Formally Verified Suffix Array Construction

Louis Cheung and Christine Rizkallah

March 17, 2025

Abstract

A suffix array [2] is a data structure that is extensively used in text retrieval and data compression applications, including query suggestion mechanisms in web search, and in bioinformatics tools for DNA sequencing and matching. This wide applicability means that algorithms for constructing suffix arrays are of great practical importance. The Suffix Array by Induced Sorting (SA-IS) algorithm [3] is a conceptually complex yet highly efficient suffix array construction technique, based on an earlier algorithm [1].

As part of this formalization, we have developed the SA-IS algorithm in Isabelle/HOL and formally verified that it is equivalent to a mathematical functional specification of suffix arrays. This required verifying a wide range of underlying properties of lists and suffixes, that could be reused in other contexts. We also used Isabelle's code extraction facilities to extract an executable Haskell implementation of SAIS. In particular, this entry includes the following: an axiomatic characterisation of suffix array construction; a formally verified encoding of a straightforward but inefficient suffix array construction algorithm (validating the specification); and a formally verified encoding of the linear time SA-IS algorithm.

Contents

1 HOL **9**

2 Natural Number Arithmetic **9**

3 Monotonic Functions **10**

4 Sets **11**

4.1 From AutoCorres 14

5 General Lists **14**

6 Find **16**

7 Filter **17**

8 Upt **22**

9 Lemmas about bijections **22**

10 Lemmas about monotone functions **26**

11 Sorting **28**
11.1 General sorting 28
11.2 Sorting on linear orders 29
11.3 Sorting on orders 31

12 Mapping elements to natural numbers **31**

13 Repeat Function At Most N Times **33**
13.1 Step and early termination lemmas 33

14 Repeat Function N Times **37**

15 Continuous Intervals **38**

16 List Slices **41**

17 Sorted List Slice **47**

18 General Non-standard Lexicographical Comparison **51**
18.1 Intro and Elimination 51
18.2 Simplification 52
18.3 Recursive version 53
18.4 Properties 54
18.5 Monotonicity 59
18.6 Other 61

19 Order definitions on lists of linorder elements **63**

20 Helper list comparison theorems **64**

21 *list-less-ns* helpers **66**

22 Lists of linorder elements are linorders with a bottom element **67**

23 Recursive Definition **68**

24 *list-less-ns-ex* helpers **69**

25 Valid List 70

26 Order Equivalence 73

27 Classical Lexicographical Order 74

28 Non-standard Lexicographical Ordering 76

29 Suffix 77

30 Valid Lists and Suffixes 79

31 Prefixes and Suffixes 79

32 Suffix Comparisons 81
- 32.1 Lexicographical Ordering 81
- 32.2 Non-standard List Ordering 82

33 List Slice 82

34 Sorting 83

35 Prefix Definition 87

36 Axiomatic Suffix Array Specification 88

37 Wrapper for Natural Number String only Algorithm 88

38 General Suffix Array Properties 89

39 Properties of Suffix Arrays on Valid Lists 90

40 Equivalence 94

41 Small and Large List Types 99

42 Suffix Type 104
- 42.1 General Suffix Type Simplifications 105
- 42.2 S-Type Simplifications 106
- 42.3 L-Type Simplifications 107
- 42.4 General Suffix Type Theories 110
- 42.5 S/L-Type Ordering 111
- 42.6 Implementation of Suffix Type Computation 113

43 SAIS Sublist Order 115

44 Sorting 115

45 LMS-Types 118
45.1 LMS-Type Simplifications . . . 118
45.2 LMS-Type Sets and Subsets . . . 120
45.3 Implementation of LMS-Types Computation . . . 120
45.3.1 Properties . . . 121
45.4 Cardinality LMS-Types . . . 121
45.5 General Properties about LMS-types . . . 123

46 Buckets 129
46.1 Entire Bucket . . . 129
46.2 L-types . . . 133
46.3 LMS-types . . . 135
46.4 S-types . . . 135

47 Continuous Buckets 143

48 Bucket Initialisation 144

49 Bucket Range 145

50 Helpers 145

51 LMS Slice 146
51.1 Find the next LMS position . . . 146
51.2 LMS Prefix . . . 150
51.3 LMS Slice . . . 151
51.4 LMS Substring butlast . . . 153
51.5 Suffix Types . . . 154

52 Ordering LMS-substrings 165

53 Mapping from suffix to lists of LMS-Substrings 174
53.1 LMS Sequence . . . 175
53.2 LMS-Substring Sequence . . . 182
53.3 LMS Map . . . 190

54 Induce Sorting 198
54.1 Bucket Insert . . . 198
54.2 Induce L-types . . . 198
54.3 Induce S-types . . . 199
54.4 Induce Sorting . . . 200

55 Rename Mapping 201

56 Rename String 201

57 Order LMS **201**

58 Extract LMS **201**

59 SAIS Definition **202**

60 Bucket Insert with Ghost State **203**

61 Simple Properties **203**

62 Invariants **203**
62.1 Defintions and Simple Helper Lemmas 203
62.1.1 Distinctness 203
62.1.2 LMS Bucket Ptr 204
62.1.3 Unknowns 205
62.1.4 Locations 205
62.1.5 Unchanged 205
62.1.6 Inserted 206
62.1.7 Sorted 206
62.2 Combined Invariant 207
62.3 Helpers 208
62.4 Establishment and Maintenance Steps 219
62.4.1 Distinctness 219
62.4.2 Bucket Ptr 223
62.4.3 Unknowns 226
62.4.4 Locations 229
62.4.5 Unchanged 232
62.4.6 Inserted 234
62.4.7 Sorted 235
62.5 Combined Establishment and Maintenance 242

63 Exhaustiveness **243**

64 Postconditions **245**

65 Abstract Induce L-types Simple Properties **253**

66 Precondition Definitions **255**

67 Invariant Definitions **261**
67.1 Distinctness 261
67.2 Predecessor 261
67.3 L Bucket Ptr 262
67.4 Unknowns 262
67.5 Indexes 263

67.6 Unchanged . . . 263
67.7 L Locations . . . 263
67.8 Seen . . . 264
67.9 Sortedness . . . 265
67.10 Permutation . . . 266

68 Invariant Helpers **267**
68.1 Distinctness of New Insert . . . 267
68.2 Bucket Ranges . . . 269
68.3 No Overwrite . . . 271
68.4 Bucket Values . . . 275
68.5 Seen . . . 280

69 Distinctness **281**
69.1 Establishment . . . 281
69.2 Maintenance . . . 283

70 Unknowns **286**
70.1 Establishment . . . 286
70.2 Maintenance . . . 286

71 Number of L-types **289**
71.1 Establishment . . . 289
71.2 Maintenance . . . 291

72 L Locations **295**
72.1 Establishment . . . 295
72.2 Maintenance . . . 295

73 Unchanged **298**
73.1 Establishment . . . 298
73.2 Maintenance . . . 299

74 Invariant about the Current Index **301**
74.1 Establishment . . . 301
74.2 Maintenance . . . 302

75 Predecessor Invariant **305**
75.1 Establishment . . . 305
75.2 Maintenance . . . 305

76 Seen Invariant **308**
76.1 Establishment . . . 308
76.2 Maintenance . . . 308

77 Permutation **310**
77.1 Establishment . . . 310
77.2 Maintenance . . . 311

78 Sorted **313**

79 L-type Exhaustiveness **333**
79.1 Case 1 . . . 334
79.2 Case 2 . . . 334
79.3 Exhaustiveness Proof . . . 336

80 Correctness and Exhaustiveness **338**

81 Abstract Induce S Simple Properties **342**

82 Preconditions **344**

83 Invariants **346**
83.1 Definitions . . . 346
83.1.1 Distinctness . . . 346
83.1.2 S Bucket Ptr . . . 346
83.1.3 Locations . . . 347
83.1.4 Unchanged . . . 348
83.1.5 Seen . . . 348
83.1.6 Predecessor . . . 348
83.1.7 Successor . . . 349
83.1.8 Combined Permutation Invariant . . . 349
83.1.9 Sorted . . . 350
83.2 Helpers . . . 351
83.3 Establishment and Maintenance Steps . . . 363
83.3.1 Distinctness . . . 363
83.3.2 Bucket Pointer . . . 366
83.3.3 Locations . . . 369
83.3.4 Unchanged . . . 373
83.3.5 Seen . . . 375
83.3.6 Predecessor . . . 406
83.3.7 Successor . . . 410
83.3.8 Combined Permutation Invariant . . . 418
83.3.9 Sorted . . . 426

84 Induce S Correctness Theorems **458**

85 Bucket Initialisation Properties **459**

86 Bucket Insert Precondition **460**

87 Induce L Precondition 460

88 Induce S Precondition 461

89 Permutation 463

90 Sorting 463

91 Extract LMS types Proofs 467

92 Order LMS-types Proofs 468

93 Rename Mapping Proofs 468

94 Rename String Proofs 472

95 SAIS General Helpers 472

96 SAIS cases simplifications 473

97 SAIS returns a permutation 474

98 SAIS Sorted Helpers 477

99 SAIS sorts suffixes 477

100 Verification of a SAIS construction algorithm 480

101 Final Theorem: Verification of a generalised SAIS construction algorithm 481

102 Bucket Insert 481

103 Suffix Types 482

104 LMS types 482

105 Extracting LMS types 483

106 LMS Substrings 483

107 Rename Mapping 483

108 Induce L Refinement 484

109 Induce S Refinement 485

110 Induce 488

111 SAIS 489

112 Bucket Insert 491

113 Induce L Refinement 493

114 Induce S Refinement 495

115 Induce 501

116 Suffix Types 503

117 LMS types 506

118 Extracting LMS types 507

119 LMS Substrings 507

120 Rename Mapping 507

121 SAIS 508

122 Correctness 513

theory *Nat-Util*
  imports *Main*
begin

1 HOL

lemma *duplicate-assms*:
  $([\![P; P]\!] \Longrightarrow Q) \equiv (P \Longrightarrow Q)$
  by *simp*

2 Natural Number Arithmetic

lemma *div-2-eq-Suc*:
  $[\![x\ div\ 2 = y\ div\ 2;\ x \neq y]\!] \Longrightarrow (y = Suc\ x) \lor (x = Suc\ y)$
  by *linarith*

lemma *Suc-m-sub-n-div-2*:
  $Suc\ ((m - n)\ div\ 2) > (m - Suc\ n)\ div\ 2$
  by (*simp add*: *div-le-mono le-Suc-eq*)

lemma *Suc-div-2-less-Suc*:
  $Suc\ x\ div\ 2 < Suc\ x$
  by *simp*

lemma *nat-x-less-y-le-Suc-x*:

$[\![x < y;\ y \leq Suc\ x]\!] \Longrightarrow y = Suc\ x$
by *simp*

lemma *nat-sub-eq-add*:
$[\![(a :: nat) - b = c - d;\ b < a]\!] \Longrightarrow a + d = c + b$
by *simp*

end
theory *Fun-Util*
imports *Main*
begin

3 Monotonic Functions

lemma *strict-mono-leD*: *strict-mono* $r \Longrightarrow m \leq n \Longrightarrow r\ m \leq r\ n$
by (*erule* (*1*) *monoD* [*OF strict-mono-mono*])

definition *map-to-nat* :: ($'a$ :: *linorder list*) $\Rightarrow$ ($'a \Rightarrow nat$)
where
map-to-nat xs = ($\lambda x.$ *card* $\{y|y.\ y \in set\ xs \wedge y < x\}$)

lemma *map-to-nat-strict-mono-on*:
strict-mono-on (*set xs*) (*map-to-nat xs*)
unfolding *strict-mono-on-def map-to-nat-def*
proof *safe*
fix $x\ y :: 'a$
assume $x < y$ $x \in set\ xs$ $y \in set\ xs$
have *finite* $\{a\ |a.\ a \in set\ xs \wedge a < y\}$
by *auto*
moreover
have $\{a\ |a.\ a \in set\ xs \wedge a < x\} \subset \{a\ |a.\ a \in set\ xs \wedge a < y\}$
proof (*intro psubsetI subsetI notI*)
fix k
assume $k \in \{a\ |a.\ a \in set\ xs \wedge a < x\}$
hence $k < x$ $k \in set\ xs$
by *simp-all*
hence $k < y$ $k \in set\ xs$
using ‹$x < y$› **by** *auto*
then show $k \in \{a\ |a.\ a \in set\ xs \wedge a < y\}$
by *simp*
next
assume $\{a\ |a.\ a \in set\ xs \wedge a < x\} = \{a\ |a.\ a \in set\ xs \wedge a < y\}$
moreover
have $x \in \{a\ |a.\ a \in set\ xs \wedge a < y\}$
using ‹$x < y$› ‹$x \in set\ xs$› **by** *auto*
moreover
have $x \notin \{a\ |a.\ a \in set\ xs \wedge a < x\}$
by *auto*
ultimately show *False*

by *simp*
qed
ultimately show $card\ \{a\ |a.\ a \in set\ xs \wedge a < x\} < card\ \{a\ |a.\ a \in set\ xs \wedge a < y\}$
using $psubset\text{-}card\text{-}mono[of\ \{a\ |a.\ a \in set\ xs \wedge a < y\}\ \{a\ |a.\ a \in set\ xs \wedge a < x\}]$
by *blast*
qed

lemma *strict-mono-on-map-set-ex*:
$\exists (f :: ('a :: linorder \Rightarrow nat)).\ strict\text{-}mono\text{-}on\ (set\ xs)\ f$
using *map-to-nat-strict-mono-on* **by** *blast*

locale *Linorder-to-Nat-List* =
fixes $map\text{-}to\text{-}nat :: 'a :: linorder\ list \Rightarrow 'a \Rightarrow nat$
and $xs :: 'a :: linorder\ list$
assumes *map-to-nat-strict-mono-on*: $strict\text{-}mono\text{-}on\ (set\ xs)\ (map\text{-}to\text{-}nat\ xs)$

context *Linorder-to-Nat-List* **begin**

lemma *strict-mono-on-Suc-map-to-nat*:
$strict\text{-}mono\text{-}on\ (set\ xs)\ (\lambda x.\ Suc\ (map\text{-}to\text{-}nat\ xs\ x))$
by (*metis* (*mono-tags*, *lifting*) *Suc-mono ord.strict-mono-on-def map-to-nat-strict-mono-on*)

end

lemma *Linorder-to-Nat-List-ex*:
$\exists \alpha.\ Linorder\text{-}to\text{-}Nat\text{-}List\ \alpha\ xs$
by (*meson Linorder-to-Nat-List.intro strict-mono-on-map-set-ex*)

end
theory *Set-Util*
imports *Main*
begin

4 Sets

lemma *pigeonhole-principle-advanced*:
assumes *finite A*
and *finite B*
and $A \cap B = \{\}$
and $card\ A > card\ B$
and $bij\text{-}betw\ f\ (A \cup B)\ (A \cup B)$
shows $\exists a \in A.\ f\ a \in A$
proof (*rule ccontr*)
assume $\neg(\exists a \in A.\ f\ a \in A)$
hence $\forall a \in A.\ f\ a \notin A$
by *blast*

```
  hence ∀ a∈A. f a ∈ B
    using assms(5) bij-betw-apply by fastforce
  hence f ‘ A ⊆ B
    by blast

  have inj-on f A
    by (meson assms(5) bij-betw-def inj-on-Un)
  hence card (f ‘ A) = card A
    using card-image by blast
  hence f ‘ A = B
    by (metis ‹f ‘ A ⊆ B› assms(2,4) card-mono leD)
  hence card B = card A
    using ‹card (f ‘ A) = card A› by blast
  then show False
    using assms(4) by linarith
qed

lemma Suc-mod-n-bij-betw:
  bij-betw (λx. Suc x mod n) {0..<n} {0..<n}
proof (intro bij-betwI')
  fix x y
  assume x ∈ {0..<n} y ∈ {0..<n}
  then show (Suc x mod n = Suc y mod n) = (x = y)
    by (simp add: mod-Suc)
next
  fix x
  assume x ∈ {0..<n}
  then show Suc x mod n ∈ {0..<n}
    by clarsimp
next
  fix y
  assume y ∈ {0..<n}
  then show ∃ x∈{0..<n}. y = Suc x mod n
   by (metis atLeastLessThan-iff bot-nat-0.extremum less-nat-zero-code mod-Suc-eq
mod-less
              mod-less-divisor mod-self not-gr-zero old.nat.exhaust)
qed



lemma subset-upt-no-Suc:
  assumes A ⊆ {1..<n}
  and     ∀ x∈A. Suc x ∉ A
  shows card A ≤ n div 2
proof (rule ccontr)
  assume ¬ card A ≤ n div 2
  hence n div 2 < card A
    by auto
```

```
have ∃ a∈A. Suc a mod n ∈ A
proof (rule pigeonhole-principle-advanced[of A {0..<n} − A (λx. Suc x mod n),
simplified])
  show finite A
    using assms(1) finite-subset by blast
next
  from card-Diff-subset
  have card ({0..<n} − A) = card {0..<n} − card A
    by (metis Diff-subset assms(1) dual-order.trans ivl-diff less-eq-nat.simps(1)
          subset-eq-atLeast0-lessThan-finite)
  moreover
  have card {0..<n} − card A < card A
    using ‹¬ card A ≤ n div 2› assms by simp
  ultimately show card ({0..<n} − A) < card A
    by simp
next
  have A ∪ {0..<n} = {0..<n}
    using assms(1) dual-order.trans by auto
  with Suc-mod-n-bij-betw[of n]
  show bij-betw (λx. Suc x mod n) (A ∪ {0..<n}) (A ∪ {0..<n})
    by simp
qed
then obtain x where
  x ∈ A
  Suc x mod n ∈ A
  by blast

show False
proof (cases n)
  case 0
  then show ?thesis
    using ‹Suc x mod n ∈ A› ‹x ∈ A› assms(2) by force
next
  case (Suc m)
  assume n = Suc m

  have x = m ∨ x < m
    using Suc ‹x ∈ A› assms(1) by auto
  then show ?thesis
  proof
    assume x = m
    then show False
      using Suc ‹Suc x mod n ∈ A› assms(1) by auto
  next
    assume x < m
    with mod-less[of Suc x Suc m]
    show False
      using Suc ‹Suc x mod n ∈ A› ‹x ∈ A› assms(2) by force
  qed
```

  qed
qed

lemma *in-set-mapD*:
  $x \in set\ (map\ f\ xs) \Longrightarrow \exists\, y \in set\ xs.\ x = f\ y$
  by (*simp add*: *image-iff*)

4.1 From AutoCorres

lemma *disjointI'*:
  assumes $\bigwedge x\ y.\ [\![\ x \in A;\ y \in B\ ]\!] \Longrightarrow x \neq y$
  shows $A \cap B = \{\}$
  using *assms* **by** *fast*

lemma *disjoint-subset2*:
  assumes $B' \subseteq B$ **and** $A \cap B = \{\}$
  shows $A \cap B' = \{\}$
  using *assms* **by** *fast*

end
theory *List-Util*
  imports *Main*
begin

5 General Lists

lemma *list-cases-3*:
  $T = [] \lor (\exists\, x.\ T = [x]) \lor (\exists\, a\ b\ xs.\ T = a\ \#\ b\ \#\ xs)$
proof (*cases T*)
  case *Nil*
  then show *?thesis* **by** *simp*
next
  case (*Cons a list*)
  then show *?thesis*
  proof (*cases list*)
    case *Nil*
    with ‹$T = a\ \#\ list$›
    show *?thesis*
      by *simp*
  next
    case (*Cons a' list'*)
    with ‹$T = a\ \#\ list$›
    show *?thesis*
      by *simp*
  qed
qed

lemma *length-cons-cons*:
$T = a \# b \# xs \Longrightarrow \exists n.\ length\ T = Suc\ (Suc\ n)$
by *simp*

lemma *length-Suc-Suc*:
$length\ T = Suc\ (Suc\ n) \Longrightarrow \exists a\ b\ xs.\ T = a \# b \# xs$
by (*metis length-Suc-conv*)

lemma *length-Suc-0*:
$length\ xs = Suc\ 0 \Longrightarrow \exists x.\ xs = [x]$
by (*simp add*: *length-Suc-conv*)

lemma *map-eq-replicate*:
$\forall x \in set\ xs.\ f\ x = k \Longrightarrow map\ f\ xs = replicate\ (length\ xs)\ k$
by (*metis map-eq-conv map-replicate-const*)

lemma *map-upt-eq-replicate*:
$\forall x \in set\ [i..<j].\ f\ x = k \Longrightarrow map\ f\ [i..<j] = replicate\ (j - i)\ k$
by (*metis length-upt map-eq-replicate*)

lemma *in-set-list-update*:
$[\![x \in set\ xs;\ xs\ !\ k \neq x]\!] \Longrightarrow x \in set\ (xs[k := y])$
by (*metis in-set-conv-nth length-list-update nth-list-update-neq*)

lemma *Max-greD*:
$i < length\ s \Longrightarrow Max\ (set\ s) \geq s\ !\ i$
by *clarsimp*

lemma *list-neq-rc1*:
$(\exists z\ zs.\ xs = ys\ @\ z \# zs) \Longrightarrow xs \neq ys$
by *fastforce*

lemma *list-neq-rc2*:
$(\exists z\ zs.\ ys = xs\ @\ z \# zs) \Longrightarrow xs \neq ys$
by *fastforce*

lemma *list-neq-rc3*:
$(\exists x\ y\ as\ bs\ cs.\ xs = as\ @\ x \# bs \wedge ys = as\ @\ y \# cs \wedge x \neq y) \Longrightarrow xs \neq ys$
by *fastforce*

lemma *list-neq-rc*:
$(\exists z\ zs.\ xs = ys\ @\ z \# zs) \vee$
$(\exists z\ zs.\ ys = xs\ @\ z \# zs) \vee$
$(\exists x\ y\ as\ bs\ cs.\ xs = as\ @\ x \# bs \wedge ys = as\ @\ y \# cs \wedge x \neq y) \Longrightarrow$
$xs \neq ys$
by (*elim disjE conjE list-neq-rc1 list-neq-rc2 list-neq-rc3*)

lemma *list-neq-fc*:

$xs \neq ys \Longrightarrow$
$(\exists z\ zs.\ xs = ys\ @\ z\ \#\ zs) \vee$
$(\exists z\ zs.\ ys = xs\ @\ z\ \#\ zs) \vee$
$(\exists x\ y\ as\ bs\ cs.\ xs = as\ @\ x\ \#\ bs \wedge ys = as\ @\ y\ \#\ cs \wedge x \neq y)$
proof (*induct xs arbitrary*: *ys*)
case *Nil*
then show *?case*
by (*metis append-Nil list.exhaust*)
next
case (*Cons a xs ys*)
note *IH* = *this*
then show *?case*
proof (*cases ys*)
case *Nil*
then show *?thesis*
by *simp*
next
case (*Cons b ys'*)
assume $ys = b\ \#\ ys'$
show *?thesis*
proof (*cases* $a = b$)
assume $a \neq b$
with ‹$ys = b\ \#\ ys'$›
show *?thesis*
by *blast*
next
assume $a = b$
with *IH*(*2*) ‹$ys = b\ \#\ ys'$›
have $xs \neq ys'$
by *simp*
with *IH*(*1*)[*of ys'*]
show *?thesis*
by (*metis Cons-eq-appendI* ‹$a = b$› *local.Cons*)
qed
qed
qed

lemma *list-neq-cases*:
$xs \neq ys \longleftrightarrow$
$(\exists z\ zs.\ xs = ys\ @\ z\ \#\ zs) \vee$
$(\exists z\ zs.\ ys = xs\ @\ z\ \#\ zs) \vee$
$(\exists x\ y\ as\ bs\ cs.\ xs = as\ @\ x\ \#\ bs \wedge ys = as\ @\ y\ \#\ cs \wedge x \neq y)$
using *list-neq-fc list-neq-rc* **by** *blast*

6 Find

lemma *findSomeD*:
find P xs = *Some x* $\Longrightarrow$ *P x* $\wedge$ $x \in$ *set xs*
by (*induct xs*) (*auto split*: *if-split-asm*)

lemma *findNoneD*:
  find P xs = *None* $\Longrightarrow \forall x \in$ *set xs*. $\neg P\ x$
  by (*induct xs*) (*auto split*: *if-split-asm*)

7 Filter

lemma *filter-update-nth-success*:
  $[\![P\ v;\ i < length\ xs]\!] \Longrightarrow$
    filter P (*xs*[*i* := *v*]) = (*filter P* (*take i xs*)) @ [*v*] @ (*filter P* (*drop* (*Suc i*) *xs*))
  by (*simp add*: *upd-conv-take-nth-drop*)

lemma *filter-update-nth-fail*:
  $[\![\neg P\ v;\ i < length\ xs]\!] \Longrightarrow$
    filter P (*xs*[*i* := *v*]) = (*filter P* (*take i xs*)) @ (*filter P* (*drop* (*Suc i*) *xs*))
  by (*simp add*: *upd-conv-take-nth-drop*)

lemma *filter-take-nth-drop-success*:
  $[\![i < length\ xs;\ P\ (xs\ !\ i)]\!] \Longrightarrow$
    filter P xs = (*filter P* (*take i xs*)) @ [*xs* ! *i*] @ (*filter P* (*drop* (*Suc i*) *xs*))
  by (*metis filter-update-nth-success list-update-id*)

lemma *filter-take-nth-drop-fail*:
  $[\![i < length\ xs;\ \neg P\ (xs\ !\ i)]\!] \Longrightarrow$
    filter P xs = (*filter P* (*take i xs*)) @ (*filter P* (*drop* (*Suc i*) *xs*))
  by (*metis filter-update-nth-fail list-update-id*)

lemma *filter-nth-1*:
  $[\![i < length\ xs;\ P\ (xs\ !\ i)]\!] \Longrightarrow$
  $\exists i'.\ i' < length\ (filter\ P\ xs) \wedge (filter\ P\ xs)\ !\ i' = xs\ !\ i$
proof (*induct xs arbitrary*: *i*)
  case *Nil*
  then show *?case*
    by *simp*
next
  case (*Cons a xs i*)
  note *IH* = *this*
  show *?case*
  proof (*cases i*)
    case *0*
    with *IH*(*3*)
    show *?thesis*
      by *fastforce*
  next
    case (*Suc n*)
    with *IH*(*1*)[*of n*] *IH*(*2*,*3*)
    have $\exists i'<length\ (filter\ P\ xs).\ filter\ P\ xs\ !\ i' = xs\ !\ n$
      by *fastforce*
    then show *?thesis*

```
      using Suc by auto
  qed
qed

lemma filter-nth-2:
  ⟦i < length (filter P xs)⟧ ⟹
   ∃ i'. i' < length xs ∧ (filter P xs) ! i = xs ! i'
proof (induct xs arbitrary: i)
  case Nil
  then show ?case
    by simp
next
  case (Cons a xs i)
  note IH = this
  then show ?case
  proof (cases i)
    case 0
    then show ?thesis
      using Cons.hyps Cons.prems by force
  next
    case (Suc n)
    with IH(1)[of n] IH(2)
    have ∃ i'<length xs. filter P xs ! n = xs ! i'
      by (metis filter.simps(2) length-Cons not-less-eq not-less-iff-gr-or-eq)
    then show ?thesis
      by (metis Cons.hyps Cons.prems Suc filter.simps(2) length-Cons not-less-eq
nth-Cons-Suc)
  qed
qed

lemma filter-nth-relative-1:
  ⟦i < length xs; P (xs ! i); j < i; P (xs ! j)⟧ ⟹
   ∃ i' j'. i' < length (filter P xs) ∧ j' < i' ∧ (filter P xs) ! i' = xs ! i ∧
    (filter P xs) ! j' = xs ! j
proof (induct xs arbitrary: i j)
  case Nil
  then show ?case
    by simp
next
  case (Cons a xs i j)
  note IH = this
  show ?case
  proof (cases i)
    case 0
    then show ?thesis
      using IH(4) by blast
  next
    case (Suc n)
    assume i = Suc n
```

show *?thesis*
proof (*cases j*)
case *0*
with *filter-nth-1*[*of n xs P*] *IH*(*2*,*3*) ‹$i = Suc\ n$› *IH*(*4*−)
show *?thesis*
by (*metis filter.simps*(*2*) *length-Cons not-less-eq nth-Cons-0 nth-Cons-Suc zero-less-Suc*)
next
case (*Suc m*)
with *IH*(*1*)[*of n m*] *IH*(*2*−) ‹$i = Suc\ n$›
show *?thesis*
by *fastforce*
qed
qed
qed

lemma *filter-nth-relative-neq-1*:
assumes $i < length\ xs$ $P\ (xs\ !\ i)$ $j < length\ xs$ $P\ (xs\ !\ j)$ $i \neq j$
shows $\exists\, i'\ j'.\ i' < length\ (filter\ P\ xs) \wedge j' < length\ (filter\ P\ xs) \wedge (filter\ P\ xs)\ !\ i' = xs\ !\ i\ \wedge$
$(filter\ P\ xs)\ !\ j' = xs\ !\ j \wedge i' \neq j'$
proof (*cases* $i < j$)
assume $i < j$
with *filter-nth-relative-1*[**where** $P = P$, *OF assms*(*3*,*4*) - *assms*(*2*)]
show *?thesis*
by (*metis* (*no-types*, *lifting*) *nat-neq-iff order-less-trans*)
next
assume $\neg\ i < j$
with *assms*(*5*)
have $j < i$
by *auto*
with *filter-nth-relative-1*[**where** $P = P$, *OF assms*(*1*,*2*) - *assms*(*4*)]
show *?thesis*
using *order-less-trans* **by** *blast*
qed

lemma *filter-nth-relative-2*:
$[\![i < length\ (filter\ P\ xs);\ j < i]\!] \Longrightarrow$
$\exists\, i'\ j'.\ i' < length\ xs \wedge j' < i' \wedge (filter\ P\ xs)\ !\ i = xs\ !\ i' \wedge (filter\ P\ xs)\ !\ j = xs\ !\ j'$
proof (*induct xs arbitrary*: *i j*)
case *Nil*
then show *?case*
by *simp*
next
case (*Cons a xs i j*)
note $IH = this$

let $?goal = \exists\, i'\ j'.\ i' < length\ (a\ \#\ xs) \wedge j' < i' \wedge filter\ P\ (a\ \#\ xs)\ !\ i = (a\ \#$

```
xs) ! i' ∧
                    filter P (a # xs) ! j = (a # xs) ! j'
  show ?case
  proof (cases i)
    case 0
    then show ?goal
      using IH(3) by blast
  next
    case (Suc n)
    assume i = Suc n
    show ?thesis
    proof (cases j)
      case 0
      assume j = 0
      from filter-nth-2[of n P xs] IH(2)
      have ∃ i'<length xs. filter P xs ! n = xs ! i'
        using Suc order-less-trans by fastforce

      show ?goal
      proof (cases P a)
        assume P a
        then show ?goal
          by (metis 0 Suc ‹∃ i'<length xs. filter P xs ! n = xs ! i'› filter.simps(2)
                length-Cons not-less-eq nth-Cons-0 nth-Cons-Suc zero-less-Suc)
      next
        assume ¬ P a
        then show ?goal
          using Cons.hyps IH(2,3) Suc-less-eq by fastforce
        qed
    next
      case (Suc m)
      with IH(1)[of n m] IH(2−) ‹i = Suc n›
      have ∃ i' j'. i' < length xs ∧ j' < i' ∧ filter P xs ! n = xs ! i' ∧
                filter P xs ! m = xs ! j'
        by (metis filter.simps(2) length-Cons not-less-eq not-less-iff-gr-or-eq)
      then obtain i' j' where
        A: i' < length xs j' < i' filter P xs ! n = xs ! i' filter P xs ! m = xs ! j'
        by blast
      show ?goal
      proof (cases P a)
        assume P a
        then show ?goal
          using A Suc ‹i = Suc n› by force
      next
        assume ¬ P a
        then show ?goal
          using Cons.hyps IH(2,3) by fastforce
      qed
    qed
```

qed
qed

lemma *filter-nth-relative-neq-2*:
  assumes $i < length\ (filter\ P\ xs)$ $j < length\ (filter\ P\ xs)$ $i \neq j$
  shows $\exists\, i'\ j'.\ i' < length\ xs \wedge j' < length\ xs \wedge xs\ !\ i' = (filter\ P\ xs)\ !\ i \wedge$
  $xs\ !\ j' = (filter\ P\ xs)\ !\ j \wedge i' \neq j'$
proof (*cases* $i < j$)
  assume $i < j$
  with *filter-nth-relative-2*[*OF assms*(*2*), *of i*]
  show *?thesis*
    by (*metis dual-order.strict-trans exists-least-iff*)
next
  assume $\neg\ i < j$
  with *assms*(*3*)
  have $j < i$
    by *auto*
  with *filter-nth-relative-2*[*OF assms*(*1*), *of j*]
  show *?thesis*
    by (*metis nat-neq-iff order-less-trans*)
qed

lemma *filter-find*:
  $filter\ P\ xs \neq [] \Longrightarrow find\ P\ xs = Some\ ((filter\ P\ xs)\ !\ 0)$
  by (*induct xs*; *auto*)

lemma *filter-nth-update-subset*:
  $set\ (filter\ P\ (xs[i := v])) \subseteq \{v\} \cup set\ (filter\ P\ xs)$
proof
  fix x
  assume $x \in set\ (filter\ P\ (xs[i := v]))$
  with *filter-set member-filter*
  have $x \in set\ (xs[i := v])$ $P\ x$
    by *auto*
  hence $\exists\, k < length\ (xs[i := v]).\ (xs[i := v])\ !\ k = x$
    by (*simp add*: *in-set-conv-nth*)
  then obtain k **where**
    $k < length\ (xs[i := v])$
    $(xs[i := v])\ !\ k = x$
    by *blast*
  hence $k < length\ xs$
    by *simp*

  from ‹$(xs[i := v])\ !\ k = x$› ‹$k < length\ (xs[i := v])$›
  have $k = i \Longrightarrow x = v$
    by *simp*
  moreover
  have $k \neq i \Longrightarrow x \in set\ (filter\ P\ xs)$
    using ‹$P\ x$› ‹$k < length\ xs$› ‹$xs[i := v]\ !\ k = x$› **by** *auto*

ultimately
show $x \in \{v\} \cup set\ (filter\ P\ xs)$
by *blast*
qed

8 Upt

lemma *card-upt*:
$card\ \{0..<n\} = n$
by *simp*

lemma *bounded-distinct-subset-upt-length*:
$[\![distinct\ xs;\ \forall\, i<length\ xs.\ xs\ !\ i < length\ xs]\!] \Longrightarrow set\ xs \subseteq \{0..<length\ xs\}$
by (*intro subsetI*; *clarsimp simp*: *in-set-conv-nth*)

lemma *bounded-distinct-eq-upt-length*:
assumes *distinct xs*
assumes $\forall\, i < length\ xs.\ xs\ !\ i < length\ xs$
shows $set\ xs = \{0..<length\ xs\}$
proof (*intro card-subset-eq finite-atLeastLessThan*
bounded-distinct-subset-upt-length[*OF assms*])
from *distinct-card*[*OF assms*(*1*)] *card-upt*[*of length xs*]
show $card\ (set\ xs) = card\ \{0..<length\ xs\}$
by *simp*
qed

lemma *set-map-nth-subset*:
assumes $n \leq length\ xs$
shows $set\ (map\ (nth\ xs)\ [0..<n]) \subseteq set\ xs$
using *assms* **by** *clarsimp*

lemma *set-map-nth-eq*:
$set\ (map\ (nth\ xs)\ [0..<length\ xs]) = set\ xs$
by (*intro equalityI set-map-nth-subset subsetI*; *simp add*: *map-nth*)

lemma *distinct-map-nth*:
assumes *distinct xs*
assumes $n \leq length\ xs$
shows $distinct\ (map\ (nth\ xs)\ [0..<n])$
using *assms* **by** (*simp add*: *distinct-conv-nth*)
end
theory *Sorting-Util*
imports *Main*
begin

9 Lemmas about bijections

A convenient definition of an inverses between two sets

definition
inverses-on ::
$('a \Rightarrow 'b) \Rightarrow ('b \Rightarrow 'a) \Rightarrow 'a\ set \Rightarrow 'b\ set \Rightarrow bool$
where
inverses-on f g A B $\longleftrightarrow$
$(\forall x \in A.\ g\ (f\ x) = x) \wedge$
$(\forall x \in B.\ f\ (g\ x) = x)$

lemmas *inverses-onD1* = *inverses-on-def*[*THEN iffD1*, *THEN conjunct1*]
lemmas *inverses-onD2* = *inverses-on-def*[*THEN iffD1*, *THEN conjunct2*]

The inverses relation over maps

lemma *inverses-on-mapD*:
assumes *inverses-on* (*map f*) (*map g*) $\{xs.\ set\ xs \subseteq A\}$ $\{xs.\ set\ xs \subseteq B\}$
shows *inverses-on f g A B*
proof −
from *inverses-onD1*[*OF assms*, *THEN bspec*, *of* [-], *simplified*]
have $\forall x \in A.\ g\ (f\ x) = x$
by *blast*
moreover
from *inverses-onD2*[*OF assms*, *THEN bspec*, *of* [-], *simplified*]
have $\forall x \in B.\ f\ (g\ x) = x$
by *blast*
ultimately show *?thesis*
by (*simp add*: *inverses-on-def*)
qed

lemma *inverses-on-map*:
assumes *inverses-on f g A B*
shows *inverses-on* (*map f*) (*map g*) $\{xs.\ set\ xs \subseteq A\}$ $\{xs.\ set\ xs \subseteq B\}$
proof −
have $\forall x \in \{xs.\ set\ xs \subseteq A\}.\ map\ g\ (map\ f\ x) = x$
using *list-eq-iff-nth-eq inverses-onD1 assms subsetD* **by** *fastforce*
moreover
have $\forall x \in \{xs.\ set\ xs \subseteq B\}.\ map\ f\ (map\ g\ x) = x$
using *list-eq-iff-nth-eq inverses-onD2 assms subsetD* **by** *fastforce*
ultimately show *?thesis*
by (*simp add*: *inverses-on-def*)
qed

Inverses are symmetric

lemma *inverses-on-sym*:
inverses-on f g A B = *inverses-on g f B A*
apply (*simp add*: *inverses-on-def*)
apply (*subst conj.commute*)
apply *simp*
done

Convenient theorem to obtain the inverse of a bijection between two sets

lemma *bij-betw-inv-alt*:

assumes *bij-betw f A B*
shows $\exists$ *g. bij-betw g B A* $\wedge$ *inverses-on f g A B*
by (*meson assms bij-betw-imp-inj-on bij-betw-inv-into inv-into-f-f bij-betw-inv-into-right inverses-on-def*[*THEN iffD2*])

Bijections over maps

lemma *bij-betw-map*:
assumes *bij-betw f A B*
shows *bij-betw* (*map f*) {*xs. set xs* $\subseteq$ *A*} {*xs. set xs* $\subseteq$ *B*}
proof (*rule bij-betwI'*)
fix *x y*
assume *x* $\in$ {*xs. set xs* $\subseteq$ *A*} *y* $\in$ {*xs. set xs* $\subseteq$ *A*}
hence *P*: *set x* $\subseteq$ *A set y* $\subseteq$ *A*
by *blast+*
note *inj* = *inj-on-eq-iff*[*THEN iffD1*, *OF bij-betw-imp-inj-on*[*OF assms*]]
from *list.inj-map-strong*[*OF inj*, *simplified*]
show (*map f x* = *map f y*) = (*x* = *y*)
using *P*(*1*) *P*(*2*) **by** *blast*
next
fix *x*
assume *x* $\in$ {*xs. set xs* $\subseteq$ *A*}
hence *set x* $\subseteq$ *A*
by *blast*
then show *map f x* $\in$ {*xs. set xs* $\subseteq$ *B*}
using *assms bij-betw-imp-surj-on* **by** *fastforce*
next
fix *y*
assume *y* $\in$ {*xs. set xs* $\subseteq$ *B*}
hence *set y* $\subseteq$ *B*
by *blast*

from *bij-betw-inv-alt*[*OF assms*, *simplified inverses-on-def*]
obtain *g* **where**
bij-betw g B A
$\forall$ *x*$\in$*A. g* (*f x*) = *x*
$\forall$ *x*$\in$*B. f* (*g x*) = *x*
by *blast*

have *set* (*map g y*) $\subseteq$ *A*
using ‹*bij-betw g B A*› ‹*set y* $\subseteq$ *B*› *bij-betw-imp-surj-on*
by *fastforce*
moreover
have *y* = *map f* (*map g y*)
proof −
have *length y* = *length* (*map f* (*map g y*))
by *simp*
moreover
have $\forall$ *i* < *length y. y* ! *i* = *map f* (*map g y*) ! *i*
using ‹$\forall$ *x*$\in$*B. f* (*g x*) = *x*› ‹*set y* $\subseteq$ *B*› *subsetD* **by** *fastforce*

ultimately show *?thesis*
using *list-eq-iff-nth-eq* **by** *blast*
qed
ultimately have $\exists x.\ set\ x \subseteq A \wedge y = map\ f\ x$
by *blast*
then show $\exists x{\in}\{xs.\ set\ xs \subseteq A\}.\ y = map\ f\ x$
by *blast*
qed

Eliminating the map from a bijection relation

lemma *bij-betw-mapD*:
assumes $bij\text{-}betw\ (map\ f)\ \{xs.\ set\ xs \subseteq A\}\ \{xs.\ set\ xs \subseteq B\}$
shows $bij\text{-}betw\ f\ A\ B$
proof (*intro bij-betwI′ iffI*)
fix $x\ y$
assume $x \in A\ \ y \in A\ \ f\ x = f\ y$
with *inj-onD*[*OF bij-betw-imp-inj-on*[*OF assms*], *of* [x] [y], *simplified*]
show $x = y$
by *blast*
next
fix x
assume $x \in A$
with *bij-betwE*[*OF assms*, *THEN bspec*, *of* [x], *simplified*]
show $f\ x \in B$
by *blast*
next
fix y
assume $y \in B$
with *bij-betw-imp-surj-on*[*OF assms*, *simplified*]
have $[y] \in map\ f\ ` \ \{xs.\ set\ xs \subseteq A\}$
by *auto*
with *image-iff*[*THEN iffD1*, *of* [y] *map f* $\{xs.\ set\ xs \subseteq A\}$]
obtain x' **where**
$x' \in \{xs.\ set\ xs \subseteq A\}$
$[y] = map\ f\ x'$
by *meson*
then show $\exists x{\in}A.\ y = f\ x$
by *auto*
next
qed(*clarsimp*)

Obtaining the inverse over map

lemma *bij-betw-inv-map*:
assumes $bij\text{-}betw\ f\ A\ B$
shows $\exists g.\ bij\text{-}betw\ (map\ g)\ \{xs.\ set\ xs \subseteq B\}\ \{xs.\ set\ xs \subseteq A\} \wedge$
$inverses\text{-}on\ (map\ f)\ (map\ g)\ \{xs.\ set\ xs \subseteq A\}\ \{xs.\ set\ xs \subseteq B\}$
proof −
from *bij-betw-inv-alt*[*OF assms*, *simplified inverses-on-def*]
obtain g **where**

bij-betw g B A
$\forall\, x \in A.\ g\ (f\ x) = x$
$\forall\, x \in B.\ f\ (g\ x) = x$
by *blast*

note *bij-betw-map*[*OF* ‹*bij-betw g B A*›]
moreover
{
have $\forall\, x.\ set\ x \subseteq A \longrightarrow map\ g\ (map\ f\ x) = x$
proof *safe*
fix *x*
assume $set\ x \subseteq A$
then show $map\ g\ (map\ f\ x) = x$
by (*clarsimp simp*: *list-eq-iff-nth-eq* ‹$\forall\, x \in A.\ g\ (f\ x) = x$› *subsetD*)
qed
moreover
have $\forall\, x.\ set\ x \subseteq B \longrightarrow map\ f\ (map\ g\ x) = x$
proof *safe*
fix *x*
assume $set\ x \subseteq B$
then show $map\ f\ (map\ g\ x) = x$
by (*clarsimp simp*: *list-eq-iff-nth-eq* ‹$\forall\, x \in B.\ f\ (g\ x) = x$› *subsetD*)
qed
ultimately have
inverses-on (*map f*) (*map g*) $\{xs.\ set\ xs \subseteq A\}$ $\{xs.\ set\ xs \subseteq B\}$
by (*simp add*: *inverses-on-def*)
}
ultimately show *?thesis*
by *blast*
qed

10 Lemmas about monotone functions

Note that the base version of monotone is used as the sorts cause some issues with the types

Essentially a general version of *strict-mono ?f* $\Longrightarrow$ (*?f ?x* $<$ *?f ?y*) $=$ (*?x* $<$ *?y*)

lemma *monotone-on-iff*:
assumes *monotone-on A orda ordb f*
and *asymp-on A orda*
and *totalp-on A orda*
and *asymp-on* (*f* ‘ *A*) *ordb*
and *totalp-on* (*f* ‘ *A*) *ordb*
and $x \in A$
and $y \in A$
shows *orda x y* $\longleftrightarrow$ *ordb* (*f x*) (*f y*)
proof (*safe*)

show *orda x y* $\Longrightarrow$ *ordb* (*f x*) (*f y*)
  by (*meson assms monotone-onD*)
next
  show *ordb* (*f x*) (*f y*) $\Longrightarrow$ *orda x y*
    by (*metis* (*full-types*) *assms*(*1*,*3*,*4*,*6*,*7*)
        asymp-onD monotone-onD totalp-onD imageI)
qed

The inverse of a monotonic function is also monotonic

lemma *monotone-on-bij-betw-inv*:
  assumes *monotone-on A orda ordb f*
  and *asymp-on A orda*
  and *totalp-on A orda*
  and *asymp-on B ordb*
  and *totalp-on B ordb*
  and *bij-betw f A B*
  and *bij-betw g B A*
  and *inverses-on f g A B*
shows *monotone-on B ordb orda g*
proof (*rule monotone-onI*)
  fix *x y*
  assume $x \in B$ $y \in B$ *ordb x y*
  moreover
  have $g\ x \in A$
    using ‹*bij-betw g B A*› *bij-betwE calculation*(*1*) **by** *auto*
  moreover
  have $f\ (g\ x) = x$
    using *assms*(*8*) *calculation*(*1*) *inverses-onD2* **by** *blast*
  moreover
  have $g\ y \in A$
    using ‹*bij-betw g B A*› *bij-betwE calculation*(*2*) **by** *auto*
  moreover
  have $f\ (g\ y) = y$
    using *assms*(*8*) *calculation*(*2*) *inverses-onD2* **by** *blast*
  ultimately show *orda* (*g x*) (*g y*)
    using *assms bij-betw-imp-surj-on monotone-on-iff* **by** *fastforce*
qed

lemma *monotone-on-bij-betw*:
  assumes *monotone-on A orda ordb f*
  and *asymp-on A orda*
  and *totalp-on A orda*
  and *asymp-on B ordb*
  and *totalp-on B ordb*
  and *bij-betw f A B*
shows $\exists$ *g. bij-betw g B A* $\wedge$ *inverses-on f g A B* $\wedge$ *monotone-on B ordb orda g*
  using *assms bij-betw-inv-alt monotone-on-bij-betw-inv* **by** *fastforce*

11 Sorting

11.1 General sorting

Intro for *sorted-wrt*

lemmas *sorted-wrtI* = *sorted-wrt-iff-nth-less*[*THEN iffD2*, *OF allI*, *OF allI*, *OF impI*, *OF impI*]

lemma *sorted-wrt-mapI*:
$(\bigwedge i\ j.\ [\![i < j; j < \mathit{length}\ xs]\!] \Longrightarrow P\ (f\ (xs\ !\ i))\ (f\ (xs\ !\ j))) \Longrightarrow$
sorted-wrt P (*map f xs*)
by (*metis* (*mono-tags*, *lifting*) *length-map nth-map order-less-trans sorted-wrtI*)

lemma *sorted-wrt-mapD*:
$(\bigwedge i\ j.\ [\![\mathit{sorted\text{-}wrt}\ P\ (\mathit{map}\ f\ xs); i < j; j < \mathit{length}\ xs]\!] \Longrightarrow P\ (f\ (xs\ !\ i))\ (f\ (xs\ !\ j)))$
by (*metis* (*mono-tags*, *lifting*) *length-map nth-map order-less-trans sorted-wrt-iff-nth-less*)

lemma *monotone-on-sorted-wrt-map*:
assumes *monotone-on A orda ordb f*
and *sorted-wrt orda xs*
and *set xs* ⊆ *A*
shows *sorted-wrt ordb* (*map f xs*)
proof (*rule sorted-wrt-mapI*)
fix *i j*
assume $i < j$ $j < \mathit{length}\ xs$
with *sorted-wrt-nth-less*[*OF assms*(*2*)]
have *orda* (*xs* ! *i*) (*xs* ! *j*)
by *blast*
with *monotone-onD*[*OF assms*(*1*), *of xs* ! *i xs* ! *j*] *assms*(*3*)
show *ordb* (*f* (*xs* ! *i*)) (*f* (*xs* ! *j*))
by (*meson* ‹$i < j$› ‹$j < \mathit{length}\ xs$› *nth-mem order-less-trans subsetD*)
qed

lemma *monotone-on-map-sorted-wrt*:
assumes *monotone-on A orda ordb f*
and *asymp-on A orda*
and *totalp-on A orda*
and *asymp-on* (*f* ‘ *A*) *ordb*
and *totalp-on* (*f* ‘ *A*) *ordb*
and *sorted-wrt ordb* (*map f xs*)
and *set xs* ⊆ *A*
shows *sorted-wrt orda xs*
proof (*rule sorted-wrtI*)
fix *i j*
assume $i < j$ $j < \mathit{length}\ xs$
with *sorted-wrt-nth-less*[*OF assms*(*6*)]
have *ordb* (*f* (*xs* ! *i*)) (*f* (*xs* ! *j*))
by *force*

with *monotone-on-iff*[*OF assms*(*1*−*3*), *of xs* ! *i xs* ! *j*]
show *orda* (*xs* ! *i*) (*xs* ! *j*)
using ‹*i* < *j*› ‹*j* < *length xs*› *assms*(*4*,*5*,*7*)
nth-mem order-less-trans **by** *blast*
qed

11.2 Sorting on linear orders

context *linorder* **begin**

abbreviation *strict-sorted xs* ≡ *sorted-wrt* (<) *xs*

lemma *sorted-nth-less-mono*:
⟦*sorted xs*; *i* < *length xs*; *j* < *length xs*; *i* ≠ *j*; *xs* ! *i* < *xs* ! *j*⟧ ⟹ *i* < *j*
by (*meson linorder-neqE-nat not-less sorted-iff-nth-mono-less*)

lemma *strict-sorted-nth-less-mono*:
⟦*strict-sorted xs*; *i* < *length xs*; *j* < *length xs*; *i* ≠ *j*; *xs* ! *i* < *xs* ! *j*⟧ ⟹ *i* < *j*
using *strict-sorted-iff sorted-nth-less-mono* **by** *blast*

lemma *strict-sorted-Min*:
⟦*strict-sorted xs*; *xs* ≠ []⟧ ⟹ *xs* ! *0* = *Min* (*set xs*)
by (*metis finite-set list.simps*(*15*) *Min-insert2 strict-sorted-imp-sorted*
nth-Cons-0 sorted-wrt.elims(*2*))

lemma *strict-sorted-take*:
assumes *strict-sorted xs*
and *i* < *length xs*
shows *set* (*take i xs*) = {*x*. *x* ∈ *set xs* ∧ *x* < *xs* ! *i*}
proof (*safe*)
fix *x*
assume *x* ∈ *set* (*take i xs*)
then show *x* ∈ *set xs*
by (*meson in-set-takeD*)
next
fix *x*
assume *x* ∈ *set* (*take i xs*)
then show *x* < *xs* ! *i*
by (*metis assms id-take-nth-drop list.set-intros*(*1*) *sorted-wrt-append*)
next
fix *x*
assume *x* ∈ *set xs x* < *xs* ! *i*
hence ∃*j* < *length xs*. *xs* ! *j* = *x*
by (*simp add*: *in-set-conv-nth*)
then obtain *j* **where**
j < *length xs xs* ! *j* = *x*
by *blast*
with *strict-sorted-nth-less-mono*[*OF assms*(*1*) - *assms*(*2*), *of j*] ‹*x* < *xs* ! *i*›
have *j* < *i*

```
    by blast
  then show x ∈ set (take i xs)
    using ‹j < length xs› ‹xs ! j = x› in-set-conv-nth by fastforce
qed

lemma strict-sorted-card-idx:
  ⟦strict-sorted xs; i < length xs⟧ ⟹ card {x. x ∈ set xs ∧ x < xs ! i} = i
 by (metis (mono-tags, lifting) distinct-card distinct-take length-take strict-sorted-iff
                                ord-class.min-def order-class.leD strict-sorted-take)

lemmas strict-sorted-distinct-set-unique =
  sorted-distinct-set-unique[OF strict-sorted-imp-sorted - strict-sorted-imp-sorted]

lemma sorted-and-distinct-imp-strict-sorted:
  ⟦sorted xs; distinct xs⟧ ⟹ strict-sorted xs
  using strict-sorted-iff
  by blast

lemma filter-sorted:
  sorted xs ⟹ sorted (filter P xs)
  using sorted-wrt-filter by blast

lemma sorted-nth-eq:
  assumes sorted xs
  and     j < length xs
  and     xs ! i = xs ! j
  and     i ≤ k
  and     k ≤ j
shows xs ! k = xs ! i
  by (metis assms sorted-iff-nth-mono preorder-class.le-less-trans nle-le)

lemma sorted-find-Min:
  sorted xs ⟹ ∃ x ∈ set xs. P x ⟹ List.find P xs = Some (Min {x∈set xs. P x})
proof (induct xs)
  case Nil then show ?case by simp
next
  case (Cons x xs) show ?case
  proof (cases P x)
    case True
    with Cons show ?thesis by (auto intro: Min-eqI [symmetric])
  next
    case False then have {y. (y = x ∨ y ∈ set xs) ∧ P y} = {y ∈ set xs. P y}
      by auto
    with Cons False show ?thesis by (simp-all)
  qed
qed

lemma sorted-cons-nil:
  xs = [] ⟹ sorted (x # xs)
```

by *simp*

lemma *sorted-consI*:
$[\![xs \neq []; \ sorted\ xs; \ x \leq xs\ !\ 0]\!] \Longrightarrow sorted\ (x\ \#\ xs)$
by (*metis list.exhaust sorted2-simps*(*2*) *nth-Cons-0*)

end

11.3 Sorting on orders

context *order* **begin**

lemma *strict-mono-strict-sorted-map-1*:
assumes *strict-mono* α
and *strict-sorted xs*
shows *strict-sorted* (*map* α *xs*)
using *assms*(*1*) *assms*(*2*) *monotone-on-sorted-wrt-map* **by** *blast*

lemma *strict-mono-sorted-map-2*:
assumes *strict-mono* α
and *strict-sorted* (*map* α *xs*)
shows *strict-sorted xs*
using *assms*(*1*) *assms*(*2*) *monotone-on-map-sorted-wrt* **by** *fastforce*

end

12 Mapping elements to natural numbers

This section contains a mapping from elements to natural numbers that maintains ordering.

definition *elm-rank* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a\ set \Rightarrow 'a \Rightarrow nat$
where
$elm\text{-}rank\ ord\ A\ x = card\ \{y.\ y \in A \wedge ord\ y\ x\}$

lemma *monotone-on-elm-rank*:
assumes *finite A*
and *transp-on A ord*
and *irreflp-on A ord*
shows *monotone-on A ord* (<) (*elm-rank ord A*)
proof (*rule monotone-onI*)
fix *a b*
assume $a \in A$ $b \in A$ *ord a b*

have $\{y.\ y \in A \wedge ord\ y\ a\} \subseteq \{y.\ y \in A \wedge ord\ y\ b\}$
proof *safe*
fix *x*
assume $x \in A$ *ord x a*
then show *ord x b*

by (*meson* ‹$a \in A$› ‹$b \in A$› ‹*ord a b*› *assms*(*2*) *transp-onD*)
qed
moreover
have $a \in \{y.\ y \in A \land ord\ y\ b\}$
by (*simp add*: ‹$a \in A$› ‹*ord a b*›)
moreover
have $a \notin \{y.\ y \in A \land ord\ y\ a\}$
using *assms*(*3*) *irreflp-onD* **by** *fastforce*
ultimately have $\{y.\ y \in A \land ord\ y\ a\} \subset \{y.\ y \in A \land ord\ y\ b\}$
by *blast*
then show *elm-rank ord A a* $<$ *elm-rank ord A b*
by (*simp add*: *elm-rank-def assms*(*1*) *psubset-card-mono*)
qed

lemma *elm-rank-insert-min*:
assumes *finite A*
and $x \notin A$
and $\forall y \in A.\ ord\ x\ y$
and $z \in A$
shows *elm-rank ord* (*insert x A*) $z = $ *Suc* (*elm-rank ord A z*)
unfolding *elm-rank-def*
proof −
have *ord x z*
by (*simp add*: *assms*(*3*,*4*))
hence $\{y \in insert\ x\ A.\ ord\ y\ z\} = insert\ x\ \{y \in A.\ ord\ y\ z\}$
by *safe*
moreover
have $x \notin \{y \in A.\ ord\ y\ z\}$
using *assms*(*2*) **by** *auto*
ultimately show $card\ \{y \in insert\ x\ A.\ ord\ y\ z\} = Suc\ (card\ \{y \in A.\ ord\ y\ z\})$
by (*simp add*: *assms*(*1*))
qed

definition (**in** *order*) *elem-rank* :: $'a\ set \Rightarrow 'a \Rightarrow nat$
where
elem-rank = *elm-rank* ($<$)

lemma (**in** *order*) *strict-mono-on-elem-rank*:
assumes *finite A*
shows *strict-mono-on A* (*elem-rank A*)
by (*simp add*: *assms elem-rank-def monotone-on-elm-rank*)

lemma (**in** *linorder*) *bij-betw-elem-rank-upt*:
assumes *finite A*
shows *bij-betw* (*elem-rank A*) *A* $\{0..<card\ A\}$
proof −
have $\bigwedge x.\ x \in A \Longrightarrow \{y.\ y \in A \land y < x\} \subset A$
by *blast*
hence $\bigwedge x.\ x \in A \Longrightarrow card\ \{y.\ y \in A \land y < x\} < card\ A$

by (*meson assms psubset-card-mono*)
hence $\bigwedge x.\ x \in A \Longrightarrow$ *elem-rank* $A\ x \in \{0..<card\ A\}$
by (*simp add*: *elm-rank-def elem-rank-def*)
moreover
have $\bigwedge x\ y.\ [\![x \in A;\ y \in A]\!] \Longrightarrow$ (*elem-rank* $A\ x$ = *elem-rank* $A\ y$) = ($x = y$)
by (*metis assms less-irrefl-nat antisym-conv3 ord.strict-mono-onD strict-mono-on-elem-rank*)
moreover
{
let *?xs* = *sorted-list-of-set A*
have *strict-sorted ?xs*
by *simp*
moreover
have $\bigwedge x.$ *elem-rank* $A\ x$ = *elem-rank* (*set ?xs*) x
using *assms* **by** *force*
moreover
have $\bigwedge y.\ y <$ *length ?xs* $\Longrightarrow y$ = *elem-rank* (*set ?xs*) (*?xs* ! y)
by (*metis* (*no-types*, *lifting*) *Collect-cong calculation*(*1*) *elem-rank-def elm-rank-def strict-sorted-card-idx*)
ultimately have $\bigwedge y.\ y \in \{0..<card\ A\} \Longrightarrow \exists x \in A.\ y$ = *elem-rank* $A\ x$
by (*metis assms length-sorted-list-of-set set-sorted-list-of-set nth-mem ord-class.atLeastLessThan-iff*)
}
ultimately show *?thesis*
using *bij-betwI'* **by** *blast*
qed

lemma (**in** *order*) *elem-rank-insert-min*:
$[\![$*finite* $A;\ x \notin A;\ \forall y \in A.\ x < y;\ z \in A]\!] \Longrightarrow$ *elem-rank* (*insert* x A) z = *Suc* (*elem-rank* $A\ z$)
by (*simp add*: *elm-rank-insert-min elem-rank-def*)
end
theory *Repeat*
imports *Main*
begin

