall ws \in \text{lists } \{x,y\}. 2 \leq |ws| \longrightarrow \text{primitive } ws \longrightarrow \text{primitive}$
(*concat* ws))
(*proof*)

7.10.2 Basic properties of *bin-prim*

lemma *bin-prim-irrefl*: $\neg \text{bin-prim } x\ x$

(*proof*)

lemma *bin-prim-symm* [*sym*]: $\text{bin-prim } x\ y \implies \text{bin-prim } y\ x$

(*proof*)

lemma *bin-prim-commutes*: $\text{bin-prim } x\ y \longleftrightarrow \text{bin-prim } y\ x$

(*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 Miscellanea

8.1.1 Mismatch additions

lemma *mismatch-pref-comm-len*: **assumes** $w1 \in \langle\{u,v\}\rangle$ **and** $w2 \in \langle\{u,v\}\rangle$ **and**
 $p \leq_p w1$
 $u \cdot p \leq_p v \cdot w2$ **and** $|v| \leq |p|$
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**
 $u \cdot w1 \cdot v \leq_p v \cdot w2 \cdot u$
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*]

8.1.2 Conjugate words with conjugate periods

lemma *conj-pers-conj-comm-aux*:
assumes $(u \cdot v)^{\textcircled{k}} \cdot u = r \cdot s$ **and** $(v \cdot u)^{\textcircled{l}} \cdot v = (s \cdot r)^{\textcircled{m}}$ **and** $0 < k \ 0 < l$
and $2 \leq m$
shows $u \cdot v = v \cdot u$

<proof>

lemma *conj-pers-conj-comm*: **assumes** $\varrho (v \cdot (u \cdot v)^{\textcircled{k}}) \sim \varrho ((u \cdot v)^{\textcircled{m}} \cdot u)$ **and**
 $0 < k$ **and** $0 < m$

shows $u \cdot v = v \cdot u$

<proof>

hide-fact *conj-pers-conj-comm-aux*

8.1.3 Covering uvvu

lemma *uv-fac-uvv*: **assumes** $p \cdot u \cdot v \leq_p u \cdot v \cdot v$ **and** $p \neq \varepsilon$ **and** $p \leq_s w$ **and** $w \in \langle \{u, v\} \rangle$

shows $u \cdot v = v \cdot u$

<proof>

lemmas *uv-fac-uvv-suf* = *uv-fac-uvv[reversed, unfolded rassoc]*

lemma $u \leq_p v \implies u' \leq_p v' \implies u \wedge_p u' \neq u \implies u \wedge_p u' \neq u' \implies u \wedge_p u' = v \wedge_p v'$

<proof>

lemma *comm-puv-pvs-eq-ug*: **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**

$p \leq_p u$ **and** $q \leq_p w$ **and** $s \leq_p w'$ **and**

$w \in \langle \{u, v\} \rangle$ **and** $w' \in \langle \{u, v\} \rangle$ **and** $|u| \leq |s|$

shows $u \cdot v = v \cdot u$

<proof>

lemma *uvu-pref-uvvu*: **assumes** $p \cdot u \cdot v \cdot v \cdot u = u \cdot v \cdot u \cdot q$ **and**

$p \leq_p u$ **and** $q \leq_p w$ **and** $w \in \langle \{u, v\} \rangle$

shows $u \cdot v = v \cdot u$

<proof>

lemma *uvu-pref-uvvu-interp*: **assumes** *interp*: $p\ u \cdot v \cdot v \cdot u\ s \sim_{\mathcal{I}}\ ws$ **and**
 $[u, v, u] \leq_p\ ws$ **and** $ws \in \text{lists } \{u, v\}$
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*: $p\ u \cdot v \cdot v \cdot u\ s \sim_{\mathcal{I}}\ ws \implies [u, v, u] \leq_s\ ws$
 $\implies ws \in \text{lists } \{u, v\} \implies u \cdot v = v \cdot u$
<proof>