13 Repeat Function At Most N Times

fun *repeatatm* :: *nat* ⇒ ('*a* ⇒ '*b* ⇒ *bool*) ⇒ ('*a* ⇒ '*b* ⇒ '*a*) ⇒ '*a* ⇒ '*b* ⇒ '*a*
where
repeatatm 0 - - acc - = acc |
repeatatm (*Suc n*) *f g acc obsv* = (*if f acc obsv then acc else repeatatm n f g* (*g acc obsv*) *obsv*)

declare *repeatatm.simps*[*simp del*]

13.1 Step and early termination lemmas

lemma *repeatatm-step-stop-Suc*:
f (*repeatatm n f g a b*) *b*

```
    ⟹ repeatatm (Suc n) f g a b = repeatatm n f g a b
proof (induct n arbitrary: a)
  case 0
  then show ?case
    by (simp add: repeatatm.simps)
next
  case (Suc n a)
  note IH = this
  show ?case
  proof (cases f a b)
    assume f a b
    then show ?case
      by (simp add: repeatatm.simps)
  next
    assume ¬ f a b
    with IH(2)
    have f (repeatatm n f g (g a b) b) b
      by (simp add: repeatatm.simps)
    with IH(1)
    have repeatatm (Suc n) f g (g a b) b = repeatatm n f g (g a b) b
      by blast
    then show ?case
      by (simp add: repeatatm.simps)
  qed
qed

lemma repeatatm-step:
  ¬f (repeatatm n f g a b) b
    ⟹ repeatatm (Suc n) f g a b = g (repeatatm n f g a b) b
proof (induct n arbitrary: a)
  case 0
  then show ?case
    by (simp add: repeatatm.simps)
next
  case (Suc n a)
  note IH = this
  show ?case
  proof (cases f a b)
    assume f a b
    with IH(2)
    show ?case
      by (simp add: repeatatm.simps)
  next
    assume ¬ f a b
    with IH(2)
    have ¬ f (repeatatm n f g (g a b) b) b
      by (simp add: repeatatm.simps)
    with IH(1)
    have repeatatm (Suc n) f g (g a b) b = g (repeatatm n f g (g a b) b) b
```

by *blast*
then show *?case*
by (*simp add*: *repeatatm.simps* ‹¬ *f a b*›)
qed
qed

lemma *repeatatm-step-forward*:
$\neg f\ a\ b \Longrightarrow repeatatm\ (Suc\ n)\ f\ g\ a\ b = repeatatm\ n\ f\ g\ (g\ a\ b)\ b$
by (*induct n arbitrary*: *a*; *simp add*: *repeatatm.simps*)

lemma *repeatatm-stop-Suc*:
$[\![f\ (repeatatm\ n\ f\ g\ a\ b)\ b]\!] \Longrightarrow f\ (repeatatm\ (Suc\ n)\ f\ g\ a\ b)\ b$
proof (*induct n arbitrary*: *a*)
case *0*
then show *?case*
by (*simp add*: *repeatatm.simps*)
next
case (*Suc n a*)
note *IH* = *this*
show *?case*
proof (*cases f a b*)
assume *f a b*
then show *?case*
by (*simp add*: *repeatatm.simps*)
next
assume ¬ *f a b*
with *IH*(*2*)
have $f\ (repeatatm\ n\ f\ g\ (g\ a\ b)\ b)\ b$
by (*simp add*: *repeatatm.simps*)
with *IH*(*1*)
have $f\ (repeatatm\ (Suc\ n)\ f\ g\ (g\ a\ b)\ b)\ b$
by *blast*
then show *?case*
by (*simp add*: *repeatatm.simps*)
qed
qed

lemma *repeatatm-stop*:
$[\![f\ (repeatatm\ n\ f\ g\ a\ b)\ b;\ n \le m]\!] \Longrightarrow f\ (repeatatm\ m\ f\ g\ a\ b)\ b$
proof (*induct m arbitrary*: *a n*)
case *0*
then show *?case*
by *simp*
next
case (*Suc m a n*)
note *IH* = *this*
show *?case*
proof (*cases* $n \le m$)
assume $n \le m$

with *repeatatm-stop-Suc*[*of f m g a b*] *IH*(*1*)[*OF IH*(*2*)]
show *?case*
by *simp*
next
assume $\neg$ $n \leq m$
with *IH*(*2*,*3*)
show *?case*
using *le-Suc-eq* **by** *blast*
qed
qed

lemma *repeatatm-step-stop*:
$[\![f\ (repeatatm\ n\ f\ g\ a\ b)\ b;\ n \leq m]\!] \Longrightarrow repeatatm\ m\ f\ g\ a\ b = repeatatm\ n\ f\ g\ a\ b$
proof (*induct m arbitrary*: *a n*)
case *0*
then show *?case*
by *simp*
next
case (*Suc m a n*)
note *IH* = *this*
show *?case*
proof (*cases* $n \leq m$)
assume $n \leq m$
with *repeatatm-step-stop-Suc*[*of f m g a b*] *IH*(*2*) *IH*(*1*)[*OF IH*(*2*)]
show *?case*
by *simp*
next
assume $\neg$ $n \leq m$
then show *?case*
using *Suc.prems*(*2*) *le-SucE* **by** *blast*
qed
qed

lemma *repeatatm-not-stop-Suc*:
$\neg f\ (repeatatm\ (Suc\ n)\ f\ g\ a\ b)\ b \Longrightarrow \neg f\ (repeatatm\ n\ f\ g\ a\ b)\ b$
apply (*rule contrapos-nn*[**where** $Q = f\ (repeatatm\ (Suc\ n)\ f\ g\ a\ b)\ b$]; *simp*)
using *repeatatm-stop-Suc*[*of f n g a b*] **by** *simp*

lemma *repeatatm-maintain-inv*:
assumes $\bigwedge a.\ P\ a \Longrightarrow P\ (g\ a\ b)$
shows $P\ a \Longrightarrow P\ (repeatatm\ n\ f\ g\ a\ b)$
proof (*induct n arbitrary*: *a*)
case *0*
then show *?case*
by (*simp add*: *repeatatm.simps*)
next
case (*Suc n a*)
note *IH* = *this*

```
  from IH(1)[OF IH(2)]
  have P (repeatatm n f g a b)
    by assumption

  with ‹⋀a. P a ⟹ P (g a b)›
  show ?case
    by (metis repeatatm-step repeatatm-step-stop-Suc)
qed
```

14 Repeat Function N Times

```
definition repeat :: nat ⇒ ('a ⇒ 'b ⇒ 'a) ⇒ 'a ⇒ 'b ⇒ 'a
  where
repeat n f a b = repeatatm n (λx y. False) f a b

lemma repeat-0:
  repeat 0 f a b = a
  by (simp add: repeat-def repeatatm.simps(1))

lemma repeat-step:
  repeat (Suc n) f a b = f (repeat n f a b) b
  unfolding repeat-def
  by (simp add: repeatatm-step)

lemma repeat-step-forward:
  repeat (Suc n) f a b = repeat n f (f a b) b
  unfolding repeat-def
  by (simp add: repeatatm-step-forward)

lemma repeat-maintain-inv:
  assumes ⋀a. P a ⟹ P (f a b)
  shows P a ⟹ P (repeat n f a b)
  by (simp add: assms repeat-def repeatatm-maintain-inv)

lemma repeat-eq-fold:
  repeat n f a b = fold (λ- a. f a b) [0..<n] a
  apply (induct n)
   apply (simp add: repeat-def repeatatm.simps)
  apply (subst repeat-step)
  apply simp
  done

end
theory Continuous-Interval
  imports Main
begin
```

15 Continuous Intervals

```
definition
  continuous-list :: (nat × nat) list ⇒ bool
where
  continuous-list xs =
     (∀ i. Suc i < length xs ⟶ fst (xs ! Suc i) = snd (xs ! i))

lemma continuous-list-nil:
  continuous-list []
  by (simp add: continuous-list-def)

lemma continuous-list-singleton:
  continuous-list [x]
  by (simp add: continuous-list-def)

lemma continuous-list-cons:
  continuous-list (x # xs) ⟹ continuous-list xs
  by (simp add: continuous-list-def)

lemma continuous-list-app:
  continuous-list (xs @ ys) ⟹ continuous-list xs ∧ continuous-list ys
proof (induct xs)
  case Nil
  then show ?case
    by (clarsimp simp: continuous-list-nil continuous-list-cons)
next
  case (Cons x1 xs)
  note IH1 = this(1) and
       IH2 = this(2)

  from continuous-list-cons IH2
  have continuous-list (xs @ ys)
    by simp
  with IH1
  have continuous-list ys
    by simp

  have xs = [] ∨ xs ≠ []
    by simp
  then show ?case
  proof
    assume xs = []
    with IH2 continuous-list-singleton
    have continuous-list (x1 # xs)
      by blast
    with ‹continuous-list ys›
    show ?thesis
      by simp
```

```
  next
    assume xs ≠ []
    hence ∃x xs'. xs = x # xs'
      using neq-Nil-conv by blast
    then obtain x xs' where
      xs = x # xs'
      by blast

    have continuous-list (x1 # (x # xs'))
      unfolding continuous-list-def
    proof(intro allI impI)
      fix i
      assume Suc i < length (x1 # (x # xs'))
      have i = 0 ∨ i ≠ 0
        by blast
      then show fst ((x1 # x # xs') ! Suc i) = snd ((x1 # x # xs') ! i)
      proof
        assume i = 0
        then show ?thesis
          using IH2 ‹xs = x # xs'› continuous-list-def by auto
      next
        assume i ≠ 0
        then show ?thesis
         using IH1 ‹Suc i < length (x1 # x # xs')› ‹continuous-list (xs @ ys)› ‹xs
= x # xs'›
              continuous-list-def
          by auto
      qed
    qed
    with ‹continuous-list ys› ‹xs = x # xs'›
    show ?thesis
      by simp
  qed
qed

lemma continuous-list-interval-1:
  assumes continuous-list xs
  and     xs ≠ []
  and     fst (hd xs) ≤ i
  and     i < snd (last xs)
  shows ∃j < length xs. fst (xs ! j) ≤ i ∧ i < snd (xs ! j)
  using assms
proof (induct xs)
  case Nil
  then show ?case
    by simp
next
  case (Cons x1 xs)
  note IH = this
```

```
have xs = [] ⟶ ?case
proof
  assume xs = []
  with IH(4,5)
  show ?case
    by simp
qed

have xs ≠ [] ⟶ ?case
proof
  assume xs ≠ []
  hence ∃ a b xs'. xs = (a, b) # xs'
    by (metis (full-types) list.exhaust surj-pair)
  then obtain a b xs' where
    xs = (a, b) # xs'
    by blast

  from IH(2)
  have continuous-list xs
    using continuous-list-cons by blast

  from IH(5) ‹xs ≠ []›
  have i < snd (last xs)
    by simp

  have a ≤ i ∨ i < a
    by linarith
  then show ?case
  proof
    assume a ≤ i
    with IH(1)
          [OF ‹continuous-list xs›
              ‹xs ≠ []› -
              ‹i < snd (last xs)›]
         ‹xs = (a, b) # xs'›
    show ?thesis
      by auto
  next
    assume i < a
    with IH(2) ‹xs = (a, b) # xs'›
    have i < snd x1
      using continuous-list-def by auto
    with IH(4)
    show ?thesis
      by auto
  qed
qed
```

with ‹*xs* = [] ⟶ *?case*›
show *?case*
by *blast*
qed

lemma *continuous-list-interval-2*:
assumes *continuous-list xs*
and *length xs* = *Suc n*
and *fst* (*xs* ! *0*) ≤ *i*
and *i* < *snd* (*xs* ! *n*)
shows ∃ *j* < *length xs*. *fst* (*xs* ! *j*) ≤ *i* ∧ *i* < *snd* (*xs* ! *j*)
proof −
from ‹*length xs* = *Suc n*›
have *xs* ≠ []
by *auto*

from ‹*fst* (*xs* ! *0*) ≤ *i*› ‹*xs* ≠ []›
have *fst* (*hd xs*) ≤ *i*
by (*simp add*: *hd-conv-nth*)

from ‹*i* < *snd* (*xs* ! *n*)›
‹*length xs* = *Suc n*› ‹*xs* ≠ []›
have *i* < *snd* (*last xs*)
by (*simp add*: *last-conv-nth*)

from *continuous-list-interval-1*
[*OF* ‹*continuous-list xs*›
‹*xs* ≠ []›
‹*fst* (*hd xs*) ≤ *i*›
‹*i* < *snd* (*last xs*)›]
show *?thesis*
by *assumption*
qed

end
theory *List-Slice*
imports *Main*
begin

16 List Slices

fun *list-slice* ::
'a list ⇒ *nat* ⇒ *nat* ⇒ *'a list*
where
list-slice xs i j = *drop i* (*take j xs*)

lemma *length-list-slice*[*simp add*]:
length (*list-slice xs i j*) = (*min j* (*length xs*)) − *i*
by *simp*

```
lemma list-slice-cons:
  fixes i j :: nat
  assumes i ≤ j
  assumes i > 0
  shows list-slice (x # xs) i j = list-slice xs (i − 1) (j − 1)
  using assms gr0-implies-Suc[OF order.strict-trans2]
  by (fastforce dest: gr0-implies-Suc)

lemma list-slice-append:
  fixes i j k :: nat
  assumes i ≤ j
  assumes j ≤ k
  shows list-slice xs i k = list-slice xs i j @ list-slice xs j k
using assms
proof (induct xs arbitrary: i j k)
  case (Cons a xs i j k)
  note IH = this
  show ?case
  proof (cases i > 0)
    assume ¬ i > 0
    hence i = 0
      by simp
    then show ?case
      unfolding list-slice.simps
      by (metis IH(3) append-take-drop-id drop0 min-def take-take)
  next
    assume i > 0
    with list-slice-cons[of i k a xs]
    have list-slice (a # xs) i k = list-slice xs (i − 1) (k − 1)
      using IH(2) IH(3) dual-order.trans by blast
    moreover
    from list-slice-cons[of i j a xs]
    have list-slice (a # xs) i j = list-slice xs (i − 1) (j − 1)
      using IH(2) ‹i > 0› by blast
    moreover
    from list-slice-cons[of j k a xs]
    have list-slice (a # xs) j k = list-slice xs (j − 1) (k − 1)
      using IH(2) IH(3) ‹0 < i› by blast
    moreover
    from IH(1)[of i−1 j−1 k−1]
    have list-slice xs (i − 1) (k − 1) =
          list-slice xs (i − 1) (j − 1) @ list-slice xs (j − 1) (k − 1)
      using IH(2) IH(3) diff-le-mono by blast
    ultimately show ?case
      by presburger
  qed
qed simp
```

lemma *list-slice-0-length*:
fixes *xs* :: *'a list*
fixes *n* :: *nat*
assumes *length xs* ≤ *n*
shows *list-slice xs 0 n* = *xs*
using *assms* **by** *simp*

lemma *list-slice-n-n*[*simp add*]:
fixes *xs* :: *'a list*
fixes *n* :: *nat*
shows *list-slice xs n n* = []
by *simp*

lemma *list-slice-nth*:
fixes *i s e* :: *nat*
fixes *xs* :: *'a list*
assumes *i* < *length xs*
assumes *s* ≤ *i*
assumes *i* < *e*
shows (*list-slice xs s e*) ! (*i* − *s*) = *xs* ! *i*
using *assms* **by** *simp*

lemma *list-slice-start-gre-length*:
fixes *xs* :: *'a list*
fixes *s* :: *nat*
assumes *length xs* ≤ *s*
shows *list-slice xs s e* = []
using *assms* **by** *simp*

lemma *list-slice-end-gre-length*:
fixes *xs* :: *'a list*
fixes *e* :: *nat*
assumes *length xs* ≤ *e*
shows *list-slice xs s e* = *list-slice xs s* (*length xs*)
using *assms* **by** *simp*

lemma *fold-list-slice*:
fixes *i j* :: *nat*
fixes *B* :: *nat list*
assumes *i* ≤ *j*
and *j* < *length B*
and *sorted B*
fixes *T zs* :: *'a list*
shows
fold (λ*x xs*. *xs* @ *list-slice T* (*B* ! *x*) (*B* ! *Suc x*)) [*i*..<*j*] *zs*
= *zs* @ (*list-slice T* (*B* ! *i*) (*B* ! *j*))
using *assms*
proof (*induct j arbitrary*: *i*)
case *0*

```
  then show ?case
    by (simp del: list-slice.simps)
next
  case (Suc j i)
  note IH = this
  show ?case
  proof (cases i ≤ j)
    assume i-leq-j: i ≤ j

    have fold (λx xs. xs @ list-slice T (B ! x) (B ! Suc x)) [i..<Suc j] zs =
          fold (λx xs. xs @ list-slice T (B ! x) (B ! Suc x)) [i..<j] zs @
            list-slice T (B ! j) (B ! Suc j)
      using ‹i ≤ j› by force
    moreover
    from IH(1)[OF ‹i ≤ j› - IH(4)]
    have fold (λx xs. xs @ list-slice T (B ! x) (B ! Suc x)) [i..<j] zs =
          zs @ list-slice T (B ! i) (B ! j)
      using Suc.prems(2) Suc-lessD by blast
    moreover
    have list-slice T (B ! i) (B ! Suc j) = list-slice T (B ! i) (B ! j) @
          list-slice T (B ! j) (B ! Suc j)
      by (meson Suc.prems(2,3) Suc-lessD i-leq-j list-slice-append
               sorted-iff-nth-Suc sorted-nth-mono)
    ultimately show ?case
      by (metis append.assoc)
  next
    assume ¬ i ≤ j
    then show ?case
      using Suc.prems(1) le-SucE by fastforce
  qed
qed

lemma nth-list-slice:
  fixes i s e :: nat
  fixes xs :: 'a list
  assumes i < length (list-slice xs s e)
  shows (list-slice xs s e) ! i = xs ! (s + i)
  using assms less-diff-conv by auto

lemma list-slice-nth-eq-iff-index-eq:
  fixes i s e j :: nat
  fixes xs :: 'a list
  assumes distinct (list-slice xs s e)
  assumes e ≤ length xs
  assumes s ≤ i and i < e
  and      s ≤ j and j < e
  shows   (xs ! i = xs ! j) ⟷ (i = j)
  using assms
  by (fastforce
```

dest: *nth-eq-iff-index-eq*[**where** $i = i - s$ **and** $j = j - s$])

lemma *distinct-list-slice*:
fixes *i j* :: *nat*
fixes *xs* :: *'a list*
assumes *distinct xs*
shows *distinct* (*list-slice xs i j*)
using *assms* **by** *simp*

lemma *list-slice-nth-mem*:
fixes *e* :: *nat*
fixes *xs* :: *'a list*
fixes *s i* :: *nat*
assumes $s \leq i$ **and** $i < e$
assumes $e \leq$ *length xs*
shows *xs* ! $i \in$ *set* (*list-slice xs s e*)
by (*metis* (*no-types*, *opaque-lifting*) *assms nat-le-iff-add*
add-diff-cancel-left' diff-less-mono nth-mem
length-list-slice min-def nth-list-slice)

lemma *nth-mem-list-slice*:
fixes *x* :: *'a*
fixes *xs* :: *'a list*
fixes *s e* :: *nat*
assumes $x \in$ *set* (*list-slice xs s e*)
shows $\exists\, i <$ *length xs*.
$s \leq i \wedge$
$i < e \wedge$
xs ! $i = x$
proof −
from *in-set-conv-nth*[*THEN iffD1*, *OF* ‹- ∈ -›]
obtain *i* **where**
$i <$ *length* (*list-slice xs s e*)
(*list-slice xs s e*) ! $i = x$
by *auto*
with *nth-list-slice*[*of i xs s e*]
have *xs* ! $(s + i) = x$
by *auto*
moreover
have $s \leq s + i$
by *linarith*
moreover
have $s + i <$ *length xs*
using ‹$i <$ *length* (*list-slice xs s e*)› **by** *auto*
moreover
have $s + i < e$
using ‹$i <$ *length* (*list-slice xs s e*)› **by** *auto*
ultimately show *?thesis*
by *blast*

qed

lemma *list-slice-subset*:
  fixes $i\ j :: nat$
  fixes $xs :: 'a\ list$
  shows $set\ (list\text{-}slice\ xs\ i\ j) \subseteq set\ xs$
  using *set-drop-subset set-take-subset* **by** *force*

lemma *list-slice-Suc*:
  fixes $i\ j :: nat$
  fixes $xs :: 'a\ list$
  assumes $i < length\ xs$
  assumes $i < j$
  shows $list\text{-}slice\ xs\ i\ j = xs\ !\ i\ \#\ list\text{-}slice\ xs\ (Suc\ i)\ j$
  by (*metis assms Cons-nth-drop-Suc Suc-diff-Suc*
      list-slice.simps take-Suc-Cons drop-take)

lemma *list-slice-update-unchanged-1*:
  fixes $xs :: 'a\ list$
  fixes $i\ j\ k :: nat$
  assumes $i < j$
  shows $list\text{-}slice\ (xs[i := x])\ j\ k = list\text{-}slice\ xs\ j\ k$
  by (*simp add*: *assms drop-take*)

lemma *list-slice-update-unchanged-2*:
  fixes $i\ j\ k :: nat$
  fixes $xs :: 'a\ list$
  assumes $k \leq i$
  shows $list\text{-}slice\ (xs[i := x])\ j\ k = list\text{-}slice\ xs\ j\ k$
  by (*metis assms list-slice.simps take-update-cancel*)

lemma *list-slice-update-changed*:
  assumes $i < length\ xs$
  assumes $j \leq i$
  assumes $i < k$
  shows $list\text{-}slice\ (xs[i := x])\ j\ k = (list\text{-}slice\ xs\ j\ k)[i - j := x]$
  using *assms*
  by (*fastforce*
    intro: *list-eq-iff-nth-eq*[*THEN iffD2*]
    simp: *nth-list-slice nth-list-update*)

lemma *list-slice-map-nth-upt*:
  assumes $j < length\ xs$
  shows $list\text{-}slice\ xs\ i\ j = map\ (nth\ xs)\ [i..<j]$
  using *assms*
  by (*fastforce intro*: *list-eq-iff-nth-eq*[*THEN iffD2*])

lemma *map-list-slice*:

map f (*list-slice xs i j*) = *list-slice* (*map f xs*) *i j*
by (*simp add*: *drop-map take-map*)

17 Sorted List Slice

lemma (**in** *linorder*) *sorted-list-slice*:
assumes *sorted xs*
shows *sorted* (*list-slice xs i j*)
by (*simp add*: *assms sorted-wrt-drop sorted-wrt-take*)

lemma (**in** *linorder*) *sorted-map-list-slice*:
assumes *sorted* (*map f xs*)
shows *sorted* (*map f* (*list-slice xs i j*))
by (*metis assms drop-map list-slice.simps local.sorted-list-slice take-map*)

lemma (**in** *linorder*) *sorted-map-filter-list-slice*:
assumes *sorted* (*map f* (*filter P xs*))
shows *sorted* (*map f* (*filter P* (*list-slice xs i j*)))
proof −
have $i \leq j \vee j < i$
using *linorder-class.le-less-linear* **by** *blast*
moreover
have $j < i \Longrightarrow$ *?thesis*
proof −
assume $j < i$
hence *list-slice xs i j* = []
by (*simp add*: *drop-take*)
then show *?thesis*
by *simp*
qed
moreover
have $i \leq j \Longrightarrow$ *?thesis*
proof −
let *?as* = *list-slice xs 0 i* **and**
?bs = *list-slice xs i j* **and**
?cs = *list-slice xs j* (*length xs*)
assume $i \leq j$
hence *xs* = *?as* @ *?bs* @ *?cs*
by (*metis le0 linorder-class.linear list-slice-0-length list-slice-append*
list-slice-start-gre-length)
hence *filter P xs* = *filter P ?as* @ *filter P ?bs* @ *filter P ?cs*
by (*metis filter-append*)
hence *map f* (*filter P xs*)
= (*map f* (*filter P ?as*)) @ (*map f* (*filter P ?bs*)) @ (*map f* (*filter P ?cs*))
by *simp*
with ‹*sorted* (*map f* (*filter P xs*))›
show *?thesis*
by (*simp add*: *local.sorted-append*)
qed

```
  ultimately show ?thesis
    by blast
qed

lemma (in linorder) list-slice-sorted-nth-mono:
  assumes sorted (list-slice xs s e)
  and     s ≤ i
  and     i ≤ j
  and     j < e
  and     j < length xs
shows xs ! i ≤ xs ! j
proof −
  have ∃ i'. i = s + i'
    using assms(2) nat-le-iff-add by blast
  then obtain i' where
    i = s + i'
    by blast

  have ∃ j'. j = s + j'
    using assms(2) assms(3) nat-le-iff-add by auto
  then obtain j' where
    j = s + j'
    by blast

  have i' ≤ j'
    using ‹i = s + i'› ‹j = s + j'› assms(3) by auto

  have j' < length (list-slice xs s e)
    using ‹j = s + j'› assms(4) assms(5) by auto
  with sorted-nth-mono[OF assms(1) ‹i' ≤ j'›]
  have list-slice xs s e ! i' ≤ list-slice xs s e ! j' .
  moreover
  have xs ! i = list-slice xs s e ! i'
    using ‹i = s + i'› assms(3−5) by force
  moreover
  have xs ! j = list-slice xs s e ! j'
    using ‹j = s + j'› ‹j' < length (list-slice xs s e)› nth-list-slice by force
  ultimately show ?thesis
    by simp
qed
end
theory List-Lexorder-Util
  imports
    HOL−Library.List-Lexorder
begin

lemma same-equiv-def:
  (∀ j<n. s ! (i + j) = s ! Suc (i + j)) = (∀ j≤n. s ! (i + j) = s ! i)
proof safe
```

fix j
assume $\forall j{<}n.\ s\ !\ (i + j) = s\ !\ Suc\ (i + j)$ $j \leq n$
then show $s\ !\ (i + j) = s\ !\ i$
proof (*induct n arbitrary*: *j*)
case *0*
then show *?case*
by *simp*
next
case (*Suc n j*)
note *IH* = *this*
show *?case*
proof (*cases j*)
case *0*
then show *?thesis*
by *simp*
next
case (*Suc m*)
with *IH*(*1*)[*of m*] *IH*(*2*,*3*)
have $s\ !\ (i + m) = s\ !\ i$
by (*meson Suc-le-mono less-Suc-eq*)
then show *?thesis*
using *Suc Suc.prems*(*1*) *Suc.prems*(*2*) **by** *force*
qed
qed
next
fix j
assume $\forall j{\leq}n.\ s\ !\ (i + j) = s\ !\ i$ $j < n$
then show $s\ !\ (i + j) = s\ !\ Suc\ (i + j)$
using *less-eq-Suc-le* **by** *fastforce*
qed

lemma *list-less-ex*:
$xs < ys \longleftrightarrow$
$(\exists\, b\ c\ as\ bs\ cs.\ xs = as\ @\ b\ \#\ bs \wedge ys = as\ @\ c\ \#\ cs \wedge b < c) \vee$
$(\exists\, c\ cs.\ ys = xs\ @\ c\ \#\ cs)$
by (*clarsimp simp*: *List-Lexorder.list-less-def lexord-def*; *blast*)

end
theory *List-Permutation-Util*
imports *HOL−Combinatorics.List-Permutation ../util/List-Util*
begin

lemma *perm-distinct-set-of-upt-iff*:
xs <~~> $[0..{<}n] \longleftrightarrow distinct\ xs \wedge set\ xs = \{0..{<}n\}$
by (*metis atLeastLessThan-upt distinct-upt perm-distinct-iff set-eq-iff-mset-eq-distinct*)

lemma *distinct-set-of-upto-length*:
$[\![distinct\ xs;\ set\ xs = \{0..{<}n\}]\!] \Longrightarrow length\ xs = n$
apply (*drule* (*1*) *iffD2*[*OF perm-distinct-set-of-upt-iff conjI*])

apply (*drule perm-length*; *simp*)
done

lemma *set-perm-upt*:
xs <~~> [*0*..<*n*] ⟹ *set xs* = {*0*..<*n*}
using *perm-distinct-set-of-upt-iff* **by** *blast*

lemma *perm-upt-length*:
xs <~~> [*0*..<*n*] ⟹ *length xs* = *n*
using *distinct-set-of-upto-length perm-distinct-set-of-upt-iff* **by** *blast*

lemma *perm-nth-ex*:
⟦*xs* <~~> [*0*..<*n*]; *i* < *n*⟧ ⟹ ∃ *k* < *n*. *xs* ! *i* = *k*
using *perm-upt-length set-perm-upt* **by** *fastforce*

lemma *ex-perm-nth*:
⟦*xs* <~~> [*0*..<*n*]; *k* < *n*⟧ ⟹ ∃ *i* < *n*. *xs* ! *i* = *k*
by (*metis atLeast-upt distinct-Ex1 distinct-upt lessThan-iff perm-distinct-iff perm-set-eq perm-upt-length*)

lemma *set-map-nth-perm-subset*:
⟦*ys* <~~> [*0*..<*n*]; *n* ≤ *length xs*⟧ ⟹ *set* (*map* (*nth xs*) *ys*) ⊆ *set xs*
by (*simp add*: *nth-image set-perm-upt set-take-subset*)

lemma *set-map-nth-perm-eq*:
ys <~~> [*0*..<*length xs*] ⟹ *set* (*map* (*nth xs*) *ys*) = *set xs*
by (*metis perm-set-eq set-map set-map-nth-eq*)

lemma *distinct-map-nth-perm*:
⟦*distinct xs*; *n* ≤ *length xs*; *ys* <~~> [*0*..<*n*]⟧ ⟹ *distinct* (*map* (*nth xs*) *ys*)
by (*metis distinct-map distinct-map-nth perm-distinct-iff perm-set-eq*)

theorem *distinct-set-imp-perm*:
assumes *distinct xs*
and *distinct ys*
and *set xs* = *set ys*
shows *xs* <~~> *ys*
proof −
from *set-eq-iff-mset-eq-distinct*[*OF assms*(*1*,*2*), *THEN iffD1*, *OF assms*(*3*)]
show *?thesis*
by *simp*
qed

theorem *perm-nth*:
assumes *xs* <~~> *ys*
and *i* < *length xs*
shows ∃ *j* < *length ys*. *ys* ! *j* = *xs* ! *i*
by (*metis assms*(*1*) *assms*(*2*) *in-set-conv-nth perm-set-eq*)

lemma *sort-perm*:
$xs <^{\sim\sim}> sort\ xs$
by *simp*

end
theory *List-Lexorder-NS*
imports
../util/Sorting-Util
../util/List-Slice
../order/List-Permutation-Util

begin

18 General Non-standard Lexicographical Comparison

This section is based on the *lexord* classical lexicographical definition in the the List library but accounts for a variant of lexicographic order defined below that we rely on for verifying sais. The main difference is that this ordering preferences the original string over its prefix. For example, "aaa" is less than "aa", which in turn is less than "a".

definition *nslexord* :: $('a \times 'a)\ set \Rightarrow ('a\ list \times 'a\ list)\ set$ **where**
$nslexord\ r = \{(x,y).\ (\exists\, a\ v.\ x = y\ @\ a\ \#\ v) \lor$
$(\exists\, u\ a\ b\ v\ w.\ (a,\ b) \in r \land x = u\ @\ a\ \#\ v \land y = u\ @\ b\ \#\ w)\}$

definition *nslexordp* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a\ list \Rightarrow 'a\ list \Rightarrow bool$
where
$nslexordp\ cmp\ xs\ ys \longleftrightarrow$
$(\exists\, b\ c\ as\ bs\ cs.\ xs = as\ @\ b\ \#\ bs \land ys = as\ @\ c\ \#\ cs \land cmp\ b\ c) \lor$
$(\exists\, c\ cs.\ xs = ys\ @\ c\ \#\ cs)$

lemma *nslexord-eq-nslexordp*:
$(xs,\ ys) \in nslexord\ \{(x,\ y).\ cmp\ x\ y\} \longleftrightarrow nslexordp\ cmp\ xs\ ys$
$(xs,\ ys) \in nslexord\ r \longleftrightarrow nslexordp\ (\lambda x\ y.\ (x,\ y) \in r)\ xs\ ys$
by (*clarsimp simp*: *nslexord-def nslexordp-def*; *blast*)+

definition *nslexordeqp* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a\ list \Rightarrow 'a\ list \Rightarrow bool$
where
$nslexordeqp\ cmp\ xs\ ys \longleftrightarrow nslexordp\ cmp\ xs\ ys \lor (xs = ys)$

18.1 Intro and Elimination

lemma *nslexordpI1*:
$\exists\, b\ c\ as\ bs\ cs.\ xs = as\ @\ b\ \#\ bs \land ys = as\ @\ c\ \#\ cs \land cmp\ b\ c \Longrightarrow nslexordp\ cmp\ xs\ ys$
by (*simp add*: *nslexordp-def*)

lemma *nslexordpI2*:
$\exists\, c\ cs.\ xs = ys\ @\ c\ \#\ cs \Longrightarrow nslexordp\ cmp\ xs\ ys$
by (*simp add*: *nslexordp-def*)

lemma *nslexordpE*:
$nslexordp\ cmp\ xs\ ys \Longrightarrow$
$(\exists\, b\ c\ as\ bs\ cs.\ xs = as\ @\ b\ \#\ bs \land ys = as\ @\ c\ \#\ cs \land cmp\ b\ c) \lor$
$(\exists\, c\ cs.\ xs = ys\ @\ c\ \#\ cs)$
by (*simp add*: *nslexordp-def*)

lemma *nslexordp-imp-eq*:
$nslexordp\ cmp\ xs\ ys \Longrightarrow nslexordeqp\ cmp\ xs\ ys$
by (*simp add*: *nslexordeqp-def*)

lemma *nslexordeqp-imp-eq-or-less*:
$nslexordeqp\ cmp\ xs\ ys \Longrightarrow xs = ys \lor nslexordp\ cmp\ xs\ ys$
using *nslexordeqp-def* **by** *auto*

18.2 Simplification

lemma *nslexord-Nil-left*[*simp*]: $([], y) \notin nslexord\ r$
by (*unfold nslexord-def*, *induct y*, *auto*)

lemma *nslexord-Nil-right*[*simp*]: $(y, []) \in nslexord\ r = (\exists\, a\ x.\ y = a\ \#\ x)$
by (*unfold nslexord-def*, *induct y*, *auto*)

lemma *nslexord-cons-cons*[*simp*]:
$(a\ \#\ x, b\ \#\ y) \in nslexord\ r \longleftrightarrow (a, b) \in r \lor (a = b \land (x, y) \in nslexord\ r)$ (**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then show *?rhs*
apply (*simp add*: *nslexord-def*)
apply (*metis hd-append list.sel*(*1*) *list.sel*(*3*) *tl-append2*)
done
qed (*auto simp add*: *nslexord-def*; (*blast* | *meson Cons-eq-appendI*))

lemma *nslexordp-cons-cons*[*simp*]:
$nslexordp\ r\ (a\ \#\ x)\ (b\ \#\ y) \longleftrightarrow r\ a\ b \lor (a = b \land nslexordp\ r\ x\ y)$
by (*metis mem-Collect-eq nslexord-cons-cons nslexord-eq-nslexordp*(*1*) *prod.simps*(*2*))

lemmas *nslexord-simps* = *nslexord-Nil-left nslexord-Nil-right nslexord-cons-cons*

lemma *nslexord-same-pref-iff*:
$(xs\ @\ ys, xs\ @\ zs) \in nslexord\ r \longleftrightarrow (\exists\, x \in set\ xs.\ (x, x) \in r) \lor (ys, zs) \in nslexord\ r$
by(*induction xs*) *auto*

lemma *nslexord-same-pref-if-irrefl*[*simp*]:

$irrefl\ r \Longrightarrow (xs\ @\ ys,\ xs\ @\ zs) \in nslexord\ r \longleftrightarrow (ys,\ zs) \in nslexord\ r$
by (*simp add*: *irrefl-def nslexord-same-pref-iff*)

lemma *nslexord-append-leftI*:
$\exists\, b\ z.\ y = b\ \#\ z \Longrightarrow (x\ @\ y,\ x) \in nslexord\ r$
by (*simp add*: *nslexordpI2 nslexord-eq-nslexordp*(*2*))

lemma *nslexord-append-left-rightI*:
$(a\ ,b) \in r \Longrightarrow (u\ @\ a\ \#\ x,\ u\ @\ b\ \#\ y) \in nslexord\ r$
by (*simp add*: *nslexord-same-pref-iff*)

lemma *nslexord-append-rightI*:
$(u,\ v) \in nslexord\ r \Longrightarrow (x\ @\ u,\ x\ @\ v) \in nslexord\ r$
by (*simp add*: *nslexord-same-pref-iff*)

lemma *nslexord-append-rightD*:
$[\![(x\ @\ u,\ x\ @\ v) \in nslexord\ r;\ (\forall\, a.\ (a,a) \notin r)\ ]\!] \Longrightarrow (u,v) \in nslexord\ r$
by (*simp add*: *nslexord-same-pref-iff*)

— nslexord is extension of partial ordering List.lex
lemma *nslexord-lex*:
$(x,y) \in lex\ r = ((x,y) \in nslexord\ r \wedge length\ x = length\ y)$
proof (*induction x arbitrary*: *y*)
case (*Cons a x y*) **then show** *?case*
by (*cases y*) (*force+*)
qed *auto*

18.3 Recursive version

fun *nslexordrec* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a\ list \Rightarrow 'a\ list \Rightarrow bool$
where
nslexordrec P [] - = *False* |
nslexordrec P - [] = *True* |
nslexordrec P (*x#xs*) (*y#ys*) = (*if P x y then True else if x = y then nslexordrec P xs ys else False*)

lemma *nslexordp-eq-nslexordrec*:
nslexordp cmp xs ys ⟷ *nslexordrec cmp xs ys*
proof (*induct xs arbitrary*: *ys*)
case *Nil*
then show *?case*
by (*simp add*: *nslexordp-def*)
next
case (*Cons a xs*)
note *IH* = *this*

then show *?case*
proof (*cases ys*)
case *Nil*

then show *?thesis*
by (*simp add*: *nslexordp-def*)
next
case (*Cons b zs*)
note $P = IH[of\ zs]$
have $cmp\ a\ b \Longrightarrow\ ?thesis$
by (*simp add*: *Cons*)
moreover
have $[\![\neg cmp\ a\ b;\ a = b]\!] \Longrightarrow\ ?thesis$
by (*simp add*: *P Cons*)
moreover
have $[\![\neg cmp\ a\ b;\ a \neq b]\!] \Longrightarrow\ ?thesis$
by (*simp add*: *local.Cons*)
ultimately show *?thesis*
by *blast*
qed
qed

lemmas *nslexordp-induct* = *nslexordrec.induct*

18.4 Properties

Useful properties for proving things about relations, such as what type of order is satisfied

lemma *nslexord-total-on*:
assumes *total-on A R*
shows *total-on* $\{xs.\ set\ xs \subseteq A\}$ (*nslexord R*)
proof (*intro total-onI*)
fix *x y*
assume $x \in \{xs.\ set\ xs \subseteq A\}$ $y \in \{xs.\ set\ xs \subseteq A\}$ $x \neq y$
hence $set\ x \subseteq A$ $set\ y \subseteq A$ $x \neq y$
by *blast+*
then show $(x, y) \in nslexord\ R \lor (y, x) \in nslexord\ R$
proof (*induct x arbitrary*: *y*)
case *Nil*
then show *?case*
by (*metis list.exhaust nslexord-Nil-right*)
next
case (*Cons a x*)
note *IH* = *this*
then show *?case*
proof (*cases y*)
case *Nil*
then show *?thesis*
by *auto*
next
case (*Cons b z*)
then show *?thesis*
by (*metis Cons.hyps IH*(*2*−*4*) *assms list.set-intros*(*1*,*2*) *nslexord-cons-cons*

subset-code(*1*)
total-on-def)
qed
qed
qed

lemma *total-on-totalp-on-eq*:
total-on A {(*x*, *y*). *R x y*} = *totalp-on A R*
by (*simp add*: *total-on-def totalp-on-def*)

lemmas *nslexordp-totalp-on* =
nslexord-total-on[*OF total-on-totalp-on-eq*[*THEN iffD2*],
simplified nslexord-eq-nslexordp(*1*) *totalp-on-total-on-eq*[*symmetric*]]

lemma *nslexord-total*:
total r ⟹ *total* (*nslexord r*)
using *nslexord-total-on* **by** *fastforce*

lemma *nslexordp-totalp*:
totalp r ⟹ *totalp* (*nslexordp r*)
using *nslexordp-totalp-on* **by** *fastforce*

corollary *nslexord-linear*:
(∀ *a b*. (*a*,*b*) ∈ *r* ∨ *a* = *b* ∨ (*b*,*a*) ∈ *r*) ⟹ (*x*,*y*) ∈ *nslexord r* ∨ *x* = *y* ∨ (*y*,*x*) ∈ *nslexord r*
using *nslexord-total* **by** (*metis UNIV-I total-on-def*)

lemma *nslexord-irrefl-on*:
assumes *irrefl-on A R*
shows *irrefl-on* {*xs*. *set xs* ⊆ *A*} (*nslexord R*)
proof (*intro irrefl-onI*)
fix *x*
assume *x* ∈ {*xs*. *set xs* ⊆ *A*}
hence *set x* ⊆ *A*
by *blast*
then show (*x*, *x*) ∉ *nslexord R*
proof (*induct x*)
case *Nil*
then show *?case*
by *auto*
next
case (*Cons a x*)
then show *?case*
by (*meson assms irrefl-onD list.set-intros*(*1*) *nslexord-cons-cons set-subset-Cons subset-code*(*1*))
qed
qed

lemma *irrefl-on-irreflp-on-eq*:

```
  irrefl-on A {(x, y). R x y} = irreflp-on A R
  by (simp add: irrefl-on-def irreflp-on-def)

lemmas nslexordp-irreflp-on =
   nslexord-irrefl-on[OF irrefl-on-irreflp-on-eq[THEN iffD2],
                simplified nslexord-eq-nslexordp(1) irreflp-on-irrefl-on-eq[symmetric]]

lemma nslexord-irreflexive:
  ∀x. (x,x) ∉ r ⟹ (xs,xs) ∉ nslexord r
  by (metis lex-take-index nslexord-lex)

lemma nslexord-irrefl:
  irrefl R ⟹ irrefl (nslexord R)
  by (simp add: irrefl-def nslexord-irreflexive)

lemma nslexordp-irreflp:
  assumes irreflp R
  shows irreflp (nslexordp R)
  using assms nslexordp-irreflp-on by force

lemma asym-on-asymp-on-eq:
  asym-on A {(x, y). R x y} = asymp-on A R
  by (simp add: asym-on-def asymp-on-def)

lemma nslexord-asym-on:
  assumes asym-on A R
  shows asym-on {xs. set xs ⊆ A} (nslexord R)
proof (intro asym-onI)
  fix x y
  assume x ∈ {xs. set xs ⊆ A} y ∈ {xs. set xs ⊆ A} (x, y) ∈ nslexord R
  hence set x ⊆ A set y ⊆ A (x, y) ∈ nslexord R
    by blast+
  then show (y, x) ∉ nslexord R
  proof (induct x arbitrary: y)
    case Nil
    then show ?case
      by force
  next
    case (Cons a x)
    note IH = this
    then show ?case
    proof (cases y)
      case Nil
      then show ?thesis
        by simp
    next
      case (Cons b z)
      hence (a # x, b # z) ∈ nslexord R
        using IH(4) by blast
```

with *nslexord-cons-cons*[*of a x b z R*]
have $(a, b) \in R \vee a = b \wedge (x, z) \in$ *nslexord R*
by *blast*
moreover
have $(a, b) \in R \Longrightarrow$ *?thesis*
by (*metis IH*(*2*,*3*) *assms asym-onD list.set-intros*(*1*) *local.Cons nslexord-cons-cons*
subset-code(*1*))
moreover
have $a = b \wedge (x, z) \in$ *nslexord R* $\Longrightarrow$ *?thesis*
using *Cons.hyps IH*(*2*,*3*) *calculation*(*2*) *local.Cons* **by** *auto*
ultimately show *?thesis*
by *blast*
qed
qed
qed

lemmas *nslexordp-asymp-on* =
nslexord-asym-on[*OF asym-on-asymp-on-eq*[*THEN iffD2*],
simplified nslexord-eq-nslexordp(*1*) *asymp-on-asym-on-eq*[*symmetric*]]

lemma *nslexord-asym*:
assumes *asym R*
shows *asym* (*nslexord R*)
using *assms nslexord-asym-on* **by** *force*

lemma *nslexordp-asymp*:
assumes *asymp R*
shows *asymp* (*nslexordp R*)
using *assms nslexordp-asymp-on* **by** *force*

lemma *nslexord-asymmetric*:
assumes *asym R* $(a, b) \in$ *nslexord R*
shows $(b, a) \notin$ *nslexord R*
by (*simp add*: *assms asymD nslexord-asym*)

lemma *trans-on-transp-on-eq*:
trans-on A $\{(x, y).\ R\ x\ y\}$ = *transp-on A R*
by (*simp add*: *trans-on-def transp-on-def*)

lemma *nslexord-trans-on*:
assumes *trans-on A R*
shows *trans-on* $\{xs.\ set\ xs \subseteq A\}$ (*nslexord R*)
proof (*intro trans-onI*)
fix *x y z*
assume $x \in \{xs.\ set\ xs \subseteq A\}$ $y \in \{xs.\ set\ xs \subseteq A\}$ $z \in \{xs.\ set\ xs \subseteq A\}$
$(x, y) \in$ *nslexord R* $(y, z) \in$ *nslexord R*
hence *set* $x \subseteq A$ *set* $y \subseteq A$ *set* $z \subseteq A$ $(x, y) \in$ *nslexord R* $(y, z) \in$ *nslexord R*
by *blast+*

```
then show (x, z) ∈ nslexord R
proof (induct x arbitrary: y z)
  case Nil
  then show ?case
    by simp
next
  case (Cons a x)
  note IH = this
  then show ?case
  proof (cases y)
    case Nil
    then show ?thesis
      using IH(6) by auto
  next
    case (Cons b y')
    hence (a # x, b # y') ∈ nslexord R
      using IH(5) by blast
    with nslexord-cons-cons[of a x b y' R]
    have P: (a, b) ∈ R ∨ a = b ∧ (x, y') ∈ nslexord R
      by blast
    then show ?thesis
    proof (cases z)
      case Nil
      then show ?thesis
        by simp
    next
      case (Cons c z')
      hence (b # y', c # z') ∈ nslexord R
        using IH(6) ‹y = b # y'› by auto
      with nslexord-cons-cons[of b y' c z' R]
      have (b, c) ∈ R ∨ b = c ∧ (y', z') ∈ nslexord R
        by blast
      moreover
      have a ∈ A set x ⊆ A
        using IH(2) by auto
      moreover
      have b ∈ A set y' ⊆ A
        using IH(3) ‹y = b # y'› by force+
      moreover
      have c ∈ A set z' ⊆ A
        using IH(4) ‹z = c # z'› by force+
      moreover
      from P
      have (b, c) ∈ R ⟹ ?thesis
      by (metis assms calculation(2,4,6) local.Cons nslexord-cons-cons trans-onD)
      moreover
      from P
      have b = c ∧ (y', z') ∈ nslexord R ⟹ ?thesis
        by (metis Cons.hyps calculation(3,5,7) local.Cons nslexord-cons-cons)
```

ultimately show *?thesis*
by *blast*
qed
qed
qed
qed

lemmas *nslexordp-transp-on* =
nslexord-trans-on[*OF trans-on-transp-on-eq*[*THEN iffD2*],
simplified nslexord-eq-nslexordp(*1*) *transp-on-trans-on-eq*[*symmetric*]]

lemma *nslexord-trans*:
assumes *trans R*
shows *trans* (*nslexord R*)
using *assms nslexord-trans-on* **by** *force*

lemma *nslexordp-transp*:
assumes *transp R*
shows *transp* (*nslexordp R*)
using *assms nslexordp-transp-on* **by** *force*

18.5 Monotonicity

Properties about monotonicity

lemma *monotone-on-nslexordp*:
assumes *monotone-on A orda ordb f*
shows *monotone-on* $\{xs.\ set\ xs \subseteq A\}$ (*nslexordp orda*) (*nslexordp ordb*) (*map f*)
proof (*rule monotone-onI*)
fix *x y*
assume $x \in \{xs.\ set\ xs \subseteq A\}$ $y \in \{xs.\ set\ xs \subseteq A\}$ *nslexordp orda x y*
hence $set\ x \subseteq A$ $set\ y \subseteq A$
by *blast+*

let $?c1 = \exists\, b\ c\ as\ bs\ cs.\ x = as\ @\ b\ \#\ bs \wedge y = as\ @\ c\ \#\ cs \wedge orda\ b\ c$
and $?c2 = \exists\, c\ cs.\ x = y\ @\ c\ \#\ cs$

let $?g$ = *nslexordp ordb* (*map f x*) (*map f y*)
from *nslexordpE*[*OF* ‹*nslexordp orda x y*›]
have $?c1 \vee ?c2$ **.**
moreover
have $?c2 \Longrightarrow ?g$
using *nslexordpI2* **by** *fastforce*
moreover
have $?c1 \Longrightarrow ?g$
proof (*rule nslexordpI1*)
assume *?c1*
then obtain *b c as bs cs* **where**
$x = as\ @\ b\ \#\ bs$
$y = as\ @\ c\ \#\ cs$

orda b c
by *blast*
moreover
have *map f x* = *map f as* @ *f b* # *map f bs*
by (*simp add*: *calculation*(*1*))
moreover
have *map f y* = *map f as* @ *f c* # *map f cs*
by (*simp add*: *calculation*(*2*))
moreover
have *ordb* (*f b*) (*f c*)
by (*metis* ‹*set x* ⊆ *A*› ‹*set y* ⊆ *A*› *assms calculation*(*1*−*3*) *in-set-conv-decomp monotone-on-def*
subset-code(*1*))
ultimately show ∃ *b c as bs cs*. *map f x* = *as* @ *b* # *bs* ∧ *map f y* = *as* @ *c* # *cs* ∧ *ordb b c*
by *blast*
qed
ultimately show *nslexordp ordb* (*map f x*) (*map f y*)
by *blast*
qed

lemma *monotone-on-bij-betw-inv-nslexordp*:
assumes *monotone-on A orda ordb f*
and *asymp-on A orda*
and *totalp-on A orda*
and *asymp-on B ordb*
and *totalp-on B ordb*
and *bij-betw f A B*
and *bij-betw g B A*
and *inverses-on f g A B*
shows *monotone-on* {*xs*. *set xs* ⊆ *B*} (*nslexordp ordb*) (*nslexordp orda*) (*map g*)
by (*metis assms monotone-on-bij-betw-inv monotone-on-nslexordp*)

lemma *monotone-on-bij-betw-nslexordp*:
assumes *monotone-on A orda ordb f*
and *asymp-on A orda*
and *totalp-on A orda*
and *asymp-on B ordb*
and *totalp-on B ordb*
and *bij-betw f A B*
shows ∃ *g*. *bij-betw* (*map g*) {*xs*. *set xs* ⊆ *B*} {*xs*. *set xs* ⊆ *A*} ∧
inverses-on (*map f*) (*map g*) {*xs*. *set xs* ⊆ *A*} {*xs*. *set xs* ⊆ *B*} ∧
monotone-on {*xs*. *set xs* ⊆ *B*} (*nslexordp ordb*) (*nslexordp orda*) (*map g*)
by (*metis assms bij-betw-inv-alt bij-betw-map inverses-on-map monotone-on-bij-betw-inv-nslexordp*)

lemma *monotone-on-iff-nslexordp*:
assumes *monotone-on A orda ordb f*
and *asymp-on A orda*
and *totalp-on A orda*

and *asymp-on B ordb*
and *totalp-on B ordb*
and *bij-betw f A B*
and *set xs* ⊆ *A*
and *set ys* ⊆ *A*
shows *nslexordp orda xs ys* ⟷ *nslexordp ordb* (*map f xs*) (*map f ys*)
proof
assume *nslexordp orda xs ys*
with *monotone-onD*[*OF monotone-on-nslexordp*[*OF assms*(*1*)], *simplified*, *OF assms*(*7*,*8*)]
show *nslexordp ordb* (*map f xs*) (*map f ys*)
by *blast*
next
assume *A*: *nslexordp ordb* (*map f xs*) (*map f ys*)

from *monotone-on-bij-betw-nslexordp*[*OF assms*(*1*−*6*)]
obtain *g* **where** *P*:
bij-betw (*map g*) {*xs*. *set xs* ⊆ *B*} {*xs*. *set xs* ⊆ *A*}
inverses-on (*map f*) (*map g*) {*xs*. *set xs* ⊆ *A*} {*xs*. *set xs* ⊆ *B*}
monotone-on {*xs*. *set xs* ⊆ *B*} (*nslexordp ordb*) (*nslexordp orda*) (*map g*)
by *blast*

have *Q*: *set* (*map f xs*) ⊆ *B*
using *assms*(*6*,*7*) *bij-betw-imp-surj-on* **by** *fastforce*

have *R*: *set* (*map f ys*) ⊆ *B*
using *assms*(*6*,*8*) *bij-betw-imp-surj-on* **by** *fastforce*

from *monotone-onD*[*OF P*(*3*), *simplified*, *OF Q R A*]
show *nslexordp orda xs ys*
by (*metis P*(*2*) *assms*(*7*,*8*) *inverses-on-def mem-Collect-eq*)
qed

18.6 Other

lemma *nslexordp-cons1-exE*:
assumes *nslexordp cmp xs* (*x* # *xs*)
shows ∃ *a as bs*. *x* # *xs* = *as* @ *x* # *a* # *bs* ∧ *cmp a x* ∧ (∀ *b* ∈ *set as*. *b* = *x*)
using *assms*
proof (*induct xs arbitrary*: *x*)
case *Nil*
then show *?case*
using *nslexord-Nil-left nslexord-eq-nslexordp*(*1*) **by** *blast*
next
case (*Cons a xs*)
note *IH* = *this*

have *cmp a x* ⟹ *?case*
by *fastforce*

moreover
have $\llbracket\neg cmp\ a\ x;\ a \neq x\rrbracket \Longrightarrow\ ?case$
using $IH(2)$ **by** *force*
moreover
have $\llbracket\neg cmp\ a\ x;\ a = x\rrbracket \Longrightarrow\ ?case$
proof −
assume $\neg cmp\ a\ x\ a = x$
with $IH(2)$
have *nslexordp cmp xs* $(x\ \#\ xs)$
by *simp*
with $IH(1)[OF\ \text{-}]\ \langle a = x\rangle$
have $\exists\,k\ as\ bs.\ a\ \#\ xs = as\ @\ x\ \#\ k\ \#\ bs \wedge cmp\ k\ x \wedge (\forall\,b{\in}set\ as.\ b = x)$
by *simp*
then obtain *k as bs* **where** *P*:
$a\ \#\ xs = as\ @\ x\ \#\ k\ \#\ bs\quad cmp\ k\ x\ \forall\,b{\in}set\ as.\ b = x$
by *blast*
then show *?case*
by (*metis Cons-eq-appendI set-ConsD*)
qed
ultimately show *?case*
by *blast*
qed

lemma *nslexordp-cons2-exE*:
assumes *nslexordp cmp* $(x\ \#\ xs)$ *xs*
shows $(\forall\,k \in set\ xs.\ k = x) \vee (\exists\,a\ as\ bs.\ x\ \#\ xs = as\ @\ x\ \#\ a\ \#\ bs \wedge cmp\ x\ a$
$\wedge\ (\forall\,b \in set\ as.\ b = x))$
using *assms*
proof (*induct xs arbitrary*: *x*)
case *Nil*
then show *?case*
by *simp*
next
case (*Cons a xs*)
note *IH* = *this*

have $cmp\ x\ a \Longrightarrow\ ?case$
by (*metis append-Nil empty-iff empty-set*)
moreover
have $\llbracket\neg cmp\ x\ a;\ a = x\rrbracket \Longrightarrow\ ?case$
proof −
assume $\neg cmp\ x\ a\ a = x$
with $IH(2)$
have *nslexordp cmp* $(x\ \#\ xs)$ *xs*
by *simp*
with $IH(1)[of\ x]\ \langle a = x\rangle$
have $(\forall\,k{\in}set\ xs.\ k = x) \vee$
$(\exists\,k\ as\ bs.\ a\ \#\ xs = as\ @\ x\ \#\ k\ \#\ bs \wedge cmp\ x\ k \wedge (\forall\,b{\in}set\ as.\ b = x))$
by *simp*

then show *?thesis*
proof
assume $\forall k \in set\ xs.\ k = x$
then show *?thesis*
by $(simp\ add: \langle a = x \rangle)$
next
assume $\exists k\ as\ bs.\ a \# xs = as\ @\ x \# k \# bs \wedge cmp\ x\ k \wedge (\forall b \in set\ as.\ b = x)$
then obtain $k\ as\ bs$ **where** P:
$a \# xs = as\ @\ x \# k \# bs\ cmp\ x\ k\ \forall b \in set\ as.\ b = x$
by *blast*
then show *?thesis*
by (*metis Cons-eq-appendI set-ConsD*)
qed
qed
moreover
have $[\![\neg cmp\ x\ a;\ a \neq x]\!] \Longrightarrow ?case$
using *Cons.prems* **by** *auto*

ultimately show *?case*
by *blast*
qed

19 Order definitions on lists of linorder elements

definition *list-less-ns* :: $('a :: linorder)\ list \Rightarrow 'a\ list \Rightarrow bool$
where
list-less-ns xs ys =
$(\exists n.\ n \leq length\ xs \wedge n \leq length\ ys \wedge$
$(\forall i < n.\ xs\ !\ i = ys\ !\ i) \wedge$
$(length\ ys = n \longrightarrow n < length\ xs) \wedge$
$(length\ ys \neq n \longrightarrow length\ xs \neq n \wedge xs\ !\ n < ys\ !\ n))$

definition *list-less-eq-ns* :: $('a :: linorder)\ list \Rightarrow 'a\ list \Rightarrow bool$
where
list-less-eq-ns xs ys =
$(\exists n.\ n \leq length\ xs \wedge n \leq length\ ys \wedge$
$(\forall i < n.\ xs\ !\ i = ys\ !\ i) \wedge$
$(length\ ys \neq n \longrightarrow length\ xs \neq n \wedge xs\ !\ n < ys\ !\ n))$