8.1.4 Conjugate words

lemma *conjug-pref-suf-mismatch*: **assumes** $w1 \in \langle \{r \cdot s, s \cdot r\} \rangle$ **and** $w2 \in \langle \{r \cdot s, s \cdot r\} \rangle$
and $r \cdot w1 = w2 \cdot s$
shows $r = s \vee r = \varepsilon \vee s = \varepsilon$
<proof>

lemma *conjug-conjug-primroots*: **assumes** $u \neq \varepsilon$ **and** $r \neq \varepsilon$ **and** $\varrho(u \cdot v) = r \cdot s$
and $\varrho(v \cdot u) = s \cdot r$
obtains $k\ m$ **where** $(r \cdot s)^{\textcircled{a}}k \cdot r = u$ **and** $(s \cdot r)^{\textcircled{a}}m \cdot s = v$
<proof>

8.1.5 Predicate “commutes”

definition *commutes* :: 'a list set \implies bool
where *commutes* $A = (\forall x\ y. x \in A \longrightarrow y \in A \longrightarrow x \cdot y = y \cdot x)$

lemma *commutesE*: *commutes* $A \implies x \in A \implies y \in A \implies x \cdot y = y \cdot x$
<proof>

lemma *commutes-root*: **assumes** *commutes* A
obtains r **where** $\bigwedge x. x \in A \implies x \in \langle \{r\} \rangle$
<proof>

lemma *commutes-primroot*: **assumes** *commutes* A
obtains r **where** $\bigwedge x. x \in A \implies x \in \langle \{r\} \rangle$ **and** *primitive* r
<proof>

lemma *commutesI* [*intro*]: $(\bigwedge x\ y. x \in A \implies y \in A \implies x \cdot y = y \cdot x) \implies \text{commutes } A$
<proof>

lemma *commutesI'*: **assumes** $x \neq \varepsilon$ **and** $\bigwedge y. y \in A \implies x \cdot y = y \cdot x$
shows *commutes* A
<proof>

lemma *commutesI-root*[*intro*]: $(\bigwedge x. x \in A \implies x \in \langle\{t\}\rangle) \implies \text{commutes } A$
 ⟨*proof*⟩

lemma *commutesI-ex-root*: $\exists t. \forall x \in A. x \in \langle\{t\}\rangle \implies \text{commutes } A$
 ⟨*proof*⟩

lemma *commutes-sub*: $\text{commutes } A \implies B \subseteq A \implies \text{commutes } B$
 ⟨*proof*⟩

lemma *commutes-insert*: $\text{commutes } A \implies x \in A \implies x \neq \varepsilon \implies x \cdot y = y \cdot x \implies$
 $\text{commutes } (\text{insert } y \ A)$
 ⟨*proof*⟩

lemma *commutes-emp* [*simp*]: $\text{commutes } \{\varepsilon, w\}$
 ⟨*proof*⟩

lemma *commutes-emp'* [*simp*]: $\text{commutes } \{w, \varepsilon\}$
 ⟨*proof*⟩

lemma *commutes-cancel*: **assumes** $y \in A$ **and** $x \cdot y \in A$ **and** $\text{commutes } A$
shows $\text{commutes } (\text{insert } x \ A)$
 ⟨*proof*⟩

lemma *commutes-cancel'*: **assumes** $x \in A$ **and** $x \cdot y \in A$ **and** $\text{commutes } A$
shows $\text{commutes } (\text{insert } y \ A)$
 ⟨*proof*⟩

8.1.6 Strong elementary lemmas

Discovered by smt

lemma *xyx-per-comm*: **assumes** $x \cdot y \cdot x \leq_p q \cdot x \cdot y \cdot x$
and $q \neq \varepsilon$ **and** $q \leq_p y \cdot q$
shows $x \cdot y = y \cdot x$
 ⟨*proof*⟩

lemma *two-elim-root-suf-comm*: **assumes** $u \leq_p v \cdot u$ **and** $v \leq_s p \cdot u$ **and** $p \in$
 $\langle\{u, v\}\rangle$
shows $u \cdot v = v \cdot u$
 ⟨*proof*⟩