— Alternative definition

definition *list-less-ns-ex* :: $('a :: linorder)\ list \Rightarrow ('a :: linorder)\ list \Rightarrow bool$
where
list-less-ns-ex xs ys $\longleftrightarrow$
$(\exists b\ c\ as\ bs\ cs.\ xs = as\ @\ b \# bs \wedge ys = as\ @\ c \# cs \wedge b < c) \vee$
$(\exists c\ cs.\ xs = ys\ @\ c \# cs)$

20 Helper list comparison theorems

lemma *list-less-ns-alt-def*:
  list-less-ns xs ys = *list-less-ns-ex xs ys*
proof
  assume *list-less-ns xs ys*
  with *list-less-ns-def*[*THEN iffD1*, *of xs ys*]
  obtain *n* **where** *P*:
    $n \leq$ *length xs* $n \leq$ *length ys* $\forall i<n.$ *xs* ! *i* = *ys* ! *i length ys* = $n \longrightarrow n <$ *length xs*
    length ys $\neq n \longrightarrow$ *length xs* $\neq n \wedge$ *xs* ! $n <$ *ys* ! *n*
    by *blast*
  show *list-less-ns-ex xs ys*
  proof (*cases length ys* = *n*)
    assume *length ys* = *n*
    then show *?thesis*
      by (*metis P*(*1*,*2*,*3*,*4*) *id-take-nth-drop list-less-ns-ex-def nth-take-lemma take-all*)
  next
    assume *length ys* $\neq$ *n*
    then show *?thesis*
      by (*metis P*(*1*,*2*,*3*,*5*) *id-take-nth-drop le-neq-implies-less list-less-ns-ex-def nth-take-lemma*)
  qed
next
  assume *list-less-ns-ex xs ys*
  hence ($\exists$ *b c as bs cs*. *xs* = *as* @ *b* # *bs* $\wedge$ *ys* = *as* @ *c* # *cs* $\wedge$ *b* < *c*) $\vee$ ($\exists$ *c cs*. *xs* = *ys* @ *c* # *cs*)
    using *list-less-ns-ex-def* **by** *blast*
  then show *list-less-ns xs ys*
  proof
    assume $\exists$ *b c as bs cs*. *xs* = *as* @ *b* # *bs* $\wedge$ *ys* = *as* @ *c* # *cs* $\wedge$ *b* < *c*
    then obtain *b c as bs cs* **where** *P*:
      xs = *as* @ *b* # *bs ys* = *as* @ *c* # *cs b* < *c*
      by *blast*
    hence *length as* < *length xs*
      by *simp*
    moreover
    have *length as* < *length ys*
      by (*simp add*: *P*(*2*))
    moreover
    have $\forall$ *i* < *length as*. *xs* ! *i* = *ys* ! *i*
      by (*simp add*: *P*(*1*) *P*(*2*) *nth-append*)
    moreover
    have *xs* ! *length as* < *ys* ! *length as*
      by (*simp add*: *P*(*1*) *P*(*2*) *P*(*3*))
    ultimately show *list-less-ns xs ys*
      using *list-less-ns-def*[*THEN iffD2*, *OF exI*, *of length as xs ys*] **by** *simp*
  next

assume ∃ *c cs*. *xs* = *ys* @ *c* # *cs*
then obtain *c cs* **where**
xs = *ys* @ *c* # *cs*
by *blast*
hence *length ys* < *length xs*
by *simp*
then show *list-less-ns xs ys*
using *list-less-ns-def*[*THEN iffD2*, *OF exI*, *of length ys xs ys*]
by (*simp add*: ‹*xs* = *ys* @ *c* # *cs*› *nth-append*)
qed
qed

lemma *nslexordp-eq-list-less-ns-ex*:
nslexordp (<) = *list-less-ns-ex*
by (*clarsimp simp*: *fun-eq-iff nslexordp-def list-less-ns-ex-def*)

lemma *nslexordp-eq-list-less-ns-ex-apply*:
nslexordp (<) *x y* = *list-less-ns-ex x y*
by (*simp add*: *nslexordp-eq-list-less-ns-ex*)

lemma *nslexordp-eq-list-less-ns*:
nslexordp (<) = *list-less-ns*
by (*clarsimp simp*: *fun-eq-iff list-less-ns-alt-def nslexordp-eq-list-less-ns-ex*)

lemma *nslexordp-eq-list-less-ns-app*:
nslexordp (<) *x y* = *list-less-ns x y*
by (*simp add*: *nslexordp-eq-list-less-ns*)

lemma *nslexordeqp-eq-list-less-eq-ns-apply*:
nslexordeqp (<) *x y* = *list-less-eq-ns x y*
proof (*cases x* = *y*)
assume *x* = *y*
then show *?thesis*
by (*simp add*: *list-less-eq-ns-def nslexordeqp-def*)
next
assume *x* ≠ *y*
hence *nslexordeqp* (<) *x y* = *nslexordp* (<) *x y*
by (*simp add*: *nslexordeqp-def*)
moreover
have *nslexordp* (<) *x y* = *list-less-ns x y*
by (*simp add*: *nslexordp-eq-list-less-ns*)
moreover
have *list-less-eq-ns x y* = *list-less-ns x y*
unfolding *list-less-eq-ns-def list-less-ns-def*
proof (*intro iffI*; *elim exE conjE*)
fix *n*
assume $n \le$ *length x* $n \le$ *length y* ∀ *i*<*n*. *x* ! *i* = *y* ! *i*
length y ≠ *n* ⟶ *length x* ≠ *n* ∧ *x* ! *n* < *y* ! *n*
then show ∃ *n*≤*length x*. $n \le$ *length y* ∧ (∀ *i*<*n*. *x* ! *i* = *y* ! *i*) ∧

$(\mathit{length}\ y = n \longrightarrow n < \mathit{length}\ x) \wedge$
$(\mathit{length}\ y \neq n \longrightarrow \mathit{length}\ x \neq n \wedge x\ !\ n < y\ !\ n)$
by (*metis* ‹$x \neq y$› *le-eq-less-or-eq nth-equalityI*)
next
fix n
assume $n \leq \mathit{length}\ x\ n \leq \mathit{length}\ y\ \forall\, i{<}n.\ x\ !\ i = y\ !\ i\ \mathit{length}\ y = n \longrightarrow n < \mathit{length}\ x$
$\mathit{length}\ y \neq n \longrightarrow \mathit{length}\ x \neq n \wedge x\ !\ n < y\ !\ n$
then show $\exists\, n{\leq}\mathit{length}\ x.\ n \leq \mathit{length}\ y \wedge (\forall\, i{<}n.\ x\ !\ i = y\ !\ i) \wedge$
$(\mathit{length}\ y \neq n \longrightarrow \mathit{length}\ x \neq n \wedge x\ !\ n < y\ !\ n)$
by *blast*
qed
ultimately show *?thesis*
by *blast*
qed

21 *list-less-ns* helpers

lemma *list-less-ns-cons-same*:
list-less-ns $(a \# xs)$ $(a \# ys)$ = *list-less-ns xs ys*
by (*metis nslexordp-cons-cons nslexordp-eq-list-less-ns order-less-irrefl*)

lemma *list-less-ns-cons-diff*:
$a < b \Longrightarrow$ *list-less-ns* $(a \# xs)$ $(b \# ys)$
using *list-less-ns-def* **by** *fastforce*

lemma *list-less-ns-cons*:
list-less-ns $(a \# xs)$ $(b \# ys)$ = $(a \leq b \wedge (a = b \longrightarrow$ *list-less-ns xs ys*$))$
by (*metis length-Cons list-less-ns-cons-diff list-less-ns-cons-same list-less-ns-def nat.simps(3)*
not-less-iff-gr-or-eq not-less-zero nth-Cons-0 order.strict-iff-order
order-class.order-eq-iff)

lemma *list-less-eq-ns-cons-same*:
list-less-eq-ns $(a \# xs)$ $(a \# ys)$ = *list-less-eq-ns xs ys*
by (*metis list.inject list-less-ns-cons-same nslexordeqp-def nslexordeqp-eq-list-less-eq-ns-apply*
nslexordp-eq-list-less-ns-app)

lemma *list-less-eq-ns-cons*:
list-less-eq-ns $(a \# xs)$ $(b \# ys)$ = $(a \leq b \wedge (a = b \longrightarrow$ *list-less-eq-ns xs ys*$))$
by (*metis list.inject list-less-ns-cons nle-le nslexordeqp-def*
nslexordeqp-eq-list-less-eq-ns-apply nslexordp-eq-list-less-ns)

lemma *list-less-ns-hd-same*:
$[\![hd\ xs = hd\ ys;\ xs \neq [\,];\ ys \neq [\,]]\!] \Longrightarrow$ *list-less-ns xs ys* = *list-less-ns* (*tl xs*) (*tl ys*)
by (*metis list.collapse list-less-ns-cons-same*)

lemma *list-less-ns-recurse*:

$[\![xs \neq []; ys \neq []]\!] \Longrightarrow$
($hd\ xs = hd\ ys \longrightarrow$ *list-less-ns xs ys* = *list-less-ns* ($tl\ xs$) ($tl\ ys$)) $\wedge$
($hd\ xs \neq hd\ ys \longrightarrow$ *list-less-ns xs ys* = ($hd\ xs < hd\ ys$))
by (*metis list.collapse list-less-ns-cons list-less-ns-hd-same nless-le*)

lemma *list-less-ns-nil*:
$xs \neq [] \Longrightarrow$ *list-less-ns xs* []
using *list-less-ns-def* **by** *auto*

lemma *list-less-ns-app*:
$bs \neq [] \Longrightarrow$ *list-less-ns* (*as* @ *bs*) *as*
by (*metis list.collapse nslexordpI2 nslexordp-eq-list-less-ns*)

22 Lists of linorder elements are linorders with a bottom element

lemma *list-less-ns-imp-less-eq-not-less-eq*:
list-less-ns x y $\Longrightarrow$ (*list-less-eq-ns x y* $\wedge \neg$ *list-less-eq-ns y x*)
apply (*clarsimp simp*: *nslexordp-eq-list-less-ns*[*symmetric*]
nslexordeqp-eq-list-less-eq-ns-apply[*symmetric*]
nslexordeqp-def
nslexord-eq-nslexordp(*1*)[*symmetric*])
apply (*rule conjI*)
apply (*erule nslexord-asymmetric*[*rotated*], *fastforce*)
by (*metis Product-Type.Collect-case-prodD fst-conv nslexord-irreflexive order-less-irrefl snd-conv*)

lemma *list-less-eq-ns-not-less-eq-imp-less*:
list-less-eq-ns x y $\wedge \neg$ *list-less-eq-ns y x* $\Longrightarrow$ *list-less-ns x y*
by (*metis nslexordeqp-eq-list-less-eq-ns-apply nslexordeqp-imp-eq-or-less nslexordp-eq-list-less-ns*)

lemma *list-less-eq-ns-trans*:
$[\![$*list-less-eq-ns x y*; *list-less-eq-ns y z*$]\!] \Longrightarrow$ *list-less-eq-ns x z*
apply (*clarsimp simp*: *nslexordp-eq-list-less-ns*[*symmetric*]
nslexordeqp-eq-list-less-eq-ns-apply[*symmetric*]
nslexordeqp-def
nslexord-eq-nslexordp(*1*)[*symmetric*])
apply *safe*
apply (*erule* (*1*) *transD*[*OF nslexord-trans*, *rotated*])
by (*metis order-less-trans transp-def transp-trans*)

lemma *list-less-eq-ns-anti-sym*:
$[\![$*list-less-eq-ns x y*; *list-less-eq-ns y x*$]\!] \Longrightarrow x = y$
by (*metis list-less-ns-imp-less-eq-not-less-eq nslexordeqp-eq-list-less-eq-ns-apply nslexordeqp-imp-eq-or-less nslexordp-eq-list-less-ns*)

lemma *list-less-eq-ns-linear*:
  list-less-eq-ns x y ∨ *list-less-eq-ns y x*
  apply (*simp add*: *nslexordp-eq-list-less-ns*[*symmetric*]
          nslexordeqp-eq-list-less-eq-ns-apply[*symmetric*]
          nslexordeqp-def
          nslexord-eq-nslexordp(*1*)[*symmetric*])
  by (*metis case-prodI linorder-neqE mem-Collect-eq nslexord-linear*)

interpretation *ordlistns*: *linorder list-less-eq-ns list-less-ns*
proof
  fix *x y z* :: *'a list*
  show *list-less-ns x y* = (*list-less-eq-ns x y* ∧ ¬ *list-less-eq-ns y x*)
    using *list-less-ns-imp-less-eq-not-less-eq list-less-eq-ns-not-less-eq-imp-less*
    by *blast*
  show *list-less-eq-ns x x*
    unfolding *list-less-eq-ns-def*
    by *simp*
  show ⟦*list-less-eq-ns x y*; *list-less-eq-ns y z*⟧ ⟹ *list-less-eq-ns x z*
    by (*rule list-less-eq-ns-trans*)
  show ⟦*list-less-eq-ns x y*; *list-less-eq-ns y x*⟧ ⟹ *x* = *y*
    by (*rule list-less-eq-ns-anti-sym*)
  show *list-less-eq-ns x y* ∨ *list-less-eq-ns y x*
    by (*rule list-less-eq-ns-linear*)
qed

interpretation *ordlistns*: *order-top list-less-eq-ns list-less-ns* []
proof
  fix *a* :: *'a list*
  show *list-less-eq-ns a* []
    unfolding *list-less-eq-ns-def*
    by *auto*
qed

23 Recursive Definition

fun *lt-ns* :: (*'a* :: *linorder*) *list* ⇒ *'a list* ⇒ *bool*
  where
lt-ns [] [] = *False* |
lt-ns [] - = *False* |
lt-ns - [] = *True* |
lt-ns (*a* # *as*) (*b* # *bs*) =
  (*if a* < *b then True*
   else if a > *b then False*
   else lt-ns as bs)

lemma *list-less-ns-lt-ns*:
  list-less-ns xs ys = *lt-ns xs ys*
  apply (*induct rule*: *lt-ns.induct*)
    apply *simp*

apply *simp*
apply (*simp add*: *list-less-ns-nil*)
apply (*simp add*: *list-less-ns-cons*)
apply (*safe*; *simp*)
done

24 *list-less-ns-ex* helpers

lemma *list-less-ns-exI1*:
$\exists\, b\ c\ as\ bs\ cs.\ xs = as\ @\ b\ \#\ bs \wedge ys = as\ @\ c\ \#\ cs \wedge b < c \Longrightarrow$ *list-less-ns-ex xs ys*
by (*simp add*: *list-less-ns-ex-def*)

lemma *list-less-ns-exI2*:
$\exists\, c\ cs.\ xs = ys\ @\ c\ \#\ cs \Longrightarrow$ *list-less-ns-ex xs ys*
by (*simp add*: *list-less-ns-ex-def*)

lemma *list-less-ns-exE*:
list-less-ns-ex xs ys $\Longrightarrow$
$(\exists\, b\ c\ as\ bs\ cs.\ xs = as\ @\ b\ \#\ bs \wedge ys = as\ @\ c\ \#\ cs \wedge b < c) \vee$
$(\exists\, c\ cs.\ xs = ys\ @\ c\ \#\ cs)$
by (*simp add*: *list-less-ns-ex-def*)

lemma *list-less-ns-app-same*:
list-less-ns (*as* @ *xs*) (*as* @ *ys*) = *list-less-ns xs ys*
apply (*induct as arbitrary*: *xs ys*)
apply *simp*
apply (*simp add*: *list-less-ns-cons-same*)
done

lemma *list-less-eq-ns-app-same*:
list-less-eq-ns (*as* @ *xs*) (*as* @ *ys*) = *list-less-eq-ns xs ys*
apply (*induct as arbitrary*: *xs ys*)
apply *simp*
apply (*simp add*: *list-less-eq-ns-cons-same*)
done

lemma *list-less-ns-cons1-exE*:
assumes *list-less-ns xs* (x # *xs*)
shows $\exists\, a\ as\ bs.\ x\ \#\ xs = as\ @\ x\ \#\ a\ \#\ bs \wedge x > a \wedge (\forall\, b \in set\ as.\ b = x)$
by (*metis assms nslexordp-cons1-exE nslexordp-eq-list-less-ns*)

lemma *list-less-ns-cons1-exI*:
assumes $\exists\, a\ as\ bs.\ x\ \#\ xs = as\ @\ x\ \#\ a\ \#\ bs \wedge x > a \wedge (\forall\, b \in set\ as.\ b = x)$
shows *list-less-ns-ex xs* (x # *xs*)
proof −
from *assms*
obtain *a as bs* **where**
$x\ \#\ xs = as\ @\ x\ \#\ a\ \#\ bs$

$a < x$
$\forall\, b \in set\ as.\ b = x$
by *blast*

have $as = [] \Longrightarrow ?thesis$
using ‹$a < x$› ‹$x \# xs = as @ x \# a \# bs$› *list-less-ns-alt-def list-less-ns-cons-diff*
by *fastforce*
moreover
have $\exists\, c\ cs.\ as = cs\ @\ [c] \Longrightarrow ?thesis$
proof −
assume $\exists\, c\ cs.\ as = cs\ @\ [c]$
then obtain *c cs* **where**
$as = cs\ @\ [c]$
by *blast*
with ‹$\forall\, b \in set\ as.\ b = x$›
have $c = x$
by *auto*
hence $x \# xs = cs\ @\ x \# x \# a \# bs$
by (*simp add*: ‹$as = cs\ @\ [c]$› ‹$x \# xs = as\ @\ x \# a \# bs$›)

have $\forall\, b \in set\ cs.\ b = x$
by (*simp add*: ‹$\forall\, b{\in}set\ as.\ b = x$› ‹$as = cs\ @\ [c]$›)
hence $\exists\, n.\ cs = replicate\ n\ x$
by (*meson replicate-eqI*)
then show *?thesis*
by (*metis* ‹$a < x$› ‹$x \# xs = cs\ @\ x \# x \# a \# bs$› *list-less-ns-alt-def list-less-ns-app-same*
list-less-ns-cons-diff list-less-ns-cons-same replicate-app-Cons-same)
qed
ultimately show *?thesis*
by (*meson rev-exhaust*)
qed

lemma *list-less-ns-cons2-ex*:
assumes *list-less-ns* $(x \# xs)\ xs$
shows $(\forall\, k \in set\ xs.\ k = x) \lor (\exists\, a\ as\ bs.\ x \# xs = as\ @\ x \# a \# bs \land x < a \land (\forall\, b \in set\ as.\ b = x))$
by (*metis assms nslexordp-cons2-exE nslexordp-eq-list-less-ns*)

end
theory *Valid-List*
imports *Main ../util/List-Util*
begin

25 Valid List

definition
valid-list :: ('*a* :: {*linorder*, *order-bot*}) *list* ⇒ *bool*
where

valid-list s = (*length* $s > 0 \wedge (\forall i < \textit{length } s - 1.\ s\ !\ i \neq \textit{bot}) \wedge \textit{last } s = \textit{bot})$

lemma *valid-list-ex-def*:
fixes s ::($'a$ :: {*linorder*, *order-bot*}) *list*
shows (*valid-list* s) =
$(\exists xs.\ s = xs\ @\ [\textit{bot}] \wedge$
$(\forall i < \textit{length } xs.\ xs\ !\ i \neq \textit{bot}))$
unfolding *valid-list-def*
proof *safe*
assume
$s \neq []$
$\forall i{<}\textit{length } s - 1.\ s\ !\ i \neq \textit{bot}$
last $s = \textit{bot}$
then
show $\exists xs.\ s = xs\ @\ [\textit{bot}] \wedge$
$(\forall i{<}\textit{length } xs.\ xs\ !\ i \neq \textit{bot})$
by (*metis append-butlast-last-id length-butlast nth-butlast*)
qed (*simp add*: *nth-append*)+

lemma *valid-list-iff-butlast-app-last*:
fixes s :: ($'a$ :: {*linorder*, *order-bot*}) *list*
shows *valid-list* $s \longleftrightarrow$
$s \neq [] \wedge$
$(\forall x \in \textit{set } (\textit{butlast } s).\ x \neq \textit{bot}) \wedge$
last $s = \textit{bot}$
by (*metis append-butlast-last-id butlast-snoc in-set-conv-nth*
valid-list-def valid-list-ex-def length-greater-0-conv)

lemma *valid-list-consI*:
fixes s :: ($'a$ :: {*linorder*, *order-bot*}) *list*
fixes a :: $'a$
assumes *valid-list* s
and $a \neq \textit{bot}$
shows *valid-list* (a # s)
using *assms*
by (*simp add*: *valid-list-iff-butlast-app-last*)

lemma *valid-list-consD*:
fixes s :: ($'a$ :: {*linorder*, *order-bot*}) *list*
fixes a :: $'a$
assumes *valid-list* (a # s)
assumes $s \neq []$
shows *valid-list* s
using *assms*
by (*simp add*: *valid-list-iff-butlast-app-last*)

lemma *Min-valid-list*:
fixes s :: ($'a$ :: {*linorder*, *order-bot*}) *list*
assumes *valid-list* s

shows *Min* (*set s*) = *bot*
by (*metis assms List.finite-set Min.in-idem last-in-set min-bot valid-list-iff-butlast-app-last*)

lemma *valid-list-length*:
fixes *s* :: ('*a* :: {*linorder*, *order-bot*}) *list*
assumes *valid-list s*
shows *length s* > *0*
using *assms*
by (*clarsimp simp*: *valid-list-ex-def*)

lemma *valid-list-length-ex*:
fixes *s* :: ('*a* :: {*linorder*, *order-bot*}) *list*
assumes *valid-list s*
shows ∃ *n*. *length s* = *Suc n*
using *assms*
by (*clarsimp simp*: *valid-list-ex-def*)

lemma *valid-list-not-nil*:
fixes *s* :: ('*a* :: {*linorder*, *order-bot*}) *list*
assumes *valid-list s*
shows *s* ≠ []
using *assms* **by** (*simp add*: *valid-list-def*)

lemma *valid-list-Suc-mapping*:
fixes *f* :: '*a* ⇒ *nat*
fixes *s* :: '*a list*
shows *valid-list* ((*map* (λ*x*. *Suc* (*f x*)) *s*) @ [*bot*])
proof (*induct s*)
case (*Cons a s*)
note *IH* = *this*
have *map* (λ*x*. *Suc* (*f x*)) (*a* # *s*) = (*Suc* (*f a*)) # *map* (λ*x*. *Suc* (*f x*)) *s*
by *simp*
moreover
have *Suc* (*f a*) ≠ *bot*
by (*simp add*: *bot-nat-def*)
hence *valid-list* ((*Suc* (*f a*)) # (*map* (λ*x*. *Suc* (*f x*)) *s*) @ [*bot*])
by (*simp add*: *IH valid-list-consI*)
ultimately
show *?case*
by *simp*
qed (*simp add*: *valid-list-ex-def*)

lemma *valid-list-app*:
assumes *valid-list* (*xs* @ *y* # *ys*)
shows *valid-list* (*y* # *ys*)
using *assms*
by (*induct xs*) (*simp add*: *valid-list-consD*)+

lemma *not-valid-list-app*:

assumes *valid-list* (*xs* @ *y* # *ys*)
shows ¬*valid-list xs*
using *assms*
proof (*induct xs*)
case *Nil*
then show *?case*
using *valid-list-not-nil* **by** *auto*
next
case (*Cons a xs*)
then show *?case*
by (*metis Groups.add-ac*(*2*) *Nil-is-append-conv One-nat-def add-diff-cancel-left′ append-Cons*
last-ConsL list.discI list.size(*4*) *nth-Cons-0 valid-list-consD valid-list-def*)
qed

lemma *valid-list-neqE*:
assumes *valid-list xs valid-list ys xs* ≠ *ys*
shows ∃ *x y as bs cs*. *xs* = *as* @ *x* # *bs* ∧ *ys* = *as* @ *y* # *cs* ∧ *x* ≠ *y*
proof −
note *cases* = *list-neq-fc*[*OF assms*(*3*)]
moreover
have ¬(∃ *z zs*. *xs* = *ys* @ *z* # *zs*)
using *assms*(*1*) *assms*(*2*) *not-valid-list-app* **by** *blast*
moreover
have ¬(∃ *z zs*. *ys* = *xs* @ *z* # *zs*)
using *assms*(*1*) *assms*(*2*) *not-valid-list-app* **by** *blast*
ultimately show *?thesis*
by *blast*
qed

end
theory *Valid-List-Util*
imports *List-Lexorder-Util List-Lexorder-NS Valid-List*
begin

26 Order Equivalence

lemma *valid-list-list-less-equiv-list-less-ns*:
assumes *valid-list s1*
and *valid-list s2*
shows *s1* < *s2* = *list-less-ns s1 s2*
proof
assume *s1* < *s2*
hence *s1* ≠ *s2*
by *simp*
with *valid-list-neqE*[*OF assms*(*1*,*2*)]
obtain *x y as bs cs* **where**
s1 = *as* @ *x* # *bs s2* = *as* @ *y* # *cs x* ≠ *y*
by *blast*

hence $x < y$
by (*metis* ‹$s1 < s2$› *linorder-less-linear list-less-ex order-less-imp-not-less*)
then have *list-less-ns-ex s1 s2*
using ‹$s1 = as\ @\ x\ \#\ bs$› ‹$s2 = as\ @\ y\ \#\ cs$› *list-less-ns-ex-def* **by** *fastforce*
then show *list-less-ns s1 s2*
by (*simp add*: *list-less-ns-alt-def*)
next
assume *list-less-ns s1 s2*
hence $s1 \neq s2$
by *fastforce*
with *valid-list-neqE*[*OF assms*(*1*,*2*)]
obtain *x y as bs cs* **where**
$s1 = as\ @\ x\ \#\ bs$ $s2 = as\ @\ y\ \#\ cs$ $x \neq y$
by *blast*
hence $x < y$
by (*metis* ‹*list-less-ns s1 s2*› *list.distinct*(*1*) *list.sel*(*1*) *list-less-ns-app-same list-less-ns-recurse*)
then show $s1 < s2$
using ‹$s1 = as\ @\ x\ \#\ bs$› ‹$s2 = as\ @\ y\ \#\ cs$› *list-less-ex* **by** *fastforce*
qed

lemma *valid-list-list-less-eq-equiv-list-less-eq-ns*:
assumes *valid-list s1*
and *valid-list s2*
shows $s1 \leq s2$ = *list-less-eq-ns s1 s2*
by (*simp add*: *assms order-le-less ordlistns.le-less valid-list-list-less-equiv-list-less-ns*)

27 Classical Lexicographical Order

lemma *valid-list-list-less-imp*:
assumes *valid-list* (*xs* @ [*bot*])
and *valid-list* (*ys* @ [*bot*])
and (*xs* @ [*bot*]) < (*ys* @ [*bot*])
shows $xs < ys$
proof −
from *assms*(*3*)
have *xs* @ [*bot*] ≠ *ys* @ [*bot*]
by *fastforce*
with *valid-list-neqE*[*OF assms*(*1*,*2*)]
obtain *x y as bs cs* **where**
xs @ [*bot*] = *as* @ *x* # *bs* *ys* @ [*bot*] = *as* @ *y* # *cs* $x \neq y$
by *blast*
hence $x < y$
by (*metis assms*(*3*) *linorder-less-linear list-less-ex order-less-imp-not-less*)
then show *?thesis*
by (*metis* ‹*xs* @ [*bot*] = *as* @ *x* # *bs*› ‹*ys* @ [*bot*] = *as* @ *y* # *cs*› *append-self-conv bot.extremum-strict butlast.simps*(*2*) *butlast-append last-snoc list-less-ex list-neq-rc1*)
qed

lemma *strict-mono-on-list-less-map*:
  fixes α :: $'a$ :: *preorder* $\Rightarrow$ $'b$ :: *ord*
  assumes *strict-mono-on A* α
  and *set xs* $\subseteq$ *A*
  and *set ys* $\subseteq$ *A*
  and *xs* $<$ *ys*
shows (*map* α *xs*) $<$ (*map* α *ys*)
  using *assms*(*2*−*4*)
proof (*induct xs arbitrary*: *ys*)
  case *Nil*
  then show *?case*
    using *list-le-def* **by** *fastforce*
next
  case (*Cons a xs*)
  note *IH* = *this*

  have $\exists$ *z zs*. *a* $\leq$ *z* $\wedge$ *ys* = *z* # *zs*
   by (*metis Cons-less-Cons IH*(*4*) *dual-order.refl dual-order.strict-iff-not neq-Nil-conv not-less-Nil*)
  then obtain *z zs* **where**
    a $\leq$ *z ys* = *z* # *zs*
    by *blast*
  then show *?case*
    using *IH assms*(*1*) *strict-mono-onD* **by** *fastforce*
qed

lemma *strict-mono-list-less-map*:
  assumes *strict-mono* α
  and *xs* $<$ *ys*
shows *map* α *xs* $<$ *map* α *ys*
  using *assms*(*1*) *assms*(*2*) *strict-mono-on-list-less-map* **by** *blast*

lemma *strict-mono-on-map-list-less*:
  fixes α :: $'a$ :: *linorder* $\Rightarrow$ $'b$ :: *order*
  assumes *strict-mono-on A* α
  and *set xs* $\subseteq$ *A*
  and *set ys* $\subseteq$ *A*
  and (*map* α *xs*) $<$ (*map* α *ys*)
shows *xs* $<$ *ys*
  using *assms*(*2*−*4*)
proof (*induct xs arbitrary*: *ys*)
case *Nil*
  then show *?case*
    using *list-le-def* **by** *fastforce*
next
  case (*Cons a xs*)
  note *IH* = *this*

have *ys* = [] ∨ (∃ *b zs*. *ys* = *b* # *zs*)
  using *neq-Nil-conv* **by** *blast*
moreover
have *ys* = [] ⟹ *?case*
  using *Cons.prems* **by** *auto*
moreover
have ∃ *b zs*. *ys* = *b* # *zs* ⟹ *?case*
 by (*metis IH*(*2*−*4*) *assms*(*1*) *linorder-neq-iff order-less-asym strict-mono-on-list-less-map*)
ultimately show *?case*
  by *blast*
qed

lemma *strict-mono-map-list-less*:
  fixes *α* :: *'a* :: *linorder* ⇒ *'b* :: *order*
  assumes *strict-mono α*
  and (*map α xs*) < (*map α ys*)
shows *xs* < *ys*
  using *assms*(*1*) *assms*(*2*) *strict-mono-on-map-list-less* **by** *blast*

28 Non-standard Lexicographical Ordering

lemma *sorted-list-less-ns*:
  assumes *sorted* (*a* # *bs* @ [*c*])
  and *c* < *d*
shows *list-less-ns* (*a* # *bs* @ [*c*, *d*] @ *xs*) (*bs* @ [*c*, *d*] @ *ys*)
  using *assms*
proof (*induct bs arbitrary*: *a*)
  case *Nil*
  then show *?case*
   by (*metis append-Cons append-Nil le-less list-less-ns-cons-diff list-less-ns-cons-same sorted2*)
next
  case (*Cons a bs x*)
  note *IH* = *this*

  from *IH*(*2*)
  have *sorted* (*a* # *bs* @ [*c*])
    by *simp*
  with *IH*(*1*)[*OF* - *assms*(*2*)]
  have *list-less-ns* (*a* # *bs* @ [*c*, *d*] @ *xs*) (*bs* @ [*c*, *d*] @ *ys*) **.**
  with *sorted-nth-mono*[*OF IH*(*2*), *of 0 Suc 0*, *simplified*]
  show *?case*
    by (*simp add*: *list-less-ns-cons*)
qed

lemma *rev-sorted-list-less-ns*:
  assumes *sorted* (*rev* (*a* # *bs* @ [*c*]))
  and *c* > *d*
shows *list-less-ns* (*bs* @ [*c*, *d*] @ *xs*) (*a* # *bs* @ [*c*, *d*] @ *ys*)

```
  using assms
proof (induct bs arbitrary: a)
  case Nil
  then show ?case
    using list-less-ns-cons list-less-ns-cons-diff by fastforce
next
  case (Cons a bs x)
  note IH = this

  from IH(2)
  have sorted (rev (a # bs @ [c]))
    by (simp add: sorted-append)
  with IH(1)[OF - assms(2)]
  have list-less-ns (bs @ [c, d] @ xs) (a # bs @ [c, d] @ ys) .
  with sorted-rev-nth-mono[OF IH(2), of 0 Suc 0, simplified]
  show ?case
    using list-less-ns-cons by auto
qed

lemma sorted-cons-list-less-ns:
  assumes sorted (a # bs)
  shows list-less-ns (a # bs) bs
  using assms
proof (induct bs arbitrary: a)
case Nil
  then show ?case
    by (simp add: list-less-ns-nil)
next
  case (Cons a bs x)
  note IH = this

  from IH(2)
  have sorted (a # bs)
    by simp
  with IH(1)
  have list-less-ns (a # bs) bs .
  with sorted-nth-mono[OF IH(2), of 0 Suc 0, simplified]
  show ?case
    by (simp add: list-less-ns-cons)
qed

end
theory Suffix
  imports Main
begin
```

29 Suffix

```
abbreviation suffix :: 'a list ⇒ nat ⇒ 'a list
```

where
suffix xs i $\equiv$ *drop i xs*

lemma *suffixes-neq*:
$[\![i < \mathit{length}\ s;\ j < \mathit{length}\ s;\ i \neq j]\!] \Longrightarrow \mathit{suffix}\ s\ i \neq \mathit{suffix}\ s\ j$
by (*metis diff-diff-cancel length-drop less-or-eq-imp-le*)

lemma *distinct-suffixes*:
$[\![\mathit{distinct}\ xs;\ \forall x \in \mathit{set}\ xs.\ x < \mathit{length}\ s]\!] \Longrightarrow \mathit{distinct}\ (\mathit{map}\ (\mathit{suffix}\ s)\ xs)$
by (*simp add*: *distinct-conv-nth suffixes-neq*)

lemma *suffix-eq-index*:
$[\![i < \mathit{length}\ xs;\ j < \mathit{length}\ xs;\ \mathit{suffix}\ xs\ i = \mathit{suffix}\ xs\ j]\!] \Longrightarrow i = j$
by (*metis diff-diff-cancel le-eq-less-or-eq length-drop*)

lemma *suffix-neq-nil*:
$i < \mathit{length}\ s \Longrightarrow \mathit{suffix}\ s\ i \neq []$
by *simp*

lemma *suffix-map*:
$\mathit{suffix}\ (\mathit{map}\ f\ xs)\ i = \mathit{map}\ f\ (\mathit{suffix}\ xs\ i)$
by (*simp add*: *drop-map*)

lemma *set-suffix-subset*:
$\mathit{set}\ (\mathit{suffix}\ s\ i) \subseteq \mathit{set}\ s$
by (*simp add*: *set-drop-subset*)
lemma *suffix-cons-suc*:
$\mathit{suffix}\ (a\ \#\ xs)\ (\mathit{Suc}\ i) = \mathit{suffix}\ xs\ i$
by *simp*

lemma *suffix-app*:
$i < \mathit{length}\ xs \Longrightarrow \mathit{suffix}\ (xs\ @\ ys)\ i = \mathit{suffix}\ xs\ i\ @\ ys$
by *simp*

lemma *suffix-cons-ex*:
$i < \mathit{length}\ T \Longrightarrow \exists x\ xs.\ \mathit{suffix}\ T\ i = x\ \#\ xs \wedge x = T\ !\ i$
by (*metis Cons-nth-drop-Suc*)

lemma *suffix-cons-Suc*:
$i < \mathit{length}\ T \Longrightarrow \mathit{suffix}\ T\ i = T\ !\ i\ \#\ \mathit{suffix}\ T\ (\mathit{Suc}\ i)$
by (*metis Cons-nth-drop-Suc*)

lemma *suffix-cons-app*:
$\mathit{suffix}\ T\ i = as\ @\ bs \Longrightarrow \mathit{suffix}\ T\ (i + \mathit{length}\ as) = bs$
by (*metis add.commute append-eq-conv-conj drop-drop*)

lemma *suffix-0*:
$\mathit{suffix}\ T\ 0 = T$
by *simp*

end
theory *Suffix-Util*
  imports
    ../util/List-Slice
    Suffix
    Valid-List
    Valid-List-Util

begin

30 Valid Lists and Suffixes

lemma *valid-suffix*:
  $[\![$*valid-list s*; $i <$ *length s*$]\!] \Longrightarrow$ *valid-list* (*suffix s i*)
  by (*clarsimp simp*: *valid-list-ex-def*)

lemma *last-suffix-index*:
  assumes *valid-list s*
  and $i <$ *length s*
  shows *hd* (*suffix s i*) = *bot* $\longleftrightarrow$ i = *length s* − *1*
proof −
  from *iffD1*[*OF valid-list-ex-def* ‹*valid-list s*›]
  obtain *xs* **where**
    s = *xs* @ [*bot*]
    $\forall$ *i* < *length xs*. *xs* ! *i* $\neq$ *bot*
    by *blast*
  show *?thesis*
  proof
    from ‹*s* = *xs* @ [*bot*]› ‹$\forall$ *i* < *length xs*. *xs* ! *i* $\neq$ *bot*›
    show *i* = *length s* − *1* $\Longrightarrow$ *hd* (*suffix s i*) = *bot*
      by *simp*
  next
    from ‹*s* = *xs* @ [*bot*]› ‹$\forall$ *i* < *length xs*. *xs* ! *i* $\neq$ *bot*› ‹*i* < *length s*›
    show *hd* (*suffix s i*) = *bot* $\Longrightarrow$ *i* = *length s* − *1*
      by (*clarsimp simp*: *hd-append hd-drop-conv-nth split*: *if-splits*)
  qed
qed

31 Prefixes and Suffixes

lemma *suffix-has-no-prefix-suffix*:
  assumes *valid-list*: *valid-list s*
  and   *i-less-len-s*: *i* < *length s*
  and   *j-less-len-s*: *j* < *length s*
  and   *i-neq-j*:   *i* $\neq$ *j*
  shows ¬ ($\exists$ *s'*. *suffix s i* = (*suffix s j*) @ *s'*)
proof

assume $\exists s'.$ *suffix s i* $=$ *suffix s j* @ s'
then obtain s' **where**
pref: *suffix s i* $=$ *suffix s j* @ s'
by *blast*
with *i-neq-j i-less-len-s j-less-len-s*
have $i < j$
by (*metis diff-less-mono2 length-append length-drop less-Suc-eq not-add-less1 not-less-eq*)
with *pref i-less-len-s j-less-len-s*
have *s-not-nil*: $s' \neq []$
by (*metis append-Nil2 diff-less-mono2 length-drop less-irrefl-nat*)

from *valid-list i-less-len-s valid-suffix*
have *valid-suf-i*: *valid-list* (*suffix s i*)
by *force*

from *valid-list j-less-len-s valid-suffix*
have *valid-list* (*suffix s j*)
by *force*
with *pref valid-list-ex-def*
have $\exists$ *xs*. *suffix s i* $=$ *xs* @ [*bot*] @ s'
using *append-assoc* **by** *auto*
then obtain *xs* **where**
suf-i: *suffix s i* $=$ *xs* @ [*bot*] @ s'
by *blast*

from *valid-suf-i valid-list-ex-def*
have $\exists$ *ys*. *suffix s i* $=$ *ys* @ [*bot*] $\wedge$ ($\forall k <$ *length ys*. *ys* ! $k \neq$ *bot*)
by *blast*
then obtain *ys* **where**
suffix s i $=$ *ys* @ [*bot*] **and**
all-ys-not-0: $\forall k <$ *length ys*. *ys* ! $k \neq$ *bot*
by *blast*
with *suf-i*
have *suf-i-eq*: *xs* @ [*bot*] @ s' $=$ *ys* @ [*bot*]
by *force*
with *s-not-nil*
have *length xs* $<$ *length ys*
by (*metis* (*no-types*, *lifting*) *append-assoc append-eq-append-conv length-append length-append-singleton less-trans-Suc linorder-neqE-nat not-add-less1 self-append-conv*)
with *suf-i-eq all-ys-not-0*
show *False*
by (*metis append-Cons butlast-snoc nth-append-length nth-butlast*)
qed

32 Suffix Comparisons

32.1 Lexicographical Ordering

lemma *suffix-less-ex*:
  fixes $s :: ('a :: \{linorder,\ order\text{-}bot\})\ list$
  assumes *valid-list s*
  and $i <$ *length s*
  and $j <$ *length s*
  and *suffix s i* $<$ *suffix s j*
  shows $\exists\, b\ c\ as\ bs\ cs.\ suffix\ s\ i = as\ @\ b\ \#\ bs\ \wedge$
    $suffix\ s\ j = as\ @\ c\ \#\ cs\ \wedge\ b < c$
proof –
  have *valid-list* (*suffix s i*)
    using *assms*(*1*) *assms*(*2*) *valid-suffix* **by** *blast*
  moreover
  have *valid-list* (*suffix s j*)
    using *assms*(*1*) *assms*(*3*) *valid-suffix* **by** *blast*
  moreover
  have *suffix s i* $\neq$ *suffix s j*
    using *assms*(*4*) *nless-le* **by** *blast*
  ultimately have
    $\exists\, b\ c\ as\ bs\ cs.\ suffix\ s\ i = as\ @\ b\ \#\ bs\ \wedge\ suffix\ s\ j = as\ @\ c\ \#\ cs\ \wedge\ b \neq c$
    using *valid-list-neqE* **by** *blast*
  then obtain *b c as bs cs* **where**
    $suffix\ s\ i = as\ @\ b\ \#\ bs\ \ suffix\ s\ j = as\ @\ c\ \#\ cs\ \ b \neq c$
    by *blast*
  hence $b < c$
    by (*metis assms*(*4*) *linorder-less-linear list-less-ex order-less-imp-not-less*)
  then show *?thesis*
    using ‹$suffix\ s\ i = as\ @\ b\ \#\ bs$› ‹$suffix\ s\ j = as\ @\ c\ \#\ cs$› **by** *blast*
qed

lemma *suffix-less-nth*:
  assumes *valid-list s*
  and $i <$ *length s*
  and $j <$ *length s*
  and *suffix s i* $<$ *suffix s j*
  shows
  $\exists\, n.\ n < length\ (suffix\ s\ i)\ \wedge$
    $n < length\ (suffix\ s\ j)\ \wedge$
    $(\forall\, k < n.\ (suffix\ s\ i)\ !\ k = (suffix\ s\ j)\ !\ k)\ \wedge$
      $(suffix\ s\ i)\ !\ n < (suffix\ s\ j)\ !\ n$
proof –
  from *suffix-less-ex*[*OF assms*]
  obtain *b c as bs cs* **where**
    suf-i: $suffix\ s\ i = as\ @\ b\ \#\ bs$ **and**
    suf-j: $suffix\ s\ j = as\ @\ c\ \#\ cs$ **and**
    b-less-c: $b < c$
    by *blast*

hence *length as* < *length* (*suffix s i*) **and**
length as < *length* (*suffix s j*) **and**
(*suffix s i*) ! *length as* = *b* **and**
(*suffix s j*) ! *length as* = *c*
by *fastforce*+
with *b-less-c suf-i suf-j*
show *?thesis*
by (*metis nth-append*)
qed

lemma *suffix-less-butlast*:
assumes *valid-list s*
and *i* < *length s*
and *j* < *length s*
and *suffix s i* < *suffix s j*
shows *butlast* (*suffix s i*) < *butlast* (*suffix s j*)
by (*metis append-butlast-last-id assms suffix-neq-nil valid-list-def valid-list-list-less-imp valid-suffix*)

32.2 Non-standard List Ordering

lemma *suffix-less-ns-ex*:
assumes *valid-list s*
and *i* < *length s*
and *j* < *length s*
and *list-less-ns* (*suffix s i*) (*suffix s j*)
shows $\exists$ *b c as bs cs.*
suffix s i = *as* @ *b* # *bs* $\wedge$
suffix s j = *as* @ *c* # *cs* $\wedge$ *b* < *c*
by (*meson assms suffix-less-ex valid-suffix valid-list-list-less-equiv-list-less-ns*)

lemma *suffix-less-ns-nth*:
assumes *valid-list s*
and *i* < *length s*
and *j* < *length s*
and *list-less-ns* (*suffix s i*) (*suffix s j*)
shows
$\exists$ *n. n* < *length* (*suffix s i*) $\wedge$
n < *length* (*suffix s j*) $\wedge$
($\forall$ *k* < *n.* (*suffix s i*) ! *k* = (*suffix s j*) ! *k*) $\wedge$
(*suffix s i*) ! *n* < (*suffix s j*) ! *n*
by (*meson assms suffix-less-nth valid-list-list-less-equiv-list-less-ns valid-suffix*)

33 List Slice

declare *list-slice.simps*[*simp del*]

lemma *list-slice-to-suffix*:
  list-slice T i j = *take* (*j* − *i*) (*suffix T i*)
  by (*simp add*: *list-slice.simps drop-take*)

lemma *suffix-eq-list-slice*:
  suffix T i = *list-slice T i* (*length T*)
  by (*simp add*: *list-slice.simps*)

lemma *list-slice-suffix*:
  list-slice T i j = *list-slice* (*suffix T i*) *0* (*j* − *i*)
  by (*metis drop0 drop-take list-slice.simps*)
lemma *suffix-to-list-slice-app*:
  $i \leq j \Longrightarrow$ *suffix T i* = (*list-slice T i j*) @ (*list-slice T j* (*length T*))
  apply (*cases* $j \leq$ *length T*)
   apply (*subst list-slice-append*[*symmetric*]; *simp?*)
   apply (*clarsimp simp*: *list-slice.simps*)
  apply (*clarsimp simp*: *not-le*)
  apply (*subst list-slice-end-gre-length*, *arith*)
  apply (*simp add*: *list-slice-start-gre-length list-slice.simps*)
  done

34 Sorting

lemma *ordlist-strict-mono-strict-sorted-1*:
  assumes *strict-mono* α
  and *strict-sorted* (*map* (*suffix* (*map* α *s*)) *xs*)
  shows *strict-sorted* (*map* (*suffix s*) *xs*)
proof (*intro sorted-wrt-mapI*)
  fix *i j*
  assume $i < j$ $j <$ *length xs*
  with *sorted-wrt-nth-less*[*OF assms*(*2*)]
  have *suffix* (*map* α *s*) (*xs* ! *i*) < *suffix* (*map* α *s*) (*xs* ! *j*)
    by *auto*
  then show *suffix s* (*xs* ! *i*) < *suffix s* (*xs* ! *j*)
    by (*metis assms*(*1*) *strict-mono-map-list-less suffix-map*)
qed

lemma *ordlist-strict-mono-on-strict-sorted-1*:
  assumes *strict-mono-on A* α
  and *set s* $\subseteq$ *A*
  and *strict-sorted* (*map* (*suffix* (*map* α *s*)) *xs*)
  shows *strict-sorted* (*map* (*suffix s*) *xs*)
proof (*intro sorted-wrt-mapI*)
  fix *i j*
  assume $i < j$ $j <$ *length xs*
  with *sorted-wrt-nth-less*[*OF assms*(*3*)]
  have *suffix* (*map* α *s*) (*xs* ! *i*) < *suffix* (*map* α *s*) (*xs* ! *j*)
    by *auto*
  hence *map* α (*suffix s* (*xs* ! *i*)) < *map* α (*suffix s* (*xs* ! *j*))

```
    by (metis suffix-map)
  then show suffix s (xs ! i) < suffix s (xs ! j)
    by (meson assms(1,2) dual-order.trans set-suffix-subset
              strict-mono-on-map-list-less)
qed

lemma ordlist-strict-mono-strict-sorted-2:
  assumes strict-mono α
  and     strict-sorted (map (suffix s) xs)
  shows strict-sorted (map (suffix (map α s)) xs)
proof (intro sorted-wrt-mapI)
  fix i j
  assume i < j j < length xs
  with sorted-wrt-nth-less[OF assms(2)]
  have suffix s (xs ! i) < suffix s (xs ! j)
    by auto
  then show suffix (map α s) (xs ! i) < suffix (map α s) (xs ! j)
    by (metis assms(1) strict-mono-list-less-map suffix-map)
qed

lemma ordlist-strict-mono-on-strict-sorted-2:
  assumes strict-mono-on A α
  and     set s ⊆ A
  and     strict-sorted (map (suffix s) xs)
  shows strict-sorted (map (suffix (map α s)) xs)
proof (intro sorted-wrt-mapI)
  fix i j
  assume i < j j < length xs
  with sorted-wrt-nth-less[OF assms(3)]
  have suffix s (xs ! i) < suffix s (xs ! j)
    by auto
  moreover
  have set (suffix s (xs ! i)) ⊆ A
    by (meson assms(2) dual-order.trans set-suffix-subset)
  moreover
  have set (suffix s (xs ! j)) ⊆ A
    by (meson assms(2) dual-order.trans set-suffix-subset)
  ultimately show suffix (map α s) (xs ! i) < suffix (map α s) (xs ! j)
    using strict-mono-on-list-less-map[OF assms(1)]
    by (metis suffix-map)
qed

lemma valid-list-ordlist-ordlistns-strict-sorted-eq:
  assumes valid-list T
  and     set xs ⊆ {0..<length T}
  shows ordlistns.strict-sorted (map (suffix T) xs) ⟷
        strict-sorted (map (suffix T) xs)
using assms
proof (safe)
```

assume *A*: *valid-list T* **and**
B: *set xs* ⊆ {*0..<length T*} **and**
C: *sorted-wrt list-less-ns* (*map* (*suffix T*) *xs*)
show *sorted-wrt* (<) (*map* (*suffix T*) *xs*)
proof (*intro sorted-wrt-mapI*)
fix *i j*
assume *i* < *j j* < *length xs*
with *sorted-wrt-nth-less*[*OF C* ‹*i* < *j*›]
have *R1*: *list-less-ns* (*suffix T* (*xs* ! *i*)) (*suffix T* (*xs* ! *j*))
by *auto*

from *B* ‹*i* < *j*› ‹*j* < *length xs*›
have *xs* ! *i* < *length T*
by (*meson atLeastLessThan-iff less-trans nth-mem subsetD*)
with *valid-suffix*[*OF A*]
have *R2*: *valid-list* (*suffix T* (*xs* ! *i*))
by *simp*

from *B* ‹*j* < *length xs*›
have *xs* ! *j* < *length T*
by (*meson atLeastLessThan-iff less-trans nth-mem subsetD*)
with *valid-suffix*[*OF A*]
have *R3*: *valid-list* (*suffix T* (*xs* ! *j*))
by *simp*

from *R1 valid-list-list-less-equiv-list-less-ns*[*OF R2 R3*]
show *suffix T* (*xs* ! *i*) < *suffix T* (*xs* ! *j*)
by *simp*
qed
next
assume *A*: *valid-list T* **and**
B: *set xs* ⊆ {*0..<length T*} **and**
C: *sorted-wrt* (<) (*map* (*suffix T*) *xs*)
show *sorted-wrt list-less-ns* (*map* (*suffix T*) *xs*)
proof (*intro sorted-wrt-mapI*)
fix *i j*
assume *i* < *j j* < *length xs*
with *sorted-wrt-nth-less*[*OF C* ‹*i* < *j*›]
have *R1*: *suffix T* (*xs* ! *i*) < *suffix T* (*xs* ! *j*)
by *auto*

from *B* ‹*i* < *j*› ‹*j* < *length xs*›
have *xs* ! *i* < *length T*
by (*meson atLeastLessThan-iff less-trans nth-mem subsetD*)
with *valid-suffix*[*OF A*]
have *R2*: *valid-list* (*suffix T* (*xs* ! *i*))
by *simp*

from *B* ‹*j* < *length xs*›

have *xs* ! *j* < *length T*
by (*meson atLeastLessThan-iff less-trans nth-mem subsetD*)
with *valid-suffix*[*OF A*]
have *R3*: *valid-list* (*suffix T* (*xs* ! *j*))
by *simp*

from *R1 valid-list-list-less-equiv-list-less-ns*[*OF R2 R3*]
show *list-less-ns* (*suffix T* (*xs* ! *i*)) (*suffix T* (*xs* ! *j*))
by *simp*
qed
qed

lemma *Min-valid-suffix*:
assumes *valid-list T*
and *length T* = *Suc n*
shows *ordlistns.Min* {*suffix T i* |*i*. *i* < *length T*} = *suffix T n*
proof −
from *assms*
have *suffix T n* = [*bot*]
by (*metis add-diff-cancel-left′ butlast-snoc length-butlast lessI list-slice-n-n nth-append-length plus-1-eq-Suc suffix-cons-Suc suffix-eq-list-slice valid-list-ex-def*)

have ∀ *i* < *n*. (*suffix T i*) ! *0* ≠ *bot*
by (*metis add-diff-cancel-left′ assms last-suffix-index less-SucI list.sel*(*1*) *nat-neq-iff nth-Cons-0 plus-1-eq-Suc suffix-cons-Suc*)
hence *A*: ∀ *i* < *n*. *bot* < (*suffix T i*) ! *0*
using *bot.not-eq-extremum* **by** *blast*

have *B*: ∀ *i* < *length T*. *length* (*suffix T i*) > *0*
by *auto*

show *?thesis*
proof (*intro ordlistns.Min-eqI conjI*)
show *finite* {*suffix T i* |*i*. *i* < *length T*}
by *simp*
next
fix *ys*
assume *ys* ∈ {*suffix T i* |*i*. *i* < *length T*}
hence ∃ *i* < *length T*. *ys* = *suffix T i*
by *blast*
then obtain *i* **where**
i < *length T*
ys = *suffix T i*
by *blast*

with ‹*ys* = *suffix T i*›
have *R1*: *i* = *n* ⟹ *list-less-eq-ns* (*suffix T n*) *ys*
by *simp*

from ‹i < *length* T› *assms*(2)
have *R2-1*: $i \neq n \Longrightarrow i < n$
by *linarith*

from *A* ‹*suffix* T n = [*bot*]› ‹i < *length* T› ‹*ys* = *suffix* T i›
have *R2-2*: $i < n \Longrightarrow$ *list-less-eq-ns* (*suffix* T n) *ys*
by (*metis list-less-ns-cons-diff nth-Cons-0 ordlistns.less-imp-le suffix-cons-ex*)

from *R1 R2-2*[*OF R2-1*]
show *list-less-eq-ns* (*suffix* T n) *ys*
by *blast*
next
show *suffix* T $n \in \{$*suffix* T i $|i.\ i <$ *length* $T\}$
using *assms*(2) **by** *auto*
qed
qed

end
theory *Prefix*
imports *Main*
begin

35 Prefix Definition

abbreviation *prefix* :: $'a$ *list* ⇒ *nat* ⇒ $'a$ *list*
where
prefix xs i ≡ *take i xs*

lemma *prefix-neq*:
assumes $i <$ *length s*
and $j <$ *length s*
and $i \neq j$
shows *prefix s i* ≠ *prefix s j*
by (*metis assms length-take less-imp-le min-absorb2*)

lemma *not-prefix-app*:
($\forall k.$ *s1* ≠ *prefix s2 k*) ⟷ ($\forall xs.$ *s2* ≠ *s1* @ *xs*)
by (*metis append-eq-conv-conj append-take-drop-id*)

lemma *not-prefix-imp-not-nil*:
$\forall k.$ *s1* ≠ *prefix s2 k* ⟹ *s1* ≠ []
by (*metis take0*)

end
theory *Prefix-Util*
imports *Prefix ../order/Suffix-Util*
begin

lemma *prefix-suffix-not-suffix*:

assumes *valid-list s*
and $i < length\ s$
and $j < length\ s$
and $i \neq j$
shows $\neg(\exists\, k.\ prefix\ (suffix\ s\ i)\ k = suffix\ s\ j)$
using *suffix-has-no-prefix-suffix assms*
by (*metis append-take-drop-id*)

end
theory *Suffix-Array*
imports
../util/Sorting-Util
../order/List-Lexorder-Util
../order/Suffix
../order/Valid-List
../order/List-Permutation-Util
begin

36 Axiomatic Suffix Array Specification

locale *Suffix-Array-General* =
fixes *sa* :: ('*a* :: {*linorder*, *order-bot*}) *list* ⇒ *nat list*
assumes *sa-g-permutation*: *sa s* <~~> [*0*..<*length s*]
and *sa-g-sorted*: *strict-sorted* (*map* (*suffix s*) (*sa s*))

locale *Suffix-Array-Restricted* =
fixes *sa* :: *nat list* ⇒ *nat list*
assumes *sa-r-permutation*: *valid-list s* ⟹ *sa s* <~~> [*0*..<*length s*]
and *sa-r-sorted*: *valid-list s* ⟹ *strict-sorted* (*map* (*suffix s*) (*sa s*))

37 Wrapper for Natural Number String only Algorithm

definition *sa-nat-wrapper* ::
('*a* :: *linorder list* ⇒ '*a* ⇒ *nat*) ⇒ (*nat list* ⇒ *nat list*) ⇒ '*a* :: *linorder list* ⇒ *nat list*
where
sa-nat-wrapper α *sa xs* =
tl (*sa* ((*map* ($\lambda x.$ *Suc* (α *xs x*)) *xs*) @ [*bot*]))

end
theory *Suffix-Array-Properties*
imports
../util/Fun-Util
../order/Suffix-Util
Suffix-Array

begin

38 General Suffix Array Properties

context *Suffix-Array-General* **begin**

lemma *sa-length*:
  length (*sa s*) = *length s*
  by (*metis Suffix-Array-General-axioms Suffix-Array-General-def length-upt minus-nat.diff-0*
    perm-length)

lemma *sa-distinct*:
  distinct (*sa s*)
  using *Suffix-Array-General.sa-g-permutation Suffix-Array-General-axioms*
    perm-distinct-set-of-upt-iff **by** *blast*

lemma *sa-set-upt*:
  $set\ (sa\ s) = \{0..<length\ s\}$
  using *Suffix-Array-General.sa-g-permutation Suffix-Array-General-axioms*
    perm-distinct-set-of-upt-iff **by** *blast*

lemma *sa-nth-ex*:
  $i < length\ s \Longrightarrow \exists\, k < length\ s.\ sa\ s\ !\ i = k$
  by (*metis atLeastLessThan-iff nth-mem sa-length sa-set-upt*)

lemma *ex-sa-nth*:
  $k < length\ s \Longrightarrow \exists\, i < length\ s.\ sa\ s\ !\ i = k$
  by (*metis atLeast0LessThan in-set-conv-nth lessThan-iff sa-length sa-set-upt*)

end

lemma *Suffix-Array-General-determinism*:
  assumes *Suffix-Array-General f*
  and *Suffix-Array-General g*
shows $f = g$
proof
  fix *s*
  from *distinct-suffixes*[*OF Suffix-Array-General.sa-distinct*[*OF assms*(*1*)], *of s s*]
    Suffix-Array-General.sa-set-upt[*OF assms*(*1*), *of s*]
  have *distinct* (*map* (*suffix s*) (*f s*))
    using *atLeastLessThan-iff* **by** *blast*
  moreover
  from *distinct-suffixes*[*OF Suffix-Array-General.sa-distinct*[*OF assms*(*2*)], *of s s*]
    Suffix-Array-General.sa-set-upt[*OF assms*(*2*), *of s*]
  have *distinct* (*map* (*suffix s*) (*g s*))
    using *atLeastLessThan-iff* **by** *blast*
  moreover
  from *Suffix-Array-General.sa-set-upt*[*OF assms*(*1*), *of s*]
    Suffix-Array-General.sa-set-upt[*OF assms*(*2*), *of s*]
  have *set* (*map* (*suffix s*) (*f s*)) = *set* (*map* (*suffix s*) (*g s*))

by *simp*
ultimately have *map* (*suffix s*) (*f s*) = *map* (*suffix s*) (*g s*)
using *strict-sorted-distinct-set-unique*[
OF Suffix-Array-General.sa-g-sorted[*of f*, *OF assms*(*1*)] -
Suffix-Array-General.sa-g-sorted[*of g*, *OF assms*(*2*)],
of s s]
by *blast*
moreover
from *Suffix-Array-General.sa-set-upt*[*OF assms*(*1*), *of s*]
Suffix-Array-General.sa-set-upt[*OF assms*(*2*), *of s*]
have *inj-on* (*suffix s*) (*set* (*f s*) ∪ *set* (*g s*))
by (*simp add*: *inj-on-def suffix-eq-index*)
ultimately show *f s* = *g s*
using *map-inj-on*[*of suffix s f s g s*]
by *blast*
qed

39 Properties of Suffix Arrays on Valid Lists

lemma *valid-list-bot-min*:
assumes *valid-list* (*s* @ [*bot*])
and *sa* (*s* @ [*bot*]) <~~> [*0*..<*length* (*s* @ [*bot*])]
and *strict-sorted* (*map* (*suffix* (*s* @ [*bot*])) (*sa* (*s* @ [*bot*])))
shows ∃ *xs*. *sa* (*s* @ [*bot*]) = *length s* # *xs*
proof −
have *suffix* (*s* @ [*bot*]) (*length s*) = [*bot*]
by *simp*

have *P*: ∀ *i* < *length s*. *suffix* (*s* @ [*bot*]) (*length s*) < *suffix* (*s* @ [*bot*]) *i*
proof(*safe*)
fix *i*
assume *i* < *length s*
have ∃ *a as*. *suffix* (*s* @ [*bot*]) *i* = *a* # *as* ∧ *a* ≠ *bot*
by (*metis Cons-nth-drop-Suc* ‹*i* < *length s*› *assms*(*1*) *butlast-snoc length-append-singleton less-SucI nth-butlast valid-list-ex-def*)
then obtain *a as* **where**
suffix (*s* @ [*bot*]) *i* = *a* # *as* ∧ *a* ≠ *bot*
by *blast*
moreover
from ‹*suffix* (*s* @ [*bot*]) (*length s*) = [*bot*]›
have *suffix* (*s* @ [*bot*]) (*length s*) = *bot* # []
by *simp*
ultimately show *suffix* (*s* @ [*bot*]) (*length s*) < *suffix* (*s* @ [*bot*]) *i*
by (*simp add*: *bot.not-eq-extremum*)
qed

have *Min* (*set* (*map* (*suffix* (*s* @ [*bot*])) (*sa* (*s* @ [*bot*]))))
= *suffix* (*s* @ [*bot*]) (*length s*)
proof (*intro Min-eqI*)

show *finite* (*set* (*map* (*suffix* (*s* @ [*bot*])) (*sa* (*s* @ [*bot*]))))
by *blast*
next
fix *y*
assume $y \in$ *set* (*map* (*suffix* (*s* @ [*bot*])) (*sa* (*s* @ [*bot*])))
with *set-perm-upt*[*OF assms*(*2*)]
have $\exists\, i <$ *length* (*s* @ [*bot*]). $y =$ *suffix* (*s* @ [*bot*]) *i*
by *auto*
then obtain *i* **where**
$i <$ *length* (*s* @ [*bot*])
$y =$ *suffix* (*s* @ [*bot*]) *i*
by *blast*
hence $i <$ *length s* $\vee$ $i =$ *length s*
by (*simp add*: *less-Suc-eq*)
moreover
have $i <$ *length s* $\Longrightarrow$ *suffix* (*s* @ [*bot*]) (*length s*) $< y$
using *P* ‹$y =$ *suffix* (*s* @ [*bot*]) *i*› *dual-order.strict-iff-order* **by** *blast*
moreover
have $i =$ *length s* $\Longrightarrow$ *suffix* (*s* @ [*bot*]) (*length s*) $\leq y$
by (*simp add*: ‹$y =$ *suffix* (*s* @ [*bot*]) *i*›)
ultimately show *suffix* (*s* @ [*bot*]) (*length s*) $\leq y$
using *nless-le* **by** *blast*
next
from *assms*
have *length s* $\in$ *set* (*sa* (*s* @ [*bot*]))
by (*metis ex-perm-nth length-append-singleton lessI nth-mem perm-upt-length*)
then show *suffix* (*s* @ [*bot*]) (*length s*) $\in$ *set* (*map* (*suffix* (*s* @ [*bot*])) (*sa* (*s* @ [*bot*])))
by *force*
qed
hence *map* (*suffix* (*s* @ [*bot*])) (*sa* (*s* @ [*bot*])) ! *0* = *suffix* (*s* @ [*bot*]) (*length s*)
using *assms*(*2*,*3*) *strict-sorted-Min* **by** *fastforce*
hence *suffix* (*s* @ [*bot*]) ((*sa* (*s* @ [*bot*])) ! *0*) = *suffix* (*s* @ [*bot*]) (*length s*)
by (*metis assms*(*1*,*2*) *nth-map perm-upt-length valid-list-length*)
hence (*sa* (*s* @ [*bot*])) ! *0* = *length s*
by (*metis Suc-le-eq* ‹*suffix* (*s* @ [*bot*]) (*length s*) = [*bot*]› *assms*(*1*) *drop-all last-suffix-index*
list.distinct(*1*) *list.sel*(*1*) *not-less-eq-eq*)
then show *?thesis*
by (*metis append-eq-Cons-conv assms*(*1*,*2*) *id-take-nth-drop perm-upt-length take0*
valid-list-length)
qed

lemma *valid-list-bot-perm*:
assumes *valid-list* (*s* @ [*bot*])
and *sa* (*s* @ [*bot*]) <~~> [*0*..<*length* (*s* @ [*bot*])]
and *strict-sorted* (*map* (*suffix* (*s* @ [*bot*])) (*sa* (*s* @ [*bot*])))
shows $\exists$ *xs*. *sa* (*s* @ [*bot*]) = *length s* # *xs* $\wedge$ *xs* <~~> [*0*..<*length s*]

proof −
  from *valid-list-bot-min*[*OF assms*(*1*), *of sa*, *OF assms*(*2*,*3*)]
  obtain *xs* **where**
    sa (*s* @ [*bot*]) = *length s* # *xs*
    by *blast*
  with *assms*(*2*)
  have *length s* # *xs* <~~> [*0*..<*length* (*s* @ [*bot*])]
    by *simp*
  then show *?thesis*
    by (*metis* ‹*sa* (*s* @ [*bot*]) = *length s* # *xs*› *assms*(*1*) *length-append-singleton* *less-Suc-eq-le*
        perm-append2-eq perm-append-single upt-Suc valid-list-length)
qed

lemma *valid-list-bot-perm-sort*:
  assumes *valid-list* (*s* @ [*bot*])
  and *sa* (*s* @ [*bot*]) <~~> [*0*..<*length* (*s* @ [*bot*])]
  and *strict-sorted* (*map* (*suffix* (*s* @ [*bot*])) (*sa* (*s* @ [*bot*])))
shows ∃ *xs*. *sa* (*s* @ [*bot*]) = *length s* # *xs* ∧ *xs* <~~> [*0*..<*length s*] ∧
        strict-sorted (*map* (*suffix s*) *xs*)
proof −
  from *valid-list-bot-perm*[*OF assms*(*1*), *of sa*, *OF assms*(*2*,*3*)]
  obtain *xs* **where**
    sa (*s* @ [*bot*]) = *length s* # *xs*
    xs <~~> [*0*..<*length s*]
    by *blast*
  with *assms*(*3*)
  have *strict-sorted* (*map* (*suffix* (*s* @ [*bot*])) (*length s* # *xs*))
    by *simp*
  hence *strict-sorted* ((*suffix* (*s* @ [*bot*]) (*length s*)) # *map* (*suffix* (*s* @ [*bot*])) *xs*)
    by *simp*
  hence *P*: *strict-sorted* (*map* (*suffix* (*s* @ [*bot*])) *xs*)
    using *strict-sorted-simps*(*2*) **by** *blast*

  have *strict-sorted* (*map* (*suffix s*) *xs*)
  proof (*intro sorted-wrt-mapI*)
    fix *i j*
    assume *i* < *j j* < *length xs*
    with *sorted-wrt-nth-less*[*OF P*, *of i j*]
    have *suffix* (*s* @ [*bot*]) (*xs* ! *i*) < *suffix* (*s* @ [*bot*]) (*xs* ! *j*)
      by *auto*
    moreover
    have *xs* ! *i* < *length s*
    using ‹*i* < *j*› ‹*j* < *length xs*› ‹*xs* <~~> [*0*..<*length s*]› *perm-distinct-set-of-upt-iff* **by** *auto*
    hence *suffix* (*s* @ [*bot*]) (*xs* ! *i*) = *suffix s* (*xs* ! *i*) @ [*bot*]
      using *suffix-app* **by** *blast*
    moreover
    have *xs* ! *j* < *length s*

```
      using ‹j < length xs› ‹xs <~~> [0..<length s]› perm-distinct-set-of-upt-iff
by auto
    hence suffix (s @ [bot]) (xs ! j) = suffix s (xs ! j) @ [bot]
      using suffix-app by blast
    moreover
    have valid-list (suffix s (xs ! i) @ [bot])
      using ‹xs ! i < length s› assms valid-suffix by fastforce
    moreover
    have valid-list (suffix s (xs ! j) @ [bot])
      using ‹xs ! j < length s› assms valid-suffix by fastforce
    ultimately show suffix s (xs ! i) < suffix s (xs ! j)
      by (simp add: valid-list-list-less-imp)
  qed
  with ‹sa (s @ [bot]) = length s # xs› ‹xs <~~> [0..<length s]›
  show ?thesis
    by blast
qed

theorem Suffix-Array-Restricted-valid-list-bot-perm-sort:
  assumes valid-list (s @ [bot])
  and     Suffix-Array-Restricted sa
shows ∃xs. sa (s @ [bot]) = length s # xs ∧ xs <~~> [0..<length s] ∧
            strict-sorted (map (suffix s) xs)
proof (rule valid-list-bot-perm-sort[OF assms(1)])
  from assms
  show sa (s @ [bot]) <~~> [0..<length (s @ [bot])]
    using Suffix-Array-Restricted-def by blast
next
  from assms
  show strict-sorted (map (suffix (s @ [bot])) (sa (s @ [bot])))
    using Suffix-Array-Restricted-def by blast
qed

lemma Suffix-Array-Restricted-wrapper-permutation:
  assumes Linorder-to-Nat-List α s
  and Suffix-Array-Restricted sa
shows sa-nat-wrapper α sa s <~~> [0..<length s]
proof −
  let ?α = α s
  let ?f = λx. Suc (?α x)
  let ?s = map ?f s

  have valid-list (?s @ [bot])
    using valid-list-Suc-mapping by blast
  with Suffix-Array-Restricted-valid-list-bot-perm-sort[OF - ‹Suffix-Array-Restricted
-›]
  obtain xs where
    sa (?s @ [bot]) = length ?s # xs
    xs <~~> [0..<length ?s]
```

strict-sorted (*map* (*suffix ?s*) *xs*)
by *blast*
then show *?thesis*
by (*simp add*: *sa-nat-wrapper-def*)
qed

lemma *Suffix-Array-Restricted-wrapper-sorted*:
assumes *Linorder-to-Nat-List* α *s*
and *Suffix-Array-Restricted sa*
shows *strict-sorted* (*map* (*suffix s*) (*sa-nat-wrapper* α *sa s*))
proof −
let *?*α = α *s*
let *?f* = λ*x*. *Suc* (*?*α *x*)
let *?s* = *map ?f s*

have *valid-list* (*?s* @ [*bot*])
using *valid-list-Suc-mapping* **by** *blast*
with *Suffix-Array-Restricted-valid-list-bot-perm-sort*[*OF* - ‹*Suffix-Array-Restricted* -›]
obtain *xs* **where** *A*:
sa (*?s* @ [*bot*]) = *length ?s* # *xs*
xs <~~> [*0*..<*length ?s*]
strict-sorted (*map* (*suffix ?s*) *xs*)
by *blast*
hence *xs* <~~> [*0*..<*length s*]
by *simp*
with *ordlist-strict-mono-on-strict-sorted-1*[
OF Linorder-to-Nat-List.strict-mono-on-Suc-map-to-nat[*OF assms*(*1*)] - *A*(*3*)]
show *?thesis*
by (*simp add*: *A*(*1*) *sa-nat-wrapper-def*)
qed

40 Equivalence

lemma *Suffix-Array-General-imp-Restrict*:
Suffix-Array-General sa-nat $\Longrightarrow$ *Suffix-Array-Restricted sa-nat*
using *Suffix-Array-General-def Suffix-Array-Restricted.intro* **by** *blast*

interpretation *Linorder-to-Nat-List map-to-nat*
proof
show *strict-mono-on* (*set xs*) (*map-to-nat xs*)
by (*simp add*: *map-to-nat-strict-mono-on*)
qed

lemma *Suffix-Array-Restricted-imp-General*:
Suffix-Array-Restricted sa $\Longrightarrow$ *Suffix-Array-General* (*sa-nat-wrapper map-to-nat sa*)
using *Linorder-to-Nat-List-axioms Suffix-Array-General-def*

Suffix-Array-Restricted-wrapper-permutation Suffix-Array-Restricted-wrapper-sorted
by *blast*

lemma *Suffix-Array-General-Restrict-determinism*:
assumes *Suffix-Array-Restricted f*
and *Suffix-Array-General g*
shows *sa-nat-wrapper map-to-nat f = g*
by (*simp add*: *Suffix-Array-General-determinism Suffix-Array-Restricted-imp-General assms*)

end
theory *Simple-SACA*
imports
../order/Suffix
../order/List-Lexorder-Util
begin

fun *gen-suffixes* :: (*'a* :: {*linorder*,*order-bot*}) *list* ⇒ *'a list list*
where
gen-suffixes s = *map* (*suffix s*) [*0*..<(*length s*)]

fun *suffix-ids* :: (*'a* :: {*linorder*,*order-bot*}) *list* ⇒ *'a list list* ⇒ *nat list*
where
suffix-ids s ss = *map* (λ*x*. *length s* − *length x*) *ss*

fun *simple-saca* :: (*'a* :: {*linorder*,*order-bot*}) *list* ⇒ *nat list*
where
simple-saca s = *suffix-ids s* (*sort* (*gen-suffixes s*))

end
theory *Simple-SACA-Verification*
imports
Simple-SACA
../spec/Suffix-Array
begin

lemma *suf-length-app*:
i < *length xs* ⟹ *length* (*suffix* (*xs* @ *ys*) *i*) = *length* (*suffix xs i*) + *length ys*
apply (*induct xs*)
apply *simp*
apply *simp*
done

lemma *distinct-natlist-add*:
distinct (*xs* :: *nat list*) ⟹ *distinct* (*map* ((+) *n*) *xs*)
apply (*induct xs arbitrary*: *n*)
apply *simp*
apply *clarsimp*
done

lemma *nat-minus-cancel-right*:
$[\![(x::nat) \leq n;\ y \leq n;\ n - x = n - y]\!] \Longrightarrow x = y$
apply (*subst* (*asm*) *le-imp-diff-is-add*, *simp*)
apply (*subst* (*asm*) *add.commute*)
apply (*subst* (*asm*) *add-diff-assoc*, *simp*)
apply (*subst* (*asm*) *add.commute*)
apply (*drule sym*)
apply (*subst* (*asm*) *Nat.le-imp-diff-is-add*, *simp*)
apply *clarsimp*
done

lemma *distinct-natlist-sub*:
$[\![distinct\ (xs :: nat\ list);\ \forall x \in set\ xs.\ x \leq n]\!] \Longrightarrow distinct\ (map\ ((-)\ n)\ xs)$
by (*meson distinct-map inj-onI nat-minus-cancel-right*)

lemma *map-suf-app*:
$n \leq length\ xs \Longrightarrow$
$map\ (length \circ suffix\ (xs\ @\ ys))\ [0..<n] = map\ ((+)\ (length\ ys))\ (map\ (length \circ (suffix\ xs))\ [0..<n])$
apply (*induct xs*)
apply *simp*
apply *clarsimp*
apply (*subst add.commute*)
apply *simp*
done

lemma *distinct-map-length-gen-suffixes*:
distinct (*map length* (*gen-suffixes s*))
apply (*induct s rule*: *rev-induct*)
apply *simp*
apply (*simp only*: *gen-suffixes.simps map-map length-append*)
apply (*subst upt-add-eq-append*; *simp only*: *map-append*)
apply (*subst map-suf-app*; *simp only*: *distinct-append*)
apply (*rule conjI*)
apply (*rule distinct-natlist-add*; *simp*)
apply (*rule conjI*; *clarsimp*)
done

lemma *different-length-different-list*:
$length\ a \notin length\ `\ set\ xs \Longrightarrow a \notin set\ xs$
apply *blast*
done

lemma *distinct-map-length-sort*:
$distinct\ (map\ length\ xs) \Longrightarrow distinct\ (map\ length\ (sort\ xs))$
apply (*induct xs*)
apply *simp*
apply *clarsimp*

apply (*rule card-distinct*)
apply *simp*
apply (*drule distinct-card*)
apply *clarsimp*
apply (*frule different-length-different-list*)
apply (*subst insort-insert-insort*[*symmetric*]; *simp*)
apply (*subst set-insort-insert*)
apply *simp*
done

lemma *suffix-ids-def'*:
suffix-ids s xs = *map* (((−) (*length s*)) ∘ *length*) *xs*
apply *simp*
done

lemma *distinct-simple-saca*:
distinct (*simple-saca s*)
apply (*subst simple-saca.simps*)
apply (*subst suffix-ids-def'*)
apply (*subst map-map*[*symmetric*])
apply (*rule distinct-natlist-sub*)
apply (*rule distinct-map-length-sort*[*OF distinct-map-length-gen-suffixes*])
apply *clarsimp*
done

lemma *suf-suffix-id-suf*:
i < *length s* ⟹ *suffix s* (*length s* − *length* (*suffix s i*)) = *suffix s i*
apply (*induct s arbitrary*: *i*)
apply *simp*
apply *clarsimp*
done

lemma *in-set-ordlist-sort*:
(*x* ∈ *set xs*) = (*x* ∈ *set* (*sort xs*))
by *simp*

lemma *ordlist-sort-conv-nth*:
(∃ *i*<*length xs*. *xs* ! *i* = *x*) = (∃ *i*<*length xs*. (*sort xs*) ! *i* = *x*)
by (*metis in-set-conv-nth length-sort set-sort*)

lemma *ordlist-sort-nth-before*:
⟦*i* < *length xs*; (*sort xs*) ! *i* = *x*⟧ ⟹
∃ *j*<*length xs*. *xs* ! *j* = *x*
apply (*subst ordlist-sort-conv-nth*)
apply *blast*
done

lemma *suf-sort-suf-nth*:
i < *length s* ⟹

```
   suffix s (length s − length ((sort (gen-suffixes s)) ! i)) =
   sort (gen-suffixes s) ! i
proof −
  assume i < length s


  have ∃ x. sort (gen-suffixes s) ! i = x
    by blast
  then obtain x where
    sort (gen-suffixes s) ! i = x
    by blast
  with ordlist-sort-nth-before[of i gen-suffixes s x]
  have ∃ j<length (gen-suffixes s). gen-suffixes s ! j = x
    by (simp add: ‹i < length s›)
  then obtain j where
    j < length (gen-suffixes s)
    gen-suffixes s ! j = x
    by blast
  hence sort (gen-suffixes s) ! i = gen-suffixes s ! j
    using ‹sort (gen-suffixes s) ! i = x› by blast
  moreover
  have j < length s
    using ‹j < length (gen-suffixes s)› by auto
  hence gen-suffixes s ! j = suffix s j
    by simp
  ultimately show ?thesis
    by (metis ‹j < length s› suf-suffix-id-suf)
qed

lemma map-suf-simple-saca:
  map (suffix s) (simple-saca s) = sort (gen-suffixes s)
  apply (simp only: simple-saca.simps suffix-ids.simps)
  apply (subst list-eq-iff-nth-eq)
  apply (rule conjI)
   apply simp
  apply (clarsimp simp del:  gen-suffixes.simps)
  apply (rule suf-sort-suf-nth; simp)
  done

interpretation simple-saca: Suffix-Array-General simple-saca
proof
  fix s :: ('a :: {linorder, order-bot}) list

  show simple-saca s <~~> [0..<length s]
  proof −
    have set (simple-saca s) = {0..<length s}
      by force
   with perm-distinct-set-of-upt-iff[THEN iffD2, OF conjI, OF distinct-simple-saca]
    show ?thesis
```

```
    by blast
  qed

  show strict-sorted (map (suffix s) (simple-saca s))
  proof -
    from ‹simple-saca s <~~> [0..<length s]›
    have set (simple-saca s) = {0..<length s}
      using perm-distinct-set-of-upt-iff by blast
    hence ∀ x ∈ set (simple-saca s). x < length s
      using atLeastLessThan-iff by blast
    with distinct-simple-saca distinct-suffixes
    have distinct (map (suffix s) (simple-saca s))
      by blast

    have sorted (map (suffix s) (simple-saca s))
      by (metis map-suf-simple-saca sorted-sort)
    with ‹distinct (map (suffix s) (simple-saca s))› show ?thesis
      using strict-sorted-iff by blast
  qed
qed

end
theory List-Type
  imports
    ../../util/Nat-Util
    ../../util/Set-Util
    ../../util/Fun-Util
    ../../util/List-Util
    ../../order/Suffix-Util
    ../../order/Valid-List-Util
    ../../spec/Suffix-Array-Properties
begin
```

This theory file contains the background theory for the SAIS algorithm (Nong et al., DCC 2009), which is essentially an optimisation of the KA algorithm (Ko et al, JDA 2005).

41 Small and Large List Types

```
datatype SL-types = S-type | L-type
```

This section contains a generalisation of the suffix types to sequences of any type and any element comparison function that satisfies certain properties given the theorem. Typical constraints involve either one or a combination of *totalp-on*, *irreflp-on*, *transp-on* and *asymp-on*.

```
definition
  list-type :: ('a ⇒ 'a ⇒ bool) ⇒ 'a list ⇒ SL-types
where
```

list-type cmp xs =
  (*if nslexordp cmp xs* (*suffix xs* (*Suc 0*))
   then S-type
   else L-type)

lemma *list-type-cons-same*:
  $[\![$*irreflp-on A cmp*; $x \in A]\!] \Longrightarrow$ *list-type cmp* (x # x # *xs*) = *list-type cmp* (x # *xs*)
  by (*clarsimp simp*: *list-type-def irreflp-onD*)

lemma *list-type-nil*:
  list-type cmp [] = *L-type*
  by (*clarsimp simp*: *list-type-def nslexordp-def*)

lemma *list-type-singleton*:
  list-type cmp [x] = *S-type*
  by (*simp add*: *nslexordp-def list-type-def*)

lemma *list-type-s-type-eq*:
  list-type cmp xs = *S-type* $\longleftrightarrow$ *nslexordp cmp xs* (*suffix xs* (*Suc 0*))
  by (*simp add*: *list-type-def*)

lemma *list-type-l-type-eq*:
  list-type cmp xs = *L-type* $\longleftrightarrow$ $\neg$*nslexordp cmp xs* (*suffix xs* (*Suc 0*))
  by (*simp add*: *list-type-def*)

lemma *list-type-cons-diff1*:
  cmp x y $\Longrightarrow$ *list-type cmp* (x # y # *xs*) = *S-type*
  by (*simp add*: *list-type-s-type-eq*)

lemma *list-type-cons-diff2*:
  $[\![\neg$*cmp x y*; $x \neq y]\!] \Longrightarrow$ *list-type cmp* (x # y # *xs*) = *L-type*
  by (*clarsimp simp add*: *list-type-l-type-eq*)

lemma *list-type-s-neq-nil*:
  list-type cmp xs = *S-type* $\Longrightarrow$ *xs* $\neq$ []
  by (*metis SL-types.simps*(*2*) *list-type-nil*)

lemma *list-type-s-hd-cmp*:
  list-type cmp (x # y # *xs*) = *S-type* $\Longrightarrow$ *cmp x y* $\vee$ $x = y$
  by (*metis SL-types.simps*(*2*) *list-type-cons-diff2*)

lemma *list-type-l-hd-cmp*:
  list-type cmp (x # y # *xs*) = *L-type* $\Longrightarrow$ $\neg$*cmp x y* $\vee$ $x = y$
  by (*metis SL-types.simps*(*2*) *list-type-cons-diff1*)

lemma *list-type-repl*:
  $[\![$*irreflp-on A cmp*; $x \in A$; *set xs* = $\{x\}]\!] \Longrightarrow$ *list-type cmp* (x # *xs*) = *S-type*
  apply (*induct xs*; *simp add*: *list-type-cons-same*)

using *list-type-singleton subset-singletonD* **by** *fastforce*

lemma *list-type-s-ex*:
assumes *list-type cmp* $(x \# xs) = S\text{-}type$
shows $(\forall a \in set\ xs.\ a = x) \lor (\exists b\ as\ bs.\ x \# xs = as\ @\ x \# b \# bs \land cmp\ x\ b \land (\forall k \in set\ as.\ k = x))$
proof −
from *list-type-s-type-eq*[*THEN iffD1*, *OF assms*(*1*)]
have *nslexordp cmp* $(x \# xs)$ *xs*
by *simp*
with *nslexordp-cons2-exE*[*of cmp x xs*]
show *?thesis*
by *blast*
qed

lemma *list-type-l-type-ex*:
assumes *list-type cmp* $(x \# xs) = L\text{-}type$
and *totalp-on A cmp*
and $x \in A$
and $set\ xs \subseteq A$
shows $\exists b\ as\ bs.\ x \# xs = as\ @\ x \# b \# bs \land cmp\ b\ x \land (\forall k \in set\ as.\ k = x)$
proof −
from *list-type-l-type-eq*[*THEN iffD1*, *OF assms*(*1*)]
have $\neg$ *nslexordp cmp* $(x \# xs)$ *xs*
by *simp*
moreover
have $x \# xs \neq xs$
by *fastforce*
ultimately have *nslexordp cmp xs* $(x \# xs)$
using *totalp-onD*[*OF nslexordp-totalp-on*[*OF assms*(*2*)]]
by (*metis assms*(*3*,*4*) *insert-subset list.simps*(*15*) *mem-Collect-eq*)
with *nslexordp-cons1-exE*[*of cmp xs x*]
show *?thesis*
by *blast*
qed

theorem *l-less-than-s-type-list-type*:
assumes *list-type cmp* $(a \# s1) = S\text{-}type$
and *list-type cmp* $(a \# s2) = L\text{-}type$
and *totalp-on A cmp*
and *transp-on A cmp*
and $a \in A$
and $set\ s1 \subseteq A$
and $set\ s2 \subseteq A$
shows *nslexordp cmp* $(a \# s2)$ $(a \# s1)$
proof −
from *list-type-l-type-ex*[*OF assms*(*2*,*3*,*5*,*7*)]
obtain *b as bs* **where**
$a \# s2 = as\ @\ a \# b \# bs$

cmp b a
$\forall k \in set\ as.\ k = a$
by *blast*
hence *S2*: *a # s2 = replicate* (*length as*) *a* @ *a # b # bs*
by (*simp add*: *replicate-length-same*)

let *?c1* = $\forall x \in set\ s1.\ x = a$
and *?c2* = $\exists d\ cs\ ds.\ a \# s1 = cs\ @\ a \# d \# ds \wedge cmp\ a\ d \wedge (\forall k \in set\ cs.\ k = a)$

from *list-type-s-ex*[*OF assms*(*1*)]
have *?c1* ∨ *?c2*
by *blast*
moreover
have *?c1* ⟹ *?thesis*
proof −
assume *?c1*

have *length s1* ≤ *length as* ⟹ *?thesis*
proof −
assume *length s1* ≤ *length as*
have *a # s1 = replicate* (*length s1*) *a* @ [*a*]
by (*metis* ‹*?c1*› *replicate-append-same replicate-length-same*)
hence ∃ *es. replicate* (*length as*) *a* @ [*a*] = *a # s1* @ *es*
by (*metis* ‹*length s1* ≤ *length as*› *le-add-diff-inverse list.simps*(*1*) *replicate-add replicate-append-same*)
then show *?thesis*
by (*metis S2 SL-types.simps*(*2*) *append.assoc append.right-neutral assms*(*1*) *assms*(*2*) *list.sel*(*3*) *neq-Nil-conv nslexordpI2 nslexordp-cons-cons replicate-app-Cons-same*)
qed
moreover
have *length s1* > *length as* ⟹ *?thesis*
proof −
assume *length s1* > *length as*
hence ∃ *es. s1 = replicate* (*length as*) *a* @ *a # es*
by (*metis* ‹*?c1*› *add-Suc-right less-iff-Suc-add replicate-Suc replicate-add replicate-length-same*)
then obtain *es* **where**
s1 = replicate (*length as*) *a* @ *a # es*
by *blast*
hence *S1*: *a # s1 = replicate* (*length as*) *a* @ *a # a # es*
by (*simp add*: *replicate-app-Cons-same*)
then show *?thesis*
by (*metis S2* ‹*cmp b a*› *nslexordpI1 nslexordp-cons-cons replicate-app-Cons-same*)
qed
ultimately show *?thesis*
by *linarith*

qed
moreover
have *?c2* $\Longrightarrow$ *?thesis*
proof −
assume *?c2*
then obtain *d cs ds* **where**
a # s1 = cs @ a # d # ds
cmp a d
∀ k∈set cs. k = a
by *blast*
hence *S1*: *a # s1 = replicate (length cs) a @ a # d # ds*
by (*simp add*: *replicate-length-same*)

from *transp-onD*[*OF assms*(*4*) - *assms*(*5*) - ‹*cmp b a*› ‹*cmp a d*›]
have *cmp b d*
by (*metis S1 S2 add-diff-cancel-left′ assms*(*6*,*7*) *length-Cons length-append less-Suc-eq-le*
list.simps(*1*) *nth-append-length nth-mem replicate-app-Cons-same subsetD zero-le*
zero-less-diff)
hence *length cs = length as* $\Longrightarrow$ *?thesis*
by (*metis S1 S2 nslexordpI1 nslexordp-cons-cons replicate-app-Cons-same*)
moreover
have *length as < length cs* $\Longrightarrow$ *?thesis*
proof −
assume *length as < length cs*
hence ∃ *es. replicate* (*length cs*) *a = replicate* (*length as*) *a @ a # es* ∧
(∀ *k* ∈ *set es. k = a*)
by (*metis* (*no-types, lifting*) *Cons-nth-drop-Suc S1* ‹*a # s1 = cs @ a # d # ds*› *add-Suc-right*
add-diff-cancel-left′ append-same-eq drop-replicate
in-set-replicate less-iff-Suc-add nth-mem replicate-add)
then obtain *es* **where**
replicate (*length cs*) *a = replicate* (*length as*) *a @ a # es*
∀ *k* ∈ *set es. k = a*
by *blast*
then show *?thesis*
by (*metis S1 S2* ‹*cmp b a*› *append.assoc append-Cons nslexordpI1 replicate-app-Cons-same*)
qed
moreover
have *length cs < length as* $\Longrightarrow$ *?thesis*
proof −
assume *length cs < length as*
hence ∃ *es. replicate* (*length as*) *a = replicate* (*length cs*) *a @ a # es* ∧
(∀ *k* ∈ *set es. k = a*)
by (*metis* (*no-types, lifting*) *Cons-nth-drop-Suc S2* ‹*a # s2 = as @ a # b # bs*› *add-Suc-right*
add-diff-cancel-left′ append-same-eq drop-replicate

in-set-replicate less-iff-Suc-add nth-mem replicate-add)
then obtain *es* **where**
replicate (*length as*) *a* = *replicate* (*length cs*) *a* @ *a* # *es*
∀ *k* ∈ *set es*. *k* = *a*
by *blast*
then show *?thesis*
by (*metis S1 S2* ‹*cmp a d*› *append.assoc append-Cons nslexordpI1 replicate-app-Cons-same*)
qed
ultimately show *?thesis*
by *linarith*
qed
ultimately show *?thesis*
by *blast*
qed

lemma *list-type-cons-diff-type1*:
⟦*list-type cmp* (*a* # *b* # *xs*) = *S-type*; *list-type cmp* (*b* # *xs*) = *L-type*⟧ ⟹
cmp a b
by (*simp add*: *list-type-l-type-eq list-type-s-type-eq*)

lemma *list-type-cons-diff-type2*:
⟦*list-type cmp* (*a* # *b* # *xs*) = *L-type*; *list-type cmp* (*b* # *xs*) = *S-type*⟧ ⟹
¬*cmp a b* ∧ *a* ≠ *b*
by (*simp add*: *list-type-l-type-eq list-type-s-type-eq*)

42 Suffix Type

This section contains the suffix type definition.

definition *suffix-type* :: (*'a* :: {*linorder*, *order-bot*}) *list* ⇒ *nat* ⇒ *SL-types*
where
suffix-type s i ≡
(*if list-less-ns* (*suffix s i*) (*suffix s* (*Suc i*)) *then S-type*
else L-type)

lemma *suffix-type-list-type-eq*:
suffix-type xs i = *list-type* (<) (*suffix xs i*)
by (*clarsimp simp*: *suffix-type-def list-type-def nslexordp-eq-list-less-ns*)

There are two types of suffixes (*SL-types*): *S-type* and *L-type*. An S-type suffix is a suffix that is strictly less than the suffix that occurs immediately after it, and an L-type suffix is a suffix that is strictly greater than the suffix that occurs immediately after it. The definition of less than used here is *list-less-ns*. Note that this definition of less than differs from lexicographical order(*list-less*, i.e. dictionary order, but it is equivalent when the both lists are valid (*valid-list*) as shown in ⟦*valid-list ?s1.0*; *valid-list ?s2.0*⟧ ⟹ (*?s1.0* < *?s2.0*) = *list-less-ns ?s1.0 ?s2.0*. There are three reasons for using the *list-less-ns* definition, and we explain in order of importance.

The first reason is that the original suffix types definition required a special case for the singleton suffix that only contains the sentinel symbol. While this special case makes sense in regards to the algorithms, i.e. it is necessary for the correctness of the algorithms, it does not naturally follow from the intuition of suffix types. In fact, it contradicts the intuitive definition that follows from the lexicographical order *list-less*. That is, a list that only consists of one element is always strictly greater than the empty list. With the alternate definition of less than *list-less-ns*, a proper prefix is always strictly greater, and so, a singleton list will always be strictly less than the empty list. Therefore, there is no need to have a special case for the singleton suffix that only contains the sentinel.

The second reason is that the SAIS algorithm uses a sublist order that depends on the suffix type definition (see Section S̈AIS Sublist Order)̈. This definition is perfectly valid for the algorithm, since the ordering is only used for sublist of the same list. However, the ordering is not easily understandable when applied to arbitrary list, even though it is equivalent to *list-less-ns*, which we prove in a later section. As an ordering, *list-less-ns* is much easier to understand. It is also used within the definition of *suffix-type*. Therefore, it makes more sense to reuse *list-less-ns*, rather than having multiple definitions of the same thing.

The third reason is that the original suffix types definition does not handle the case where the suffix is not terminated by sentinel symbol. The reason for this is that it is assumed that all lists are terminated by the sentinel. This assumption is very important to the SAIS algorithm as it is central to its correctness argument. That being said, in terms of elegance and consistency, using *list-less-ns* requires the least amount of special cases.

42.1 General Suffix Type Simplifications

This section contains theorems that simplify the use of the definition *suffix-type*.

lemma *suffix-type-cons-suc*:
  suffix-type (*a* # *s*) (*Suc i*) = *suffix-type s i*
  by (*simp add*: *suffix-type-def*)

lemma *suffix-type-cons-same*:
  suffix-type (*x* # *x* # *xs*) *0* = *suffix-type* (*x* # *xs*) *0*
  by (*simp add*: *list-less-ns-cons-same suffix-type-def*)

lemma *suffix-type-suffix*:
  suffix-type s i = *suffix-type* (*suffix s i*) *0*
  by (*simp add*: *suffix-type-list-type-eq*)

lemma *suffix-type-suffix-gen*:
  suffix-type (*suffix s n*) *i* = *suffix-type s* (*i* + *n*)

by (*simp add*: *suffix-type-list-type-eq*)

lemma *suffix-type-eq-Suc*:
suffix-type xs n = *suffix-type xs* (*Suc n*) $\Longrightarrow$
suffix-type xs n = *S-type* ∨ *suffix-type xs* (*Suc n*) = *L-type*
using *SL-types.exhaust* **by** *auto*

42.2 S-Type Simplifications

This subsection contains theorems about facts that can be derived S-type suffixes and vice versa.

lemma *suffix-is-bot*:
suffix s i = [*bot*] $\Longrightarrow$ *suffix-type s i* = *S-type*
by (*simp add*: *list-type-singleton suffix-type-list-type-eq*)

lemma *suffix-is-singleton*:
suffix s i = [*x*] $\Longrightarrow$ *suffix-type s i* = *S-type*
by (*simp add*: *list-type-singleton suffix-type-list-type-eq*)

lemma *suffix-type-last*:
length xs = *Suc n* $\Longrightarrow$ *suffix-type xs n* = *S-type*
by (*simp add*: *list-less-ns-nil suffix-type-def*)

lemma *s-type-list-less-ns*:
suffix-type s i = *S-type* $\longleftrightarrow$ *list-less-ns* (*suffix s i*) (*suffix s* (*Suc i*))
by (*metis SL-types.simps*(*2*) *suffix-type-def*)

lemma *nth-less-imp-s-type*:
⟦*Suc i* < *length s*; *s* ! *i* < *s* ! *Suc i*⟧ $\Longrightarrow$ *suffix-type s i* = *S-type*
by (*metis Cons-nth-drop-Suc Suc-lessD less-imp-le list-less-ns-cons neq-iff s-type-list-less-ns*)

lemma *sl-type-hd-less*:
⟦*Suc i* < *length s*; *hd* (*suffix s i*) < *hd* (*suffix s* (*Suc i*))⟧ $\Longrightarrow$
suffix-type s i = *S-type*
by (*simp add*: *hd-drop-conv-nth nth-less-imp-s-type*)

lemma *suffix-type-cons-less*:
x < *y* $\Longrightarrow$ *suffix-type* (*x* # *y* # *xs*) *0* = *S-type*
by (*clarsimp simp*: *suffix-type-def list-less-ns-cons-diff*)

lemma *suffix-type-s-bound*:
suffix-type s i = *S-type* $\Longrightarrow$ *i* < *length s*
using *ordlistns.less-asym s-type-list-less-ns* **by** *fastforce*

lemma *s-type-letter-le-Suc*:
⟦*Suc i* < *length T*; *suffix-type T i* = *S-type*⟧ $\Longrightarrow$
T ! *i* ≤ *T* ! (*Suc i*)
by (*metis Cons-nth-drop-Suc Suc-lessD leI list-less-ns-cons-diff ordlistns.less-asym s-type-list-less-ns*)

lemma *s-type-ex*:
  assumes *suffix-type* $(x \# xs)$ $0 = S\text{-}type$
  shows $(\forall a \in set\ xs.\ a = x) \vee (\exists b\ as\ bs.\ x \# xs = as\ @\ x \# b \# bs \wedge x < b \wedge (\forall k \in set\ as.\ k = x))$
  by (*metis assms drop0 list-type-s-ex suffix-type-list-type-eq*)

42.3 L-Type Simplifications

This subsection contains theorems about facts that can be derived from L-type suffixes and vice versa.

lemma *suffix-is-nil*:
  suffix s i $= [] \Longrightarrow$ *suffix-type s i* $= L\text{-}type$
  by (*clarsimp simp*: *suffix-type-def split*: *if-splits*)

lemma *l-type-list-less-ns*:
  suffix-type s i $= L\text{-}type \longleftrightarrow$ *list-less-ns* (*suffix s* (*Suc i*)) (*suffix s i*) $\vee$ *suffix s i* $= []$
  by (*metis Cons-nth-drop-Suc SL-types.distinct*(*1*) *drop-Nil drop-eq-Nil linorder-le-less-linear not-Cons-self2 ordlistns.less-imp-not-less ordlistns.neqE suffix-type-def suffix-type-suffix*)

lemma *nth-gr-imp-l-type*:
  $[\![Suc\ i < length\ s;\ s\ !\ i > s\ !\ Suc\ i]\!] \Longrightarrow$ *suffix-type s i* $= L\text{-}type$
  by (*metis Cons-nth-drop-Suc Suc-lessD list-less-ns-cons-diff ordlistns.less-asym suffix-type-def*)

lemma *sl-type-hd-greater*:
  $[\![Suc\ i < length\ s;\ hd\ (suffix\ s\ i) > hd\ (suffix\ s\ (Suc\ i))]\!] \Longrightarrow$
   suffix-type s i $= L\text{-}type$
  by (*simp add*: *hd-drop-conv-nth nth-gr-imp-l-type*)

lemma *suffix-type-cons-greater*:
  $x > y \Longrightarrow$ *suffix-type* $(x \# y \# xs)$ $0 = L\text{-}type$
  by (*simp add*: *list-type-cons-diff2 suffix-type-list-type-eq*)

lemma *l-type-letter-gre-Suc*:
  $[\![i < length\ T;\ suffix\text{-}type\ T\ i = L\text{-}type]\!] \Longrightarrow$
    $T\ !\ (Suc\ i) \leq T\ !\ i$
  by (*metis SL-types.distinct*(*1*) *Suc-lessI not-less nth-less-imp-s-type suffix-type-last*)

lemma *l-type-ex*:
  assumes *suffix-type* $(x \# xs)$ $0 = L\text{-}type$
  shows $\exists b\ as\ bs.\ x \# xs = as\ @\ x \# b \# bs \wedge x > b \wedge (\forall k \in set\ as.\ k = x)$
  by (*metis assms drop0 drop-Suc-Cons l-type-list-less-ns list.discI list-less-ns-cons1-exE*)

An overlooked property, but one that is crucial for completeness of the SAIS algorithm

lemma *suffix-max-hd-is-l-type*:

```
  assumes valid-list s
  and     i < length s
  and     length s > Suc 0
  and     hd (suffix s i) = Max (set s)
shows suffix-type s i = L-type
  using assms
proof (induct s arbitrary: i)
  case Nil
  then show ?case
    by simp
next
  case (Cons a s i)
  note IH = this
  show ?case
  proof (cases s)
    case Nil
    then show ?thesis
      using IH(4) by auto
  next
    case (Cons b xs)
    assume s = b # xs
    show ?thesis
    proof (cases xs)
      case Nil
      hence s = [b]
        by (simp add: local.Cons)
      moreover
      have b = bot
        by (metis IH(2) last.simps local.Cons local.Nil not-Cons-self
                valid-list-iff-butlast-app-last)
      moreover
      have a > b
      by (metis IH(2,4) One-nat-def antisym-conv3 bot.extremum-strict nth-Cons-0
valid-list-def
                zero-less-diff calculation(2))
      ultimately show ?thesis
      by (metis IH(2,3,5) Max-greD add-diff-cancel-left' last-suffix-index length-Cons
less-Suc0
              linorder-not-less list.size(3) not-less-eq not-less-iff-gr-or-eq nth-Cons-0
                plus-1-eq-Suc suffix-type-cons-greater)
    next
      case (Cons c ys)
      hence s = b # c # ys
        by (simp add: ‹s = b # xs›)
      show ?thesis
      proof (cases i)
        case 0
        hence a ≥ b
          by (metis IH(4,5) List.finite-set Max-ge ‹s = b # xs› drop0 list.sel(1)
```

nth-Cons-0
          nth-Cons-Suc nth-mem)
    hence $a > b \vee a = b$
      using *antisym-conv1* **by** *blast*
    then show *?thesis*
    proof
      assume $b < a$
      then show *?thesis*
        by (*simp add*: *0* ‹$s = b \# xs$› *suffix-type-cons-greater*)
    next
      assume $a = b$
      hence *Max* (*set s*) = *b*
        using *0 IH*(*5*) ‹$s = b \# xs$› **by** *auto*
      with *IH*(*1*)[*of 0*]
      have *suffix-type s 0* = *L-type*
          by (*metis IH*(*2*) *Suc-less-eq2* ‹$s = b \# xs$› *drop0 drop-Suc-Cons*
length-Cons list.sel(*1*)
          local.Cons valid-suffix zero-less-Suc)
      then show *?thesis*
        by (*simp add*: *0* ‹$a = b$› ‹$s = b \# xs$› *suffix-type-cons-same*)
    qed
  next
    case (*Suc n*)
    assume $i = Suc\ n$
    have *valid-list s*
      using *IH*(*2*) ‹$s = b \# xs$› *valid-list-consD* **by** *blast*
    moreover
    have $n < length\ s$
      using *IH*(*3*) *Suc* **by** *auto*
    moreover
    have $Suc\ 0 < length\ s$
      by (*simp add*: ‹$s = b \# xs$› *local.Cons*)
    moreover
    {
      have *hd* (*suffix s n*) = *hd* (*suffix* (*a* # *s*) *i*)
        using *Suc* **by** *fastforce*
      moreover
      have *hd* (*suffix s n*) ∈ *set s*
        by (*simp add*: ‹$n < length\ s$› *hd-drop-conv-nth*)
      with *IH*(*5*)
      have *Max* (*set* (*a* # *s*)) ∈ *set s*
        using *calculation* **by** *argo*
      hence *Max* (*set* (*a* # *s*)) = *Max* (*set s*)
        using *IH*(*5*) *max.cobounded1*[*of a Max* (*set s*)]
        by (*metis List.finite-set Max-greD Max-insert Suc-lessD* ‹$Suc\ 0 < length$
s›
          ‹$n < length\ s$› *calculation hd-drop-conv-nth length-greater-0-conv*
list.set(*2*)
          max-def set-empty)

```
      ultimately have hd (suffix s n) = Max (set s)
        using IH(5) by presburger
      }
      ultimately have suffix-type s n = L-type
        using Cons.hyps by blast
      then show ?thesis
        by (simp add: Suc suffix-type-cons-suc)
    qed
  qed
 qed
qed
```

42.4 General Suffix Type Theories

This subsection contains the background theory needed to prove that computing the suffix types of a list can be achieved in linear time by starting from the end of the list (lemma 1, Ko et al., JDA 2005).