8.1.7 Binary words without a letter square

lemma *no-repetition-list*:
assumes $\text{set } ws \subseteq \{a, b\}$
and *not-per*: $\neg ws \leq_p [a, b] \cdot ws \neg ws \leq_p [b, a] \cdot ws$
and *not-square*: $\neg [a, a] \leq_f ws$ **and** $\neg [b, b] \leq_f ws$
shows *False*
 ⟨*proof*⟩

lemma *no-repetition-list-bin*:

fixes $ws :: \text{binA list}$

assumes *not-square*: $\bigwedge c. \neg [c,c] \leq f ws$

shows $ws \leq p [hd ws, 1-(hd ws)] \cdot ws$

<proof>

lemma *per-root-hd-last-root*: **assumes** $ws \leq p [a,b] \cdot ws$ **and** $hd ws \neq last ws$

shows $ws \in \langle \{[a,b]\} \rangle$

<proof>

lemma *no-cyclic-repetition-list*:

assumes *set* $ws \subseteq \{a,b\}$ $ws \notin \langle \{[a,b]\} \rangle$ $ws \notin \langle \{[b,a]\} \rangle$ $hd ws \neq last ws$

$\neg [a,a] \leq f ws \neg [b,b] \leq f ws$

shows *False*

<proof>

8.1.8 Three covers

lemma *three-covers-example*:

assumes

$v = a$ **and**

$t = (b \cdot a^{(j+1)})^{(m+l+1)} \cdot b \cdot a$ **and**

$r = a \cdot b \cdot (a^{(j+1)} \cdot b)^{(m+l+1)}$ **and**

$t' = (b \cdot a^{(j+1)})^{(m+l+1)} \cdot b \cdot a$ **and**

$r' = a \cdot b \cdot (a^{(j+1)} \cdot b)^{(m+l+1)}$ **and**

$w = a \cdot (b \cdot a^{(j+1)})^{(m+l+1)} \cdot b \cdot a$

shows $w = v \cdot t$ **and** $w = r \cdot v$ **and** $w = r' \cdot v^{(j+1)} \cdot t'$ **and** $t' <_p t$ **and** r'

$<_s r$

<proof>

lemma *three-covers-pers*: — alias Old Good Lemma

assumes $w = v \cdot t$ **and** $w = r' \cdot v^{(j)} \cdot t'$ **and** $w = r \cdot v$ **and** $0 < j$ **and**

$r' <_s r$ **and** $t' <_p t$

shows *period* w $(|t| - |t'|)$ **and** *period* w $(|r| - |r'|)$ **and**

$(|t| - |t'|) + (|r| - |r'|) = |w| + j \cdot |v| - 2 \cdot |v|$

<proof>

lemma *three-covers-per0*: **assumes** $w = v \cdot t$ **and** $w = r' \cdot v^{(j)} \cdot t'$ **and** $w = r \cdot v$ **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* $w \leq_p r \cdot w$ $r \neq \varepsilon$
obtains p q i **where** $w = (p \cdot q)^{\textcircled{i}} \cdot p$ $p \cdot q = r$
<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** $t = (q \cdot p)^{\textcircled{m+k}}$ **and** $r = (p \cdot q)^{\textcircled{m+k}}$ **and**
 $t' = (q \cdot p)^{\textcircled{k}}$ **and** $r' = (p \cdot q)^{\textcircled{m}}$ **and** $v = (p \cdot q)^{\textcircled{i}} \cdot p$ **and**
 $w = (p \cdot q)^{\textcircled{m+i+k}} \cdot p$ **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** $w \leq_f$
 $y^{\textcircled{k}}$ **and** $|y| \leq |x|$
shows $x \cdot y = y \cdot x$
<proof>