The main intuition is that the suffix type of the (i+1)th suffix is known and the ith suffix starts with same symbol of the (i+1)th suffix, then the ith suffix will have the same type.

```
theorem sl-type-hd-equal:
  ⟦Suc i < length s; hd (suffix s i) = hd (suffix s (Suc i))⟧ ⟹
   suffix-type s i = suffix-type s (Suc i)
   by (metis Cons-nth-drop-Suc Suc-lessD hd-drop-conv-nth l-type-letter-gre-Suc
list-less-ns-cons
         suffix-type-def)

corollary sl-type-prefix-equal:
  ⟦i + n ≤ length s; ∀ j < n. hd (suffix s (i + j)) = hd (suffix s i)⟧ ⟹
   ∀ j < n. suffix-type s (i + j) = suffix-type s i
proof (induct n)
  case 0
  then show ?case
    by blast
next
  case (Suc n)
  note IH = this
  hence ∀ j<n. suffix-type s (i + j) = suffix-type s i
    by (metis add-Suc-right less-Suc-eq linorder-not-less)
  show ?case
  proof safe
    fix j
    assume j < Suc n
    then show suffix-type s (i + j) = suffix-type s i
    proof (cases j < n)
      assume j < n
      then show ?thesis
        by (simp add: ‹∀ j<n. suffix-type s (i + j) = suffix-type s i›)
```

next
assume $\neg\ j < n$
hence $j = n$
using ‹$j < Suc\ n$› **by** *auto*
show *?thesis*
proof (*cases j*)
case *0*
then show *?thesis*
by *simp*
next
case (*Suc m*)
then show *?thesis*
by (*metis Suc.prems*(*1*,*2*) *Suc-lessD Suc-less-SucD* ‹$j < Suc\ n$› ‹$j = n$› *add-Suc-right*
‹$\forall j<n.$ *suffix-type s* $(i + j) =$ *suffix-type s i*› *le-imp-less-Suc sl-type-hd-equal*)
qed
qed
qed
qed

corollary *sl-type-prefix-equal-nth*:
$[\![i + n \leq length\ s; \forall j < n.\ (suffix\ s\ i)\ !\ j = (suffix\ s\ i)\ !\ 0]\!] \Longrightarrow$
$\forall j < n.$ *suffix-type s* $(i + j) =$ *suffix-type s i*
by (*rule sl-type-prefix-equal*, *assumption*, *clarsimp simp*: *hd-conv-nth*)

corollary *sl-type-prefix-replicate*:
$\forall i < n.$ *suffix-type* (*replicate n a @ as*) $i =$ *suffix-type* (*replicate n a @ as*) *0*
by (*rule sl-type-prefix-equal-nth*[**where** $i = 0$, *simplified*]; *clarsimp simp*: *nth-append*)

lemma *suffix-type-neq*:
$[\![$*suffix-type T j* $\neq$ *suffix-type T* (*Suc j*); *Suc j* $<$ *length T*$]\!] \Longrightarrow T\ !\ j \neq T\ !\ Suc\ j$
by (*metis Cons-nth-drop-Suc Suc-lessD l-type-letter-gre-Suc list-less-ns-cons suffix-type-def*)

42.5 S/L-Type Ordering

This section contains the crucial theorem that L-type suffixes are always less than S-type suffixes if they start with the same symbol (lemma 2, Ko et al., JDA 2005).

theorem *l-less-than-s-type-general*:
assumes *suffix-type* (*a # s1*) *0* = *S-type*
and *suffix-type* (*a # s2*) *0* = *L-type*
shows *list-less-ns* (*a # s2*) (*a # s1*)
proof −
from *suffix-type-list-type-eq*[*of a # s1 0*]
have *suffix-type* (*a # s1*) *0* = *list-type* (<) (*a # s1*)
by *simp*
hence *list-type* (<) (*a # s1*) = *S-type*

    using *assms*(*1*) **by** *auto*
  moreover
  from *suffix-type-list-type-eq*[*of a # s2 0*]
  have *suffix-type* (*a # s2*) *0* = *list-type* (<) (*a # s2*)
    by *simp*
  hence *list-type* (<) (*a # s2*) = *L-type*
    using *assms*(*2*) **by** *auto*
  ultimately show *?thesis*
    using *l-less-than-s-type-list-type*[*of* (<) *a s1 s2*]
   by (*meson UNIV-I nslexordp-eq-list-less-ns-app top-greatest totalp-on-less transp-on-less*)
qed

corollary *l-less-than-s-type-suffix*:
  assumes *i* < *length s*
  and *j* < *length s*
  and *s* ! *i* = *s* ! *j*
  and *suffix-type s i* = *S-type*
  and *suffix-type s j* = *L-type*
shows *list-less-ns* (*suffix s j*) (*suffix s i*)
  by (*metis Cons-nth-drop-Suc assms l-less-than-s-type-general suffix-type-suffix*)

theorem *l-less-than-s-type*:
  assumes *valid-list s*
  and *i* < *length s*
  and *j* < *length s*
  and *hd* (*suffix s i*) = *hd* (*suffix s j*)
  and *suffix-type s i* = *S-type*
  and *suffix-type s j* = *L-type*
shows *list-less-ns* (*suffix s j*) (*suffix s i*)
  by (*metis hd-drop-conv-nth assms*(*2*−) *l-less-than-s-type-suffix*)

corollary (**in** *Suffix-Array-General*) *same-hd-s-after-l*:
  assumes *valid-list*: *valid-list s*
  and *i-less-len-s*: *i* < *length s*
  and *j-less-len-s*: *j* < *length s*
  and *i-neq-j*: *i* ≠ *j*
  and *suf-i-type*: *suffix-type s* ((*sa s*)! *i*) = *L-type*
  and *suf-j-type*: *suffix-type s* ((*sa s*)! *j*) = *S-type*
  and *hd-eq*: *hd* (*suffix s* ((*sa s*) ! *i*)) = *hd* (*suffix s* ((*sa s*) ! *j*))
shows *i* < *j*
proof −
  have *A*: (*sa s*) ! *i* < *length s*
    using *i-less-len-s sa-nth-ex* **by** *auto*

  from *sa-set-upt*[**where** *s* = *s*] *sa-length*[**where** *s* = *s*] *j-less-len-s*
  have *B*: (*sa s*) ! *j* < *length s*
    by (*metis atLeast0LessThan lessThan-iff nth-mem*)

  from *l-less-than-s-type*[*OF valid-list B A hd-eq*[*symmetric*] *suf-j-type suf-i-type*]

```
  have suf-i-less-suf-j: list-less-ns (suffix s ((sa s)! i)) (suffix s ((sa s)! j))
    by simp

  from sorted-nth-less-mono[OF strict-sorted-imp-sorted[OF sa-g-sorted],
                              simplified length-map sa-length,
                              OF i-less-len-s j-less-len-s i-neq-j]
    nth-map[where f = suffix s and xs = sa s, simplified sa-length, OF i-less-len-s]
    nth-map[where f = suffix s and xs = sa s, simplified sa-length, OF j-less-len-s]
      valid-list-list-less-equiv-list-less-ns[OF valid-suffix,
                                              OF valid-list A valid-suffix,
                                              OF valid-list B,
                                              THEN iffD2,
                                              OF suf-i-less-suf-j]
  show ?thesis by simp
qed
```

42.6 Implementation of Suffix Type Computation

This subsection contain a shallow embedding of a function that would compute the suffix types for a list.

```
fun abs-get-suffix-types :: ('a :: {linorder, order-bot}) list ⇒ SL-types list
  where
abs-get-suffix-types [] = [] |
abs-get-suffix-types ([-]) = [S-type] |
abs-get-suffix-types (a # b # xs) =
  (let ys = abs-get-suffix-types (b # xs)
   in
     (if a < b then S-type # ys
      else if a > b then L-type # ys
      else hd (ys) # ys))

lemma length-abs-get-suffix-types:
  length (abs-get-suffix-types s) = length s
  by (induct s rule: abs-get-suffix-types.induct; clarsimp simp: Let-def)

lemma abs-get-suffix-types-correct-nth:
  i < length s ⟹ abs-get-suffix-types s ! i = suffix-type s i
proof (induct s arbitrary: i rule: abs-get-suffix-types.induct)
  case 1
  then show ?case
    by simp
next
  case (2 uu i)
  then show ?case
    by (simp add: suffix-is-singleton)
next
  case (3 a b xs i)
  note IH = this
```

have $i \neq 0 \Longrightarrow$ *?case*
proof $-$
  assume $i \neq 0$
  hence $\exists n.\ i = Suc\ n$
    using *not0-implies-Suc* **by** *blast*
  then obtain n **where**
    $i = Suc\ n$
    by *blast*
  hence *abs-get-suffix-types* $(a \# b \# xs)\ !\ i =$ *abs-get-suffix-types* $(b \# xs)\ !\ n$
    by (*clarsimp simp*: *Let-def*)
  moreover
  have *suffix-type* $(a \# b \# xs)\ i =$ *suffix-type* $(b \# xs)\ n$
    by (*simp add*: ‹$i = Suc\ n$› *suffix-type-cons-suc*)
  moreover
  from *IH*(*1*)[*of n*] *IH*(*2*) ‹$i = Suc\ n$›
  have *abs-get-suffix-types* $(b \# xs)\ !\ n =$ *suffix-type* $(b \# xs)\ n$
    by *simp*
  ultimately show *?thesis*
    by *simp*
qed
moreover
have $i = 0 \Longrightarrow$ *?case*
proof $-$
  assume $i = 0$

  have $a < b \vee b < a \vee a = b$
    by *fastforce*
  then show *?case*
  proof (*elim disjE*)
    assume $a < b$
    then show *?case*
      by (*simp add*: ‹$i = 0$› *suffix-type-cons-less*)
  next
    assume $b < a$
    then show *?case*
      using ‹$i = 0$› *order-less-imp-triv suffix-type-cons-greater* **by** *fastforce*
  next
    assume $a = b$

    have *abs-get-suffix-types* $(b \# xs) \neq []$
      by (*metis Zero-neq-Suc length-Cons length-abs-get-suffix-types list.size*(*3*))
    hence *abs-get-suffix-types* $(a \# b \# xs)\ !\ i =$ *abs-get-suffix-types* $(b \# xs)\ !\ 0$
      unfolding *abs-get-suffix-types.simps*(*3*)[*of a b xs*, *simplified Let-def*]
      by (*simp add*: ‹$a = b$› ‹$i = 0$› *hd-conv-nth*)
    moreover
    have *suffix-type* $(a \# b \# xs)\ i =$ *suffix-type* $(b \# xs)\ 0$
      by (*simp add*: ‹$a = b$› ‹$i = 0$› *suffix-type-cons-same*)
    ultimately show *?case*
      using *IH*(*1*)[*of 0*, *simplified*] **by** *presburger*

qed
qed
ultimately show *?case*
by *blast*
qed

lemma *get-suffix-types-correct*:
$\forall\, i < \mathit{length}\ s.\ (\mathit{abs\text{-}get\text{-}suffix\text{-}types}\ s)\ !\ i = \mathit{suffix\text{-}type}\ s\ i$
by (*simp add*: *abs-get-suffix-types-correct-nth*)

43 SAIS Sublist Order

This section contains the sublist ordering used in SAIS (definition 2.3, Nong et al., DCC 2009). Note that this generalised so that it is not a ternary relation but a binary relation.

fun *ss-order-less* :: ($'a$ :: {*linorder*, *order-bot*}) *list* $\Rightarrow$ $'a$ *list* $\Rightarrow$ *bool*
where
ss-order-less [] - = *False* |
ss-order-less - [] = *True* |
ss-order-less (*a* # *as*) (*b* # *bs*) =
(*if a* < *b then True*
else if a > *b then False*
else if suffix-type (*a* # *as*) *0* = *suffix-type* (*b* # *bs*) *0 then ss-order-less as bs*
else if suffix-type (*a* # *as*) *0* = *L-type then True*
else False)

As described in section "Suffix Type", the SAIS sublist ordering (*ss-order-less*) is equivalent to *list-less-ns*.

lemma *ss-order-less-equiv-list-less-ns*:
ss-order-less s1 s2 = *list-less-ns s1 s2*
proof (*induct rule*: *ss-order-less.induct*)
case (*1 uu*)
then show *?case*
by *simp*
next
case (*2 v va*)
then show *?case*
by (*simp add*: *list-less-ns-nil*)
next
case (*3 a as b bs*)
then show *?case*
by (*metis antisym-conv3 l-less-than-s-type-general list-less-ns-cons-diff list-less-ns-cons-same ordlistns.less-asym ss-order-less.simps*(*3*) *suffix-type-def*)
qed

44 Sorting

lemma *sorted-letters-s-types*:

assumes $\forall k \geq i.\ k < j \longrightarrow$ *suffix-type* $T\ k = S\text{-}type$
and $j \leq$ *length* T
shows *sorted* (*list-slice* $T\ i\ j$)
proof (*intro sorted-iff-nth-mono*[*THEN iffD2*] *allI impI*)
fix $x\ y$
assume $x \leq y\ y <$ *length* (*list-slice* $T\ i\ j$)

have *list-slice* $T\ i\ j\ !\ x = T\ !\ (i + x)$
by (*meson* ‹$x \leq y$› ‹$y <$ *length* (*list-slice* $T\ i\ j$)› *dual-order.strict-trans2* *nth-list-slice*)
moreover
have *list-slice* $T\ i\ j\ !\ y = T\ !\ (i + y)$
using ‹$y <$ *length* (*list-slice* $T\ i\ j$)› *nth-list-slice*
by *blast*
moreover
have $i + y < j$
using ‹$y <$ *length* (*list-slice* $T\ i\ j$)›
by (*simp add*: *assms*(*2*))
have $i + x \leq i + y$
by (*simp add*: ‹$x \leq y$›)
with ‹$i + y < j$›
have $T\ !\ (i + x) \leq T\ !\ (i + y)$
proof (*induct* $y - x$ *arbitrary*: $x\ y$)
case *0*
then show *?case* **by** *simp*
next
case (*Suc* z)
note $IH = this$
from *IH*(*2*)
have $\exists y'.\ y = Suc\ y'$
by *presburger*
then obtain y' **where**
$y = Suc\ y'$
by *blast*
hence $z = y' - x$
using *IH*(*2*) **by** *linarith*
moreover
have $i + y' < j$
using *Suc.prems*(*1*) ‹$y = Suc\ y'$› **by** *linarith*
moreover
have $i + x \leq i + y'$
using *Suc.hyps*(*2*) ‹$y = Suc\ y'$› **by** *linarith*
ultimately have $T\ !\ (i + x) \leq T\ !\ (i + y')$
using *Suc.hyps*(*1*) **by** *blast*
moreover
from *assms*(*1*)
have *suffix-type* $T\ (i + y') = S\text{-}type$
by (*simp add*: ‹$i + y' < j$›)
hence $T\ !\ (i + y') \leq T\ !\ (i + y)$

using *Suc.prems*(*1*) ‹$y = Suc\ y'$› *assms*(*2*) *less-le-trans s-type-letter-le-Suc*
by *fastforce*
ultimately show *?case*
using *order.trans* **by** *blast*
qed
ultimately show *list-slice T i j* ! $x \leq$ *list-slice T i j* ! y
by *simp*
qed

lemma *sorted-letters-l-types*:
assumes $\forall k \geq i.\ k < j \longrightarrow$ *suffix-type T k = L-type*
and $j \leq$ *length T*
shows *sorted* ((*rev* (*list-slice T i j*)))
proof (*intro sorted-rev-iff-nth-mono*[*THEN iffD2*] *allI impI*)
fix *x y*
assume $x \leq y$ $y <$ *length* (*list-slice T i j*)

have *length* (*list-slice T i j*) $= j - i$
by (*simp add*: *assms*(*2*))

have $i + y < j$
using ‹*length* (*list-slice T i j*) $= j - i$› ‹$y <$ *length* (*list-slice T i j*)› **by** *linarith*
with ‹$x \leq y$›
have $T\ !\ (i + y) \leq T\ !\ (i + x)$
proof (*induct* $y - x$ *arbitrary*: *x y*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc z x y*)
note *IH = this*
have $\exists y'.\ y = Suc\ y' \wedge x \leq y' \wedge z = y' - x$
by (*metis Suc.hyps*(*2*) *Suc.prems*(*1*) *add-Suc-right add-diff-cancel-left′ add-diff-cancel-right′ diff-le-self ordered-cancel-comm-monoid-diff-class.add-diff-inverse*)
then obtain y' **where**
$y = Suc\ y'$ $x \leq y'$ $z = y' - x$
by *blast*
with *IH*(*1*) *IH*(*4*)
have $T\ !\ (i + x) \geq T\ !\ (i + y')$
by *simp*
moreover
from *assms*(*1*)
have *suffix-type T* $(i + y')$ = *L-type*
using *Suc.prems*(*2*) ‹$y = Suc\ y'$› **by** *auto*
hence $T\ !\ (i + y') \geq T\ !\ (i + y)$
using *Suc.prems*(*2*) ‹$y = Suc\ y'$› *assms*(*2*) *nth-less-imp-s-type order-less-le-trans*
by *fastforce*
ultimately show *?case*
by *auto*
qed

moreover
have *list-slice T i j* ! $x =$ T ! $(i + x)$
by (*metis* ‹$x \leq y$› ‹$y <$ *length* (*list-slice T i j*)› *nth-list-slice order-le-less-trans*)
moreover
have *list-slice T i j* ! $y = T$! $(i + y)$
using ‹$y <$ *length* (*list-slice T i j*)› *nth-list-slice* **by** *blast*
ultimately show *list-slice T i j* ! $y \leq$ *list-slice T i j* ! x
by *presburger*
qed

45 LMS-Types

This section contains the definition of an LMS-type; standing for large, middle and small. It also contains lemmas pertaining to these types.

definition
abs-is-lms :: (*'a* :: {*linorder*, *order-bot*}) *list* ⇒ *nat* ⇒ *bool*
where
abs-is-lms s i ≡
(*suffix-type s i* = *S-type*) ∧
(∃ *j*. *i* = *Suc j* ∧
suffix-type s j = *L-type*)

LMS-types are subtypes of *S-type*. This is because these are *S-type*, but they are also immediately succeed *L-type*.

45.1 LMS-Type Simplifications

This subsection contains theorems about facts that can be derived from the *abs-is-lms* definition and vice versa.

lemma *lms-type-list-less-ns*:
abs-is-lms s i = (∃ *j*. *i* = *Suc j* ∧ *list-less-ns* (*suffix s i*) (*suffix s j*) ∧
list-less-ns (*suffix s i*) (*suffix s* (*Suc i*)))
by (*metis SL-types.simps*(*2*) *abs-is-lms-def l-type-list-less-ns ordlistns.antisym-conv3*

s-type-list-less-ns)

lemma *abs-is-lms-0*:
¬*abs-is-lms s 0*
apply (*clarsimp simp*: *abs-is-lms-def*)
done

lemma *abs-is-lms-cons-suc*:
$i > 0 \Longrightarrow$ *abs-is-lms* (*a* # *s*) (*Suc i*) = *abs-is-lms s i*
apply (*drule gr0-implies-Suc*; *clarsimp*)
apply (*clarsimp simp*: *abs-is-lms-def suffix-type-cons-suc*)
done

lemma *i-s-type-imp-Suc-i-not-lms*:
suffix-type s i = *S-type* $\Longrightarrow$ $\neg$*abs-is-lms s* (*Suc i*)
by (*simp add*: *abs-is-lms-def*)

lemma *suffix-type-same-imp-not-lms*:
suffix-type s i = *suffix-type s* (*Suc i*) $\Longrightarrow$ $\neg$*abs-is-lms s* (*Suc i*)
by (*simp add*: *abs-is-lms-def*)

lemma *abs-is-lms-consec*:
abs-is-lms xs i $\Longrightarrow$ $\neg$*abs-is-lms xs* (*Suc i*)
abs-is-lms xs (*Suc i*) $\Longrightarrow$ $\neg$*abs-is-lms xs i*
by (*clarsimp simp*: *abs-is-lms-def*)+

lemma *abs-is-lms-gre-length*:
n $\geq$ *length xs* $\Longrightarrow$ $\neg$*abs-is-lms xs n*
by (*metis SL-types.distinct*(*1*) *drop-eq-Nil abs-is-lms-def l-type-list-less-ns*)

lemma *abs-is-lms-suffix*:
abs-is-lms (*suffix s n*) *i* $\Longrightarrow$ *abs-is-lms s* (*i* + *n*)
by (*clarsimp simp*: *abs-is-lms-def suffix-type-suffix-gen*)

lemma *abs-is-lms-i-gr-0*:
i > *0* $\Longrightarrow$ *abs-is-lms* (*suffix s n*) *i* = *abs-is-lms s* (*i* + *n*)
apply *safe*
 apply (*erule abs-is-lms-suffix*)
apply (*clarsimp simp*: *abs-is-lms-def*)
apply (*rule conjI*)
 apply (*subst suffix-type-suffix-gen*; *simp*)
by (*metis* (*no-types*, *opaque-lifting*) *add.commute add-Suc-right add-right-cancel*
less-Suc-eq-le
less-iff-Suc-add ordered-cancel-comm-monoid-diff-class.diff-add suffix-type-suffix-gen)

lemma *set-abs-is-lms-suffix*:
$\{$*i*. *abs-is-lms* (*suffix s n*) (*i* − *n*)$\}$ = $\{$*i*. *abs-is-lms s i* $\wedge$ *i* > *n*$\}$
apply *safe*
  apply (*metis abs-is-lms-0 abs-is-lms-suffix le-less-linear nat-diff-split*
ordered-cancel-comm-monoid-diff-class.diff-add)
 apply (*metis bot-nat-0.not-eq-extremum abs-is-lms-0 zero-less-diff*)
apply (*cases n*)
 apply *clarsimp*
by (*metis* (*no-types*, *lifting*) *Suc-diff-le abs-is-lms-def less-Suc-eq-le less-or-eq-imp-le*
ordered-cancel-comm-monoid-diff-class.diff-add suffix-type-suffix-gen)

lemma *abs-is-lms-set-less-length*:
n $\geq$ *length xs* $\Longrightarrow$ $\{$*i*. *abs-is-lms xs i* $\wedge$ *i* < *n*$\}$ = $\{$*i*. *abs-is-lms xs i*$\}$
by (*meson dual-order.trans abs-is-lms-gre-length le-less-linear*)

lemma *abs-is-lms-suffix-Suc*:
abs-is-lms (*suffix s n*) (*Suc i*) = *abs-is-lms s* (*Suc* (*i* + *n*))

apply *safe*
apply (*drule abs-is-lms-suffix*; *simp*)
apply (*clarsimp simp*: *abs-is-lms-def*)
apply (*subst suffix-type-suffix-gen*)+
apply *simp*
done

45.2 LMS-Type Sets and Subsets

This subsection contains lemmas about sets and subsets of LMS-types.

lemma *set-lms-gr-0*:
$\{i.\ abs\text{-}is\text{-}lms\ xs\ i \wedge 0 < i\} = \{i.\ abs\text{-}is\text{-}lms\ xs\ i\}$
using *bot-nat-0.not-eq-extremum abs-is-lms-0* **by** *blast*

lemma *set-lms-n-subset*:
$\{i.\ abs\text{-}is\text{-}lms\ xs\ i \wedge i > n\} \subseteq \{i.\ abs\text{-}is\text{-}lms\ xs\ i\}$
by *blast*

lemma *set-lms-Suc-subset*:
$\{i.\ abs\text{-}is\text{-}lms\ xs\ i \wedge i > Suc\ n\} \subseteq \{i.\ abs\text{-}is\text{-}lms\ xs\ i \wedge i > n\}$
by (*simp add*: *Collect-mono*)

lemma *set-lms-Suc-insert*:
$abs\text{-}is\text{-}lms\ xs\ (Suc\ n) \Longrightarrow \{i.\ abs\text{-}is\text{-}lms\ xs\ i \wedge i > n\} = insert\ (Suc\ n)\ \{i.\ abs\text{-}is\text{-}lms\ xs\ i \wedge i > Suc\ n\}$
using *Collect-cong* **by** *auto*

lemma *lms-finite*:
$finite\ \{i.\ abs\text{-}is\text{-}lms\ xs\ i\}$
by (*metis finite-nat-set-iff-bounded abs-is-lms-def mem-Collect-eq suffix-type-s-bound*)

lemma *lms-set-empty*:
$[\![length\ xs = Suc\ n;\ m \geq n]\!] \Longrightarrow \{i.\ abs\text{-}is\text{-}lms\ xs\ i \wedge i > m\ \} = \{\}$
by (*metis* (*no-types*, *lifting*) *Collect-empty-eq Suc-leI diff-diff-cancel abs-is-lms-gre-length less-imp-diff-less*)

45.3 Implementation of LMS-Types Computation

This section contains a shallow embedding of a function that would compute all the LMS-types of an ordered list.

fun *get-lms* :: ($'a$ :: {*linorder*, *order-bot*}) *list* $\Rightarrow$ *nat* $\Rightarrow$ *nat list*
where
$get\text{-}lms\ xs\ 0 = []\ |$
$get\text{-}lms\ xs\ (Suc\ n) = (if\ abs\text{-}is\text{-}lms\ xs\ n\ then\ n\ \#\ get\text{-}lms\ xs\ n\ else\ get\text{-}lms\ xs\ n)$

lemma *get-lms-correct*:
$get\text{-}lms\ xs\ n = rev\ (filter\ (abs\text{-}is\text{-}lms\ xs)\ [0..{<}n])$
apply (*induct n*)

apply *simp*
apply *clarsimp*
done

45.3.1 Properties

This subsection contains miscellaneous lemmas about facts that can be derived from the shallow embedding and vice versa.

lemma *get-lms-element-bound*:
$x \in set\ (get\text{-}lms\ xs\ n) \Longrightarrow x < n \wedge x > 0$
apply (*induct n*; *simp*)
apply (*clarsimp split*: *if-splits*)
apply (*erule disjE*; *clarsimp*)
apply (*cases x*; *clarsimp simp*: *abs-is-lms-0*)
done

lemma *distinct-get-lms*:
distinct (*get-lms xs n*)
apply (*induct n*; *clarsimp*)
apply (*drule get-lms-element-bound*)
by *blast*

lemma *get-lms-abs-is-lms*:
$x \in set\ (get\text{-}lms\ xs\ n) \longleftrightarrow abs\text{-}is\text{-}lms\ xs\ x \wedge x < n$
apply (*subst get-lms-correct*)
apply *clarsimp*
by *blast*

lemma *lms-le-length*:
$x \in set\ (get\text{-}lms\ xs\ n) \Longrightarrow x < length\ xs$
by (*simp add*: *get-lms-abs-is-lms abs-is-lms-def suffix-type-s-bound*)

lemma *get-lms-set*:
$set\ (get\text{-}lms\ xs\ n) = \{i.\ abs\text{-}is\text{-}lms\ xs\ i \wedge i < n\}$
apply (*induct n*)
apply *simp*
apply *safe*
using *get-lms-abs-is-lms* **by** *blast+*

lemma *get-lms-set-n-gre-length*:
$n \geq length\ xs \Longrightarrow set\ (get\text{-}lms\ xs\ n) = \{i.\ abs\text{-}is\text{-}lms\ xs\ i\}$
apply (*simp add*: *get-lms-set*)
by (*meson dual-order.trans abs-is-lms-gre-length not-less*)

45.4 Cardinality LMS-Types

This section contains lemmas about how many LMS-types exist (lemma 2.1, Nonge et al., DCC2009). These lemmas are particularly important when proving that the SAIS is O(n) in space (bytes) and time complexity (lemma 3.1, Nong et al., DCC 2009).

lemma *num-lms-bound-1*:
  length (*get-lms xs n*) $\leq$ *n div 2*
proof –
  have *card* (*set* (*get-lms xs n*)) $\leq$ *n div 2*
  proof (*intro ballI subset-upt-no-Suc*[*of set* (*get-lms xs n*) *n*])
    show *set* (*get-lms xs n*) $\subseteq$ {*1..<n*}
      by (*simp add*: *Suc-leI get-lms-element-bound subset-code*(*1*))
  next
    fix *x*
    assume *x* $\in$ *set* (*get-lms xs n*)
    hence *abs-is-lms xs x*
      by (*simp add*: *get-lms-abs-is-lms*)
    then show *Suc x* $\notin$ *set* (*get-lms xs n*)
      using *get-lms-abs-is-lms abs-is-lms-consec*(*2*) **by** *blast*
  qed
  with *distinct-card*[*OF distinct-get-lms*[*of xs n*]]
  show *?thesis*
    by *presburger*
qed

lemma *num-lms-bound-2*:
  length (*get-lms xs n*) $\leq$ *length xs div 2*
proof –
  have *card* (*set* (*get-lms xs n*)) $\leq$ *length xs div 2*
  proof (*intro ballI subset-upt-no-Suc*[*of set* (*get-lms xs n*) *length xs*])
    show *set* (*get-lms xs n*) $\subseteq$ {*1..<length xs*}
      by (*metis One-nat-def Suc-leI atLeastLessThan-iff get-lms-element-bound lms-le-length subsetI*)
  next
    fix *x*
    assume *x* $\in$ *set* (*get-lms xs n*)
    hence *abs-is-lms xs x*
      by (*simp add*: *get-lms-abs-is-lms*)
    then show *Suc x* $\notin$ *set* (*get-lms xs n*)
      using *get-lms-abs-is-lms abs-is-lms-consec*(*2*) **by** *blast*
  qed
  with *distinct-card*[*OF distinct-get-lms*[*of xs n*]]
  show *?thesis*
    by *simp*
qed

lemma *card-abs-is-lms-bound*:
  xs $\neq$ [] $\Longrightarrow$ *card* {*i. abs-is-lms xs i*} < *length xs*
  by (*metis* (*no-types*, *opaque-lifting*) *One-nat-def distinct-card distinct-get-lms div-less-dividend*
    get-lms-set-n-gre-length le-less-linear le-less-trans length-greater-0-conv lessI num-lms-bound-2 numeral-2-eq-2)

lemma *card-abs-is-lms-bound-length-div-2*:
card $\{i.$ *abs-is-lms xs i*$\} \leq$ *length xs div 2*
by (*metis distinct-card distinct-get-lms get-lms-set-n-gre-length linear num-lms-bound-2*)

lemma *length-filter-lms*:
$T \neq [] \Longrightarrow$ *length* (*filter* (*abs-is-lms T*) [*0..<length T*]) < *length T*
by (*metis diff-zero abs-is-lms-0 length-filter-less length-greater-0-conv length-upt nth-Cons-0*
nth-mem upt-rec)

45.5 General Properties about LMS-types

lemma *abs-is-lms-imp-le-nth*:
⟦*abs-is-lms T i*; *Suc i* < *length T*⟧ $\Longrightarrow$ *T* ! *i* $\leq$ *T* ! *Suc i*
by (*metis SL-types.simps*(*2*) *abs-is-lms-def not-less nth-gr-imp-l-type*)

lemma *abs-is-lms-neq*:
abs-is-lms T (*Suc i*) $\Longrightarrow$ *T* ! *Suc i* < *T* ! *i*
unfolding *abs-is-lms-def*
proof(*safe*)
assume *suffix-type T* (*Suc i*) = *S-type suffix-type T i* = *L-type*

from ‹*suffix-type T* (*Suc i*) = *S-type*›
have *Suc i* < *length T*
by (*simp add*: *suffix-type-s-bound*)
with *suffix-type-neq* ‹*suffix-type T* (*Suc i*) = *S-type*› ‹*suffix-type T i* = *L-type*›
show *?thesis*
by (*metis SL-types.simps*(*2*) *not-less-iff-gr-or-eq nth-less-imp-s-type*)
qed

lemma *abs-is-lms-last*:
⟦*valid-list T*; *length T* = *Suc* (*Suc n*)⟧ $\Longrightarrow$ *abs-is-lms T* (*Suc n*)
proof (*induct n arbitrary*: *T*)
case *0*
note *IH* = *this*

have *T* ! *Suc 0* = *bot*
by (*metis IH Zero-neq-Suc diff-Suc-1 last-conv-nth list.size*(*3*) *valid-list-def*)
with *IH*(*1*)[*simplified valid-list-ex-def*]
have *T* ! *0* > *T* ! *Suc 0*
by (*metis IH*(*2*) *One-nat-def bot.not-eq-extremum butlast-snoc diff-Suc-1' le-less-Suc-eq length-butlast less-or-eq-imp-le nth-butlast*)
with *suffix-type-last*[*OF IH*(*2*)]
show *?case*
using *abs-is-lms-def nth-gr-imp-l-type suffix-type-s-bound* **by** *blast*
next
case (*Suc n*)
note *IH* = *this*

show *?case*
proof (*cases T*)
case *Nil*
then show *?thesis*
by (*simp add*: *Suc.prems*(*1*) *valid-list-not-nil*)
next
case (*Cons a T′*)
with *IH valid-list-consD*[*of a T′*]
have *abs-is-lms T′* (*Suc n*)
by *fastforce*
then show *?thesis*
by (*simp add*: *abs-is-lms-def local.Cons suffix-type-cons-suc*)
qed
qed

lemma *abs-is-lms-imp-less-length*:
$abs\text{-}is\text{-}lms\ T\ i \Longrightarrow i < length\ T$
using *abs-is-lms-gre-length le-less-linear* **by** *blast*

lemma *s-type-and-not-lms-Suc*:
$[\![\neg abs\text{-}is\text{-}lms\ T\ (Suc\ i);\ suffix\text{-}type\ T\ (Suc\ i) = S\text{-}type]\!] \Longrightarrow suffix\text{-}type\ T\ i = S\text{-}type$
by (*meson abs-is-lms-def suffix-type-def*)

lemma *no-lms-imp-all-s-type*:
assumes $j < length\ T$
and $i \le j$
and $\forall k{>}i.\ k \le j \longrightarrow \neg abs\text{-}is\text{-}lms\ T\ k$
and $suffix\text{-}type\ T\ j = S\text{-}type$
and $i \le k$
and $k \le j$
shows $suffix\text{-}type\ T\ k = S\text{-}type$
using *assms*
proof (*induct* $j - k$ *arbitrary*: *j*)
case *0*
then show *?case*
by *auto*
next
case (*Suc x*)
note $IH = this$

have $\exists j'.\ j = Suc\ j'$
using *Suc.hyps*(*2*) **by** *presburger*
then obtain j' **where**
$j = Suc\ j'$
by *blast*
hence $x = j' - k$
using *Suc.hyps*(*2*) **by** *linarith*
moreover

have $j' <$ *length T*
  using *Suc.prems*(*1*) *Suc-lessD* ‹j = *Suc* j'› **by** *blast*
moreover
have $i \leq j'$
  using *Suc.hyps*(*2*) ‹j = *Suc* j'› *assms*(*5*) **by** *linarith*
moreover
have $\forall k>i.\ k \leq j' \longrightarrow \neg$ *abs-is-lms T k*
  by (*simp add*: *Suc.prems*(*3*) ‹j = *Suc* j'›)
moreover
from *IH*(*5*)
have ¬*abs-is-lms T* (*Suc* j')
  by (*simp add*: ‹j = *Suc* j'› *calculation*(*3*) *le-imp-less-Suc*)
with *IH*(*6*) ‹j = *Suc* j'› *s-type-and-not-lms-Suc*[*of T* j']
have *suffix-type T* j' = *S-type*
  by *blast*
moreover
have $k \leq j'$
  using *Suc.hyps*(*2*) ‹j = *Suc* j'› **by** *linarith*
ultimately show *?case*
  using *IH*(*1*)[*of* j'] *assms*(*5*) **by** *blast*
qed

lemma *first-l-type-after-s-type*:
  assumes $j <$ *length T*
  and $i \leq j$
  and $\forall k>i.\ k \leq j \longrightarrow \neg$*abs-is-lms T k*
  and *suffix-type T j* = *L-type*
  and *suffix-type T i* = *S-type*
shows $\exists l \geq i.\ l \leq j \wedge (\forall k<l.\ i \leq k \longrightarrow$ *suffix-type T k* = *S-type*) ∧ *suffix-type T l* = *L-type*
  using *assms*
proof (*induct* $j - i$ *arbitrary*: *j*)
  case *0*
  then show *?case*
    by *auto*
next
  case (*Suc x*)
  note *IH* = *this*

  have $\exists j'.\ j$ = *Suc* j'
    by (*metis SL-types.distinct*(*1*) *Suc.prems*(*2*) *Suc.prems*(*4*) *assms*(*5*) *diff-is-0-eq diff-zero*
          less-imp-Suc-add neq0-conv)
  then obtain j' **where**
    j = *Suc* j'
    by *blast*
  hence $x = j' - i$
    using *Suc.hyps*(*2*) ‹j = *Suc* j'› **by** *linarith*

```
  have j' < length T
    using Suc.prems(1) Suc-lessD ‹j = Suc j'› by blast

  have i ≤ j'
    using Suc.hyps(2) ‹j = Suc j'› by linarith

  have P: ∀k>i. k ≤ j' ⟶ ¬ abs-is-lms T k
    by (simp add: Suc.prems(3) ‹j = Suc j'›)

  have suffix-type T j' = S-type ∨ suffix-type T j' = L-type
    using SL-types.exhaust by blast
  moreover
  have suffix-type T j' = S-type ⟹ ?case
  proof −
    assume suffix-type T j' = S-type
    hence ∀k<j. i ≤ k ⟶ suffix-type T k = S-type
      using P ‹j' < length T› no-lms-imp-all-s-type  ‹j = Suc j'› less-Suc-eq-le by
auto
    then show ?thesis
      using Suc.prems(2) Suc.prems(4) by blast
  qed
  moreover
  have suffix-type T j' = L-type ⟹ ?case
  proof −
    assume suffix-type T j' = L-type
    with IH(1)[OF ‹x = -› ‹j' < -› ‹i ≤ j'› P - assms(5)]
     have ∃l≥i. l ≤ j' ∧ (∀k<l. i ≤ k ⟶ suffix-type T k = S-type) ∧ suffix-type
T l = L-type .
    then show ?thesis
      using ‹j = Suc j'› by auto
  qed
  ultimately show ?case
    by blast
qed

lemma no-lms-imp-and-s-imp-all-s-below:
  assumes ∀k. i ≤ k ∧ k < j ⟶ ¬abs-is-lms T k
  and     suffix-type T k = S-type
  and     i ≤ k
  and     k < j
shows ⟦i ≤ k'; k' ≤ k⟧ ⟹ suffix-type T k' = S-type
proof (induct k − k' arbitrary: k')
  case 0
  with assms
  show ?case
    by auto
next
  case (Suc x)
  note IH = this
```

from $IH(2)$
have $x = k - Suc\ k'$
by *linarith*

from $IH(3)$
have $i \leq Suc\ k'$
by *simp*

from $IH(2)$
have $Suc\ k' \leq k$
by *linarith*

from $IH(1)[OF\ \langle x = k - Suc\ k' \rangle\ \langle i \leq Suc\ k' \rangle\ \langle Suc\ k' \leq k \rangle]$
have $suffix\text{-}type\ T\ (Suc\ k') = S\text{-}type$
by *assumption*
with $assms(1)\ \langle i \leq k' \rangle\ \langle k' \leq k \rangle\ \langle Suc\ k' \leq k \rangle\ \langle i \leq Suc\ k' \rangle\ assms(4)$
show *?case*
by (*meson SL-types.exhaust abs-is-lms-def order.strict-trans1*)
qed

lemma *no-lms-imp-and-l-imp-all-l-above*:
assumes $\forall k.\ i \leq k \wedge k < j \longrightarrow \neg abs\text{-}is\text{-}lms\ T\ k$
and $suffix\text{-}type\ T\ k = L\text{-}type$
and $i \leq k$
and $k < j$
shows $[\![k \leq k';\ k' < j]\!] \Longrightarrow suffix\text{-}type\ T\ k' = L\text{-}type$
proof (*induct* $k' - k$ *arbitrary*: k')
case *0*
with *assms*
show *?case*
by *auto*
next
case (*Suc x*)
note $IH = this$
from $IH(2)$
have $\exists n.\ k' = Suc\ n$
by (*metis less-Suc-eq less-Suc-eq-0-disj zero-diff*)
then obtain n **where**
$k' = Suc\ n$
by *blast*
with $IH(2)$
have $x = n - k$
by *linarith*

from $IH(2)\ \langle k' = Suc\ n \rangle$
have $k \leq n$
by *linarith*

from $IH(4)$ ‹$k' = Suc\ n$›
have $n < j$
by *linarith*

from $IH(1)[OF$ ‹$x = n - k$› ‹$k \leq n$› ‹$n < j$›$]$
have *suffix-type* $T\ n = L\text{-}type$
by *assumption*
with $assms(1)$ ‹$k' = Suc\ n$› ‹$k' < j$› ‹$k \leq k'$› $assms(3)$
show *?case*
by (*meson SL-types.exhaust abs-is-lms-def le-trans*)
qed

lemma *lms-sublist-helper*:
assumes $\forall k.$ *suffix-type* $T\ k = S\text{-}type \longrightarrow Suc\ k < n \longrightarrow i \leq k \longrightarrow$ *suffix-type* $T\ (Suc\ k) \neq L\text{-}type$
and *suffix-type* $T\ i = S\text{-}type$
shows $[\![i \leq k;\ k < n]\!] \Longrightarrow$ *suffix-type* $T\ k = S\text{-}type$
proof (*induct* $k - i$ *arbitrary*: k)
case *0*
then show *?case*
using $assms(2)$ **by** *auto*
next
case (*Suc* x)
note $IH = this$
from $IH(2)$
have $\exists k'.\ k = Suc\ k'$
by *presburger*
then obtain k' **where**
$k = Suc\ k'$
by *blast*
with $IH(2)$
have $x = k' - i$
by *linarith*

from $IH(2)$ ‹$k = Suc\ k'$›
have $i \leq k'$
by *linarith*

from $IH(4)$ ‹$k = Suc\ k'$›
have $k' < n$
by *linarith*

from $IH(1)[OF$ ‹$x = k' - i$› ‹$i \leq k'$› ‹$k' < n$›$]$ $assms(1)$ $IH(4)$ ‹$k = Suc\ k'$› ‹$i \leq k'$›
show *?case*
using *SL-types.exhaust* **by** *blast*
qed

end

theory *Buckets*
imports
../../util/Continuous-Interval
List-Type
begin

46 Buckets

46.1 Entire Bucket

definition *bucket* :: $('a :: \{linorder,order\text{-}bot\} \Rightarrow nat) \Rightarrow 'a\ list \Rightarrow nat \Rightarrow nat\ set$
where
$bucket\ \alpha\ T\ b \equiv \{k\ |k.\ k < length\ T \wedge \alpha\ (T\ !\ k) = b\}$

definition *bucket-size* :: $('a :: \{linorder,order\text{-}bot\} \Rightarrow nat) \Rightarrow 'a\ list \Rightarrow nat \Rightarrow nat$
where
$bucket\text{-}size\ \alpha\ T\ b \equiv card\ (bucket\ \alpha\ T\ b)$

definition *bucket-upt* :: $('a :: \{linorder,order\text{-}bot\} \Rightarrow nat) \Rightarrow 'a\ list \Rightarrow nat \Rightarrow nat\ set$
where
$bucket\text{-}upt\ \alpha\ T\ b = \{k\ |k.\ k < length\ T \wedge \alpha\ (T\ !\ k) < b\}$

definition *bucket-start* :: $('a :: \{linorder,order\text{-}bot\} \Rightarrow nat) \Rightarrow 'a\ list \Rightarrow nat \Rightarrow nat$
where
$bucket\text{-}start\ \alpha\ T\ b \equiv card\ (bucket\text{-}upt\ \alpha\ T\ b)$

definition *bucket-end* :: $('a :: \{linorder,order\text{-}bot\} \Rightarrow nat) \Rightarrow 'a\ list \Rightarrow nat \Rightarrow nat$
where
$bucket\text{-}end\ \alpha\ T\ b \equiv card\ (bucket\text{-}upt\ \alpha\ T\ (Suc\ b))$

lemma *bucket-upt-subset*:
$bucket\text{-}upt\ \alpha\ T\ b \subseteq \{0..<length\ T\}$
by (*rule subsetI*, *simp add*: *bucket-upt-def*)

lemma *bucket-upt-subset-Suc*:
$bucket\text{-}upt\ \alpha\ T\ b \subseteq bucket\text{-}upt\ \alpha\ T\ (Suc\ b)$
by (*rule subsetI*, *simp add*: *bucket-upt-def*)

lemma *bucket-upt-un-bucket*:
$bucket\text{-}upt\ \alpha\ T\ b \cup bucket\ \alpha\ T\ b = bucket\text{-}upt\ \alpha\ T\ (Suc\ b)$
apply (*clarsimp simp*: *bucket-upt-def bucket-def*)
apply (*intro equalityI subsetI*; *clarsimp*)
apply (*erule disjE*; *clarsimp*)
done

lemma *bucket-0*:
assumes $valid\text{-}list\ T\ \alpha\ bot = 0\ strict\text{-}mono\ \alpha\ length\ T = Suc\ k$

shows *bucket* α *T 0* = {*k*}
proof *safe*
fix *x*
assume *x* ∈ *bucket* α *T 0*
then show *x* = *k*
by (*metis* (*mono-tags*, *lifting*) *assms bucket-def diff-Suc-1 le-neq-trans le-simps*(*2*) *mem-Collect-eq strict-mono-eq valid-list-def*)
next
show *k* ∈ *bucket* α *T 0*
by (*metis* (*mono-tags*, *lifting*) *assms*(*1*,*2*,*4*) *bucket-def diff-Suc-1 last-conv-nth lessI*
list.size(*3*) *mem-Collect-eq order-less-irrefl valid-list-def*)
qed

lemma *finite-bucket*:
finite (*bucket* α *T x*)
by (*clarsimp simp*: *bucket-def*)

lemma *finite-bucket-upt*:
finite (*bucket-upt* α *T b*)
by (*meson bucket-upt-subset subset-eq-atLeast0-lessThan-finite*)

lemma *bucket-start-Suc*:
bucket-start α *T* (*Suc b*) = *bucket-start* α *T b* + *bucket-size* α *T b*
apply (*clarsimp simp*: *bucket-start-def bucket-size-def*)
apply (*subst card-Un-disjoint*[*symmetric*])
apply (*meson bucket-upt-subset subset-eq-atLeast0-lessThan-finite*)
apply (*simp add*: *finite-bucket*)
apply (*rule disjointI′*)
apply (*metis* (*mono-tags*, *lifting*) *bucket-def bucket-upt-def less-irrefl-nat mem-Collect-eq*)
apply (*simp add*: *bucket-upt-un-bucket*)
done

lemma *bucket-start-le*:
b ≤ *b′* ⟹ *bucket-start* α *T b* ≤ *bucket-start* α *T b′*
apply (*clarsimp simp*: *bucket-start-def*)
by (*meson bucket-upt-subset bucket-upt-subset-Suc card-mono lift-Suc-mono-le subset-eq-atLeast0-lessThan-finite*)

lemma *bucket-start-Suc-eq-bucket-end*:
bucket-start α *T* (*Suc b*) = *bucket-end* α *T b*
by (*simp add*: *bucket-end-def bucket-start-def*)

lemma *bucket-end-le-length*:
bucket-end α *T b* ≤ *length T*
apply (*clarsimp simp*: *bucket-end-def*)
apply (*insert card-mono*[*OF - bucket-upt-subset*[*of* α *T Suc b*]])
apply (*erule meta-impE*, *simp*)
apply (*erule order.trans*)

apply *simp*
done

lemma *bucket-start-le-end*:
bucket-start α *T b* $\leq$ *bucket-end* α *T b*
by (*metis Suc-n-not-le-n bucket-start-Suc-eq-bucket-end bucket-start-le nat-le-linear*)

lemma *le-bucket-start-le-end*:
$b \leq b' \Longrightarrow$ *bucket-start* α *T b* $\leq$ *bucket-end* α *T* b'
using *bucket-start-le bucket-start-le-end le-trans* **by** *blast*

lemma *bucket-end-le*:
$b \leq b' \Longrightarrow$ *bucket-end* α *T b* $\leq$ *bucket-end* α *T* b'
by (*metis bucket-start-Suc-eq-bucket-end bucket-start-le-end lift-Suc-mono-le*)

lemma *less-bucket-end-le-start*:
$b < b' \Longrightarrow$ *bucket-end* α *T b* $\leq$ *bucket-start* α *T* b'
by (*metis Suc-leI bucket-start-Suc-eq-bucket-end bucket-start-le*)

lemma *bucket-end-def'*:
bucket-end α *T b* = *bucket-start* α *T b* + *bucket-size* α *T b*
by (*metis bucket-start-Suc bucket-start-Suc-eq-bucket-end*)

lemma *valid-list-bucket-start-0*:
$[\![$*valid-list T*; *strict-mono* α; α *bot* = *0*$]\!] \Longrightarrow$
bucket-start α *T 0* = *0*
by (*clarsimp simp*: *bucket-start-def bucket-upt-def*)

lemma *bucket-upt-0*:
bucket-upt α *T 0* = $\{\}$
by (*clarsimp simp*: *bucket-upt-def*)

lemma *bucket-start-0*:
bucket-start α *T 0* = *0*
by (*clarsimp simp*: *bucket-start-def bucket-upt-def*)

lemma *valid-list-bucket-upt-Suc-0*:
$[\![$*valid-list T*; *strict-mono* α; α *bot* = *0*; *length T* = *Suc n*$]\!] \Longrightarrow$
bucket-upt α *T* (*Suc 0*) = $\{n\}$
apply (*clarsimp simp*: *bucket-upt-def*)
apply (*intro equalityI subsetI*)
apply (*clarsimp simp*: *valid-list-def*)
apply (*metis less-antisym strict-mono-eq*)
apply (*clarsimp simp*: *valid-list-ex-def*)
done

lemma *valid-list-bucket-end-0*:
$[\![$*valid-list T*; *strict-mono* α; α *bot* = *0*$]\!] \Longrightarrow$
bucket-end α *T 0* = *1*

apply (*clarsimp simp*: *bucket-end-def*)
apply (*frule valid-list-length-ex*)
apply *clarsimp*
apply (*frule* (*3*) *valid-list-bucket-upt-Suc-0*)
apply *simp*
done

lemma *nth-Max*:
$T \neq [] \Longrightarrow \exists\, i < \textit{length}\ T.\ T\ !\ i = \textit{Max}\ (\textit{set}\ T)$
by (*metis List.finite-set Max-in in-set-conv-nth set-empty*)

lemma *bucket-upt-Suc-Max*:
$\textit{strict-mono}\ \alpha \Longrightarrow \textit{bucket-upt}\ \alpha\ T\ (\textit{Suc}\ (\alpha\ (\textit{Max}\ (\textit{set}\ T)))) = \{0..<\textit{length}\ T\}$
apply (*intro equalityI subsetI*)
apply (*erule bucket-upt-subset*[*THEN subsetD*])
by (*clarsimp simp*: *bucket-upt-def less-Suc-eq-le strict-mono-less-eq*)

lemma *bucket-end-Max*:
$\textit{strict-mono}\ \alpha \Longrightarrow \textit{bucket-end}\ \alpha\ T\ (\alpha\ (\textit{Max}\ (\textit{set}\ T))) = \textit{length}\ T$
apply (*clarsimp simp*: *bucket-end-def*)
apply (*drule bucket-upt-Suc-Max*[**where** $T = T$])
apply *clarsimp*
done

lemma *bucket-end-eq-length*:
$[\![\textit{strict-mono}\ \alpha;\ b \leq \alpha\ (\textit{Max}\ (\textit{set}\ T));\ T \neq [];\ \textit{bucket-end}\ \alpha\ T\ b = \textit{length}\ T]\!] \Longrightarrow$
$b = \alpha\ (\textit{Max}\ (\textit{set}\ T))$
proof $-$
assume $\textit{strict-mono}\ \alpha\ b \leq \alpha\ (\textit{Max}\ (\textit{set}\ T))\ \textit{bucket-end}\ \alpha\ T\ b = \textit{length}\ T\ T \neq []$
show $b = \alpha\ (\textit{Max}\ (\textit{set}\ T))$
proof (*rule ccontr*)
assume $b \neq \alpha\ (\textit{Max}\ (\textit{set}\ T))$
with $\langle b \leq \text{-}\rangle$
have $b < \alpha\ (\textit{Max}\ (\textit{set}\ T))$
by *simp*
hence $\exists\, b'.\ b' = \alpha\ (\textit{Max}\ (\textit{set}\ T))$
by *blast*
then obtain b' **where**
$b' = \alpha\ (\textit{Max}\ (\textit{set}\ T))$
by *blast*
with $\langle b < \text{-}\rangle$
have $b < b'$
by *blast*
hence $\textit{bucket-end}\ \alpha\ T\ b \leq \textit{bucket-start}\ \alpha\ T\ b'$
by (*simp add*: *less-bucket-end-le-start*)
moreover
from *nth-Max*[*OF* $\langle T \neq []\rangle$]
obtain i **where**
$i < \textit{length}\ T$

$T ! i = Max (set\ T)$
by *blast*
with ‹$b' = \alpha$ (*Max* (*set* T))› ‹*strict-mono* α›
have $i \in$ *bucket* α T b'
by (*simp add*: *bucket-def*)
hence *bucket-start* α T b' < *bucket-end* α T b'
by (*metis add-diff-cancel-left′ bucket-end-def′ bucket-size-def bucket-start-le-end card-gt-0-iff diff-is-0-eq′ empty-iff finite-bucket nat-less-le*)
moreover
have *bucket-end* α T $b' \leq$ *length* T
using *bucket-end-le-length* **by** *blast*
ultimately
show *False*
using ‹*bucket-end* α T b = *length* T›
by *linarith*
qed
qed

46.2 L-types

definition *l-bucket* :: ($'a$ :: {*linorder*,*order-bot*} $\Rightarrow$ *nat*) $\Rightarrow$ $'a$ *list* $\Rightarrow$ *nat* $\Rightarrow$ *nat set*
where
l-bucket α T b = {k |k. $k \in$ *bucket* α T b $\wedge$ *suffix-type* T k = *L-type*}

definition *l-bucket-size* :: ($'a$ :: {*linorder*,*order-bot*} $\Rightarrow$ *nat*) $\Rightarrow$ $'a$ *list* $\Rightarrow$ *nat* $\Rightarrow$ *nat*
where
l-bucket-size α T b $\equiv$ *card* (*l-bucket* α T b)

definition *l-bucket-end* :: ($'a$ :: {*linorder*,*order-bot*} $\Rightarrow$ *nat*) $\Rightarrow$ $'a$ *list* $\Rightarrow$ *nat* $\Rightarrow$ *nat*
where
l-bucket-end α T b = *bucket-start* α T b + *l-bucket-size* α T b

lemma *l-bucket-subset-bucket*:
l-bucket α T $b \subseteq$ *bucket* α T b
by (*rule subsetI*, *simp add*: *l-bucket-def*)

lemma *bucket-upt-int-l-bucket*:
strict-mono $\alpha \Longrightarrow$ *bucket-upt* α T b $\cap$ *l-bucket* α T b = {}
apply (*rule disjoint-subset2*[**where** B = *bucket* α T b])
apply (*simp add*: *l-bucket-def*)
apply (*simp add*: *bucket-def bucket-upt-def*)
apply (*rule disjointI′*)
apply *clarsimp*
done

lemma *subset-l-bucket*:
⟦$\forall k <$ *length* *ls*. *ls* ! k < *length* T $\wedge$ *suffix-type* T (*ls* ! k) = *L-type* $\wedge$ α (T ! (*ls*

! k)) = x;
    distinct ls⟧ ⟹
  set ls ⊆ *l-bucket* α *T x*
  apply (*rule subsetI*)
  apply (*clarsimp simp*: *l-bucket-def bucket-def in-set-conv-nth*)
  done

lemma *finite-l-bucket*:
  finite (*l-bucket* α *T x*)
  apply (*clarsimp simp*: *finite-bucket l-bucket-def*)
  done

lemma *l-bucket-list-eq*:
  ⟦∀ *k* < *length ls*. *ls* ! *k* < *length T* ∧ *suffix-type T* (*ls* ! *k*) = *L-type* ∧ α (*T* ! (*ls* ! *k*)) = *x*;
    distinct ls; *length ls* = *l-bucket-size* α *T x*⟧ ⟹
  set ls = *l-bucket* α *T x*
  apply (*frule* (*1*) *subset-l-bucket*)
  apply (*frule distinct-card*)
  apply (*insert finite-l-bucket*[*of* α *T x*])
  by (*simp add*: *card-subset-eq l-bucket-size-def*)

lemma *l-bucket-le-bucket-size*:
  l-bucket-size α *T b* ≤ *bucket-size* α *T b*
  apply (*clarsimp simp*: *l-bucket-size-def bucket-size-def*)
  apply (*rule card-mono*[*OF finite-bucket l-bucket-subset-bucket*])
  done

lemma *l-bucket-not-empty*:
  ⟦*i* < *length T*; *suffix-type T i* = *L-type*⟧ ⟹ *0* < *l-bucket-size* α *T* (α (*T* ! *i*))
  apply (*clarsimp simp*: *l-bucket-size-def*)
  apply (*subst card-gt-0-iff*)
  apply (*intro conjI finite-l-bucket*)
  apply (*clarsimp simp*: *l-bucket-def bucket-def*)
  apply *blast*
  done

lemma *l-bucket-end-le-bucket-end*:
  l-bucket-end α *T b* ≤ *bucket-end* α *T b*
  apply (*clarsimp simp*: *l-bucket-end-def*)
  apply (*rule order.trans*[**where** *b* = *bucket-start* α *T b* + *bucket-size* α *T b*])
   apply (*simp add*: *l-bucket-le-bucket-size*)
  by (*metis bucket-start-Suc bucket-start-Suc-eq-bucket-end le-refl*)

lemma *l-bucket-Max*:
  assumes *valid-list T*
  and   *Suc 0* < *length T*
  and   *strict-mono* α
  shows *l-bucket* α *T* (α (*Max* (*set T*))) = *bucket* α *T* (α (*Max* (*set T*)))

proof (*intro subsetI equalityI*)
  let *?b* = α (*Max* (*set T*))
  fix *x*
  assume $x \in$ *l-bucket* α *T ?b*
  then show $x \in$ *bucket* α *T ?b*
    using *l-bucket-subset-bucket* **by** *blast*
next
  let *?b* = α (*Max* (*set T*))
  fix *x*
  assume $x \in$ *bucket* α *T ?b*
  hence $x <$ *length T* α (*T* ! *x*) = *?b*
    using *bucket-def* **by** *blast+*
  with *suffix-max-hd-is-l-type*[*OF assms*(*1*) - *assms*(*2*)]
  have *suffix-type T x* = *L-type*
    by (*metis Cons-nth-drop-Suc assms*(*3*) *list.sel*(*1*) *strict-mono-eq*)
  then show $x \in$ *l-bucket* α *T ?b*
    using ‹$x \in$ *bucket* α *T* (α (*Max* (*set T*)))› *l-bucket-def* **by** *blast*
qed

46.3 LMS-types

definition *lms-bucket* :: (*'a* :: {*linorder*,*order-bot*} $\Rightarrow$ *nat*) $\Rightarrow$ *'a list* $\Rightarrow$ *nat* $\Rightarrow$ *nat set*
  where
lms-bucket α *T b* = {*k* |*k*. $k \in$ *bucket* α *T b* $\wedge$ *abs-is-lms T k*}

definition *lms-bucket-size* :: (*'a* :: {*linorder*,*order-bot*} $\Rightarrow$ *nat*) $\Rightarrow$ *'a list* $\Rightarrow$ *nat* $\Rightarrow$ *nat*
  where
lms-bucket-size α *T b* $\equiv$ *card* (*lms-bucket* α *T b*)

lemma *lms-bucket-subset-bucket*:
  lms-bucket α *T b* $\subseteq$ *bucket* α *T b*
  by (*simp add*: *lms-bucket-def*)

lemma *finite-lms-bucket*:
  finite (*lms-bucket* α *T b*)
  by (*clarsimp simp*: *lms-bucket-def finite-bucket*)

lemma *disjoint-l-lms-bucket*:
 l-bucket α *T b* $\cap$ *lms-bucket* α *T b* = {}
  apply (*rule disjointI'*)
  by (*clarsimp simp*: *l-bucket-def lms-bucket-def abs-is-lms-def*)

46.4 S-types

definition *s-bucket* :: (*'a* :: {*linorder*,*order-bot*} $\Rightarrow$ *nat*) $\Rightarrow$ *'a list* $\Rightarrow$ *nat* $\Rightarrow$ *nat set*
  where
s-bucket α *T b* = {*k* |*k*. $k \in$ *bucket* α *T b* $\wedge$ *suffix-type T k* = *S-type*}

definition *s-bucket-size* :: ($'a$ :: {*linorder*,*order-bot*} ⇒ *nat*) ⇒ $'a$ *list* ⇒ *nat* ⇒ *nat*
  where
s-bucket-size α *T b* ≡ *card* (*s-bucket* α *T b*)

definition *s-bucket-start* :: ($'a$ :: {*linorder*,*order-bot*} ⇒ *nat*) ⇒ $'a$ *list* ⇒ *nat* ⇒ *nat*
  where
s-bucket-start α *T b* ≡ *bucket-start* α *T b* + *l-bucket-size* α *T b*

lemma *finite-s-bucket*:
 finite (*s-bucket* α *T b*)
  by (*clarsimp simp*: *s-bucket-def finite-bucket*)

lemma *disjoint-l-s-bucket*:
 l-bucket α *T b* ∩ *s-bucket* α *T b* = {}
  apply (*rule disjointI'*)
  by (*clarsimp simp*: *l-bucket-def s-bucket-def*)

lemma *lms-subset-s-bucket*:
  lms-bucket α *T b* ⊆ *s-bucket* α *T b*
  by (*clarsimp simp*: *s-bucket-def lms-bucket-def abs-is-lms-def*)

lemma *l-un-s-bucket*:
  bucket α *T b* = *l-bucket* α *T b* ∪ *s-bucket* α *T b*
  apply (*rule equalityI*; *clarsimp simp*: *l-bucket-def s-bucket-def*)
  by (*meson suffix-type-def*)

lemma *s-bucket-Max*:
  assumes *valid-list T*
  and *length T* > *Suc 0*
  and *strict-mono* α
shows *s-bucket* α *T* (α (*Max* (*set T*))) = {}
proof −
  let *?b* = α (*Max* (*set T*))
  from *l-bucket-Max*[*OF assms*]
  have *l-bucket* α *T ?b* = *bucket* α *T ?b* **.**
  moreover
  from *l-un-s-bucket*
  have *bucket* α *T ?b* = *l-bucket* α *T ?b* ∪ *s-bucket* α *T ?b* **.**
  hence *s-bucket* α *T ?b* ⊆ *bucket* α *T ?b*
    by *blast*
  moreover
  from *disjoint-l-s-bucket*
  have *l-bucket* α *T ?b* ∩ *s-bucket* α *T ?b* = {} **.**
  ultimately
  show *?thesis*
    by *blast*

qed

lemma *s-bucket-0*:
  assumes *valid-list* T
  and *strict-mono* α
  and α *bot* $= 0$
  and *length* $T = $ *Suc* n
shows *s-bucket* α T $0 = \{n\}$
proof −
  have *suffix-type* T n = *S-type*
    using *assms*(*4*) *suffix-type-last* **by** *blast*
  moreover
  have T ! n = *bot*
      by (*metis* *assms*(*1*) *assms*(*4*) *diff-Suc-1* *last-conv-nth* *length-greater-0-conv* *valid-list-def*)
  hence α (T ! n) $= 0$
    by (*simp add*: *assms*(*3*))
  ultimately have $n \in$ *s-bucket* α T 0
    by (*simp add*: *assms*(*4*) *bucket-def* *s-bucket-def*)
  hence $\{n\} \subseteq$ *s-bucket* α T 0
    by *blast*
  moreover
  have *s-bucket* α T $0 \subseteq \{n\}$
    unfolding *s-bucket-def*
  proof *safe*
    fix k
    assume $k \in$ *bucket* α T 0 *suffix-type* T k = *S-type*
    hence $k \leq n$
      by (*simp add*: *assms*(*4*) *bucket-def*)
    have α (T ! k) $= 0$
      using ‹$k \in$ *bucket* α T 0› *bucket-def* **by** *blast*
    hence T ! k = *bot*
      by (*metis* *assms*(*2*) *assms*(*3*) *strict-mono-eq*)
    show $k = n$
    proof (*rule ccontr*)
      assume $k \neq n$
      hence $k < n$
        by (*simp add*: ‹$k \leq n$› *le-neq-implies-less*)
      then show *False*
        using ‹$k \in$ *bucket* α T 0› ‹$k \neq n$› *assms* *bucket-0* **by** *blast*
    qed
  qed
  ultimately show *?thesis*
    by *blast*
qed

lemma *s-bucket-successor*:
  $[\![$*valid-list* T; *strict-mono* α; α *bot* $= 0$; $b \neq 0$; $x \in$ *s-bucket* α T $b]\!] \Longrightarrow$
    Suc $x \in$ *s-bucket* α T $b \vee (\exists b'.\ b < b' \wedge$ *Suc* $x \in$ *bucket* α T $b')$

```
proof −
  assume valid-list T strict-mono α α bot = 0 b ≠ 0 x ∈ s-bucket α T b
  hence suffix-type T x = S-type
    by (simp add: s-bucket-def)

  from valid-list-length-ex[OF ‹valid-list -›]
  obtain n where
    length T = Suc n
    by blast
  moreover
  from ‹x ∈ s-bucket α T b›
  have x < length T α (T ! x) = b
    by (simp add: s-bucket-def bucket-def)+
  ultimately have Suc x < length T
    by (metis Suc-lessI ‹α bot = 0› ‹b ≠ 0› ‹valid-list T› diff-Suc-1 last-conv-nth
list.size(3)
          valid-list-def)

  have T ! x ≤ T ! Suc x
   by (simp add: ‹Suc x < length T› ‹suffix-type T x = S-type› s-type-letter-le-Suc)
  hence T ! x < T ! Suc x ∨ T ! x = T ! Suc x
    using le-neq-trans by blast
  moreover
  have T ! x < T ! Suc x ⟹ ?thesis
  proof −
    assume T ! x < T ! Suc x
    hence α (T ! x) < α (T ! Suc x)
      by (simp add: ‹strict-mono α› strict-mono-less)
    hence b < α (T ! Suc x)
      by (simp add: ‹α (T ! x) = b›)
    with ‹Suc x < length T›
    have Suc x ∈ bucket α T (α (T ! Suc x))
      by (simp add: bucket-def)
    with ‹b < α (T ! Suc x)›
    show ?thesis
      by blast
  qed
  moreover
  have T ! x = T ! Suc x ⟹ ?thesis
  proof −
    assume T ! x = T ! Suc x
    hence α (T ! Suc x) = b
      using ‹α (T ! x) = b› by auto
    moreover
    from ‹Suc x < length T› ‹T ! x = T ! Suc x› ‹suffix-type T x = S-type›
    have suffix-type T (Suc x) = S-type
      using suffix-type-neq by force
    ultimately show ?thesis
      by (simp add: ‹Suc x < length T› bucket-def s-bucket-def)
```

qed
ultimately show *?thesis*
by *blast*
qed

lemma *subset-s-bucket-successor*:
$\llbracket$*valid-list* T; *strict-mono* α; α *bot* $= 0$; $b \neq 0$; $A \subseteq$ *s-bucket* α T b; $A \neq \{\}\rrbracket \Longrightarrow$
$\exists x \in A.$ *Suc* $x \in$ *s-bucket* α T $b - A \vee (\exists b'.\ b < b' \wedge$ *Suc* $x \in$ *bucket* α T $b')$
proof $-$
assume *valid-list* T *strict-mono* α α *bot* $= 0$ $b \neq 0$ $A \subseteq$ *s-bucket* α T b $A \neq \{\}$

have *finite* A
using ‹$A \subseteq$ *s-bucket* α T b› *finite-s-bucket finite-subset* **by** *blast*

let *?B* $=$ *s-bucket* α T $b - A$

have $\exists x \in A.$ *Suc* $x \notin A$
proof (*rule ccontr*)
assume $\neg$ ($\exists x{\in}A.$ *Suc* $x \notin A$)
hence $\forall x{\in}A.$ *Suc* $x \in A$
by *clarsimp*
hence $\neg$ *finite* A
using *Max.coboundedI Max-in Suc-n-not-le-n* ‹$A \neq \{\}$› **by** *blast*
with ‹*finite* A›
show *False*
by *blast*
qed
then obtain x **where**
$x \in A$
Suc $x \notin A$
by *blast*
with *s-bucket-successor*[*OF* ‹*valid-list* -› ‹*strict-mono* -› ‹α - = -› ‹$b \neq$ -›, *of* x]
‹$A \subseteq$ *s-bucket* α T b›
have *Suc* $x \in$ *s-bucket* α T $b \vee (\exists b'.\ b < b' \wedge$ *Suc* $x \in$ *bucket* α T $b')$
by *blast*
moreover
have *Suc* $x \in$ *s-bucket* α T $b \Longrightarrow$ *?thesis*
proof $-$
assume *Suc* $x \in$ *s-bucket* α T b
with ‹*Suc* $x \notin A$›
show *?thesis*
using ‹$x \in A$› **by** *blast*
qed
moreover
have ($\exists b'.\ b < b' \wedge$ *Suc* $x \in$ *bucket* α T b') $\Longrightarrow$ *?thesis*
using ‹$x \in A$› **by** *blast*
ultimately show *?thesis*
by *blast*
qed

lemma *valid-list-s-bucket-start-0*:
$[\![$*valid-list T*; *strict-mono* α; α *bot* = *0*$]\!] \Longrightarrow$
s-bucket-start α *T 0* = *0*
apply (*clarsimp simp*: *s-bucket-start-def bucket-start-0*)
apply (*frule valid-list-length-ex*)
apply *clarsimp*
apply (*frule* (*3*) *bucket-0*)
apply (*frule suffix-type-last*)
apply (*clarsimp simp*: *l-bucket-size-def l-bucket-def*)
done

definition *pure-s-bucket* :: (*'a* :: {*linorder*,*order-bot*} ⇒ *nat*) ⇒ *'a list* ⇒ *nat* ⇒ *nat set*
where
pure-s-bucket α *T b* = {*k* |*k*. *k* ∈ *s-bucket* α *T b* ∧ *k* ∉ *lms-bucket* α *T b*}

definition *pure-s-bucket-size* :: (*'a* :: {*linorder*,*order-bot*} ⇒ *nat*) ⇒ *'a list* ⇒ *nat* ⇒ *nat*
where
pure-s-bucket-size α *T b* ≡ *card* (*pure-s-bucket* α *T b*)

lemma *finite-pure-s-bucket*:
finite (*pure-s-bucket* α *T b*)
by (*clarsimp simp*: *pure-s-bucket-def finite-s-bucket*)

lemma *pure-s-subset-s-bucket*:
pure-s-bucket α *T b* ⊆ *s-bucket* α *T b*
by (*clarsimp simp*: *s-bucket-def pure-s-bucket-def*)

lemma *disjoint-lms-pure-s-bucket*:
lms-bucket α *T b* ∩ *pure-s-bucket* α *T b* = {}
apply (*rule disjointI'*)
by (*clarsimp simp*: *lms-bucket-def pure-s-bucket-def*)

lemma *disjoint-pure-s-lms-bucket*:
pure-s-bucket α *T b* ∩ *lms-bucket* α *T b* = {}
apply (*subst Int-commute*)
by (*rule disjoint-lms-pure-s-bucket*)

lemma *s-eq-pure-s-un-lms-bucket*:
s-bucket α *T b* = *pure-s-bucket* α *T b* ∪ *lms-bucket* α *T b*
apply (*intro equalityI*; *clarsimp simp*: *pure-s-subset-s-bucket lms-subset-s-bucket*)
apply (*clarsimp simp*: *s-bucket-def lms-bucket-def pure-s-bucket-def*)
done

lemma *l-pl-pure-s-pl-lms-size*:
bucket-size α *T b* = *l-bucket-size* α *T b* + *pure-s-bucket-size* α *T b* + *lms-bucket-size* α *T b*

apply (*clarsimp simp*: *bucket-size-def l-bucket-size-def pure-s-bucket-size-def lms-bucket-size-def*)
apply (*subst add.assoc*)
apply (*subst card-Un-disjoint*[*symmetric*,
OF finite-pure-s-bucket finite-lms-bucket disjoint-pure-s-lms-bucket])
apply (*subst s-eq-pure-s-un-lms-bucket*[*symmetric*])
apply (*subst card-Un-disjoint*[*symmetric*,
OF finite-l-bucket finite-s-bucket disjoint-l-s-bucket])
apply (*clarsimp simp*: *l-un-s-bucket*)
done

lemma *s-bucket-start-eq-l-bucket-end*:
s-bucket-start α *T b* = *l-bucket-end* α *T b*
by (*simp add*: *l-bucket-end-def s-bucket-start-def*)

lemma *s-eq-pure-pl-lms-size*:
s-bucket-size α *T b* = *pure-s-bucket-size* α *T b* + *lms-bucket-size* α *T b*
by (*simp add*: *card-Un-disjoint disjoint-pure-s-lms-bucket finite-lms-bucket finite-pure-s-bucket*
lms-bucket-size-def pure-s-bucket-size-def s-bucket-size-def s-eq-pure-s-un-lms-bucket)

lemma *bucket-end-eq-s-start-pl-size*:
bucket-end α *T b* = *s-bucket-start* α *T b* + *s-bucket-size* α *T b*
by (*simp add*: *bucket-end-def′ l-bucket-end-def l-pl-pure-s-pl-lms-size*
s-bucket-start-eq-l-bucket-end s-eq-pure-pl-lms-size)

lemma *bucket-start-le-s-bucket-start*:
bucket-start α *T b* $\leq$ *s-bucket-start* α *T b*
by (*simp add*: *s-bucket-start-def*)

lemma *bucket-0-size1*:
assumes *valid-list T*
and *strict-mono* α
and α *bot* = *0*
shows *bucket-size* α *T 0* = *Suc 0* $\wedge$ *l-bucket-size* α *T 0* = *0*
proof −
from *valid-list-length-ex*[*OF assms*(*1*)]
obtain *n* **where**
length T = *Suc n*
by *blast*
with *bucket-0*[*OF assms*(*1*,*3*,*2*)]
have *bucket* α *T 0* = {*n*}
by *blast*
hence *bucket-size* α *T 0* = *Suc 0*
by (*simp add*: *bucket-size-def*)
moreover
have *suffix-type T n* = *S-type*
by (*simp add*: ‹*length T* = *Suc n*› *suffix-type-last*)
hence *n* $\notin$ *l-bucket* α *T 0*

by (*simp add*: *l-bucket-def*)
hence *l-bucket-size* α *T 0* = *0*
proof −
have *l-bucket* α *T 0* $\subseteq$ $\{n\}$
by (*metis* ‹*bucket* α *T 0* = $\{n\}$› *l-bucket-subset-bucket*)
hence $\forall$ *n*. *n* $\notin$ *l-bucket* α *T 0*
using ‹*n* $\notin$ *l-bucket* α *T 0*› **by** *blast*
then show *?thesis*
by (*simp add*: *l-bucket-size-def*)
qed
ultimately
show *?thesis*
by *blast*
qed

lemma *bucket-0-size2*:
assumes *valid-list T*
and *strict-mono* α
and α *bot* = *0*
and *length T* = *Suc* (*Suc n*)
shows *bucket-size* α *T 0* = *Suc 0* $\wedge$ *l-bucket-size* α *T 0* = *0* $\wedge$ *lms-bucket-size* α *T 0* = *Suc 0* $\wedge$
pure-s-bucket-size α *T 0* = *0*
proof −
from *bucket-0*[*OF assms*(*1*,*3*,*2*,*4*)]
have *bucket* α *T 0* = $\{$*Suc n*$\}$ **.**

have *abs-is-lms T* (*Suc n*)
using *assms*(*1*,*4*) *abs-is-lms-last* **by** *blast*
hence *lms-bucket* α *T 0* = $\{$*Suc n*$\}$
using *lms-bucket-subset-bucket*[*of* α *T 0*] ‹*bucket* α *T 0* = $\{$*Suc n*$\}$›
by (*simp add*: *lms-bucket-def subset-antisym*)
hence *lms-bucket-size* α *T 0* = *Suc 0*
by (*simp add*: *lms-bucket-size-def*)
moreover
from ‹*bucket* α *T 0* = $\{$*Suc n*$\}$› ‹*lms-bucket* α *T 0* = $\{$*Suc n*$\}$›
have *pure-s-bucket* α *T 0* = $\{\}$
by (*metis disjoint-insert*(*1*) *disjoint-l-lms-bucket disjoint-lms-pure-s-bucket l-bucket-subset-bucket l-un-s-bucket pure-s-subset-s-bucket singletonI subset-singletonD sup-bot.left-neutral*)
hence *pure-s-bucket-size* α *T 0* = *0*
by (*simp add*: *pure-s-bucket-size-def*)
moreover
from *bucket-0-size1*[*OF assms*(*1*−*3*)]
have *bucket-size* α *T 0* = *Suc 0* $\wedge$ *l-bucket-size* α *T 0* = *0* **.**
ultimately
show *?thesis*
by *blast*
qed

definition *lms-bucket-start* :: ($'a$ :: {*linorder*,*order-bot*} ⇒ *nat*) ⇒ $'a$ *list* ⇒ *nat* ⇒ *nat*
  where
lms-bucket-start α *T b* = *bucket-start* α *T b* + *l-bucket-size* α *T b* + *pure-s-bucket-size* α *T b*

lemma *l-bucket-end-le-lms-bucket-start*:
  l-bucket-end α *T b* ≤ *lms-bucket-start* α *T b*
  by (*simp add*: *l-bucket-end-def lms-bucket-start-def*)

lemma *lms-bucket-start-le-bucket-end*:
  lms-bucket-start α *T b* ≤ *bucket-end* α *T b*
  by (*simp add*: *bucket-end-def′ lms-bucket-start-def l-pl-pure-s-pl-lms-size*)

lemma *lms-bucket-pl-size-eq-end*:
  lms-bucket-start α *T b* + *lms-bucket-size* α *T b* = *bucket-end* α *T b*
  by (*simp add*: *bucket-end-def′ l-pl-pure-s-pl-lms-size lms-bucket-start-def*)

47 Continuous Buckets

lemma *continuous-buckets*:
  continuous-list (*map* (λ*b*. (*bucket-start* α *T b*, *bucket-end* α *T b*)) [*i*..<*j*])
  by (*clarsimp simp*: *continuous-list-def bucket-start-Suc-eq-bucket-end*)

lemma *index-in-bucket-interval-gen*:
  ⟦*i* < *length T*; *strict-mono* α⟧ ⟹
    ∃ *b* ≤ α (*Max* (*set T*)). *bucket-start* α *T b* ≤ *i* ∧ *i* < *bucket-end* α *T b*
  apply (*insert continuous-buckets*[*of* α *T 0 Suc* (α (*Max* (*set T*)))])
  apply (*drule continuous-list-interval-2*[**where** *n* = α (*Max* (*set T*)) **and** *i* = *i*])
     apply *clarsimp*
    apply (*subst nth-map*)
     apply *clarsimp*
    apply (*clarsimp split*: *prod.split simp*: *upt-rec bucket-start-0*)
    apply (*subst nth-map*)
    apply *clarsimp*
   apply (*clarsimp split*: *prod.split simp*: *nth-append bucket-end-Max*)
  apply *clarsimp*
  apply (*clarsimp simp*: *nth-append split*: *if-splits prod.splits*)
  apply (*meson dual-order.order-iff-strict*)
  by *blast*

lemma *index-in-bucket-interval*:
  ⟦*i* < *length T*; *valid-list T*; α *bot* = *0*; *strict-mono* α⟧ ⟹
    ∃ *b* ≤ α (*Max* (*set T*)). *bucket-start* α *T b* ≤ *i* ∧ *i* < *bucket-end* α *T b*
  using *index-in-bucket-interval-gen* **by** *blast*

48 Bucket Initialisation

definition *lms-bucket-init* :: $('a :: \{linorder,order\text{-}bot\} \Rightarrow nat) \Rightarrow 'a\ list \Rightarrow nat\ list \Rightarrow bool$
where
$lms\text{-}bucket\text{-}init\ \alpha\ T\ B =$
$(\alpha\ (Max\ (set\ T)) < length\ B\ \wedge$
$(\forall\ b \leq \alpha\ (Max\ (set\ T)).\ B\ !\ b = bucket\text{-}end\ \alpha\ T\ b))$

lemma *lms-bucket-init-length*:
$lms\text{-}bucket\text{-}init\ \alpha\ T\ B \Longrightarrow \alpha\ (Max\ (set\ T)) < length\ B$
using *lms-bucket-init-def* **by** *blast*

lemma *lms-bucket-initD*:
$[\![lms\text{-}bucket\text{-}init\ \alpha\ T\ B;\ b \leq \alpha\ (Max\ (set\ T))]\!] \Longrightarrow B\ !\ b = bucket\text{-}end\ \alpha\ T\ b$
using *lms-bucket-init-def* **by** *blast*

definition *l-bucket-init* :: $('a :: \{linorder,order\text{-}bot\} \Rightarrow nat) \Rightarrow 'a\ list \Rightarrow nat\ list \Rightarrow bool$
where
$l\text{-}bucket\text{-}init\ \alpha\ T\ B =$
$(\alpha\ (Max\ (set\ T)) < length\ B\ \wedge$
$(\forall\ b \leq \alpha\ (Max\ (set\ T)).\ B\ !\ b = bucket\text{-}start\ \alpha\ T\ b))$

lemma *l-bucket-init-length*:
$l\text{-}bucket\text{-}init\ \alpha\ T\ B \Longrightarrow \alpha\ (Max\ (set\ T)) < length\ B$
using *l-bucket-init-def* **by** *blast*

lemma *l-bucket-initD*:
$[\![l\text{-}bucket\text{-}init\ \alpha\ T\ B;\ b \leq \alpha\ (Max\ (set\ T))]\!] \Longrightarrow B\ !\ b = bucket\text{-}start\ \alpha\ T\ b$
using *l-bucket-init-def* **by** *blast*

definition *s-bucket-init*
where
$s\text{-}bucket\text{-}init\ \alpha\ T\ B =$
$(\alpha\ (Max\ (set\ T)) < length\ B\ \wedge$
$(\forall\ b{\leq}\alpha\ (Max\ (set\ T)).$
$(b > 0 \longrightarrow B\ !\ b = bucket\text{-}end\ \alpha\ T\ b)\ \wedge$
$(b = 0 \longrightarrow B\ !\ b = 0)$
$)$
$)$

lemma *s-bucket-init-length*:
$s\text{-}bucket\text{-}init\ \alpha\ T\ B \Longrightarrow \alpha\ (Max\ (set\ T)) < length\ B$
using *s-bucket-init-def* **by** *blast*

lemma *s-bucket-initD*:
$[\![s\text{-}bucket\text{-}init\ \alpha\ T\ B;\ b \leq \alpha\ (Max\ (set\ T));\ b > 0]\!] \Longrightarrow B\ !\ b = bucket\text{-}end\ \alpha\ T\ b$

$[\![$*s-bucket-init* α *T B*; $b \le \alpha$ (*Max* (*set T*)); $b = 0]\!] \Longrightarrow B\ !\ b = 0$
using *s-bucket-init-def* **by** *auto*

49 Bucket Range

definition *in-s-current-bucket*
where
in-s-current-bucket α *T B b i* $\equiv$ ($b \le \alpha$ (*Max* (*set T*)) $\wedge$ $B\ !\ b \le i \wedge i <$ *bucket-end* α *T b*)

lemma *in-s-current-bucketD*:
in-s-current-bucket α *T B b i* $\Longrightarrow b \le \alpha$ (*Max* (*set T*))
in-s-current-bucket α *T B b i* $\Longrightarrow B\ !\ b \le i$
in-s-current-bucket α *T B b i* $\Longrightarrow i <$ *bucket-end* α *T b*
by (*simp-all add*: *in-s-current-bucket-def*)

definition *in-s-current-buckets*
where
in-s-current-buckets α *T B i* $\equiv$ ($\exists$ *b*. *in-s-current-bucket* α *T B b i*)

lemma *in-s-current-bucket-list-slice*:
assumes *length SA* = *length T*
and *in-s-current-bucket* α *T B b i*
and *SA* ! *i* = *x*
shows $x \in$ *set* (*list-slice SA* (*B* ! *b*) (*bucket-end* α *T b*))
by (*metis assms bucket-end-le-length in-s-current-bucket-def list-slice-nth-mem*)

definition *in-l-bucket*
where
in-l-bucket α *T b i* $\equiv$ ($b \le \alpha$ (*Max* (*set T*)) $\wedge$ *bucket-start* α *T b* $\le i \wedge i <$ *l-bucket-end* α *T b*)

end
theory *LMS-List-Slice-Util*
imports *List-Type*
begin

50 Helpers

lemma *filter-abs-is-lms-upt-0*:
filter (*abs-is-lms xs*) [*0*..<*n*] = *filter* (*abs-is-lms xs*) [*Suc 0*..<*n*]
by (*metis filter.simps*(*2*) *abs-is-lms-0 tl-upt upt-0 upt-rec*)

lemma *filter-abs-is-lms-upt-hd*:
$[\![$*abs-is-lms xs i*; $i < n]\!] \Longrightarrow$
filter (*abs-is-lms xs*) [*i*..<*n*] = *i* # *filter* (*abs-is-lms xs*) [*Suc i*..<*n*]
by (*metis filter.simps*(*2*) *upt-rec*)

51 LMS Slice

51.1 Find the next LMS position

fun
abs-find-index' :: $('a \Rightarrow bool) \Rightarrow 'a\ list \Rightarrow nat \Rightarrow nat$
where
abs-find-index' P xs i =
(*case xs of*
[] ⇒ *i*
| *x#xs'* ⇒
(*if P x*
then i
else abs-find-index' P xs' (*Suc i*)))

definition
abs-find-next-lms :: $('a :: \{linorder, order\text{-}bot\})\ list \Rightarrow nat \Rightarrow nat$
where
abs-find-next-lms T i =
(*case find* (λ*j. abs-is-lms T j*) [*Suc i..<length T*] *of*
Some j ⇒ *j*
| - ⇒ *length T*)

lemma *abs-find-next-lms-le-length*:
abs-find-next-lms T i ≤ *length T*
unfolding *abs-find-next-lms-def*
apply (*clarsimp split*: *option.split*)
by (*metis find-Some-iff abs-is-lms-gre-length not-less order.order-iff-strict*)

lemma *abs-find-next-lms-abs-is-lms*:
abs-is-lms T (*Suc i*) ⟹ *abs-find-next-lms T i* = *Suc i*
unfolding *abs-find-next-lms-def*
apply (*frule abs-is-lms-imp-less-length*)
apply (*clarsimp split*: *option.split simp*: *upt-rec*)
done

lemma *Suc-not-lms-imp-abs-find-next-eq-Suc*:
¬ *abs-is-lms T* (*Suc i*) ⟹ *abs-find-next-lms T i* = *abs-find-next-lms T* (*Suc i*)
unfolding *abs-find-next-lms-def*
by (*simp add*: *upt-rec*)

lemma *abs-find-next-lms-lower-bound-1*:
i < *length T* ⟹ *i* < *abs-find-next-lms T i*
unfolding *abs-find-next-lms-def*
apply (*clarsimp split*: *option.split*)
using *findSomeD* **by** *fastforce*

lemma *abs-find-next-lms-lower-bound-2*:
length T ≤ *i* ⟹ *length T* ≤ *abs-find-next-lms T i*
unfolding *abs-find-next-lms-def*

by (*clarsimp split*: *option.split*)

lemma *abs-find-next-lms-le-Suc*:
abs-find-next-lms T i ≤ *abs-find-next-lms T* (*Suc i*)
apply (*cases Suc i* < *length T*)
apply (*metis find.simps*(*2*) *abs-find-next-lms-def abs-find-next-lms-abs-is-lms abs-find-next-lms-lower-bound-1*
le-less upt-rec)
by (*simp add*: *abs-find-next-lms-def*)

lemma *no-lms-between-i-and-next*:
⟦*i* < *k*; *k* < *abs-find-next-lms T i*⟧ ⟹ ¬*abs-is-lms T k*
unfolding *abs-find-next-lms-def*
apply (*clarsimp split*: *option.splits*)
apply (*drule findNoneD*)
apply (*erule ballE*[*of* - - *k*])
apply *blast*
apply *simp*
apply (*drule find-Some-iff*[*THEN iffD1*])
apply *clarsimp*
apply (*erule allE*[*of* - *k* − *Suc i*])
apply *clarsimp*
done

lemma *abs-find-next-lms-less-length-abs-is-lms*:
abs-find-next-lms T i < *length T* ⟹
abs-is-lms T (*abs-find-next-lms T i*)
unfolding *abs-find-next-lms-def*
apply (*clarsimp split*: *option.splits*)
apply (*drule find-Some-iff*[*THEN iffD1*])
apply *clarsimp*
done

lemma *abs-find-next-lms-strict-upper-imp-lower-bound*:
abs-find-next-lms T i < *length T* ⟹
i < *abs-find-next-lms T i*
unfolding *abs-find-next-lms-def*
apply (*clarsimp split*: *option.splits*)
using *findSomeD* **by** *fastforce*

lemma *abs-find-next-lms-suffix*:
assumes *i* ≤ *length T*
shows *abs-find-next-lms T i* =
i + *abs-find-next-lms* (*suffix T i*) *0*
proof −
from *abs-is-lms-i-gr-0*[*of* - *i T*] *no-lms-between-i-and-next*[*of i* - *T*]
have *P*: ⋀*k*. ⟦*0* < *k*; *k* < *abs-find-next-lms T i* − *i*⟧ ⟹ ¬*abs-is-lms* (*suffix T i*) *k*
by (*meson less-add-same-cancel2 less-diff-conv*)

have *abs-find-next-lms T i = length T* ∨ *abs-find-next-lms T i < length T*
using *abs-find-next-lms-le-length le-neq-implies-less* **by** *blast*
moreover
have *abs-find-next-lms T i = length T* $\Longrightarrow$ *?thesis*
proof −
assume *abs-find-next-lms T i = length T*
hence $\bigwedge k$. ⟦*0 < k*; *k < length T − i*⟧ $\Longrightarrow$ ¬*abs-is-lms* (*suffix T i*) *k*
using *P* **by** *presburger*
hence *abs-find-next-lms* (*suffix T i*) *0 = length T − i*
by (*metis abs-find-next-lms-le-length abs-find-next-lms-less-length-abs-is-lms*
abs-find-next-lms-strict-upper-imp-lower-bound le-neq-implies-less
length-drop)
then show *?thesis*
by (*simp add*: ‹*abs-find-next-lms T i = length T*› *assms*)
qed
moreover
have *abs-find-next-lms T i < length T* $\Longrightarrow$ *?thesis*
proof −
assume *abs-find-next-lms T i < length T*
hence *abs-is-lms T* (*abs-find-next-lms T i*)
using *abs-find-next-lms-less-length-abs-is-lms* **by** *blast*
hence *abs-is-lms* (*suffix T i*) (*abs-find-next-lms T i − i*)
by (*simp add*: *abs-is-lms-i-gr-0* ‹*abs-find-next-lms T i < length T*›
abs-find-next-lms-strict-upper-imp-lower-bound less-or-eq-imp-le)
with *P*
show *?thesis*
by (*metis add.commute abs-find-next-lms-le-length abs-find-next-lms-less-length-abs-is-lms*
abs-find-next-lms-strict-upper-imp-lower-bound abs-is-lms-imp-less-length
le-neq-implies-less length-drop less-or-eq-imp-le nat-neq-iff
no-lms-between-i-and-next ordered-cancel-comm-monoid-diff-class.diff-add
zero-less-diff)
qed
ultimately show *?thesis*
by *blast*
qed

lemma *abs-find-next-lms-cons-Suc*:
assumes *i* ≤ *length xs*
shows *abs-find-next-lms* (*x # xs*) (*Suc i*) =
Suc (*abs-find-next-lms xs i*)
proof −
have *abs-find-next-lms xs i = length xs* ∨ *abs-find-next-lms xs i < length xs*
using *abs-find-next-lms-le-length le-neq-implies-less* **by** *blast*
moreover
have *abs-find-next-lms xs i = length xs* $\Longrightarrow$ *?thesis*
by (*metis Suc-le-mono add.assoc assms drop-Suc-Cons*
abs-find-next-lms-suffix length-Cons plus-1-eq-Suc)
moreover

have *abs-find-next-lms xs i* $<$ *length xs* $\Longrightarrow$ *?thesis*
by (*metis* (*no-types*, *lifting*) *Suc-le-mono add.assoc length-Cons assms drop-Suc-Cons abs-find-next-lms-suffix plus-1-eq-Suc*)
ultimately show *?thesis*
by *blast*
qed

lemma *abs-find-next-lms-funpow-Suc*:
$((\textit{abs-find-next-lms } T)^{\frown\frown}(\textit{Suc } k))\ i =$
$\textit{abs-find-next-lms } T\ (((\textit{abs-find-next-lms } T)^{\frown\frown}k)\ i)$
by *simp*

lemma *abs-find-next-lms-funpow-le*:
$i < \textit{length } T \Longrightarrow$
$((\textit{abs-find-next-lms } T)^{\frown\frown}k)\ i \leq$
$((\textit{abs-find-next-lms } T)^{\frown\frown}(\textit{Suc } k))\ i$
apply (*induct k*; *clarsimp*)
apply (*simp add*: *abs-find-next-lms-lower-bound-1 less-or-eq-imp-le*)
by (*simp add*: *abs-find-next-lms-le-Suc lift-Suc-mono-le*)

lemma *no-lms-between-i-and-next-funpow*:
$[\![((\textit{abs-find-next-lms } T)^{\frown\frown}k)\ i <$
$((\textit{abs-find-next-lms } T)^{\frown\frown}(\textit{Suc } k))\ i;$
$((\textit{abs-find-next-lms } T)^{\frown\frown}k)\ i < j;$
$j < ((\textit{abs-find-next-lms } T)^{\frown\frown}(\textit{Suc } k))\ i]\!] \Longrightarrow$
$\neg\ \textit{abs-is-lms } T\ j$
by (*simp add*: *no-lms-between-i-and-next*)

lemma *abs-find-next-lms-eq-Suc*:
$xs \neq [] \Longrightarrow \exists k.\ \textit{abs-find-next-lms } xs\ i = \textit{Suc } k$
by (*metis abs-find-next-lms-less-length-abs-is-lms abs-is-lms-0 length-greater-0-conv not0-implies-Suc*)

lemma *filter-no-lms1*:
$[\![\textit{abs-is-lms } xs\ i;\ i < k;\ k \leq \textit{abs-find-next-lms } xs\ i]\!] \Longrightarrow$
$\textit{filter } (\textit{abs-is-lms } xs)\ [\textit{Suc } i..<k] = []$
proof (*induct k*)
case *0*
then show *?case*
by *simp*
next
case (*Suc k*)
then show *?case*
by (*metis Suc-leD Suc-le-eq append-Nil filter.simps*(*1*,*2*) *upt-Suc filter-append no-lms-between-i-and-next*)
qed

lemma *filter-no-lms2*:
$[\![\neg\textit{abs-is-lms } xs\ i;\ i < k;\ k \leq \textit{abs-find-next-lms } xs\ i]\!] \Longrightarrow$

```
  filter (abs-is-lms xs) [i..<k] = []
proof (induct k)
  case 0
  then show ?case
    by simp
next
  case (Suc k)
  then show ?case
    by (metis Suc-le-eq append-Nil filter.simps(1)
              filter.simps(2) filter-append
              not-less-eq-eq upt.simps(2) nle-le
              no-lms-between-i-and-next upt-conv-Cons)
qed
```

51.2 LMS Prefix

```
fun
  closest-lms ::
    ('a :: {linorder, order-bot}) list ⇒ nat ⇒ nat
where
  closest-lms T i =
     (if abs-is-lms T i
      then i
      else abs-find-next-lms T i)

definition
  lms-prefix ::
     ('a :: {linorder, order-bot}) list ⇒ nat ⇒ 'a list
where
  lms-prefix T i =
    list-slice T i (Suc (closest-lms T i))


lemma lms-lms-prefix:
  abs-is-lms T i ⟹ lms-prefix T i = [T ! i]
  unfolding lms-prefix-def
  by (simp add: abs-is-lms-imp-less-length list-slice-Suc)

lemma suffix-to-lms-prefix:
  i < length T ⟹
     suffix T i =
     lms-prefix T i @
       (list-slice T (Suc (closest-lms T i)) (length T))
  unfolding lms-prefix-def
  apply clarsimp
  apply (intro impI conjI)
   apply (rule suffix-to-list-slice-app)
   apply linarith
  by (meson abs-find-next-lms-lower-bound-1 less-SucI
```

less-or-eq-imp-le suffix-to-list-slice-app)

lemma *abs-find-next-lms-funpow-all-lms*:
⟦*abs-is-lms xs* ((*abs-find-next-lms xs* ^^ *Suc k*) *x*);
i ≤ *k*⟧ ⟹
abs-is-lms xs ((*abs-find-next-lms xs* ^^ *Suc i*) *x*)
proof (*induct k arbitrary*: *i*)
case *0*
then show *?case*
by *blast*
next
case (*Suc k*)
note *IH* = *this*
hence (*abs-find-next-lms xs* ^^ *Suc* (*Suc k*)) *x* < *length xs*
using *abs-is-lms-imp-less-length* **by** *blast*
moreover
have *x* < *length xs*
by (*metis calculation abs-find-next-lms-funpow-Suc*
abs-find-next-lms-le-length abs-find-next-lms-lower-bound-2
abs-find-next-lms-strict-upper-imp-lower-bound funpow-swap1
linorder-not-less nat-neq-iff)
ultimately
have (*abs-find-next-lms xs* ^^ *Suc k*) *x* < *length xs*
using *abs-find-next-lms-funpow-le order.strict-trans1* **by** *blast*
hence *P*: *abs-is-lms xs* ((*abs-find-next-lms xs* ^^ *Suc k*) *x*)
by (*simp add*: *abs-find-next-lms-less-length-abs-is-lms*)

from *IH*(*3*)
have *i* ≤ *k* ∨ *i* = *Suc k*
by (*meson le-Suc-eq le-neq-implies-less*)
moreover
from *IH*(*1*)[*OF P*, *of i*]
have *i* ≤ *k* ⟹ *?case*
by *blast*
moreover
from *IH*(*2*)
have *i* = *Suc k* ⟹ *?case*
by *simp*
ultimately show *?case*
by *blast*
qed

51.3 LMS Slice

definition
lms-slice :: ('*a* :: {*linorder*, *order-bot*}) *list* ⇒ *nat* ⇒ '*a list*
where
lms-slice T i =
list-slice T i (*Suc* (*abs-find-next-lms T i*))

lemma *suffix-to-lms-slice*:
  $i < length\ T \Longrightarrow$
   suffix T i =
   lms-slice T i @
     (*list-slice T* (*Suc* (*abs-find-next-lms T i*)) (*length T*))
  unfolding *lms-slice-def*
  apply (*rule suffix-to-list-slice-app*)
  by (*simp add*: *abs-find-next-lms-lower-bound-1*
                le-Suc-eq less-or-eq-imp-le)

lemma *suffix-to-lms-slice-app-suffix*:
  $i < length\ T \Longrightarrow$
   suffix T i =
   lms-slice T i @
     (*suffix T* (*Suc* (*abs-find-next-lms T i*)))
  by (*metis suffix-eq-list-slice suffix-to-lms-slice*)

lemma *lms-slice-cons*:
  $[\![i < length\ T;\ suffix\text{-}type\ T\ i = S\text{-}type]\!] \Longrightarrow$
   lms-slice T i =
   T ! *i* # *lms-slice T* (*Suc i*)
  using *abs-is-lms-def Suc-not-lms-imp-abs-find-next-eq-Suc*
        abs-find-next-lms-lower-bound-1 i-s-type-imp-Suc-i-not-lms
        list-slice-Suc Suc-not-lms-imp-abs-find-next-eq-Suc
  by (*clarsimp simp*: *lms-slice-def*) *fastforce*

lemma *lms-slice-hd*:
  $i < length\ T \Longrightarrow$
   $\exists$ *xs*. *lms-slice T i* = *T* ! *i* # *xs*
 by (*simp add*: *abs-find-next-lms-lower-bound-1 less-SucI list-slice-Suc lms-slice-def*)

lemma *lms-slice-suffix*:
  assumes $i \leq length\ T$
  shows *lms-slice* (*suffix T i*) *0* =
        lms-slice T i
proof −
  from *list-slice-suffix*[*of T i Suc* (*abs-find-next-lms T i*)]
       lms-slice-def[*of T i*]
       abs-find-next-lms-suffix[*OF assms*]
       lms-slice-def[*of suffix T i 0*]
  show *?thesis*
    by (*metis add-Suc-right add-diff-cancel-left'*)
qed

lemma *lms-slice-suffix-gen*:
  assumes $i \leq length\ T$
  and $j \leq length\ T - i$
shows *lms-slice* (*suffix T i*) *j* =

lms-slice T (*i* + *j*)
proof −
have *lms-slice T* (*i* + *j*) =
lms-slice (*suffix T* (*i* + *j*)) *0*
by (*metis add.commute assms lms-slice-suffix le-diff-conv2*)
hence *lms-slice T* (*i* + *j*) =
lms-slice (*suffix* (*suffix T i*) *j*) *0*
by (*simp add*: *add.commute*)
moreover
have *lms-slice* (*suffix T i*) *j* = *lms-slice* (*suffix* (*suffix T i*) *j*) *0*
by (*metis assms*(*2*) *length-drop lms-slice-suffix*)
ultimately show *?thesis*
by *simp*
qed

lemma *lms-slice-cons-Suc*:
$i \leq$ *length xs* $\Longrightarrow$ *lms-slice* (*x* # *xs*) (*Suc i*) = *lms-slice xs i*
by (*metis Suc-le-mono drop-Suc-Cons length-Cons lms-slice-suffix*)

51.4 LMS Substring butlast

definition *lms-slice-butlast* :: (*'a* :: {*linorder*, *order-bot*}) *list* ⇒ *nat* ⇒ *'a list*
where
lms-slice-butlast T i = *list-slice T i* (*abs-find-next-lms T i*)

lemma *lms-slice-to-butlast-app*:
abs-find-next-lms T i < *length T* $\Longrightarrow$
lms-slice T i = *lms-slice-butlast T i* @ [*T* ! *abs-find-next-lms T i*]
unfolding *lms-slice-def lms-slice-butlast-def*
apply (*subst list-slice-append*[*of - abs-find-next-lms T i*])
apply (*simp add*: *abs-find-next-lms-strict-upper-imp-lower-bound order.strict-implies-order*)
apply *simp*
by (*simp add*: *list-slice-Suc*)

lemma *lms-slice-eq-butlast*:
length T $\leq$ *abs-find-next-lms T i* $\Longrightarrow$
lms-slice T i = *lms-slice-butlast T i*
by (*metis le-SucI list-slice-end-gre-length lms-slice-butlast-def lms-slice-def*)

lemma *lms-slice-eq-suffix*:
length T $\leq$ *abs-find-next-lms T i* $\Longrightarrow$
lms-slice T i = *suffix T i*
by (*simp add*: *list-slice.simps lms-slice-butlast-def lms-slice-eq-butlast*)

lemma *suffix-abs-find-next-lms*:
abs-find-next-lms T i < *length T* $\Longrightarrow$
suffix T i = *lms-slice-butlast T i* @ *suffix T* (*abs-find-next-lms T i*)
by (*simp add*: *abs-find-next-lms-strict-upper-imp-lower-bound less-or-eq-imp-le list-slice-append*

lms-slice-butlast-def suffix-eq-list-slice)

51.5 Suffix Types

lemma *suffix-type-lms-slice-l-s*:
  assumes *suffix-type T i = L-type*
  and *suffix-type T (Suc i) = S-type*
  shows *suffix-type (lms-slice T i) 0 = suffix-type T i*
proof −
  have *Suc i < length T*
    by (*simp add*: *assms*(*2*) *suffix-type-s-bound*)

  have *abs-is-lms T (Suc i)*
    by (*simp add*: *assms abs-is-lms-def*)
  hence *abs-find-next-lms T i = Suc i*
    by (*simp add*: *abs-find-next-lms-abs-is-lms*)
  hence *lms-slice T i = [T ! i, T ! Suc i]*
    by (*metis Suc-n-not-le-n ‹Suc i < length T› le-Suc-eq not-le*
        list-slice-Suc list-slice-n-n lms-slice-def)
  moreover
  have *T ! i > T ! Suc i*
    using ‹*abs-is-lms T (Suc i)*› *abs-is-lms-neq* **by** *blast*
  ultimately show *?thesis*
    by (*simp add*: ‹*suffix-type T i = L-type*› *suffix-type-cons-greater*)
qed

lemma *abs-find-next-lms-same-types*:
  assumes $\forall k.\ i \leq k \wedge k <$ *length T* $\longrightarrow$ *suffix-type T k = suffix-type T i*
  and $i \leq j$
shows *abs-find-next-lms T j = length T*
proof (*cases find (abs-is-lms T) [Suc j..<length T]*)
  assume *find (abs-is-lms T) [Suc j..<length T] = None*
  then show *abs-find-next-lms T j = length T*
    by (*simp add*: *abs-find-next-lms-def*)
next
  fix *x*
  assume *find (abs-is-lms T) [Suc j..<length T] = Some x*
  hence *x < length T Suc j ≤ x abs-is-lms T x*
   using ‹*find (abs-is-lms T) [Suc j..<length T] = Some x*› *findSomeD* **by** *force+*
  hence $\exists y.\ x = $ *Suc y*
    using *abs-is-lms-def* **by** *blast*
  then obtain *y* **where**
    x = Suc y
    by *blast*
  hence *suffix-type T y = L-type suffix-type T (Suc y) = S-type*
    using ‹*abs-is-lms T x*› *abs-is-lms-def* **by** *auto*

  have $i \leq y$
    using ‹*Suc j ≤ x*› ‹*x = Suc y*› *assms*(*2*) *le-trans* **by** *blast*

moreover
have $y < length\ T$
using *Suc-lessD* ‹$x < length\ T$› ‹$x = Suc\ y$› **by** *blast*
ultimately have *suffix-type* $T\ i = L\text{-}type$
by (*metis* ‹*suffix-type* $T\ y = L\text{-}type$› *assms*(*1*))

have $i \leq Suc\ y$
by (*simp add*: ‹$i \leq y$› *le-SucI*)
moreover
have $Suc\ y < length\ T$
using ‹$x < length\ T$› ‹$x = Suc\ y$› **by** *blast*
ultimately have *suffix-type* $T\ i = S\text{-}type$
by (*metis* ‹*suffix-type* $T\ (Suc\ y) = S\text{-}type$› *assms*(*1*))
with ‹*suffix-type* $T\ i = L\text{-}type$›
have $x = length\ T$
by *simp*
then show *?thesis*
using ‹$x < length\ T$› **by** *blast*
qed

lemma *lms-slice-same-types*:
assumes $\forall k.\ i \leq k \wedge k < length\ T \longrightarrow$ *suffix-type* $T\ k =$ *suffix-type* $T\ i$
and $i \leq j$
shows *lms-slice* $T\ j =$ *suffix* $T\ j$
proof −
have *abs-find-next-lms* $T\ j = length\ T$
using *assms abs-find-next-lms-same-types* **by** *blast*
then show *?thesis*
by (*metis le0 le-add-same-cancel2 list-slice-end-gre-length lms-slice-def plus-1-eq-Suc suffix-eq-list-slice*)
qed

lemma *all-l-types-up-to-next-lms*:
$[\![i \leq k;\ k <$ *abs-find-next-lms* $T\ i;$ *suffix-type* $T\ i = L\text{-}type]\!] \Longrightarrow$ *suffix-type* $T\ k = L\text{-}type$
proof(*induct* $k - i$ *arbitrary*: k)
case *0*
then show *?case* **by** *simp*
next
case (*Suc* x)
have $\exists k'.\ k = Suc\ k'$
using *Suc.hyps*(*2*) *Suc-le-D diff-le-self* **by** *presburger*
then obtain k' **where**
$k = Suc\ k'$
by *blast*
hence $x = k' - i$
using *Suc.hyps*(*2*) **by** *linarith*
moreover
have $i \leq k'$

    using *Suc.hyps*(*2*) ‹$k = Suc\ k'$› **by** *linarith*
  moreover
  have $k' <$ *abs-find-next-lms T i*
    using *Suc.prems*(*2*) *Suc-lessD* ‹$k = Suc\ k'$› **by** *blast*
  ultimately have *suffix-type T k'* = *L-type*
    using *Suc.hyps*(*1*) *Suc.prems*(*3*) **by** *blast*

  have $i < k$
    by (*simp add*: ‹$i \le k'$› ‹$k = Suc\ k'$› *le-imp-less-Suc*)
  with *Suc.prems*(*2*) *no-lms-between-i-and-next*[*of i k T*]
  have ¬ *abs-is-lms T k*
    by *blast*
  with ‹*suffix-type T k'* = *L-type*› ‹$k = Suc\ k'$›
  show *?case*
    using *SL-types.exhaust abs-is-lms-def* **by** *blast*
qed

lemma *abs-find-next-lms-eq-length*:
  assumes *abs-find-next-lms T i* = *length T*
  and $i <$ *length T*
shows *suffix-type T i* = *S-type*
proof (*rule ccontr*)
  assume *suffix-type T i* ≠ *S-type*
  hence *suffix-type T i* = *L-type*
    using *SL-types.exhaust* **by** *blast*
  moreover
  have ∃ *k*. *abs-find-next-lms T i* = *Suc k*
    by (*metis assms*(*1*) *assms*(*2*) *not-less0 old.nat.exhaust*)
  then obtain *k* **where**
    abs-find-next-lms T i = *Suc k*
    by *blast*
  moreover
  have $i \le k$
    using *assms*(*1*,*2*) *calculation* **by** *linarith*
  ultimately have *suffix-type T k* = *L-type*
    by (*metis all-l-types-up-to-next-lms lessI*)
  moreover
  have *length T* = *Suc k*
    using ‹*abs-find-next-lms T i* = *Suc k*› *assms*(*1*) **by** *auto*
  ultimately show *False*
    using *suffix-type-last*[*of T k*]
    by *simp*
qed

lemma *abs-find-next-lms-eq-length-all-s-types*:
  assumes *abs-find-next-lms T i* = *length T*
  and $i \le j$
  and $j <$ *length T*
shows *suffix-type T j* = *S-type*

by (*metis assms abs-find-next-lms-eq-length abs-find-next-lms-le-length abs-find-next-lms-less-length-abs-is-lms abs-find-next-lms-lower-bound-1 le-neq-implies-less no-lms-between-i-and-next order.strict-trans1*)

lemma *abs-find-next-lms-first-l-after-s-type*:
assumes *abs-find-next-lms* T i $<$ *length* T
and *suffix-type* T i $=$ *S-type*
shows $\exists j>i.\ j <$ *abs-find-next-lms* T i $\wedge$ ($\forall k<j.\ i \le k \longrightarrow$ *suffix-type* T k $=$ *S-type*) $\wedge$
suffix-type T j $=$ *L-type*
proof −
have $\exists j.$ *abs-find-next-lms* T i $=$ *Suc* j
by (*metis assms*(*2*) *abs-find-next-lms-lower-bound-1 not-less0 old.nat.exhaust suffix-type-s-bound*)
then obtain j **where**
abs-find-next-lms T i $=$ *Suc* j
by *blast*
hence *abs-is-lms* T (*Suc* j)
using *assms*(*1*) *abs-find-next-lms-less-length-abs-is-lms* **by** *fastforce*
hence *suffix-type* T j $=$ *L-type*
using *SL-types.exhaust i-s-type-imp-Suc-i-not-lms* **by** *auto*
moreover
have $j <$ *length* T
using ‹*abs-find-next-lms* T i $=$ *Suc* j› *assms*(*1*) **by** *linarith*
moreover
have $i \le j$
by (*metis* ‹*abs-find-next-lms* T i $=$ *Suc* j› *assms*(*1*) *abs-find-next-lms-strict-upper-imp-lower-bound less-Suc-eq-le*)
moreover
have $\forall k>i.\ k \le j \longrightarrow \neg$ *abs-is-lms* T k
by (*simp add*: ‹*abs-find-next-lms* T i $=$ *Suc* j› *no-lms-between-i-and-next*)
ultimately show *?thesis*
using *first-l-type-after-s-type*[*OF - - - -assms*(*2*), *of* j]
by (*metis SL-types.simps*(*2*) ‹*abs-find-next-lms* T i $=$ *Suc* j› *assms*(*2*) *dual-order.order-iff-strict le-imp-less-Suc*)
qed

lemma *lms-slice-type*:
assumes $i <$ *length* T
shows *suffix-type* (*lms-slice* T i) *0* $=$ *suffix-type* T i
proof −
have $\exists k.$ *abs-find-next-lms* T i $=$ *Suc* k
by (*meson Nat.lessE assms abs-find-next-lms-lower-bound-1*)
then obtain k **where**
abs-find-next-lms T i $=$ *Suc* k
by *blast*

have *suffix-type* T i $=$ *L-type* $\vee$ *suffix-type* T i $=$ *S-type*
using *SL-types.exhaust* **by** *blast*

moreover
have *suffix-type T i = L-type* $\Longrightarrow$ *?thesis*
proof –
assume *suffix-type T i = L-type*

have *suffix-type T (Suc i) = L-type* $\vee$ *suffix-type T (Suc i) = S-type*
using *SL-types.exhaust* **by** *blast*
moreover
have *suffix-type T (Suc i) = S-type* $\Longrightarrow$ *?thesis*
by (*simp add*: ‹*suffix-type T i = L-type*› *suffix-type-lms-slice-l-s*)
moreover
have *suffix-type T (Suc i) = L-type* $\Longrightarrow$ *?thesis*
proof –
assume *suffix-type T (Suc i) = L-type*

from ‹*abs-find-next-lms T i = Suc k*›
have *P*: $\forall k' \geq i.\ k' < Suc\ k \longrightarrow$ *suffix-type T k' = L-type*
by (*simp add*: ‹*suffix-type T i = L-type*› *all-l-types-up-to-next-lms*)

have *lms-slice T i = list-slice T i (Suc (Suc k))*
by (*simp add*: *lms-slice-def* ‹*abs-find-next-lms T i = Suc k*›)
moreover
{
have $i < k$
by (*metis SL-types.simps(2) Suc-lessI* ‹*abs-find-next-lms T i = Suc k*›
‹*suffix-type T (Suc i) = L-type*› ‹*suffix-type T i = L-type*› *assms*
abs-find-next-lms-less-length-abs-is-lms abs-find-next-lms-lower-bound-1
less-antisym
suffix-type-last suffix-type-same-imp-not-lms)
hence *list-slice T i (Suc (Suc k)) = list-slice T i k @ list-slice T k (Suc (Suc*
k))
by (*meson dual-order.order-iff-strict le-Suc-eq list-slice-append*)
moreover
have *list-slice T k (Suc (Suc k)) = [T ! k, T ! (Suc k)]*
by (*metis SL-types.simps(2)* ‹*abs-find-next-lms T i = Suc k*› ‹*suffix-type*
T i = L-type›
all-l-types-up-to-next-lms assms dual-order.order-iff-strict
abs-find-next-lms-le-length less-Suc-eq-le list-slice-Suc list-slice-n-n
not-less-eq suffix-type-last)
moreover
have *list-slice T i k = T ! i # list-slice T (Suc i) k*
using ‹$i < k$› *assms list-slice-Suc* **by** *blast*
ultimately have
list-slice T i (Suc (Suc k)) = T ! i # (list-slice T (Suc i) k) @ [T ! k, T !
Suc k]
by *simp*
}
ultimately have
lms-slice T i = T ! i # (list-slice T (Suc i) k) @ [T ! k, T ! Suc k]

by *simp*

let *?bs* = *list-slice T* (*Suc i*) *k*

have *abs-is-lms T* (*Suc k*)
by (*metis SL-types.simps*(*2*) ‹*abs-find-next-lms T i* = *Suc k*› ‹*suffix-type T i* = *L-type*›
all-l-types-up-to-next-lms assms dual-order.order-iff-strict suffix-type-last
abs-find-next-lms-le-length abs-find-next-lms-less-length-abs-is-lms less-Suc-eq-le)
hence *T* ! *k* > *T* ! *Suc k*
using *abs-is-lms-neq* **by** *blast*
moreover
from *sorted-letters-l-types*[*OF P*[*simplified* ‹*abs-find-next-lms T i* = *Suc k*›]]
have *sorted* (*rev* (*list-slice T i* (*Suc k*)))
using ‹*abs-is-lms T* (*Suc k*)› *abs-is-lms-gre-length linear* **by** *blast*
moreover
have *list-slice T i* (*Suc k*) = *T* ! *i* # (*list-slice T* (*Suc i*) *k*) @ [*T* ! *k*]
by (*metis SL-types.simps*(*2*) ‹*abs-find-next-lms T i* = *Suc k*› ‹*abs-is-lms T* (*Suc k*)›
‹*suffix-type T* (*Suc i*) = *L-type*› *assms dual-order.order-iff-strict not-less-eq*
abs-find-next-lms-le-length abs-find-next-lms-lower-bound-1 abs-is-lms-def list-slice-Suc not-less
list-slice-append list-slice-n-n)
ultimately have
list-less-ns (*?bs* @ [*T* ! *k*, *T* ! *Suc k*]) (*T* ! *i* # *?bs* @ [*T* ! *k*, *T* ! *Suc k*])
using *rev-sorted-list-less-ns*[*of T* ! *i ?bs T* ! *k T* ! *Suc k* [] []]
by *simp*
moreover
have *suffix* (*lms-slice T i*) *0* = *T* ! *i* # *?bs* @ [*T* ! *k*, *T* ! *Suc k*]
by (*simp add*: ‹*lms-slice T i* = *T* ! *i* # *?bs* @ [*T* ! *k*, *T* ! *Suc k*]›)
moreover
have *suffix* (*lms-slice T i*) (*Suc 0*) = *?bs* @ [*T* ! *k*, *T* ! *Suc k*]
by (*simp add*: ‹*lms-slice T i* = *T* ! *i* # *list-slice T* (*Suc i*) *k* @ [*T* ! *k*, *T* ! *Suc k*]›)
ultimately show *?thesis*
by (*metis* ‹*suffix-type T i* = *L-type*› *ordlistns.less-asym suffix-type-def*)
qed
ultimately show *?thesis*
by *blast*
qed
moreover
have *suffix-type T i* = *S-type* ⟹ *?thesis*
proof −
assume *suffix-type T i* = *S-type*
hence *lms-slice T i* = *T* ! *i* # *lms-slice T* (*Suc i*)
using *assms lms-slice-cons* **by** *blast*

have *abs-find-next-lms T i* < *length T* ∨ *abs-find-next-lms T i* = *length T*
by (*simp add*: *abs-find-next-lms-le-length nat-less-le*)
moreover
have *abs-find-next-lms T i* < *length T* ⟹ *?thesis*
proof −
assume *abs-find-next-lms T i* < *length T*
with *abs-find-next-lms-first-l-after-s-type*[*OF* - ‹*suffix-type T i* = *S-type*›]
obtain *j* **where**
i < *j*
j < *abs-find-next-lms T i*
∀ *k*<*j*. *i* ≤ *k* ⟶ *suffix-type T k* = *S-type*
suffix-type T j = *L-type*
by *blast*
hence *sorted* (*list-slice T i j*)
by (*meson* ‹*abs-find-next-lms T i* < *length T*› *dual-order.order-iff-strict order.strict-trans*
sorted-letters-s-types)
have ∃ *l*. *j* = *Suc l*
using ‹*i* < *j*› *less-imp-Suc-add* **by** *blast*
then obtain *l* **where**
j = *Suc l*
by *blast*

let *?xs* = *list-slice T* (*Suc i*) (*Suc l*)
and *?ys* = *list-slice T* (*Suc* ((*Suc l*))) (*Suc* (*Suc k*))

have *lms-slice T i* = *list-slice T i j* @ *list-slice T j* (*Suc* (*Suc k*))
by (*metis* ‹*abs-find-next-lms T i* = *Suc k*› ‹*i* < *j*› ‹*j* < *abs-find-next-lms T i*› *less-SucI*
less-imp-le-nat list-slice-append lms-slice-def)
moreover
have *list-slice T i j* = *T* ! *i* # *?xs*
using ‹*i* < *j*› ‹*j* = *Suc l*› *assms list-slice-Suc* **by** *blast*
moreover
have *list-slice T j* (*Suc* (*Suc k*)) = *T* ! *Suc l* # *?ys*
by (*metis* ‹*abs-find-next-lms T i* < *length T*› ‹*abs-find-next-lms T i* = *Suc k*› ‹*j* < *abs-find-next-lms T i*›
‹*j* = *Suc l*› *less-SucI list-slice-Suc order.strict-trans*)
ultimately have *lms-slice T i* = *T* ! *i* # *?xs* @ [*T* ! *Suc l*] @ *?ys*
by *auto*

have *i* = *l* ∨ *i* < *l*
using ‹*i* < *j*› *less-antisym* ‹*j* = *Suc l*› **by** *blast*
moreover
have *i* = *l* ⟹ *?thesis*
proof −
assume *i* = *l*
hence *?xs* = []

using *list-slice-n-n* **by** *blast*
hence *lms-slice T i = T ! i # [T ! Suc l] @ ?ys*
using ‹*lms-slice T i = T ! i # ?xs @ [T ! Suc l] @ ?ys*›
by *simp*
moreover
have *T ! i < T ! Suc l*
by (*metis SL-types.simps*(*2*) ‹*abs-find-next-lms T i < length T*› ‹*i = l*› ‹*j < abs-find-next-lms T i*›
‹*j = Suc l*› ‹*suffix-type T i = S-type*› ‹*suffix-type T j = L-type*› *suffix-type-neq*
le-imp-less-or-eq order.strict-trans s-type-letter-le-Suc)
ultimately show *?thesis*
by (*simp add*: ‹*suffix-type T i = S-type*› *suffix-type-cons-less*)
qed
moreover
have *i < l* $\Longrightarrow$ *?thesis*
proof −
let *?zs = list-slice T* (*Suc i*) *l*
assume *i < l*
hence *?xs = ?zs @ [T ! l]*
by (*metis Suc-leI Suc-n-not-le-n* ‹*abs-find-next-lms T i < length T*› ‹*j < abs-find-next-lms T i*›
‹*j = Suc l*› *lessI less-le-trans linear list-slice-Suc list-slice-append list-slice-n-n*)
hence *lms-slice T i = T ! i # ?zs @ [T ! l, T ! Suc l] @ ?ys*
using ‹*lms-slice T i = T ! i # ?xs @ [T ! Suc l] @ ?ys*›
by *simp*
moreover
have *suffix-type T l = S-type*
by (*simp add*: ‹$\forall$ *k<j. i* $\leq$ *k* $\longrightarrow$ *suffix-type T k = S-type*› ‹*i < l*› ‹*j = Suc l*› *less-or-eq-imp-le*)
hence *T ! l < T ! Suc l*
by (*metis SL-types.simps*(*2*) ‹*abs-find-next-lms T i < length T*› ‹*j < abs-find-next-lms T i*›
‹*j = Suc l*› ‹*suffix-type T j = L-type*› *order.strict-iff-order order.strict-trans*
s-type-letter-le-Suc suffix-type-neq)
ultimately show *?thesis*
using ‹*sorted* (*list-slice T i j*)›
sorted-list-less-ns[*of T ! i ?zs T ! l T ! Suc l ?ys ?ys*]
by (*metis* ‹*?xs = ?zs @ [T ! l]*› ‹*list-slice T i j = T ! i # ?xs*› *suffix-0 length-Cons*
‹*suffix-type T i = S-type*› *list.inject suffix-0 suffix-cons-Suc suffix-type-def zero-less-Suc*)
qed
ultimately show *?thesis*
by *blast*
qed
moreover

have *abs-find-next-lms T i = length T* $\Longrightarrow$ *?thesis*
proof –
assume *abs-find-next-lms T i = length T*
hence *P*: $\forall k' \geq i.\ k' <$ *length T* $\longrightarrow$ *suffix-type T k' = S-type*
using *abs-find-next-lms-eq-length-all-s-types* **by** *blast*

have *lms-slice T i = T ! i # list-slice T (Suc i) (length T)*
by (*metis* ‹*abs-find-next-lms T i = length T*› *assms leI less-not-refl list-slice-Suc*
lms-slice-butlast-def lms-slice-eq-butlast)
with *sorted-letters-s-types*[*OF P*]
sorted-cons-list-less-ns[*of T ! i list-slice T (Suc i) (length T)*]
show *?thesis*
by (*metis assms list-slice-Suc suffix-eq-list-slice suffix-type-suffix*)
qed

ultimately show *?thesis*
by *blast*
qed
ultimately show *?thesis*
by *blast*
qed

lemma *lms-slice-l-less-than-s-type-gen*:
assumes *suffix-type (a # as) 0 = L-type*
and *suffix-type (a # bs) 0 = S-type*
shows *list-less-ns (lms-slice (a # as) 0) (lms-slice (a # bs) 0)*
proof –
from *lms-slice-type*[*of 0 a # as*] *assms*(*1*)
have *suffix-type (lms-slice (a # as) 0) 0 = L-type*
by *simp*
moreover
have $\exists$*xs. lms-slice (a # as) 0 = a # xs*
by (*simp add*: *list-slice-Suc lms-slice-def*)
then obtain *xs* **where**
lms-slice (a # as) 0 = a # xs
by *blast*
moreover
from *lms-slice-type*[*of 0 a # bs*] *assms*(*2*)
have *suffix-type (lms-slice (a # bs) 0) 0 = S-type*
by *simp*
moreover
have $\exists$*xs. lms-slice (a # bs) 0 = a # xs*
by (*simp add*: *assms*(*2*) *lms-slice-cons*)
then obtain *ys* **where**
lms-slice (a # bs) 0 = a # ys
by *blast*
ultimately show *?thesis*
by (*simp add*: *l-less-than-s-type-general*)

qed

lemma *lms-slice-l-less-than-s-type*:
assumes $i < length\ T$
and $j < length\ T$
and $T\ !\ i = T\ !\ j$
and $suffix\text{-}type\ T\ i = L\text{-}type$
and $suffix\text{-}type\ T\ j = S\text{-}type$
shows *list-less-ns* (*lms-slice* $T\ i$) (*lms-slice* $T\ j$)
by (*metis assms abs-find-next-lms-lower-bound-1 l-less-than-s-type-general less-SucI list-slice-Suc*
lms-slice-def lms-slice-type)

lemma *lms-prefix-type*:
assumes $i < length\ T$
shows *suffix-type* (*lms-prefix* $T\ i$) 0 = *suffix-type* $T\ i$
proof −
have *abs-is-lms* $T\ i \lor \neg$*abs-is-lms* $T\ i$
by *blast*
moreover
have $\neg$*abs-is-lms* $T\ i \Longrightarrow$ *?thesis*
by (*metis assms closest-lms.simps lms-prefix-def lms-slice-def*
lms-slice-type)
moreover
have *abs-is-lms* $T\ i \Longrightarrow$ *?thesis*
by (*simp add*: *abs-is-lms-def lms-lms-prefix suffix-type-last*)
ultimately show *?thesis*
by *blast*
qed

lemma *lms-prefix-l-less-than-s-type-gen*:
assumes *suffix-type* (a # *as*) 0 = *L-type*
and *suffix-type* (a # *bs*) 0 = *S-type*
shows *list-less-ns* (*lms-prefix* (a # *as*) 0) (*lms-prefix* (a # *bs*) 0)
by (*metis assms closest-lms.simps lms-prefix-def abs-is-lms-def lessI lms-slice-def*
lms-slice-l-less-than-s-type-gen not-gr-zero not-less-iff-gr-or-eq)

lemma *lms-prefix-l-less-than-s-type*:
assumes $i < length\ T$
and $j < length\ T$
and $T\ !\ i = T\ !\ j$
and $suffix\text{-}type\ T\ i = L\text{-}type$
and $suffix\text{-}type\ T\ j = S\text{-}type$
shows *list-less-ns* (*lms-prefix* $T\ i$) (*lms-prefix* $T\ j$)
proof −
let $?a = T\ !\ i$

have $\exists$ *as*. *lms-prefix* $T\ i$ = $?a$ # *as*
by (*simp add*: *assms*(*1*) *lms-prefix-def abs-find-next-lms-lower-bound-1 less-SucI*

list-slice-Suc)
then obtain *as* **where**
lms-prefix T i = *?a # as*
by *blast*
hence *suffix-type* (*?a # as*) *0* = *L-type*
using *assms*(*1*,*4*) *lms-prefix-type* **by** *fastforce*

have ∃ *bs*. *lms-prefix T j* = *?a # bs*
by (*simp add*: *assms*(*2*,*3*) *lms-prefix-def abs-find-next-lms-lower-bound-1 less-SucI list-slice-Suc*)
then obtain *bs* **where**
lms-prefix T j = *?a # bs*
by *blast*
hence *suffix-type* (*?a # bs*) *0* = *S-type*
using *assms*(*2*) *assms*(*5*) *lms-prefix-type* **by** *fastforce*
with *l-less-than-s-type-general*[*OF* - ‹*suffix-type* (*?a # as*) *0* = -›, *of bs*]
have *list-less-ns* (*?a # as*) (*?a # bs*) **.**
then show *?thesis*
by (*simp add*: ‹*lms-prefix T i* = *T* ! *i # as*› ‹*lms-prefix T j* = *T* ! *i # bs*›)
qed

lemma *l-type-lms-prefix-cons*:
assumes *suffix-type T i* = *L-type*
and *i* < *length T*
shows *lms-prefix T i* = *T* ! *i # lms-prefix T* (*Suc i*)
proof −
have *suffix-type T* (*Suc i*) = *L-type* ∨ *suffix-type T* (*Suc i*) = *S-type*
using *SL-types.exhaust* **by** *blast*
moreover
have *suffix-type T* (*Suc i*) = *L-type* ⟹ *?thesis*
proof −
assume *suffix-type T* (*Suc i*) = *L-type*
hence ¬*abs-is-lms T* (*Suc i*)
by (*simp add*: ‹*suffix-type T i* = *L-type*› *suffix-type-same-imp-not-lms*)
with *Suc-not-lms-imp-abs-find-next-eq-Suc*
have *abs-find-next-lms T i* = *abs-find-next-lms T* (*Suc i*) **.**
hence *closest-lms T i* = *abs-find-next-lms T* (*Suc i*)
by (*simp add*: ‹*suffix-type T i* = *L-type*› *abs-is-lms-def*)
hence *lms-prefix T i* = *list-slice T i* (*Suc* (*abs-find-next-lms T* (*Suc i*)))
by (*simp add*: *lms-prefix-def*)
moreover
have *Suc i* < *Suc* (*abs-find-next-lms T* (*Suc i*))
using ‹*abs-find-next-lms T i* = *abs-find-next-lms T* (*Suc i*)› *assms*(*2*) *abs-find-next-lms-lower-bound-1*
by *force*
ultimately have
lms-prefix T i = *T* ! *i # list-slice T* (*Suc i*) (*Suc* (*abs-find-next-lms T* (*Suc i*)))
by (*simp add*: *assms*(*2*) *list-slice-Suc*)
then show *?thesis*

by (*simp add*: ‹¬ *abs-is-lms T* (*Suc i*)› *lms-prefix-def*)
qed
moreover
have *suffix-type T* (*Suc i*) = *S-type* ⟹ *?thesis*
proof −
assume *suffix-type T* (*Suc i*) = *S-type*
hence *abs-is-lms T* (*Suc i*)
by (*simp add*: *assms*(*1*) *abs-is-lms-def*)
hence *abs-find-next-lms T i* = *Suc i*
by (*simp add*: *abs-find-next-lms-abs-is-lms*)
hence *lms-prefix T i* = [*T* ! *i*, *T* ! *Suc i*]
by (*metis Suc-lessD* ‹*abs-is-lms T* (*Suc i*)› *assms*(*2*) *closest-lms.simps lms-prefix-def abs-is-lms-consec*(*1*) *lessI list-slice-Suc lms-lms-prefix*)
moreover
have *lms-prefix T* (*Suc i*) = [*T* ! *Suc i*]
by (*simp add*: ‹*abs-is-lms T* (*Suc i*)› *lms-lms-prefix*)
ultimately show *?thesis*
by *simp*
qed
ultimately show *?thesis*
by *blast*
qed