8.1.9 Binary Equality Words

definition *binary-equality-generator* :: *binA list* \Rightarrow *bool* **where**
binary-equality-generator $w \equiv (\text{set } w = \text{UNIV}) \wedge (\exists (g :: \text{binA list} \Rightarrow \text{nat list})$
 $h. \text{binary-code-morphism } g \wedge \text{binary-code-morphism } h \wedge g \neq h \wedge w \in g =_M h)$

definition *canonical-binary-equality-generator* :: *binA list* \Rightarrow *bool* **where**
canonical-binary-equality-generator $w = (\exists (g :: \text{binA list} \Rightarrow \text{nat list})$
 $h. \text{binary-code-morphism } g \wedge \text{binary-code-morphism } h \wedge w \in g =_M h \wedge |h \mathbf{a}| < |g \mathbf{a}| \wedge$
 $|g \mathbf{b}| < |h \mathbf{b}| \wedge |g \mathbf{a}| \leq |h \mathbf{b}|)$

lemma *begE*: **assumes** *binary-equality-generator* w
obtains g h **where** *binary-code-morphism* $(g :: \text{binA list} \Rightarrow \text{nat list})$ **and** *bi-*
nary-code-morphism h **and** $g \neq h$ **and** $w \in g =_M h$ **and** $\text{set } w = \text{UNIV}$
<proof>

lemma *cbegE*: **assumes** *canonical-binary-equality-generator* w
obtains g h **where** *binary-code-morphism* $(g :: \text{binA list} \Rightarrow \text{nat list})$ **and** *bi-*
nary-code-morphism h **and** $w \in g =_M h$ **and** $|h \mathbf{a}| < |g \mathbf{a}|$ **and** $|g \mathbf{b}| < |h \mathbf{b}|$ **and**
 $|g \mathbf{a}| \leq |h \mathbf{b}|$

<proof>

lemma *cbeg-is-beg*: **assumes** *canonical-binary-equality-generator w* **shows** *binary-equality-generator w*
<proof>

lemma *beg-productE*: **assumes** *binary-equality-generator (u.v)* **and** $u \neq \varepsilon$ **and** $v \neq \varepsilon$

obtains $g \ h$ **where** *binary-code-morphism (g :: binA list \Rightarrow nat list)* **and** *binary-code-morphism h* **and** $g \neq h$ **and** $(u.v) \in g =_M h$ **and** $|g \ u| < |h \ u|$ **and** $|h \ v| < |g \ v|$
<proof>

lemma *bew-baiba-eq'*: **assumes** $|y| < |v|$ **and** $x \leq_s y$ **and** $u \leq_s v$ **and**

$$y \cdot x^{\textcircled{k}} \cdot y = v \cdot u^{\textcircled{k}} \cdot v$$

shows *commutes {x,y,u,v}*

<proof>

lemma *bew-baiba-eq*: **assumes** $y \cdot x \neq v \cdot u$ **and**

$$y \cdot x^{\textcircled{k+1}} \cdot y \cdot x = v \cdot u^{\textcircled{k+1}} \cdot v \cdot u$$

shows *commutes {x,y,u,v}*

<proof>

lemmas *less-mult-le [intro] = mult-le-mono1[OF Suc-leI, unfolded mult-Suc]*

lemma *less-mult-le' [intro]*: $m < (k::nat) \implies m*r + r \leq k*r$

<proof>

lemma *bew-baibaib-eq-aux*: **assumes** $|x| < |u|$ **and** $1 < i$ **and**

$$eq: x \cdot y^{\textcircled{i}} \cdot x \cdot y^{\textcircled{i}} \cdot x = u \cdot v^{\textcircled{i}} \cdot u \cdot v^{\textcircled{i}} \cdot u$$

shows *commutes {x,y,u,v}*

<proof>

lemma *bew-baibaib-eq*: **assumes** $1 < i$ **and** $x \neq u$ **and**

$$eq: x \cdot y^{\textcircled{i}} \cdot x \cdot y^{\textcircled{i}} \cdot x = u \cdot v^{\textcircled{i}} \cdot u \cdot v^{\textcircled{i}} \cdot u$$

shows *commutes {x,y,u,v}*

<proof>

theorem *not-beg-abiab*: \neg *binary-equality-generator (a . b[ⓐ](i+1) . a . b)*

<proof>

theorem *not-beg-baibaib*: **assumes** $1 < i$

shows \neg *binary-equality-generator (b . a[ⓐ]i . b . a[ⓐ]i . 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.