52 Ordering LMS-substrings

This section contains theorems about how LMS-substrings and suffixes are ordered.

lemma *lms-slice-eq-suffix-less*:
assumes *lms-slice T i* = *lms-slice T j*
shows *list-less-ns* (*suffix T i*) (*suffix T j*) ⟷
list-less-ns (*suffix T* (*abs-find-next-lms T i*)) (*suffix T* (*abs-find-next-lms T j*))
proof −
have ⟦*abs-find-next-lms T i* < *length T*; *abs-find-next-lms T j* < *length T*⟧ ⟹ *?thesis*
proof −
assume *A*: *abs-find-next-lms T i* < *length T abs-find-next-lms T j* < *length T*
have *suffix T i* = *lms-slice-butlast T i* @ *suffix T* (*abs-find-next-lms T i*)
using *A*(*1*) *suffix-abs-find-next-lms* **by** *blast*
moreover
have *suffix T j* = *lms-slice-butlast T j* @ *suffix T* (*abs-find-next-lms T j*)
using *A*(*2*) *suffix-abs-find-next-lms* **by** *blast*
moreover
have *lms-slice-butlast T i* = *lms-slice-butlast T j*
by (*metis A*(*1*) *A*(*2*) *assms butlast-snoc lms-slice-to-butlast-app*)
ultimately show *?thesis*
by (*simp add*: *list-less-ns-app-same*)
qed

moreover
have ⟦*abs-find-next-lms T i* = *length T*; *abs-find-next-lms T j* = *length T*⟧ ⟹
?thesis
by (*metis assms dual-order.refl lms-slice-eq-suffix ordlistns.less-irrefl*)
moreover
have ⟦*abs-find-next-lms T i* = *length T*; *abs-find-next-lms T j* < *length T*⟧ ⟹
?thesis
proof −
assume *A*: *abs-find-next-lms T i* = *length T abs-find-next-lms T j* < *length T*
have *suffix T i* = *lms-slice T i*
by (*simp add*: *A*(*1*) *lms-slice-eq-suffix*)
moreover
have *suffix T j* = *lms-slice T j* @ *suffix T* (*Suc* (*abs-find-next-lms T j*))
by (*meson A*(*2*) *abs-find-next-lms-lower-bound-2 linorder-not-less suffix-to-lms-slice-app-suffix*)
ultimately show *?thesis*
by (*metis A*(*1*) *append.right-neutral assms cancel-comm-monoid-add-class.diff-cancel length-0-conv length-drop list-less-ns-app ordlistns.not-less-iff-gr-or-eq ordlistns.top.extremum-strict*)
qed
moreover
have ⟦*abs-find-next-lms T i* < *length T*; *abs-find-next-lms T j* = *length T*⟧ ⟹
?thesis
proof −
assume *A*: *abs-find-next-lms T i* < *length T abs-find-next-lms T j* = *length T*
have *suffix T i* = *lms-slice T i* @ *suffix T* (*Suc* (*abs-find-next-lms T i*))
by (*meson A*(*1*) *abs-find-next-lms-lower-bound-2 linorder-not-less suffix-to-lms-slice-app-suffix*)
moreover
have *suffix T j* = *lms-slice T j*
by (*simp add*: *A*(*2*) *lms-slice-eq-suffix*)
ultimately show *?thesis*
by (*metis A append-Nil2 assms drop-eq-Nil2 abs-find-next-lms-lower-bound-2
linorder-le-less-linear
list-less-ns-app nless-le ordlistns.top.not-eq-extremum suffix-neq-nil
suffixes-neq*)
qed
ultimately show *?thesis*
by (*meson abs-find-next-lms-le-length le-neq-implies-less*)
qed

lemma *lms-slice-eq-suffix-less-funpow′*:
assumes $\forall k < n$. *lms-slice T* (((*abs-find-next-lms T*)^^*k*) *i*) =
lms-slice T (((*abs-find-next-lms T*)^^*k*) *j*)
and $k < n$
shows *list-less-ns* (*suffix T i*) (*suffix T j*) ⟷
list-less-ns (*suffix T* (((*abs-find-next-lms T*)^^*k*) *i*)) (*suffix T* (((*abs-find-next-lms
T*)^^*k*) *j*))
using *assms*

proof (*induct n arbitrary*: *k*)
  case *0*
  then show *?case*
    by *blast*
next
  case (*Suc n*)
  note *IH* = *this*
  have $k < n \Longrightarrow$ *?case*
    by (*simp add*: *Suc.hyps Suc.prems*(*1*))
  moreover
  have $k = n \Longrightarrow$ *?case*
  proof −
    assume $k = n$

    have $k = 0 \Longrightarrow$ *?thesis*
      by *auto*
    moreover
    have $\exists m.\ k = Suc\ m \Longrightarrow$ *?thesis*
    proof −
      assume $\exists m.\ k = Suc\ m$
      then obtain *m* **where** $k = Suc\ m$
        by *blast*
      hence $m < n$
        using ‹$k = n$› **by** *blast*
      with *IH*(*1*)[*of m*]
      have *list-less-ns* (*suffix T i*) (*suffix T j*) =
            list-less-ns (*suffix T* ((*abs-find-next-lms T* ^^ *m*) *i*))
                        (*suffix T* ((*abs-find-next-lms T* ^^ *m*) *j*))
        using *Suc.prems*(*1*) *less-Suc-eq-le less-or-eq-imp-le* **by** *presburger*
      moreover
      have *list-less-ns* (*suffix T* ((*abs-find-next-lms T* ^^ *m*) *i*))
                        (*suffix T* ((*abs-find-next-lms T* ^^ *m*) *j*)) =
              list-less-ns (*suffix T* (*abs-find-next-lms T* ((*abs-find-next-lms T* ^^ *m*)
i)))
                        (*suffix T* (*abs-find-next-lms T* ((*abs-find-next-lms T* ^^ *m*) *j*)))
       using *Suc.prems*(*1*,*2*) *Suc-lessD* ‹$k = Suc\ m$› *lms-slice-eq-suffix-less* **by** *blast*
      ultimately show *?thesis*
        by (*simp add*: ‹$k = Suc\ m$›)
    qed
    ultimately show *?thesis*
      using *not0-implies-Suc* **by** *blast*
  qed
  ultimately show *?case*
    using *Suc.prems*(*2*) *less-Suc-eq* **by** *blast*
qed

lemma *lms-slice-eq-suffix-less-funpow*:
  assumes $\forall k < n.$ *lms-slice T* (((*abs-find-next-lms T*)^^*k*) *i*) =
                  lms-slice T (((*abs-find-next-lms T*)^^*k*) *j*)

shows *list-less-ns* (*suffix T i*) (*suffix T j*) $\longleftrightarrow$
list-less-ns (*suffix T* (((*abs-find-next-lms T*)$\widehat{\ }\widehat{\ }$*n*) *i*)) (*suffix T* (((*abs-find-next-lms T*)$\widehat{\ }\widehat{\ }$*n*) *j*))
proof (*cases n*)
case *0*
then show *?thesis*
by *auto*
next
case (*Suc m*)
then show *?thesis*
by (*metis assms abs-find-next-lms-funpow-Suc lessI lms-slice-eq-suffix-less lms-slice-eq-suffix-less-funpow'*)
qed

lemma *list-slice-single*:
$i < length\ xs \Longrightarrow list\text{-}slice\ xs\ i\ (Suc\ i) = [xs\ !\ i]$
by (*simp add*: *list-slice-Suc*)

lemma *less-lms-slice-imp-suffix*:
assumes $i < length\ T$
and $j < length\ T$
and *list-less-ns* (*lms-slice T i*) (*lms-slice T j*)
shows *list-less-ns* (*suffix T i*) (*suffix T j*)
proof −

let $?c1 = \exists\, b\ c\ as\ bs\ cs.\ lms\text{-}slice\ T\ i = as\ @\ b\ \#\ bs\ \wedge$
$lms\text{-}slice\ T\ j = as\ @\ c\ \#\ cs\ \wedge\ b < c$
let $?c2 = \exists\, c\ cs.\ lms\text{-}slice\ T\ i = lms\text{-}slice\ T\ j\ @\ c\ \#\ cs$

from *list-less-ns-exE*[*OF assms*(*3*)[*simplified list-less-ns-alt-def*]]
have $?c1 \vee ?c2$ **.**
moreover
have $?c1 \Longrightarrow ?thesis$
by (*metis append.assoc append-Cons assms*(*1*,*2*) *list-less-ns-app-same list-less-ns-cons-diff suffix-to-lms-slice-app-suffix*)
moreover
have $?c2 \Longrightarrow ?thesis$
proof −
assume *?c2*
then obtain *c cs* **where**
$lms\text{-}slice\ T\ i = lms\text{-}slice\ T\ j\ @\ c\ \#\ cs$
by *blast*
moreover
from *lms-slice-hd*[*OF assms*(*2*)]
obtain *xs* **where**
Sj: $lms\text{-}slice\ T\ j = T\ !\ j\ \#\ xs$
by *blast*
ultimately have *Si*: $lms\text{-}slice\ T\ i = T\ !\ j\ \#\ xs\ @\ c\ \#\ cs$
by *simp*

let *?i* = *abs-find-next-lms T i*
let *?j* = *abs-find-next-lms T j*
have ∃ *k*. *?j* = *Suc k*
  by (*meson Nat.lessE assms*(*1*) *dual-order.strict-trans1*
      abs-find-next-lms-strict-upper-imp-lower-bound linorder-not-less)
then obtain *k* **where**
  ?j = *Suc k*
  by *blast*
hence $j \le k$
  using *assms*(*2*) *abs-find-next-lms-lower-bound-1* **by** *force*

have *?j* = *length T* ⟹ *?thesis*
  by (*metis append-Nil2 assms*(*1*,*3*) *abs-find-next-lms-le-length list-less-ns-app*
      lms-slice-eq-suffix ordlistns.dual-order.strict-trans
      suffix-to-lms-slice-app-suffix)
moreover
have *?j* < *length T* ⟹ *?thesis*
proof −
  assume *?j* < *length T*
  with *lms-slice-to-butlast-app*
  have *lms-slice T j* = *lms-slice-butlast T j* @ [*T* ! *Suc k*]
    using ‹*?j* = *Suc k*› **by** *fastforce*
  moreover
  have ∃ *ys*. *lms-slice-butlast T j* = *ys* @ [*T* ! *k*]
  proof (*cases j* = *k*)
    case *True*
    hence *lms-slice-butlast T j* = *list-slice T k* (*Suc k*)
      by (*metis* ‹*?j* = *Suc k*› *lms-slice-butlast-def*)
    moreover
    have *list-slice T k* (*Suc k*) = [*T* ! *k*]
      using *True assms*(*2*) *list-slice-single* **by** *auto*
    ultimately show *?thesis*
      by *simp*
  next
    case *False*
    hence *j* < *k*
      by (*simp add*: ‹$j \le k$› *le-neq-implies-less*)
    hence *list-slice T j* (*Suc k*) = *list-slice T j k* @ [*T* ! *k*]
        by (*metis* ‹*?j* < *length T*› ‹*?j* = *Suc k*› ‹$j \le k$› *le-SucI le-add2*
list-slice-append
        list-slice-single not-less plus-1-eq-Suc)
    then show *?thesis*
      by (*simp add*: ‹*?j* = *Suc k*› *lms-slice-butlast-def*)
  qed
  then obtain *ys* **where**
    lms-slice-butlast T j = *ys* @ [*T* ! *k*]
    by *blast*
  ultimately have *Sj'*: *lms-slice T j* = *ys* @ [*T* ! *k*, *T* ! *Suc k*]

```
      by simp

    have Si': lms-slice T i = ys @ T ! k # T ! Suc k # c # cs
      using Si Sj Sj' by auto

    have suffix-type T k = L-type
    by (metis ‹?j < length T› ‹?j = Suc k› abs-find-next-lms-less-length-abs-is-lms
            abs-is-lms-neq nth-gr-imp-l-type)

    have abs-is-lms T (Suc k)
    by (metis ‹?j < length T› ‹?j = Suc k› abs-find-next-lms-less-length-abs-is-lms)
    hence T ! k > T ! Suc k
      using abs-is-lms-neq by blast

    have Si: suffix T i = ys @ T ! k # T ! Suc k # c # cs @ suffix T (Suc ?i)
      by (simp add: Si' assms(1) suffix-to-lms-slice-app-suffix)
    moreover
    have suffix T j = ys @ T ! k # T ! Suc k # suffix T (Suc ?j)
      by (simp add: ‹lms-slice T j = ys @ [T ! k, T ! Suc k]› assms(2)
                suffix-to-lms-slice-app-suffix)
    ultimately have list-less-ns (suffix T i) (suffix T j) ⟷
                  list-less-ns (T ! k # T ! Suc k # c # cs @ suffix T (Suc ?i))
                              (T ! k # T ! Suc k # suffix T (Suc ?j))
      by (simp add: list-less-ns-app-same)
    moreover
    have suffix-type (T ! k # T ! Suc k # c # cs @ suffix T (Suc ?i)) (Suc 0)
= L-type ⟹ ?thesis
       by (metis Cons-nth-drop-Suc ‹?j < length T› ‹?j = Suc k› ‹abs-is-lms T
(Suc k)› calculation
                 abs-is-lms-def l-less-than-s-type-general list-less-ns-cons-same
suffix-type-cons-suc
            suffix-type-suffix)
    moreover
    have suffix-type (T ! k # T ! Suc k # c # cs @ suffix T (Suc ?i)) (Suc 0)
= S-type ⟹ ?thesis
    proof -
      assume suffix-type (T ! k # T ! Suc k # c # cs @ suffix T (Suc ?i)) (Suc
0) = S-type
      with Si
      have suffix-type T (Suc (i + length ys)) = S-type
        by (metis One-nat-def plus-1-eq-Suc suffix-cons-app suffix-type-suffix-gen)
      moreover
       have suffix-type (T ! k # T ! Suc k # c # cs @ suffix T (Suc ?i)) 0 =
L-type
        using ‹T ! Suc k < T ! k› suffix-type-cons-greater by blast
      hence suffix-type T (i + length ys) = L-type
        by (simp add: Si suffix-cons-app suffix-type-list-type-eq)
      ultimately have abs-is-lms T (Suc (i + length ys))
        using abs-is-lms-def by blast
```

```
        moreover
        {
          have i < ?i
            by (simp add: assms(1) abs-find-next-lms-lower-bound-1)

          from ‹lms-slice T i = ys @ T ! k # T ! Suc k # c # cs›
          have Suc (Suc (length ys)) < length (lms-slice T i)
            by simp

          have ?i = length T ⟹ Suc (i + length ys) < ?i
         by (simp add: ‹abs-is-lms T (Suc (i + length ys))› abs-is-lms-imp-less-length)
          moreover
          have ?i < length T ⟹ Suc (i + length ys) < ?i
          proof -
            assume ?i < length T
            hence length (lms-slice T i) = Suc ?i - i
              by (simp add: lms-slice-def Suc-leI)
            hence Suc (Suc (length ys)) < Suc ?i - i
              using ‹Suc (Suc (length ys)) < length (lms-slice T i)› by presburger
            then show ?thesis
              by linarith
          qed
          ultimately have Suc (i + length ys) < ?i
            using abs-find-next-lms-le-length le-neq-implies-less by blast
        }
        ultimately show ?thesis
           using no-lms-between-i-and-next[of i Suc (i + length ys)] less-add-Suc1
by blast
      qed
      ultimately show ?thesis
        using SL-types.exhaust by blast
    qed
    ultimately show ?thesis
      using abs-find-next-lms-le-length le-neq-implies-less by blast
  qed
  ultimately show ?thesis
    by blast
qed

lemma lms-slice-list-less-ns-suffix:
  assumes abs-is-lms T i
  and     abs-is-lms T j
  and     list-less-ns (lms-slice T i) (lms-slice T j)
shows list-less-ns (suffix T i) (suffix T j)
  by (simp add: assms abs-is-lms-imp-less-length less-lms-slice-imp-suffix)

lemma less-suffix-imp-lms-slice:
  assumes i < length T
  and     j < length T
```

and *lms-slice T i* $\neq$ *lms-slice T j*
and *list-less-ns* (*suffix T i*) (*suffix T j*)
shows *list-less-ns* (*lms-slice T i*) (*lms-slice T j*)
by (*meson assms less-lms-slice-imp-suffix ordlistns.less-asym ordlistns.neqE*)

lemma *not-lms-imp-next-eq-lms-prefix*:
$\neg$*abs-is-lms T i* $\Longrightarrow$ *lms-slice T i* = *lms-prefix T i*
by (*simp add*: *lms-prefix-def lms-slice-def*)

lemma *lms-slice-last*:
assumes *valid-list T*
and *length T* = *Suc n*
shows *lms-slice T n* = [*bot*]
by (*metis add-diff-cancel-left′ assms butlast-snoc abs-find-next-lms-lower-bound-1 le-Suc-eq*
length-butlast less-Suc-eq list-slice-Suc list-slice-start-gre-length lms-slice-def nth-append-length plus-1-eq-Suc valid-list-ex-def)

lemma *Min-valid-lms-slice*:
assumes *valid-list T*
and *length T* = *Suc n*
shows *ordlistns.Min* {*lms-slice T i* |*i*. *i* < *length T*} = *lms-slice T n*
proof −
from *lms-slice-last*[*OF assms*]
have *lms-slice T n* = [*bot*]
by *assumption*

have $\forall$ *i* < *n*. (*lms-slice T i*) ! *0* $\neq$ *bot*
by (*metis add-diff-cancel-left′ assms abs-find-next-lms-lower-bound-1 less-SucI list-slice-Suc*
lms-slice-def nth-Cons-0 plus-1-eq-Suc valid-list-def)
hence *A*: $\forall$ *i* < *n*. *bot* < (*lms-slice T i*) ! *0*
using *bot.not-eq-extremum* **by** *blast*

have *B*: $\forall$ *i* < *length T*. *length* (*lms-slice T i*) > *0*
by (*simp add*: *abs-find-next-lms-lower-bound-1 less-SucI list-slice-Suc lms-slice-def*)

show *?thesis*
proof (*intro ordlistns.Min-eqI conjI*)
show *finite* {*lms-slice T i* |*i*. *i* < *length T*}
using *finite-image-set* **by** *blast*
next
fix *ys*
assume *ys* $\in$ {*lms-slice T i* |*i*. *i* < *length T*}
hence $\exists$ *i* < *length T*. *ys* = *lms-slice T i*
by *blast*
then obtain *i* **where**
i < *length T*
ys = *lms-slice T i*

by *blast*

with ‹$ys = lms\text{-}slice\ T\ i$›
have *R1*: $i = n \Longrightarrow list\text{-}less\text{-}eq\text{-}ns\ (lms\text{-}slice\ T\ n)\ ys$
by *simp*

from ‹$i < length\ T$› *assms*(*2*)
have *R2-1*: $i \neq n \Longrightarrow i < n$
by *linarith*

from *A* ‹$lms\text{-}slice\ T\ n = [bot]$› ‹$i < length\ T$› ‹$ys = lms\text{-}slice\ T\ i$›
have *R2-2*: $i < n \Longrightarrow list\text{-}less\text{-}eq\text{-}ns\ (lms\text{-}slice\ T\ n)\ ys$
using *list-less-eq-ns-def* **by** *fastforce*

from *R1 R2-2*[*OF R2-1*]
show *list-less-eq-ns* (*lms-slice T n*) *ys*
by *blast*
next
show $lms\text{-}slice\ T\ n \in \{lms\text{-}slice\ T\ i\ |i.\ i < length\ T\}$
using *assms*(*2*) **by** *auto*
qed
qed

lemma *unique-valid-lms-slice*:
assumes *valid-list T*
and $length\ T = Suc\ n$
shows $\forall i < n.\ lms\text{-}slice\ T\ i \neq lms\text{-}slice\ T\ n$
proof (*intro allI impI*)
fix *i*
assume $i < n$
from *lms-slice-last*[*OF assms*]
have $lms\text{-}slice\ T\ n = [bot]$
by *assumption*

have $\forall i < n.\ (lms\text{-}slice\ T\ i)\ !\ 0 \neq bot$
by (*metis add-diff-cancel-left′ assms abs-find-next-lms-lower-bound-1 less-SucI list-slice-Suc*
lms-slice-def nth-Cons-0 plus-1-eq-Suc valid-list-def)
hence $\forall i < n.\ bot < (lms\text{-}slice\ T\ i)\ !\ 0$
using *bot.not-eq-extremum* **by** *blast*
with ‹$lms\text{-}slice\ T\ n = [bot]$› ‹$i < n$›
show $lms\text{-}slice\ T\ i \neq lms\text{-}slice\ T\ n$
by *auto*
qed

lemma *strict-Min-valid-lms-slice*:
assumes *valid-list T*
and $length\ T = Suc\ n$
shows $\forall i < n.\ list\text{-}less\text{-}ns\ (lms\text{-}slice\ T\ n)\ (lms\text{-}slice\ T\ i)$

by (*metis add-diff-cancel-left′ assms bot.not-eq-extremum abs-find-next-lms-lower-bound-1 less-Suc-eq*
list-less-ns-cons-diff list-slice-Suc lms-slice-def lms-slice-last plus-1-eq-Suc valid-list-def)

lemma *ordlistns-lms-slice-imp-suffix-strict-sorted*:
assumes *set xs* $\subseteq$ {*i*. *abs-is-lms T i*} *ordlistns.strict-sorted* (*map* (*lms-slice T*) *xs*)
shows *ordlistns.strict-sorted* (*map* (*suffix T*) *xs*)
proof (*intro sorted-wrt-mapI*)
fix *i j*
assume *i* < *j j* < *length xs*
with *sorted-wrt-mapD*[*OF assms*(*2*), *of i j*]
have *list-less-ns* (*lms-slice T* (*xs* ! *i*)) (*lms-slice T* (*xs* ! *j*))
by *blast*
moreover
have *abs-is-lms T* (*xs* ! *i*)
using ‹*i* < *j*› ‹*j* < *length xs*› *assms*(*1*) *subsetD* **by** *fastforce*
hence *xs* ! *i* < *length T*
by (*simp add*: *abs-is-lms-imp-less-length*)
moreover
have *abs-is-lms T* (*xs* ! *j*)
using ‹*j* < *length xs*› *assms*(*1*) *nth-mem* **by** *auto*
hence *xs* ! *j* < *length T*
by (*simp add*: *abs-is-lms-imp-less-length*)
ultimately show *list-less-ns* (*suffix T* (*xs* ! *i*)) (*suffix T* (*xs* ! *j*))
using *less-lms-slice-imp-suffix* **by** *blast*
qed

53 Mapping from suffix to lists of LMS-Substrings

This section contains the mapping from LMS-type suffixes to suffixes of the reduced sequence. The mapping is constructed in 3 major steps. 1) From suffix ID to a sequence of LMS-type suffix IDs 2) From a sequence of LMS-type suffix IDs to a sequence of LMS-substrings 3) From a LMS-type suffix to a reduced suffix using the mappings 1, 2 and *ordlistns.elem-rank* The mapping is also shown to be monotonic.

abbreviation *lms-substrs xs* ≡ *lms-slice xs* ‘ {*i*. *abs-is-lms xs i*}
abbreviation *lms-suffixes xs* ≡ *suffix xs* ‘ {*i*. *abs-is-lms xs i*}

abbreviation *nth-lms xs i* ≡ (*abs-find-next-lms xs* ^^ *Suc i*) *0*

abbreviation *lms0 xs* ≡ *abs-find-next-lms xs 0*
abbreviation *lms0-suffix xs* ≡ *suffix xs* (*lms0 xs*)
abbreviation *lms0-substr xs* ≡ *lms-slice xs* (*lms0 xs*)

53.1 LMS Sequence

definition *lms-seq* :: *'a* :: {*linorder*,*order-bot*} *list* $\Rightarrow$ *nat* $\Rightarrow$ *nat list*
  where
lms-seq xs i = *filter* (*abs-is-lms xs*) [*i*..<*length xs*]

lemma *lms-seq-distinct*:
  distinct (*lms-seq xs i*)
  by (*simp add*: *lms-seq-def*)

lemma *lms-seq-sorted*:
  sorted (*lms-seq xs i*)
  by (*simp add*: *filter-sorted lms-seq-def*)

lemma *lms-seq-strict-sorted*:
  strict-sorted (*lms-seq xs i*)
 by (*simp add*: *lms-seq-distinct lms-seq-sorted sorted-and-distinct-imp-strict-sorted*)

lemma *lms-seq-abs-is-lms-hd*:
  abs-is-lms xs i $\Longrightarrow$ $\exists$ *ys*. *lms-seq xs i* = *i* # *ys*
  by (*simp add*: *filter-abs-is-lms-upt-hd abs-is-lms-imp-less-length lms-seq-def*)

lemma *length-lms-seq*:
  assumes *abs-is-lms xs i*
  shows *length* (*lms-seq xs i*) = *card* {*j*. *abs-is-lms xs j* $\wedge$ *i* $\leq$ *j*}
proof −
  from *distinct-length-filter*[*of* [*i*..<*length xs*] *abs-is-lms xs*]
  have *length* (*lms-seq xs i*) = *card* ({*x*. *abs-is-lms xs x*} $\cap$ {*i*..<*length xs*})
    by (*simp add*: *lms-seq-def*)
  moreover
  have {*x*. *abs-is-lms xs x*} $\cap$ {*i*..<*length xs*} = {*x*. *abs-is-lms xs x* $\wedge$ *i* $\leq$ *x*}
    by (*safe*; *clarsimp simp*: *abs-is-lms-imp-less-length*)
  ultimately show *?thesis*
    by *simp*
qed

lemma *length-lms-seq-less*:
  assumes *abs-is-lms xs i*
  and   *abs-is-lms xs j*
  and   *i* < *j*
shows *length* (*lms-seq xs j*) < *length* (*lms-seq xs i*)
proof −
  have {*k*. *abs-is-lms xs k* $\wedge$ *j* $\leq$ *k*} $\subseteq$ {*j*. *abs-is-lms xs j* $\wedge$ *i* $\leq$ *j*}
    using *assms*(*3*) **by** *force*
  moreover
  have *i* $\in$ {*j*. *abs-is-lms xs j* $\wedge$ *i* $\leq$ *j*}
    using *assms*(*1*) *less-or-eq-imp-le* **by** *blast*
  moreover
  have *i* $\notin$ {*k*. *abs-is-lms xs k* $\wedge$ *j* $\leq$ *k*}
    using *assms*(*3*) *linorder-not-less* **by** *blast*

ultimately have $\{k.\ \mathit{abs\text{-}is\text{-}lms}\ xs\ k \wedge j \leq k\} \subset \{j.\ \mathit{abs\text{-}is\text{-}lms}\ xs\ j \wedge i \leq j\}$
  by *blast*
hence $\mathit{card}\ \{k.\ \mathit{abs\text{-}is\text{-}lms}\ xs\ k \wedge j \leq k\} < \mathit{card}\ \{j.\ \mathit{abs\text{-}is\text{-}lms}\ xs\ j \wedge i \leq j\}$
  by (*simp add*: *lms-finite psubset-card-mono*)
then show *?thesis*
  by (*simp add*: *assms*(*1*) *assms*(*2*) *length-lms-seq*)
qed

lemma *lms-seq-nth-0*:
lms-seq xs (*Suc k*) ≠ [] ⟹ *lms-seq xs* (*Suc k*) ! *0* = *abs-find-next-lms xs k*
unfolding *lms-seq-def*
apply (*simp add*: *abs-find-next-lms-funpow-Suc abs-find-next-lms-def*)
apply (*drule filter-find*)
by (*clarsimp split*: *option.splits*)

lemma *lms-seq-eq-cons-lms*:
assumes *abs-is-lms xs i* $i < k$ $k \leq$ *abs-find-next-lms xs i*
shows *lms-seq xs i* = *i* # *lms-seq xs k*
proof −
have *filter* (*abs-is-lms xs*) [*Suc i*..<*k*] = []
  using *assms*(*1*) *assms*(*2*) *assms*(*3*) *filter-no-lms1* **by** *blast*
moreover
have [*i*..<*length xs*] = *i* # [*Suc i*..<*k*] @ [*k*..<*length xs*]
  by (*metis Suc-leI assms dual-order.trans abs-find-next-lms-le-length abs-is-lms-imp-less-length le-add-diff-inverse upt-add-eq-append upt-conv-Cons*)
hence *lms-seq xs i* = *i* # (*filter* (*abs-is-lms xs*) [*Suc i*..<*k*]) @ (*filter* (*abs-is-lms xs*) [*k*..<*length xs*])
  by (*simp add*: *assms*(*1*) *lms-seq-def*)
ultimately show *?thesis*
  by (*metis append-Nil lms-seq-def*)
qed

lemma *lms-seq-not-lms*:
assumes ¬*abs-is-lms xs i* $i < k$ $k \leq$ *abs-find-next-lms xs i*
shows *lms-seq xs i* = *lms-seq xs k*
proof −
have *filter* (*abs-is-lms xs*) [*i*..<*k*] = []
  using *assms filter-no-lms2* **by** *blast*
moreover
have [*i*..<*length xs*] = [*i*..<*k*] @ [*k*..<*length xs*]
  by (*metis assms*(*2*,*3*) *dual-order.trans abs-find-next-lms-le-length le-add-diff-inverse less-or-eq-imp-le upt-add-eq-append*)
ultimately show *?thesis*
  by (*simp add*: *lms-seq-def*)
qed

lemma *lms-seq-eq-cons*:
assumes *lms-seq xs* (*Suc i*) ≠ []
shows *lms-seq xs* (*Suc i*) = *abs-find-next-lms xs i* # *lms-seq xs* (*Suc* (*abs-find-next-lms*

xs i))
proof −
  from *lms-seq-nth-0*[*OF assms*]
  have *lms-seq xs* (*Suc i*) ! *0* = *abs-find-next-lms xs i* **.**
  moreover
  have *i* < *abs-find-next-lms xs i*
   by (*metis Suc-lessD assms filter.simps*(*1*) *abs-find-next-lms-lower-bound-1 lms-seq-def upt-rec*)
  hence *Suc i* ≤ *abs-find-next-lms xs i*
    by *simp*
  hence *lms-seq xs* (*Suc i*) = *lms-seq xs* (*abs-find-next-lms xs i*)
    by (*metis abs-find-next-lms-abs-is-lms abs-find-next-lms-le-Suc lms-seq-not-lms order-le-imp-less-or-eq*)
  ultimately show *?thesis*
    by (*metis Suc-leI assms filter.simps*(*1*) *abs-find-next-lms-less-length-abs-is-lms abs-find-next-lms-lower-bound-1 lessI lms-seq-def lms-seq-eq-cons-lms upt-rec*)
qed

lemma *lms-seq-nth-abs-is-lms*:
  i < *length* (*lms-seq xs k*) ⟹ *abs-is-lms xs* ((*lms-seq xs k*) ! *i*)
  unfolding *lms-seq-def*
  using *nth-mem* **by** *fastforce*

lemma *lms-seq-0*:
  lms-seq xs 0 = *lms-seq xs* (*Suc 0*)
  by (*metis filter-abs-is-lms-upt-0 lms-seq-def*)

lemma *lms-seq-nth*:
  i < *length* (*lms-seq xs* (*Suc k*)) ⟹ *lms-seq xs* (*Suc k*) ! *i* = ((*abs-find-next-lms xs*)^^(*Suc i*)) *k*
proof (*induct i arbitrary*: *k*)
  case *0*
  then show *?case*
    by (*simp add*: *lms-seq-nth-0*)
next
  case (*Suc i*)
  note *IH* = *this*
  let *?j* = *abs-find-next-lms xs k*
  have *lms-seq xs* (*Suc k*) = *?j* # *lms-seq xs* (*Suc ?j*)
    using *Suc.prems lms-seq-eq-cons* **by** *force*
  with *IH*(*1*)[*of ?j*]
  have *lms-seq xs* (*Suc ?j*) ! *i* = (*abs-find-next-lms xs* ^^ *Suc i*) *?j*
    using *Suc.prems* **by** *fastforce*
  then show *?case*
    by (*simp add*: ‹*lms-seq xs* (*Suc k*) = *?j* # *lms-seq xs* (*Suc ?j*)› *funpow-swap1*)
qed

lemma *inj-on-lms-seq*:

inj-on (*lms-seq xs*) {*i*. *abs-is-lms xs i*}
by (*metis* (*mono-tags*, *lifting*) *inj-onI list.inject lms-seq-abs-is-lms-hd mem-Collect-eq*)

lemma *list-app-imp-suffix*:
xs = *ys* @ *zs* $\Longrightarrow$ *suffix xs* (*length ys*) = *zs*
by *auto*

abbreviation *nth-lms-seq xs i* $\equiv$ *lms-seq xs* (*nth-lms xs i*)

abbreviation *lms0-seq xs* $\equiv$ *lms-seq xs* (*lms0 xs*)

lemma *lms-seq-0-zeroth-lms*:
lms-seq xs 0 = *lms0-seq xs*
by (*metis gr-zeroI abs-is-lms-0 le-refl lms-seq-not-lms*)

lemma *lms-seq-set*:
set (*lms-seq xs i*) = {*k*. *abs-is-lms xs k* $\wedge$ *i* $\leq$ *k*}
by (*intro equalityI subsetI*; *clarsimp simp add*: *abs-is-lms-def suffix-type-s-bound lms-seq-def*)

lemma *lms-seq-last-eq-length*:
length (*lms-seq xs i*) = *Suc n* $\Longrightarrow$
abs-find-next-lms xs ((*lms-seq xs i*) ! *n*) = *length xs*
proof −
let *?k* = (*lms-seq xs i*) ! *n*
assume *length* (*lms-seq xs i*) = *Suc n*
hence *i* $\leq$ *?k*
by (*metis* (*no-types*, *lifting*) *lessI lms-seq-set mem-Collect-eq nth-mem*)
have $\forall$ *j* < *length xs*. *?k* < *j* $\longrightarrow$ $\neg$*abs-is-lms xs j*
proof *safe*
fix *j*
assume *j* < *length xs ?k* < *j abs-is-lms xs j*
hence *j* $\in$ *set* (*lms-seq xs i*)
using ‹*i* $\leq$ *lms-seq xs i* ! *n*› *lms-seq-set* **by** *fastforce*
hence $\exists$ *j'*< *length* (*lms-seq xs i*). (*lms-seq xs i*) ! *j'* = *j*
by (*simp add*: *in-set-conv-nth*)
then obtain *j'* **where**
j' < *length* (*lms-seq xs i*)
(*lms-seq xs i*) ! *j'* = *j*
by *blast*
with *strict-sorted-nth-less-mono*[*OF lms-seq-strict-sorted*[*of xs i*], *of n j'*]
have *n* < *j'*
using ‹*length* (*lms-seq xs i*) = *Suc n*› ‹*lms-seq xs i* ! *n* < *j*› **by** *fastforce*
then show *False*
using ‹*j'* < *length* (*lms-seq xs i*)› ‹*length* (*lms-seq xs i*) = *Suc n*› **by** *linarith*
qed
then show *?thesis*
by (*meson abs-find-next-lms-le-length abs-find-next-lms-less-length-abs-is-lms abs-find-next-lms-strict-upper-imp-lower-bound order.strict-iff-order*)

qed

lemma *lms0-seq-has-all-lms*:
  set (*lms0-seq xs*) = {*i*. *abs-is-lms xs i*}
  by (*metis* (*mono-tags*, *lifting*) *Collect-cong linorder-le-less-linear lms-seq-set mem-Collect-eq no-lms-between-i-and-next set-lms-gr-0*)

lemma *lms0-seq-length*:
  length (*lms0-seq xs*) = *card* {*i*. *abs-is-lms xs i*}
  by (*metis distinct-card lms0-seq-has-all-lms lms-seq-distinct*)

lemma *lms0-seq-nth*:
  $i <$ *card* {*i*. *abs-is-lms xs i*} $\Longrightarrow$ *lms0-seq xs* ! i = *nth-lms xs i*
  by (*metis lms0-seq-length lms-seq-0 lms-seq-0-zeroth-lms lms-seq-nth*)

lemma *lms-seq-Suc1*:
  assumes *abs-is-lms xs i*
  shows *lms-seq xs i* = *i* # *lms-seq xs* (*Suc i*)
  by (*simp add*: *assms filter-abs-is-lms-upt-hd abs-is-lms-imp-less-length lms-seq-def*)

lemma *lms-seq-Suc2*:
  assumes ¬*abs-is-lms xs i*
  shows *lms-seq xs i* = *lms-seq xs* (*Suc i*)
  by (*metis* (*no-types*, *lifting*) *assms dual-order.strict-trans2 filter.simps*(*2*) *lessI less-or-eq-imp-le lms-seq-def upt-rec*)

lemma *lms-seq-suf*:
  $i \le j \Longrightarrow \exists$ *ys*. *lms-seq xs i* = *ys* @ *lms-seq xs j*
proof (*induct* $j - i$ *arbitrary*: *i j*)
  case *0*
  then show *?case*
    by *force*
next
  case (*Suc x*)
  note *IH* = *this*
  hence $\exists j'$. j = *Suc* j'
    by (*metis Suc-le-D diff-le-self*)
  then obtain j' **where**
    A: j = *Suc* j'
    by *blast*
  with *IH*(*1*)[*of* j' *i*] *IH*(*2*,*3*)
  have *B*: $\exists$ *ys*. *lms-seq xs i* = *ys* @ *lms-seq xs* j'
    by (*metis Suc-diff-le Suc-eq-plus1 add.commute add-left-imp-eq antisym-conv2
le-Suc-eq
        zero-less-Suc zero-less-diff*)
  show *?case*
  proof (*cases abs-is-lms xs* j')
    assume *abs-is-lms xs* j'
    with *A B lms-seq-Suc1*[*of xs* j']

```
    show ?thesis
      by auto
  next
    assume ¬abs-is-lms xs j'
    with A B lms-seq-Suc2[of xs j']
    show ?thesis
      by simp
  qed
qed

lemma lms-lms-seq-is-suffix:
  assumes abs-is-lms xs i
  shows ∃ k < length (lms0-seq xs).
          suffix (lms0-seq xs) k = lms-seq xs i
proof −
  have lms0 xs ≤ i
   by (metis assms bot-nat-0.not-eq-extremum abs-is-lms-0 linorder-not-less no-lms-between-i-and-next)
  with lms-seq-suf[of lms0 xs i xs]
  show ?thesis
   by (metis assms length-Cons length-append length-greater-0-conv less-add-same-cancel1
                  less-numeral-extra(1) list.size(3) list-app-imp-suffix lms-seq-Suc1
plus-1-eq-Suc
              zero-eq-add-iff-both-eq-0)
qed

lemma nth-lms:
  i < card {i. abs-is-lms xs i} ⟹
   abs-is-lms xs (nth-lms xs i)
proof −
  assume i < card {i. abs-is-lms xs i}
  hence i < length (lms0-seq xs)
    by (metis distinct-card lms0-seq-has-all-lms lms-seq-distinct)
  moreover
  have ∃ k. lms0 xs = Suc k
    by (metis calculation filter.simps(1) abs-find-next-lms-eq-Suc less-nat-zero-code
list.size(3)
              lms-seq-def upt.simps(1))
  then obtain k where lms0 xs = Suc k by blast
  ultimately show ?thesis
    by (metis lms-seq-0 lms-seq-0-zeroth-lms lms-seq-nth lms-seq-nth-abs-is-lms)
qed

lemma card-abs-find-next-lms-funpow:
  i < card {k. abs-is-lms xs k} ⟹
   card {k. abs-is-lms xs k ∧ k < nth-lms xs i} = i
proof (induct i)
  case 0
  then show ?case
    by (metis (mono-tags, lifting) Collect-empty-eq card-eq-0-iff comp-apply fun-
```

pow.simps(*2*)
funpow-0 gr-zeroI abs-is-lms-0 no-lms-between-i-and-next)
next
case (*Suc i*)
note *IH* = *this*
let *?i* = *nth-lms xs i*
let *?j* = *nth-lms xs* (*Suc i*)
let *?A* = {*k*. *abs-is-lms xs k* ∧ *k* < *?i*}
let *?B* = {*k*. *abs-is-lms xs k* ∧ *k* < *?j*}

from *IH*
have *A*: *card ?A* = *i*
using *Suc-lessD* **by** *blast*
moreover
have *P*: *?i* < *?j*
by (*metis Suc.prems Suc-lessD abs-find-next-lms-funpow-Suc abs-find-next-lms-lower-bound-1 abs-is-lms-imp-less-length nth-lms*)
with *no-lms-between-i-and-next-funpow*
have *B*: ∀ *j*. *?i* < *j* ∧ *j* < *?j* ⟶ *j* ∉ *?B*
by *blast*

have *C*: *?i* ∈ *?B*
using *P Suc.prems Suc-lessD nth-lms* **by** *fastforce*

have *D*: *?A* ⊆ *?B*
using *P* **by** *force*

have *?B* = *insert ?i ?A*
using *B nat-neq-iff C D*
by (*intro equalityI subsetI insert-iff*[*THEN iffD2*]; *auto*)
moreover
have *?i* ∉ *?A*
by *blast*
hence *card* (*insert ?i ?A*) = *Suc* (*card ?A*)
by *simp*
ultimately show *?case*
by *simp*
qed

lemma *lms-seq-nth-suffix*:
i < *card* {*i*. *abs-is-lms xs i*} ⟹
suffix (*lms0-seq xs*) *i* = *nth-lms-seq xs i*
proof −
let *?i* = *lms0 xs*
let *?j* = *nth-lms xs i*
assume *A*: *i* < *card* {*i*. *abs-is-lms xs i*}
from *nth-lms*[*OF A*]
have *abs-is-lms xs ?j* **.**
moreover

```
  have ?i ≤ ?j
    by (metis bot-nat-0.not-eq-extremum calculation abs-is-lms-0 linorder-not-less
           no-lms-between-i-and-next)
  hence [?i..<length xs] = [?i..<?j] @ [?j..<length xs]
   by (metis abs-find-next-lms-funpow-Suc abs-find-next-lms-le-length le-add-diff-inverse
           upt-add-eq-append)
  moreover
  have length (filter (abs-is-lms xs) [?i..<?j]) = card {k. abs-is-lms xs k ∧ k < ?j}
  proof −
    from filter-no-lms2[of xs 0 ?i]
    have filter (abs-is-lms xs) [0..<?i] = []
      using abs-is-lms-0 by fastforce
    moreover
    have [0..<?j] = [0..<?i] @ [?i..<?j]
     by (metis ‹?i ≤ ?j› bot-nat-0.not-eq-extremum le-add-diff-inverse less-or-eq-imp-le
             upt-add-eq-append)
    ultimately have filter (abs-is-lms xs) [0..<?j] = filter (abs-is-lms xs) [?i..<?j]
      by fastforce
    moreover
     have length (filter (abs-is-lms xs) [0..<?j]) = card {k. abs-is-lms xs k ∧ k <
?j}
    proof −
      from length-filter-conv-card[of abs-is-lms xs [0..<?j], simplified length-upt]
      have length (filter (abs-is-lms xs) [0..<?j]) = card {k. k < ?j ∧ abs-is-lms xs
([0..<?j] ! k)}
        by simp
      moreover
      have {k. k < ?j ∧ abs-is-lms xs ([0..<?j] ! k)} = {k. abs-is-lms xs k ∧ k <
?j}
        by force
      ultimately show ?thesis
        by simp
    qed
    ultimately show ?thesis
      by presburger
  qed
  ultimately show ?thesis
    by (metis (no-types, lifting) A Collect-cong card-abs-find-next-lms-funpow fil-
ter-append
                    list-app-imp-suffix lms-seq-def)
qed
```

53.2 LMS-Substring Sequence

```
definition lms-substr-seq :: 'a :: {linorder,order-bot} list ⇒ nat ⇒ 'a list list
  where
lms-substr-seq xs i = map (lms-slice xs) (lms-seq xs i)

lemma lms-substr-seq-length:
```

length (*lms-substr-seq xs i*) = *length* (*lms-seq xs i*)
by (*simp add*: *lms-substr-seq-def*)

lemma *inj-on-map-lms-slice-lms-seq*:
inj-on (*map* (*lms-slice xs*)) (*lms-seq xs* ‘ {*i*. *abs-is-lms xs i*})
proof (*intro inj-onI*)
fix *x y*
assume $x \in$ *lms-seq xs* ‘ {*i*. *abs-is-lms xs i*}
$y \in$ *lms-seq xs* ‘ {*i*. *abs-is-lms xs i*}
map (*lms-slice xs*) *x* = *map* (*lms-slice xs*) *y*
then obtain *i j* **where**
x = *lms-seq xs i y* = *lms-seq xs j*
map (*lms-slice xs*) (*lms-seq xs i*) = *map* (*lms-slice xs*) (*lms-seq xs j*)
abs-is-lms xs i abs-is-lms xs j
by *clarsimp*+
then have *lms-seq xs i* = *lms-seq xs j*
by (*metis length-lms-seq-less length-map nat-neq-iff*)
then show *x* = *y*
by (*simp add*: ‹*x* = *lms-seq xs i*› ‹*y* = *lms-seq xs j*›)
qed

lemma *inj-on-lms-substr-seq*:
inj-on (*lms-substr-seq xs*) {*i*. *abs-is-lms xs i*}
unfolding *lms-substr-seq-def*
using *comp-inj-on*[*OF inj-on-lms-seq inj-on-map-lms-slice-lms-seq*, *simplified o-def*]
by *blast*

lemma *lms-substr-seq-nth*:
$i <$ *length* (*lms-substr-seq xs* (*Suc k*)) $\Longrightarrow$
lms-substr-seq xs (*Suc k*) ! *i* = *lms-slice xs* ((*abs-find-next-lms xs* ^^ *Suc i*) *k*)
by (*simp add*: *lms-seq-nth lms-substr-seq-def*)

lemma *lms-substr-seq-nth-abs-is-lms*:
$i <$ *length* (*lms-substr-seq xs k*) $\Longrightarrow$
(*lms-substr-seq xs k*) ! $i \in$ *lms-substrs xs*
by (*simp add*: *lms-seq-nth-abs-is-lms lms-substr-seq-def*)

definition *suffix-to-id*
where
suffix-to-id xs ys = *length xs* − *length ys*

lemma *suffix-lengths-neq*:
⟦$i < j$; $j <$ *length xs*⟧ $\Longrightarrow$ *length* (*suffix xs i*) > *length* (*suffix xs j*)
by *simp*

lemma *inj-on-suffix-to-id*:
inj-on (*suffix-to-id xs*) (*suffix xs* ‘ {*i*. *abs-is-lms xs i*})
by (*intro inj-onI*; *clarsimp simp*: *suffix-to-id-def abs-is-lms-imp-less-length less-or-eq-imp-le*)

lemma *suffix-id-suffix*:
  $i < length\ xs \Longrightarrow suffix\text{-}to\text{-}id\ xs\ (suffix\ xs\ i) = i$
  by (*simp add*: *suffix-to-id-def*)

lemma *suffix-to-id-image*:
  suffix-to-id xs ‘ suffix xs ‘ $\{i.\ abs\text{-}is\text{-}lms\ xs\ i\} = \{i.\ abs\text{-}is\text{-}lms\ xs\ i\}$
proof *safe*
  fix *i*
  assume *abs-is-lms xs i*
  then show *abs-is-lms xs* (*suffix-to-id xs* (*suffix xs i*))
    by (*simp add*: *abs-is-lms-imp-less-length suffix-id-suffix*)
next
  fix *i*
  assume *abs-is-lms xs i*
  then show $i \in$ *suffix-to-id xs ‘ lms-suffixes xs*
    by (*simp add*: *image-iff abs-is-lms-imp-less-length suffix-id-suffix*)
qed

abbreviation *lms-substr-seq-id xs* $\equiv$ (*lms-substr-seq xs*) $\circ$ (*suffix-to-id xs*)

lemma *lms-subtrs-seq-id-suffix*:
  lms-substr-seq-id xs (*suffix xs i*) = *lms-substr-seq xs i*
  apply *simp*
  apply (*cases* $i < length\ xs$)
   apply (*simp add*: *suffix-id-suffix*)
  by (*simp add*: *lms-seq-def lms-substr-seq-def suffix-to-id-def*)

lemma *lms-substr-seq-id-nth-abs-is-lms*:
  $i <$ *length* (*lms-substr-seq-id xs* (*suffix xs k*)) $\Longrightarrow$
   (*lms-substr-seq-id xs* (*suffix xs k*)) ! $i \in$ *lms-substrs xs*
  by (*simp add*: *lms-seq-nth-abs-is-lms lms-substr-seq-def*)

lemma *inj-on-lms-substr-seq-o-suffix-to-id*:
  inj-on (*lms-substr-seq-id xs*) (*lms-suffixes xs*)
proof −
  have *lms-substr-seq xs* = *map* (*lms-slice xs*) $\circ$ *lms-seq xs*
    using *lms-substr-seq-def* **by** *fastforce*
  with *comp-inj-on*[*OF inj-on-lms-seq inj-on-map-lms-slice-lms-seq*, *of xs*]
  have *inj-on* (*lms-substr-seq xs*) $\{i.\ abs\text{-}is\text{-}lms\ xs\ i\}$
    by *simp*
  with *comp-inj-on*[*OF inj-on-suffix-to-id*[*of xs*], *simplified suffix-to-id-image*[*of xs*]]
  show *?thesis*
    by *blast*
qed

lemma *list-less-ns-lms-substr-seq-suffix*:
  assumes *abs-is-lms xs i*
  and *abs-is-lms xs j*
  and *nslexordp list-less-ns* (*lms-substr-seq xs i*) (*lms-substr-seq xs j*)

shows *list-less-ns* (*suffix xs i*) (*suffix xs j*)
proof −
  have $\exists\, i'.\ i = Suc\ i'$
    using *assms*(*1*) *abs-is-lms-def* **by** *blast*
  then obtain i' **where**
    $i = Suc\ i'$
    by *blast*
  hence *abs-find-next-lms xs* $i' = i$
    using *assms*(*1*) *abs-find-next-lms-abs-is-lms* **by** *blast*

  have *A*: $\bigwedge k.\ k <$ *length* (*lms-substr-seq xs i*) $\Longrightarrow$
      lms-substr-seq xs i ! k = *lms-slice xs* ((*abs-find-next-lms xs* ^^ k) i)
   by (*simp add*: ‹*abs-find-next-lms xs* $i' = i$› ‹$i = Suc\ i'$› *funpow-swap1 lms-substr-seq-nth*)

  have $\exists\, j'.\ j = Suc\ j'$
    using *assms*(*2*) *abs-is-lms-def* **by** *blast*
  then obtain j' **where**
    $j = Suc\ j'$
    by *blast*
  hence *abs-find-next-lms xs* $j' = j$
    using *assms*(*2*) *abs-find-next-lms-abs-is-lms* **by** *blast*

  have *B*: $\bigwedge k.\ k <$ *length* (*lms-substr-seq xs j*) $\Longrightarrow$
      lms-substr-seq xs j ! k = *lms-slice xs* ((*abs-find-next-lms xs* ^^ k) j)
   by (*simp add*: ‹*abs-find-next-lms xs* $j' = j$› ‹$j = Suc\ j'$› *funpow-swap1 lms-substr-seq-nth*)

  let *?c1* = $\exists$ *b c as bs cs*.
      lms-substr-seq xs i = *as* @ *b* # *bs* $\wedge$ *lms-substr-seq xs j* = *as* @ *c* # *cs*
$\wedge$
      list-less-ns b c
  let *?c2* = $\exists$ *c cs*. *lms-substr-seq xs i* = *lms-substr-seq xs j* @ *c* # *cs*
  from *nslexordpE*[*OF assms*(*3*)]
  have *?c1* $\vee$ *?c2* **.**
  moreover
  have *?c1* $\Longrightarrow$ *?thesis*
  proof −
    assume *?c1*
    then obtain *b c as bs cs* **where**
      lms-substr-seq xs i = *as* @ *b* # *bs*
      lms-substr-seq xs j = *as* @ *c* # *cs*
      list-less-ns b c
      by *blast*

    let *?b* = *lms-slice xs* ((*abs-find-next-lms xs* ^^ (*length as*)) *i*)
    from *lms-substr-seq-nth*[*of length as xs* i'] ‹$i = Suc\ i'$›
      ‹*lms-substr-seq xs i* = *as* @ *b* # *bs*›
    have *b* = *lms-slice xs* ((*abs-find-next-lms xs* ^^ *Suc* (*length as*)) i')
      by *simp*
    with ‹*abs-find-next-lms xs* $i' = i$›

```
have b = ?b
  by (simp add: funpow-swap1)

let ?c = lms-slice xs ((abs-find-next-lms xs ^^ (length as)) j)
from lms-substr-seq-nth[of length as xs j'] ‹j = Suc j'›
       ‹lms-substr-seq xs j = as @ c # cs›
have c = lms-slice xs ((abs-find-next-lms xs ^^ Suc (length as)) j')
  by simp
with ‹abs-find-next-lms xs j' = j›
have c = lms-slice xs ((abs-find-next-lms xs ^^ (length as)) j)
  by (simp add: funpow-swap1)

have P: ∀k<length as. lms-slice xs ((abs-find-next-lms xs ^^ k) i) =
                     lms-slice xs ((abs-find-next-lms xs ^^ k) j)
proof (safe)
  fix k
  assume k < length as
  with ‹lms-substr-seq xs i = as @ b # bs› A[of k]
  have as ! k = lms-slice xs ((abs-find-next-lms xs ^^ k) i)
    by (simp add: nth-append)
  moreover
  from ‹lms-substr-seq xs j = as @ c # cs› ‹k < length as› B[of k]
  have as ! k = lms-slice xs ((abs-find-next-lms xs ^^ k) j)
    by (simp add: nth-append)
  ultimately show
    lms-slice xs ((abs-find-next-lms xs ^^ k) i) =
     lms-slice xs ((abs-find-next-lms xs ^^ k) j)
    by simp
qed

have Q: list-less-ns (suffix xs ((abs-find-next-lms xs ^^ (length as)) i))
                    (suffix xs ((abs-find-next-lms xs ^^ (length as)) j))
 by (metis ‹b = ?b› ‹c = ?c› ‹list-less-ns b c› drop-eq-Nil abs-find-next-lms-lower-bound-2
           less-lms-slice-imp-suffix list-less-ns-nil lms-slice-eq-suffix
           not-le-imp-less ordlistns.less-asym)

show ?thesis
proof (cases length as)
  case 0
  then show ?thesis
    using Q by force
next
  case (Suc n)
  assume length as = Suc n
  moreover
  from lms-slice-eq-suffix-less-funpow[OF P]
  have list-less-ns (suffix xs i) (suffix xs j) =
       list-less-ns (suffix xs ((abs-find-next-lms xs ^^ (length as)) i))
                   (suffix xs ((abs-find-next-lms xs ^^ (length as)) j))
```

using *lessI* **by** *presburger*
ultimately show *?thesis*
using *Q* **by** *blast*
qed
qed
moreover
have *?c2* $\Longrightarrow$ *?thesis*
proof −
assume *?c2*
then obtain *c cs* **where**
lms-substr-seq xs i = *lms-substr-seq xs j* @ *c* # *cs*
by *blast*

have *lms-substr-seq xs j* $\neq$ []
by (*metis assms*(*2*) *list.distinct*(*1*) *list.map-disc-iff lms-seq-abs-is-lms-hd lms-substr-seq-def*)
hence $\exists$ *n*. *length* (*lms-substr-seq xs j*) = *Suc n*
using *not0-implies-Suc* **by** *auto*
then obtain *n* **where**
length (*lms-substr-seq xs j*) = *Suc n*
by *blast*

have *P*: $\forall$ *k* < *Suc n*. *lms-slice xs* ((*abs-find-next-lms xs* ^^ *k*) *i*) =
lms-slice xs ((*abs-find-next-lms xs* ^^ *k*) *j*)
proof *safe*
fix *k*
assume *k* < *Suc n*
hence *lms-substr-seq xs j* ! *k* = *lms-slice xs* ((*abs-find-next-lms xs* ^^ *k*) *j*)
by (*simp add*: *B* ‹*length* (*lms-substr-seq xs j*) = *Suc n*›)
moreover
from ‹*lms-substr-seq xs i* = *lms-substr-seq xs j* @ *c* # *cs*› ‹*k* < *Suc n*›
have *lms-substr-seq xs i* ! *k* = *lms-slice xs* ((*abs-find-next-lms xs* ^^ *k*) *j*)
by (*simp add*: *B* ‹*length* (*lms-substr-seq xs j*) = *Suc n*› *nth-append*)
moreover
have *lms-substr-seq xs i* ! *k* = *lms-slice xs* ((*abs-find-next-lms xs* ^^ *k*) *i*)
using *A* ‹*k* < *Suc n*› ‹*length* (*lms-substr-seq xs j*) = *Suc n*›
‹*lms-substr-seq xs i* = *lms-substr-seq xs j* @ *c* # *cs*› **by** *auto*
ultimately show
lms-slice xs ((*abs-find-next-lms xs* ^^ *k*) *i*) =
lms-slice xs ((*abs-find-next-lms xs* ^^ *k*) *j*)
by *simp*
qed

have *list-less-ns* (*suffix xs i*) (*suffix xs j*) =
list-less-ns (*suffix xs* ((*abs-find-next-lms xs* ^^ *Suc n*) *i*))
(*suffix xs* ((*abs-find-next-lms xs* ^^ *Suc n*) *j*))
using *lms-slice-eq-suffix-less-funpow*[*OF P*]
by *blast*
moreover

from *lms-seq-last-eq-length*[*of xs j n*]
have (*abs-find-next-lms xs* ^^ *Suc n*) *j* = *length xs*
by (*metis* ‹*abs-find-next-lms xs j'* = *j*› ‹*j* = *Suc j'*› ‹*length* (*lms-substr-seq xs j*) = *Suc n*›
funpow-swap1 length-map lessI lms-seq-nth lms-substr-seq-def)
hence *suffix xs* ((*abs-find-next-lms xs* ^^ *Suc n*) *j*) = []
by *force*
ultimately show *?thesis*
by (*metis P* ‹*lms-substr-seq xs i* = *lms-substr-seq xs j* @ *c* # *cs*› *append-self-conv assms*(*1*,*2*)
abs-is-lms-imp-less-length list.distinct(*1*) *list-less-ns-nil suffix-id-suffix*
lms-slice-eq-suffix-less-funpow ordlistns.not-less-iff-gr-or-eq)
qed
ultimately show *?thesis*
by *blast*
qed

lemma *monotone-on-lms-substr-seq-id*:
monotone-on (*lms-suffixes xs*) *list-less-ns* (*nslexordp list-less-ns*) (*lms-substr-seq-id xs*)
(**is** *monotone-on ?A ?orda ?ordb ?f*)
proof −
let *?B* = *?f* ‘ *?A*

from *inj-on-imp-bij-betw*[*OF inj-on-lms-substr-seq-o-suffix-to-id*]
have *A*: *bij-betw ?f ?A ?B* **.**
with *bij-betw-inv-alt*
have ∃ *g*. *bij-betw g ?B ?A* ∧ *inverses-on ?f g ?A ?B*
by *blast*
then obtain *g* **where** *B*:
bij-betw g ?B ?A
inverses-on ?f g ?A ?B
by *blast*

have *C*: *monotone-on ?B ?ordb ?orda g*
proof (*intro monotone-onI*)
fix *x y*
assume *x* ∈ *?B y* ∈ *?B nslexordp list-less-ns x y*
moreover
have ∃ *i*. *abs-is-lms xs i* ∧ *g x* = *suffix xs i*
using ‹*x* ∈ *?B*› *bij-betw-apply B*(*1*) **by** *fastforce*
then obtain *i* **where**
abs-is-lms xs i g x = *suffix xs i*
by *blast*
moreover
have ∃ *j*. *abs-is-lms xs j* ∧ *g y* = *suffix xs j*
using ‹*y* ∈ *?B*› *bij-betw-apply B*(*1*) **by** *fastforce*
then obtain *j* **where**
abs-is-lms xs j g y = *suffix xs j*

by *blast*
ultimately show *list-less-ns* (*g x*) (*g y*)
using *list-less-ns-lms-substr-seq-suffix*[*of xs i j*]
by (*metis* (*mono-tags*, *lifting*) *B*(*2*) *comp-def inverses-onD2 abs-is-lms-imp-less-length suffix-id-suffix*)
qed

from *nslexordp-asymp*[*of list-less-ns*]
have *asymp-on ?B ?ordb*
using *asympD* **by** *fastforce*

from *totalp-on-subset*[*OF nslexordp-totalp*[*of list-less-ns*]]
have *totalp-on ?B ?ordb*
using *ordlistns.totalp-on-less* **by** *blast*

note *D* = ‹*asymp-on ?B ?ordb*› ‹*totalp-on ?B ?ordb*›

from *monotone-on-bij-betw-inv*[*OF C D - - B*(*1*) *A inverses-on-sym*[*THEN iffD1*, *OF B*(*2*)],
simplified]
show *?thesis* **.**
qed

lemma *list-less-ns-suffix-lms-substr-seq*:
assumes *abs-is-lms xs i*
and *abs-is-lms xs j*
and *list-less-ns* (*suffix xs i*) (*suffix xs j*)
shows *nslexordp list-less-ns* (*lms-substr-seq xs i*) (*lms-substr-seq xs j*)
using *monotone-onD*[*OF monotone-on-lms-substr-seq-id - - assms*(*3*)]
assms(*1*,*2*) *abs-is-lms-imp-less-length suffix-id-suffix* **by** *fastforce*

lemma *lms-substr-seq-suf*:
$i \le j \Longrightarrow \exists\, ys.$ *lms-substr-seq xs i* = *ys* @ *lms-substr-seq xs j*
unfolding *lms-substr-seq-def*
by (*frule lms-seq-suf*[*of - - xs*]; *clarsimp*)

lemma *lms-lms-substr-seq-is-suffix*:
assumes *abs-is-lms xs i*
shows $\exists\, k <$ *length* (*lms-substr-seq xs* (*abs-find-next-lms xs 0*)).
suffix (*lms-substr-seq xs* (*abs-find-next-lms xs 0*)) *k* = *lms-substr-seq xs i*
unfolding *lms-substr-seq-def*
by (*metis assms length-map lms-lms-seq-is-suffix*[*of xs i*] *suffix-map*)

lemma *lms-substr-seq-nth-suffix*:
$i <$ *card* {*i*. *abs-is-lms xs i*} $\Longrightarrow$
suffix (*lms-substr-seq xs* (*abs-find-next-lms xs 0*)) *i* =
lms-substr-seq xs ((*abs-find-next-lms xs* ^^ *Suc i*) *0*)
by (*simp add*: *lms-seq-nth-suffix lms-substr-seq-def suffix-map*)

53.3 LMS Map

lemma *finite-lms-substrs*:
  finite (*lms-substrs xs*)
  by (*simp add*: *lms-finite*)

definition *lms-map* :: ($'a$ :: {*linorder*, *order-bot*}) *list* ⇒ $'a$ *list* ⇒ *nat list*
  where
lms-map xs ≡ (*map* (*ordlistns.elem-rank* (*lms-substrs xs*))) ∘ (*lms-substr-seq-id xs*)

lemma *lms-substr-seq-o-suffix-to-id-range*:
  (*lms-substr-seq xs* ∘ *suffix-to-id xs*) ‘ *lms-suffixes xs* ⊆ {*ys*. *set ys* ⊆ *lms-substrs xs*}
  unfolding *lms-substr-seq-def suffix-to-id-def*
  by (*safe*; *clarsimp simp*: *lms-seq-set*)

lemma *lms-map-o-def*:
  lms-map xs ys = *map* (*ordlistns.elem-rank* (*lms-substrs xs*)) (*lms-substr-seq-id xs ys*)
  by (*simp add*: *lms-map-def*)

lemma *inj-on-lms-map*:
  inj-on (*lms-map xs*) (*lms-suffixes xs*)
proof −
  note *A* = *comp-inj-on*[*OF inj-on-lms-substr-seq-o-suffix-to-id*]
  note *B* = *inj-on-subset*[*OF bij-betw-imp-inj-on*[*OF bij-betw-map*[*OF ordlistns.bij-betw-elem-rank-upt*]]]
  from *A*[*OF B*, *OF finite-lms-substrs lms-substr-seq-o-suffix-to-id-range*, *of xs*]
  show *?thesis*
    by (*simp add*: *lms-map-def*)
qed

lemma *lms-map-length*:
  length (*lms-map xs ys*) = *length* (*lms-substr-seq xs* (*suffix-to-id xs ys*))
  by (*simp add*: *lms-map-def*)

lemma *lms-map-nth-suffix*:
  i < *card* {*i*. *abs-is-lms xs i*} ⟹
   suffix (*lms-map xs* (*suffix xs* (*abs-find-next-lms xs 0*))) *i* =
   lms-map xs (*suffix xs* ((*abs-find-next-lms xs* ^^ *Suc i*) *0*))
  by (*simp add*: *abs-find-next-lms-le-length lms-map-def lms-seq-nth-suffix lms-substr-seq-def suffix-map*
        suffix-to-id-def)

lemma *lms-lms-map-is-suffix*:
  assumes *abs-is-lms xs i*
  shows ∃*k* < *length* (*lms-map xs* (*suffix xs* (*abs-find-next-lms xs 0*))).
        suffix (*lms-map xs* (*suffix xs* (*abs-find-next-lms xs 0*))) *k* = *lms-map xs* (*suffix xs i*)

proof −

have *suffix-to-id xs* (*suffix xs i*) = *i*
by (*simp add*: *assms abs-is-lms-imp-less-length suffix-id-suffix*)
moreover
have *suffix-to-id xs* (*suffix xs* (*abs-find-next-lms xs 0*)) = *abs-find-next-lms xs 0*
by (*simp add*: *abs-find-next-lms-le-length suffix-to-id-def*)
moreover
from *lms-lms-substr-seq-is-suffix*[*OF assms*]
obtain *k* **where**
k < *length* (*lms-substr-seq xs* (*abs-find-next-lms xs 0*))
suffix (*lms-substr-seq xs* (*abs-find-next-lms xs 0*)) *k* = *lms-substr-seq xs i*
by *blast*
moreover
have *k* < *length* (*lms-map xs* (*suffix xs* (*abs-find-next-lms xs 0*)))
by (*simp add*: *calculation*(*2*) *calculation*(*3*) *lms-map-length*)
moreover
have *suffix* (*lms-map xs* (*suffix xs* (*abs-find-next-lms xs 0*))) *k* = *lms-map xs* (*suffix xs i*)
by (*simp add*: *lms-map-def calculation suffix-map*)
ultimately show *?thesis*
by *blast*
qed

lemma *length-reduced-seq*:
length (*lms-map xs* (*suffix xs* (*abs-find-next-lms xs 0*))) = *card* (*lms-suffixes xs*)
apply (*simp add*: *lms-map-length lms-substr-seq-length*)
apply (*cases abs-find-next-lms xs 0* < *length xs*)
apply (*simp add*: *suffix-id-suffix*)
apply (*subst distinct-card*[*OF lms-seq-distinct*[*of xs abs-find-next-lms xs 0*], *symmetric*])
apply (*simp add*: *lms0-seq-has-all-lms*)
apply (*metis card-image inj-on-suffix-to-id suffix-to-id-image*)
by (*metis card-eq-0-iff diff-diff-cancel empty-set filter.simps*(*1*) *abs-find-next-lms-le-length lms0-seq-has-all-lms image-is-empty length-drop linorder-le-less-linear list.size*(*3*) *lms-seq-def suffix-to-id-def upt-eq-Nil-conv*)

corollary *lms-lms-map-in-suffixes*:
abs-is-lms xs i $\Longrightarrow$
lms-map xs (*suffix xs i*) $\in$
suffix (*lms-map xs* (*suffix xs* (*abs-find-next-lms xs 0*))) ‘ {*0*..<*card* (*lms-suffixes xs*)}
by (*metis atLeastLessThan-iff imageI length-reduced-seq lms-lms-map-is-suffix zero-le*)

lemma *card-lms-suffixes*:
card (*lms-suffixes xs*) = *card* {*i*. *abs-is-lms xs i*}
by (*metis card-image inj-on-suffix-to-id suffix-to-id-image*)

lemma *lms-map-image*:

lms-map xs ‘ lms-suffixes xs =
suffix (*lms-map xs* (*suffix xs* (*abs-find-next-lms xs 0*))) ‘ {*0..<card* (*lms-suffixes xs*)}
proof (*safe*)
fix *i*
assume *abs-is-lms xs i*
then show *lms-map xs* (*suffix xs i*) ∈
suffix (*lms-map xs* (*lms0-suffix xs*)) ‘ {*0..<card* (*lms-suffixes xs*)}
using *lms-lms-map-in-suffixes* **by** *blast*
next
fix *i*
assume *i* ∈ {*0..<card* (*lms-suffixes xs*)}
with *card-lms-suffixes*
have *i* < *card* {*i. abs-is-lms xs i*}
by (*metis atLeastLessThan-iff*)
with *lms-map-nth-suffix*[*of i xs*]
have *suffix* (*lms-map xs* (*lms0-suffix xs*)) *i* = *lms-map xs* (*suffix xs* (*nth-lms xs i*))
by *blast*
moreover
have *suffix xs* (*nth-lms xs i*) ∈ *lms-suffixes xs*
using ‹*i* < *card* {*i. abs-is-lms xs i*}› *nth-lms* **by** *fastforce*
ultimately show *suffix* (*lms-map xs* (*lms0-suffix xs*)) *i* ∈ *lms-map xs ‘ lms-suffixes xs*
by *blast*
qed

lemma *monotone-on-lms-map*:
monotone-on (*lms-suffixes xs*) *list-less-ns list-less-ns* (*lms-map xs*)
proof (*intro monotone-onI*)
fix *x y*
assume *x* ∈ *lms-suffixes xs y* ∈ *lms-suffixes xs list-less-ns x y*
with *monotone-onD*[*OF monotone-on-lms-substr-seq-id*, *of x xs y*]
have *nslexordp list-less-ns* (*lms-substr-seq-id xs x*) (*lms-substr-seq-id xs y*)
by *blast*
moreover
have ⋀*x. lms-map xs x* = *map* (*ordlistns.elem-rank* (*lms-substrs xs*)) (*lms-substr-seq-id xs x*)
by (*simp add*: *lms-map-def*)
moreover
{
have *set* (*lms-substr-seq-id xs x*) ⊆ *lms-substrs xs*
by (*simp add*: *Collect-mono-iff image-mono lms-seq-set lms-substr-seq-def*)
moreover
have *set* (*lms-substr-seq-id xs y*) ⊆ *lms-substrs xs*
by (*simp add*: *Collect-mono-iff image-mono lms-seq-set lms-substr-seq-def*)
ultimately have
nslexordp list-less-ns (*lms-substr-seq-id xs x*) (*lms-substr-seq-id xs y*) =
nslexordp (<) (*map* (*ordlistns.elem-rank* (*lms-substrs xs*)) (*lms-substr-seq-id*

xs x))
(*map* (*ordlistns.elem-rank* (*lms-substrs xs*)) (*lms-substr-seq-id xs y*))
using *monotone-on-iff-nslexordp*[*OF ordlistns.strict-mono-on-elem-rank*, *simplified*,
OF finite-lms-substrs[*of xs*] *ordlistns.bij-betw-elem-rank-upt*,
OF finite-lms-substrs[*of xs*]]
by *blast*
}
ultimately show *list-less-ns* (*lms-map xs x*) (*lms-map xs y*)
by (*simp add*: *nslexordp-eq-list-less-ns*)
qed

lemma *list-less-ns-lms-map-suffix*:
assumes *abs-is-lms xs i*
and *abs-is-lms xs j*
and *list-less-ns* (*lms-map xs* (*suffix xs i*)) (*lms-map xs* (*suffix xs j*))
shows *list-less-ns* (*suffix xs i*) (*suffix xs j*)
using *monotone-on-iff*[*OF monotone-on-lms-map*, *simplified*] *assms* **by** *blast*

abbreviation
lms0-map xs ≡
lms-map xs (*lms0-suffix xs*)

lemma *sorted-reduced-seq-imp-lms*:
assumes *ordlistns.strict-sorted* (*map* (*suffix* (*lms0-map xs*)) *ys*)
and $\forall y \in$ *set ys*. $y <$ *card* {*i*. *abs-is-lms xs i*}
shows *ordlistns.strict-sorted* (*map* (*suffix xs*) (*map* ((!) (*lms0-seq xs*)) *ys*))
proof (*intro sorted-wrt-mapI*)
fix *i j*
assume $i < j$ $j <$ *length* (*map* ((!) (*lms0-seq xs*)) *ys*)
hence *A*: $i < j$ $j <$ *length ys*
by *simp-all*
with *sorted-wrt-mapD*[*OF assms*(*1*)]
have *list-less-ns* (*suffix* (*lms0-map xs*) (*ys* ! *i*)) (*suffix* (*lms0-map xs*) (*ys* ! *j*)) **.**
moreover
from *lms-map-nth-suffix*[*of ys* ! *i xs*]
have *suffix* (*lms0-map xs*) (*ys* ! *i*) = *lms-map xs* (*suffix xs* (*nth-lms xs* (*ys* ! *i*)))
using *A*(*1*) *A*(*2*) *assms*(*2*) **by** *fastforce*
moreover
from *lms-map-nth-suffix*[*of ys* ! *j xs*]
have *suffix* (*lms0-map xs*) (*ys* ! *j*) = *lms-map xs* (*suffix xs* (*nth-lms xs* (*ys* ! *j*)))
using *A*(*2*) *assms*(*2*) **by** *fastforce*
moreover
have *abs-is-lms xs* (*nth-lms xs* (*ys* ! *i*))
by (*meson A*(*1*) *A*(*2*) *assms*(*2*) *nth-lms nth-mem order.strict-trans*)
hence *suffix xs* (*nth-lms xs* (*ys* ! *i*)) ∈ *lms-suffixes xs*
by *blast*
moreover

have *abs-is-lms xs* (*nth-lms xs* (*ys* ! *j*))
by (*meson A*(*2*) *assms*(*2*) *nth-lms nth-mem order.strict-trans*)
hence *suffix xs* (*nth-lms xs* (*ys* ! *j*)) ∈ *lms-suffixes xs*
by *blast*
ultimately have
list-less-ns (*suffix xs* (*nth-lms xs* (*ys* ! *i*))) (*suffix xs* (*nth-lms xs* (*ys* ! *j*)))
using *monotone-on-iff*[*OF monotone-on-lms-map*, *simplified*] **by** *auto*
moreover
from *lms0-seq-nth*[*of ys* ! *i xs*]
have *lms0-seq xs* ! (*ys* ! *i*) = *nth-lms xs* (*ys* ! *i*)
using *A*(*1*) *A*(*2*) *assms*(*2*) **by** *force*
moreover
from *lms0-seq-nth*[*of ys* ! *j xs*]
have *lms0-seq xs* ! (*ys* ! *j*) = *nth-lms xs* (*ys* ! *j*)
using *A*(*2*) *assms*(*2*) **by** *force*
ultimately have
list-less-ns (*suffix xs* (*lms0-seq xs* ! (*ys* ! *i*))) (*suffix xs* (*lms0-seq xs* ! (*ys* ! *j*)))
by *presburger*
then show *list-less-ns* (*suffix xs* (*map* ((!) (*lms0-seq xs*)) *ys* ! *i*))
(*suffix xs* (*map* ((!) (*lms0-seq xs*)) *ys* ! *j*))
using *A*(*1*) *A*(*2*) **by** *fastforce*
qed

lemma *sorted-distinct-lms-substr*:
assumes *ordlistns.sorted* (*map* (*lms-slice xs*) *ys*)
and *distinct* (*map* (*lms-slice xs*) *ys*)
and ∀ *y* ∈ *set ys*. *y* < *length xs*
shows *ordlistns.sorted* (*map* (*suffix xs*) *ys*)
proof (*intro sorted-wrt-mapI*)
fix *i j*
assume *i* < *j j* < *length ys*
with *sorted-wrt-mapD*[*OF assms*(*1*)]
have *list-less-eq-ns* (*lms-slice xs* (*ys* ! *i*)) (*lms-slice xs* (*ys* ! *j*)) **.**
moreover
have *lms-slice xs* (*ys* ! *i*) ≠ *lms-slice xs* (*ys* ! *j*)
using ‹*i* < *j*› ‹*j* < *length ys*› *assms*(*2*) *nth-eq-iff-index-eq* **by** *fastforce*
ultimately have *list-less-ns* (*lms-slice xs* (*ys* ! *i*)) (*lms-slice xs* (*ys* ! *j*))
using *ordlistns.nless-le* **by** *blast*
then show *list-less-eq-ns* (*suffix xs* (*ys* ! *i*)) (*suffix xs* (*ys* ! *j*))
by (*metis* ‹*i* < *j*› ‹*j* < *length ys*› *less-lms-slice-imp-suffix assms*(*3*) *dual-order.strict-trans*
nth-mem ordlistns.dual-order.strict-implies-order)
qed

lemma *distinct-lms0-map*:
assumes *distinct* (*lms0-map xs*)
shows *distinct* (*map* (*lms-slice xs*) (*lms0-seq xs*))
proof (*intro distinct-conv-nth*[*THEN iffD2*] *allI impI*)
fix *i j*
assume *i* < *length* (*map* (*lms-slice xs*) (*lms0-seq xs*))

j < *length* (*map* (*lms-slice xs*) (*lms0-seq xs*))
i ≠ *j*
hence *A*: *i* < *length* (*lms0-seq xs*) *j* < *length* (*lms0-seq xs*) *i* ≠ *j*
by *simp-all*
with *distinct-conv-nth*[*THEN iffD1*, *OF assms*]
have *B*: *lms0-map xs* ! *i* ≠ *lms0-map xs* ! *j*
by (*metis card-lms-suffixes length-reduced-seq lms0-seq-length*)
moreover
have *lms-substr-seq-id xs* (*lms0-suffix xs*) = *map* (*lms-slice xs*) (*lms0-seq xs*)
by (*metis lms-substr-seq-def lms-subtrs-seq-id-suffix*)
hence *lms0-map xs* = *map* (*ordlistns.elem-rank* (*lms-substrs xs*)) (*map* (*lms-slice xs*) (*lms0-seq xs*))
by (*simp add*: *lms-map-o-def*)
with *A*
have *lms0-map xs* ! *i* = *ordlistns.elem-rank* (*lms-substrs xs*) (*lms-slice xs* (*lms0-seq xs* ! *i*))
lms0-map xs ! *j* = *ordlistns.elem-rank* (*lms-substrs xs*) (*lms-slice xs* (*lms0-seq xs* ! *j*))
by *auto*
ultimately have *lms-slice xs* (*lms0-seq xs* ! *i*) ≠ *lms-slice xs* (*lms0-seq xs* ! *j*)
by *fastforce*
then show *map* (*lms-slice xs*) (*lms0-seq xs*) ! *i* ≠ *map* (*lms-slice xs*) (*lms0-seq xs*) ! *j*
by (*simp add*: *A*(*1*) *A*(*2*))
qed

lemma *sorted-distinct-lms-substr-perm*:
assumes *ordlistns.sorted* (*map* (*lms-slice xs*) *ys*)
and *distinct* (*lms0-map xs*)
and *ys* <~~> *lms0-seq xs*
shows *ordlistns.sorted* (*map* (*suffix xs*) *ys*)
by (*metis sorted-distinct-lms-substr*[*OF assms*(*1*)] *distinct-lms0-map*[*OF assms*(*2*)] *assms*(*3*)
distinct-map abs-is-lms-imp-less-length lms0-seq-has-all-lms mem-Collect-eq perm-distinct-iff perm-set-eq)

lemma *list-less-ns-suffix-lms-map*:
assumes *abs-is-lms xs i*
and *abs-is-lms xs j*
and *list-less-ns* (*suffix xs i*) (*suffix xs j*)
shows *list-less-ns* (*lms-map xs* (*suffix xs i*)) (*lms-map xs* (*suffix xs j*))
using *monotone-on-iff*[*OF monotone-on-lms-map*, *simplified*] *assms* **by** *blast*

lemma *valid-list-lms-map*:
assumes *valid-list* (*a* # *b* # *xs*)
and *abs-is-lms* (*a* # *b* # *xs*) *i*
shows *valid-list* (*lms-map* (*a* # *b* # *xs*) (*suffix* (*a* # *b* # *xs*) *i*))
proof −
let *?xs* = *a* # *b* # *xs*

have $\exists\, n.\ length\ ?xs = Suc\ n$
  by *simp*
then obtain n **where**
  $length\ ?xs = Suc\ n$
  by *blast*
hence *abs-is-lms ?xs n*
  using *assms*(*1*) *abs-is-lms-last* **by** *fastforce*

have *lms-slice ?xs n* = [*bot*]
  using ‹*length ?xs = Suc n*› *assms*(*1*) *lms-slice-last* **by** *blast*

have $\exists\, m.\ i = Suc\ m$
  using *assms*(*2*) *lms-type-list-less-ns* **by** *auto*
then obtain m **where**
  $i = Suc\ m$
  by *blast*

have *P*: $\forall\, x \in \{k.\ abs\text{-}is\text{-}lms\ ?xs\ k\}.\ x \neq n \longrightarrow$
    list-less-ns (*lms-slice ?xs n*) (*lms-slice ?xs x*)
proof *safe*
  fix x
  assume *abs-is-lms ?xs x* $x \neq n$
  hence $\exists\, ys.$ *lms-slice ?xs x* = *?xs* ! *x* # *ys*
    using *abs-is-lms-imp-less-length lms-slice-hd* **by** *blast*
  then obtain *ys* **where**
    lms-slice ?xs x = *?xs* ! *x* # *ys*
    by *blast*
  moreover
  have *bot* < *?xs* ! *x*
  by (*metis* ‹*abs-is-lms ?xs x*› ‹*length ?xs = Suc n*› ‹$x \neq n$› *assms*(*1*) *bot.not-eq-extremum*
    diff-Suc-1 hd-drop-conv-nth abs-is-lms-imp-less-length last-suffix-index)
  ultimately show *list-less-ns* (*lms-slice ?xs n*) (*lms-slice ?xs x*)
    by (*simp add*: ‹*lms-slice ?xs n* = [*bot*]› *list-less-ns-cons-diff*)
qed

have *Q*: *ordlistns.elem-rank* (*lms-substrs ?xs*) (*lms-slice ?xs n*) = *0*
  unfolding *ordlistns.elem-rank-def elm-rank-def*
proof −
  have {$y \in$ *lms-substrs ?xs. list-less-ns y* (*lms-slice ?xs n*)} = {}
    using *P* **by** *auto*
  then show *card* {$y \in$ *lms-substrs ?xs. list-less-ns y* (*lms-slice ?xs n*)} = *0*
    by (*metis card.empty*)
qed

have *R*: $\forall\, x \in$ *lms-substrs ?xs. list-less-ns* (*lms-slice ?xs n*) $x \longrightarrow$
    ordlistns.elem-rank (*lms-substrs ?xs*) $x > 0$
  using ‹*abs-is-lms ?xs n*› *finite-lms-substrs ordlistns.strict-mono-onD*
    ordlistns.strict-mono-on-elem-rank **by** *fastforce*

have [*i*..<*length ?xs*] = [*i*..<*n*] @ [*n*..<*Suc n*]
  using ‹*length ?xs* = *Suc n*› *assms*(*2*) *abs-is-lms-imp-less-length* **by** *fastforce*
hence *last* (*lms-seq ?xs i*) = *n*
  unfolding *lms-seq-def*
  by (*simp add*: ‹*abs-is-lms ?xs n*›)
hence *last* (*lms-substr-seq ?xs i*) = *lms-slice ?xs n*
  by (*metis assms*(*2*) *last-map list.discI lms-seq-Suc1 lms-substr-seq-def*)
hence *last* (*lms-substr-seq-id ?xs* (*suffix ?xs i*)) = *lms-slice ?xs n*
  by (*metis lms-subtrs-seq-id-suffix*)
hence *last* (*lms-map ?xs* (*suffix ?xs i*)) = *0*
  unfolding *lms-map-o-def*
  by (*metis assms*(*2*) *Q last-map length-0-conv list.discI lms-seq-Suc1 lms-substr-seq-length lms-subtrs-seq-id-suffix*)
moreover
have *butlast* (*lms-seq ?xs i*) = *filter* (*abs-is-lms ?xs*) [*i*..<*n*]
  unfolding *lms-seq-def*
  using ‹[*i*..<*length ?xs*] = [*i*..<*n*] @ [*n*..<*Suc n*]› ‹*abs-is-lms ?xs n*› **by** *auto*
hence ∀ *x* ∈ *set* (*butlast* (*lms-seq ?xs i*)). *x* ≠ *n* ∧ *abs-is-lms ?xs x*
  by *clarsimp*
hence ∀ *x* ∈ *set* (*butlast* (*lms-substr-seq ?xs i*)).
    list-less-ns (*lms-slice ?xs n*) *x*
  by (*clarsimp simp*: *P map-butlast*[*symmetric*] *lms-substr-seq-def*)
hence *S*: ∀ *x* ∈ *set* (*butlast* (*lms-substr-seq-id ?xs* (*suffix ?xs i*))).
    list-less-ns (*lms-slice ?xs n*) *x*
  by (*metis lms-subtrs-seq-id-suffix*)
have ∀ *x* ∈ *set* (*butlast* (*lms-map ?xs* (*suffix ?xs i*))). *x* > *0*
proof *safe*
  fix *x*
  assume *x* ∈ *set* (*butlast* (*lms-map ?xs* (*suffix ?xs i*)))
  moreover
  have *butlast* (*lms-map ?xs* (*suffix ?xs i*)) =
      map (*ordlistns.elem-rank* (*lms-substrs ?xs*)) (*butlast* (*lms-substr-seq-id ?xs* (*suffix ?xs i*)))
    unfolding *lms-map-o-def*
    by (*simp add*: *map-butlast*)
  ultimately have
    ∃ *y* ∈ *set* (*butlast* (*lms-substr-seq-id ?xs* (*suffix ?xs i*))).
      x = *ordlistns.elem-rank* (*lms-substrs ?xs*) *y*
    by (*simp add*: *in-set-mapD*)
  then obtain *y* **where** *A*:
    y ∈ *set* (*butlast* (*lms-substr-seq-id ?xs* (*suffix ?xs i*)))
    x = *ordlistns.elem-rank* (*lms-substrs ?xs*) *y*
    by *blast*
  moreover
  have *y* ∈ *lms-substrs* (*a* # *b* # *xs*)
  by (*metis calculation*(*1*) *in-set-butlastD in-set-conv-nth lms-substr-seq-id-nth-abs-is-lms*)
  ultimately show *0* < *x*
    using *S R* **by** *blast*
qed

moreover
have *lms-map ?xs* (*suffix ?xs i*) $\neq$ []
unfolding *lms-map-o-def*
by (*metis assms*(*2*) *list.discI list.map-disc-iff lms-seq-Suc1 lms-substr-seq-def lms-subtrs-seq-id-suffix*)
ultimately show *?thesis*
unfolding *valid-list-iff-butlast-app-last*
by (*metis bot-nat-def less-numeral-extra*(*3*))
qed

end
theory *Abs-SAIS*
imports *../prop/Buckets*
../prop/LMS-List-Slice-Util
../../util/Repeat
begin

54 Induce Sorting

54.1 Bucket Insert

```
fun abs-bucket-insert ::
  (('a :: {linorder, order-bot}) ⇒ nat) ⇒
   'a list ⇒
   nat list ⇒
   nat list ⇒
   nat list ⇒
   nat list
  where
abs-bucket-insert α T - SA [] = SA |
abs-bucket-insert α T B SA (x # xs) =
  (let b = α (T ! x);
       k = B ! b − Suc 0;
       SA' = SA[k := x];
       B' = B[b := k]
   in abs-bucket-insert α T B' SA' xs)
```

54.2 Induce L-types

```
fun abs-induce-l-step ::
  nat list × nat list × nat ⇒
   (('a :: {linorder, order-bot}) ⇒ nat) × 'a list ⇒
   nat list × nat list × nat
  where
abs-induce-l-step (B, SA, i) (α, T) =
  (if i < length SA ∧ SA ! i < length T
   then
     (case SA ! i of
        Suc j ⇒
```

$(case\ suffix\text{-}type\ T\ j\ of$
$L\text{-}type \Rightarrow$
$(let\ k = \alpha\ (T\ !\ j);$
$l = B\ !\ k$
$in\ (B[k := Suc\ l],\ SA[l := j],\ Suc\ i))$
$|\ \text{-} \Rightarrow (B,\ SA,\ Suc\ i))$
$|\ \text{-} \Rightarrow (B,\ SA,\ Suc\ i))$
$else\ (B,\ SA,\ Suc\ i))$

definition *abs-induce-l-base* ::
$(('a :: \{linorder,\ order\text{-}bot\}) \Rightarrow nat) \Rightarrow$
$'a\ list \Rightarrow$
$nat\ list \Rightarrow$
$nat\ list \Rightarrow$
$nat\ list \times nat\ list \times nat$
where
$abs\text{-}induce\text{-}l\text{-}base\ \alpha\ T\ B\ SA = repeat\ (length\ T)\ abs\text{-}induce\text{-}l\text{-}step\ (B,\ SA,\ 0)\ (\alpha,\ T)$

definition *abs-induce-l* ::
$(('a :: \{linorder,\ order\text{-}bot\}) \Rightarrow nat) \Rightarrow$
$'a\ list \Rightarrow$
$nat\ list \Rightarrow$
$nat\ list \Rightarrow$
$nat\ list$
where
$abs\text{-}induce\text{-}l\ \alpha\ T\ B\ SA =$
$(let\ (B',\ SA',\ i) = abs\text{-}induce\text{-}l\text{-}base\ \alpha\ T\ B\ SA$
$in\ SA')$

54.3 Induce S-types

fun *abs-induce-s-step* ::
$nat\ list \times nat\ list \times nat \Rightarrow$
$(('a :: \{linorder,\ order\text{-}bot\}) \Rightarrow nat) \times 'a\ list \Rightarrow$
$nat\ list \times nat\ list \times nat$
where
$abs\text{-}induce\text{-}s\text{-}step\ (B,\ SA,\ i)\ (\alpha,\ T) =$
$(case\ i\ of$
$Suc\ n \Rightarrow$
$(if\ Suc\ n < length\ SA \wedge SA\ !\ Suc\ n < length\ T\ then$
$(case\ SA\ !\ Suc\ n\ of$
$Suc\ j \Rightarrow$
$(case\ suffix\text{-}type\ T\ j\ of$
$S\text{-}type \Rightarrow$
$(let\ b = \alpha\ (T\ !\ j);$
$k = B\ !\ b - Suc\ 0$
$in\ (B[b := k],\ SA[k := j],\ n)$
$)$

```
          | - ⇒ (B, SA, n)
        )
      | - ⇒ (B, SA, n)
    )
    else
      (B, SA, n)
   )
 | -  ⇒ (B, SA, 0)
)
```

definition *abs-induce-s-base* ::
((′*a* :: {*linorder*, *order-bot*}) ⇒ *nat*) ⇒
′*a list* ⇒
nat list ⇒
nat list ⇒
nat list × *nat list* × *nat*
where
abs-induce-s-base α *T B SA* = *repeat* (*length T*) *abs-induce-s-step* (*B*, *SA*, *length T*) (α, *T*)

definition *abs-induce-s* ::
((′*a* :: {*linorder*, *order-bot*}) ⇒ *nat*) ⇒
′*a list* ⇒
nat list ⇒
nat list ⇒
nat list
where
abs-induce-s α *T B SA* =
(*let* (*B*′, *SA*′, *i*) = *abs-induce-s-base* α *T B SA*
in SA′)

54.4 Induce Sorting

definition *abs-sa-induce* ::
((′*a* :: {*linorder*, *order-bot*}) ⇒ *nat*) ⇒
′*a list* ⇒
nat list ⇒
nat list
where
abs-sa-induce α *T LMS* =
(*let*
B0 = *map* (*bucket-end* α *T*) [*0*..<*Suc* (α (*Max* (*set T*)))];
B1 = *map* (*bucket-start* α *T*) [*0*..<*Suc* (α (*Max* (*set T*)))];

— Initialise SA
SA = *replicate* (*length T*) (*length T*);

— Insert the LMS types into the suffix array
SA = *abs-bucket-insert* α *T B0 SA* (*rev LMS*);

— Insert the L types into the suffix array
SA = abs-induce-l α T B1 SA

— Insert the S types into the suffix array
in abs-induce-s α T (B0[0 := 0]) SA)

55 Rename Mapping

fun *abs-rename-mapping′* ::
('*a* :: {*linorder*, *order-bot*}) *list* ⇒
nat list ⇒
nat list ⇒
nat ⇒
nat list
where
abs-rename-mapping′ - [] *names* - = *names* |
abs-rename-mapping′ - [*x*] *names i* = *names*[*x* := *i*] |
abs-rename-mapping′ T (*a* # *b* # *xs*) *names i* =
(*if lms-slice T a* = *lms-slice T b*
then abs-rename-mapping′ T (*b* # *xs*) (*names*[*a* := *i*]) *i*
else abs-rename-mapping′ T (*b* # *xs*) (*names*[*a* := *i*]) (*Suc i*))

definition *abs-rename-mapping* :: ('*a* :: {*linorder*, *order-bot*}) *list* ⇒ *nat list* ⇒ *nat list*
where
abs-rename-mapping T LMS = *abs-rename-mapping′ T LMS* (*replicate* (*length T*) (*length T*)) *0*

56 Rename String

fun *rename-string* :: *nat list* ⇒ *nat list* ⇒ *nat list*
where
rename-string [] - = [] |
rename-string (*x*#*xs*) *names* = (*names* ! *x*) # *rename-string xs names*

57 Order LMS

fun *order-lms* :: *nat list* ⇒ *nat list* ⇒ *nat list*
where
order-lms LMS [] = [] |
order-lms LMS (*x* # *xs*) = *LMS* ! *x* # *order-lms LMS xs*

58 Extract LMS

abbreviation *abs-extract-lms* :: ('*a* :: {*linorder*, *order-bot*}) *list* ⇒ *nat list* ⇒ *nat list*

where
abs-extract-lms ≡ *filter* ∘ *abs-is-lms*

59 SAIS Definition

function *abs-sais* ::
nat list ⇒
nat list
where
abs-sais [] = [] |
abs-sais [*x*] = [*0*] |
abs-sais (*a* # *b* # *xs*) =
(*let*
T = *a* # *b* # *xs*;

— Extract the LMS types
LMS0 = *abs-extract-lms T* [*0*..<*length T*];

— Induce the prefix ordering based on LMS
SA = *abs-sa-induce id T LMS0*;

— Extract the LMS types
LMS = *abs-extract-lms T SA*;

— Create a new alphabet
names = *abs-rename-mapping T LMS*;

— Make a reduced string
T′ = *rename-string LMS0 names*;

— Obtain the correct ordering of LMS-types
LMS = (*if distinct T′ then LMS else order-lms LMS0* (*abs-sais T′*))

— Induce the suffix ordering based of LMS
in abs-sa-induce id T LMS)
by *pat-completeness blast+*

end
theory *Abs-Bucket-Insert-Verification*
imports
../abs−def/Abs-SAIS
../../util/List-Util
../../util/List-Slice

begin

60 Bucket Insert with Ghost State

fun *bucket-insert-abs′* ::
  $(('a :: \{linorder,\ order\text{-}bot\}) \Rightarrow nat) \Rightarrow$
   $'a\ list \Rightarrow$
   $nat\ list \Rightarrow$
   $nat\ list \Rightarrow$
   $nat\ list \Rightarrow$
   $nat\ list \Rightarrow$
   $nat\ list \times nat\ list \times nat\ list$
where
  $bucket\text{-}insert\text{-}abs'\ \alpha\ T\ B\ SA\ gs\ []\ = (SA,\ B,\ gs)\ |$
  $bucket\text{-}insert\text{-}abs'\ \alpha\ T\ B\ SA\ gs\ (x\ \#\ xs) =$
   $(let\ b = \alpha\ (T\ !\ x);$
     $k = B\ !\ b - Suc\ 0;$
     $SA' = SA[k := x];$
     $B' = B[b := k];$
     $gs' = gs\ @\ [x]$
    $in\ bucket\text{-}insert\text{-}abs'\ \alpha\ T\ B'\ SA'\ gs'\ xs)$

61 Simple Properties

lemma *abs-bucket-insert-length*:
  $length\ (abs\text{-}bucket\text{-}insert\ \alpha\ T\ B\ SA\ xs) = length\ SA$
  by (*induct xs arbitrary*: *B SA*; *simp add*: *Let-def*)

lemma *abs-bucket-insert-equiv*:
  $abs\text{-}bucket\text{-}insert\ \alpha\ T\ B\ SA\ xs = fst\ (bucket\text{-}insert\text{-}abs'\ \alpha\ T\ B\ SA\ gs\ xs)$
  by (*induct xs arbitrary*: *B SA gs*; *simp add*: *Let-def*)

62 Invariants

62.1 Defintions and Simple Helper Lemmas

62.1.1 Distinctness

definition *lms-distinct-inv* ::
  $('a :: \{linorder,\ order\text{-}bot\})\ list \Rightarrow nat\ list \Rightarrow nat\ list \Rightarrow bool$
where
  $lms\text{-}distinct\text{-}inv\ T\ SA\ LMS =$
   $distinct\ ((filter\ (\lambda x.\ x < length\ T)\ SA)\ @\ LMS)$

lemma *lms-inv-distinct-inv-helper*:
  assumes $lms\text{-}distinct\text{-}inv\ T\ SA\ LMS$
  shows $distinct\ (filter\ (\lambda x.\ x < length\ T)\ SA) \wedge$
    $distinct\ LMS \wedge$
    $set\ (filter\ (\lambda x.\ x < length\ T)\ SA) \cap set\ LMS = \{\}$
  using *assms distinct-append lms-distinct-inv-def* **by** *blast*

62.1.2 LMS Bucket Ptr

definition *cur-lms-types* ::
  $('a :: \{linorder, order\text{-}bot\} \Rightarrow nat) \Rightarrow 'a\ list \Rightarrow nat\ list \Rightarrow nat \Rightarrow nat\ set$
where
  $cur\text{-}lms\text{-}types\ \alpha\ T\ SA\ b =$
    $\{i|i.\ i \in set\ SA\ \wedge$
    $i \in lms\text{-}bucket\ \alpha\ T\ b\ \}$

lemma *cur-lms-subset-SA*:
  $cur\text{-}lms\text{-}types\ \alpha\ T\ SA\ b \subseteq set\ SA$
  using *cur-lms-types-def* **by** *blast*

lemma *cur-lms-subset-lms-bucket*:
  $cur\text{-}lms\text{-}types\ \alpha\ T\ SA\ b \subseteq lms\text{-}bucket\ \alpha\ T\ b$
  using *cur-lms-types-def* **by** *blast*

definition *num-lms-types* ::
  $('a :: \{linorder, order\text{-}bot\} \Rightarrow nat) \Rightarrow 'a\ list \Rightarrow nat\ list \Rightarrow nat \Rightarrow nat$
where
  $num\text{-}lms\text{-}types\ \alpha\ T\ SA\ b =$
   $card\ (cur\text{-}lms\text{-}types\ \alpha\ T\ SA\ b)$

lemma *num-lms-types-upper-bound*:
  $num\text{-}lms\text{-}types\ \alpha\ T\ SA\ b \le lms\text{-}bucket\text{-}size\ \alpha\ T\ b$
  by (*metis not-le cur-lms-subset-lms-bucket num-lms-types-def*
      finite-lms-bucket lms-bucket-size-def card-mono)

definition *lms-bucket-ptr-inv* ::
  $('a :: \{linorder, order\text{-}bot\} \Rightarrow nat) \Rightarrow$
   $'a\ list \Rightarrow nat\ list \Rightarrow nat\ list \Rightarrow bool$
where
  $lms\text{-}bucket\text{-}ptr\text{-}inv\ \alpha\ T\ B\ SA \equiv$
   $(\forall\, b \le \alpha\ (Max\ (set\ T)).$
    $B\ !\ b + num\text{-}lms\text{-}types\ \alpha\ T\ SA\ b = bucket\text{-}end\ \alpha\ T\ b)$

lemma *lms-bucket-ptr-invD*:
  assumes $lms\text{-}bucket\text{-}ptr\text{-}inv\ \alpha\ T\ B\ SA$
  and $b \le \alpha\ (Max\ (set\ T))$
  shows $B\ !\ b + num\text{-}lms\text{-}types\ \alpha\ T\ SA\ b = bucket\text{-}end\ \alpha\ T\ b$
  using *assms lms-bucket-ptr-inv-def* **by** *blast*

lemma *lms-bucket-ptr-lower-bound*:
  assumes $lms\text{-}bucket\text{-}ptr\text{-}inv\ \alpha\ T\ B\ SA$
  and $b \le \alpha\ (Max\ (set\ T))$
  shows $lms\text{-}bucket\text{-}start\ \alpha\ T\ b \le B\ !\ b$
proof −
  from *lms-bucket-ptr-invD*[*OF assms*]
  have $B\ !\ b + num\text{-}lms\text{-}types\ \alpha\ T\ SA\ b = bucket\text{-}end\ \alpha\ T\ b$ **.**
  then show *?thesis*

by (*metis add.commute add-le-cancel-left lms-bucket-pl-size-eq-end num-lms-types-upper-bound*)
qed

lemma *lms-bucket-ptr-upper-bound*:
assumes *lms-bucket-ptr-inv* α T B SA
and $b \leq \alpha$ (*Max* (*set* T))
shows B ! $b \leq$ *bucket-end* α T b
by (*metis assms le-iff-add lms-bucket-ptr-inv-def*)

62.1.3 Unknowns

definition *lms-unknowns-inv* ::
('*a* :: {*linorder*, *order-bot*} $\Rightarrow$ *nat*) $\Rightarrow$
'*a list* $\Rightarrow$ *nat list* $\Rightarrow$ *nat list* $\Rightarrow$ *bool*
where
lms-unknowns-inv α T B SA $\equiv$
($\forall b \leq \alpha$ (*Max* (*set* T)).
($\forall i$. *lms-bucket-start* α T $b \leq i$ $\wedge$
$i < B$! $b \longrightarrow$ SA ! i = *length* T))

lemma *lms-unknowns-invD*:
assumes *lms-unknowns-inv* α T B SA
and $b \leq \alpha$ (*Max* (*set* T))
and *lms-bucket-start* α T $b \leq i$
and $i < B$! b
shows SA ! i = *length* T
using *assms lms-unknowns-inv-def* **by** *blast*

62.1.4 Locations

definition *lms-locations-inv* ::
('*a* :: {*linorder*, *order-bot*} $\Rightarrow$ *nat*) $\Rightarrow$
'*a list* $\Rightarrow$ *nat list* $\Rightarrow$ *nat list* $\Rightarrow$ *bool*
where
lms-locations-inv α T B SA $\equiv$
($\forall b \leq \alpha$ (*Max* (*set* T)).
($\forall i$. B ! $b \leq i$ $\wedge$
$i <$ *bucket-end* α T $b \longrightarrow SA$! $i \in$ *lms-bucket* α T b))

lemma *lms-locations-invD*:
assumes *lms-locations-inv* α T B SA
and $b \leq \alpha$ (*Max* (*set* T))
and B ! $b \leq i$
and $i <$ *bucket-end* α T b
shows SA ! $i \in$ *lms-bucket* α T b
using *assms lms-locations-inv-def* **by** *blast*

62.1.5 Unchanged

definition *lms-unchanged-inv* ::

$('a :: \{linorder,\ order\text{-}bot\} \Rightarrow nat) \Rightarrow$
$'a\ list \Rightarrow nat\ list \Rightarrow nat\ list \Rightarrow nat\ list \Rightarrow bool$
where
$lms\text{-}unchanged\text{-}inv\ \alpha\ T\ B\ SA\ SA' \equiv$
$(\forall b \le \alpha\ (Max\ (set\ T)).$
$(\forall i.\ bucket\text{-}start\ \alpha\ T\ b \le i \wedge$
$i < B\ !\ b \longrightarrow\ SA'\ !\ i = SA\ !\ i))$

lemma *lms-unchanged-invD*:
assumes $lms\text{-}unchanged\text{-}inv\ \alpha\ T\ B\ SA\ SA'$
and $b \le \alpha\ (Max\ (set\ T))$
and $bucket\text{-}start\ \alpha\ T\ b \le i$
and $i < B\ !\ b$
shows $SA'\ !\ i = SA\ !\ i$
using *assms lms-unchanged-inv-def* **by** *blast*

62.1.6 Inserted

definition *lms-inserted-inv* ::
$nat\ list \Rightarrow nat\ list \Rightarrow nat\ list \Rightarrow nat\ list \Rightarrow bool$
where
$lms\text{-}inserted\text{-}inv\ LMS\ SA\ LMSa\ LMSb \equiv$
$LMS = LMSa\ @\ LMSb\ \wedge$
$set\ LMSa \subseteq set\ SA$

lemma *lms-inserted-invD*:
$\bigwedge LMS\ SA\ LMSa\ LMSb.\ lms\text{-}inserted\text{-}inv\ LMS\ SA\ LMSa\ LMSb \Longrightarrow LMS = LMSa\ @\ LMSb$
$\bigwedge LMS\ SA\ LMSa\ LMSb.\ lms\text{-}inserted\text{-}inv\ LMS\ SA\ LMSa\ LMSb \Longrightarrow set\ LMSa \subseteq set\ SA$
unfolding *lms-inserted-inv-def* **by** *blast+*

62.1.7 Sorted

definition *lms-sorted-inv* :: $('a :: \{linorder,\ order\text{-}bot\})\ list \Rightarrow nat\ list \Rightarrow nat\ list \Rightarrow bool$
where
$lms\text{-}sorted\text{-}inv\ T\ LMS\ SA \equiv$
$(\forall j < length\ SA.$
$\forall i < j.$
$SA\ !\ i \in set\ LMS \wedge SA\ !\ j \in set\ LMS \longrightarrow$
$(T\ !\ (SA\ !\ i) \ne T\ !\ (SA\ !\ j) \longrightarrow T\ !\ (SA\ !\ i) < T\ !\ (SA\ !\ j)) \wedge$
$(T\ !\ (SA\ !\ i) = T\ !\ (SA\ !\ j) \longrightarrow$
$(\exists j' < length\ LMS.\ \exists i' < j'.\ LMS\ !\ i' = SA\ !\ j \wedge LMS\ !\ j' = SA\ !\ i))$
$)$

lemma *lms-sorted-invD*:
$[\![lms\text{-}sorted\text{-}inv\ T\ LMS\ SA;\ j < length\ SA;\ i < j;\ SA\ !\ i \in set\ LMS;\ SA\ !\ j \in set\ LMS]\!] \Longrightarrow$
$(T\ !\ (SA\ !\ i) \ne T\ !\ (SA\ !\ j) \longrightarrow T\ !\ (SA\ !\ i) < T\ !\ (SA\ !\ j)) \wedge$

$(T ! (SA ! i) = T ! (SA ! j) \longrightarrow$
$(\exists j' < \mathit{length}\ LMS.\ \exists i' < j'.\ LMS ! i' = SA ! j \wedge LMS ! j' = SA ! i))$
using *lms-sorted-inv-def* **by** *blast*

lemma *lms-sorted-invD1*:
$[\![$*lms-sorted-inv* T LMS SA; $j <$ *length* SA; $i < j$;
$SA ! i \in$ *set* LMS; $SA ! j \in$ *set* LMS;
$T ! (SA ! i) \neq T ! (SA ! j)]\!] \Longrightarrow$
$T ! (SA ! i) < T ! (SA ! j)$
using *lms-sorted-inv-def* **by** *blast*

lemma *lms-sorted-invD2*:
$[\![$*lms-sorted-inv* T LMS SA; $j <$ *length* SA; $i < j$; $SA ! i \in$ *set* LMS; $SA ! j \in$ *set* LMS;
$T ! (SA ! i) = T ! (SA ! j)]\!] \Longrightarrow$
$\exists j' < \mathit{length}\ LMS.\ \exists i' < j'.\ LMS ! i' = SA ! j \wedge LMS ! j' = SA ! i$
using *lms-sorted-inv-def* **by** *blast*

62.2 Combined Invariant

definition *lms-inv* ::
$('a :: \{\mathit{linorder},\ \mathit{order\text{-}bot}\} \Rightarrow \mathit{nat}) \Rightarrow$
$'a\ \mathit{list} \Rightarrow$
$\mathit{nat\ list} \Rightarrow$
$\mathit{nat\ list} \Rightarrow$
$\mathit{nat\ list} \Rightarrow$
$\mathit{nat\ list} \Rightarrow$
$\mathit{nat\ list} \Rightarrow$
$\mathit{nat\ list} \Rightarrow$
bool
where
lms-inv α T B LMS $LMSa$ $LMSb$ $SA0$ $SA \equiv$
lms-distinct-inv T SA $LMSb$ $\wedge$
lms-bucket-ptr-inv α T B SA $\wedge$
lms-unknowns-inv α T B SA $\wedge$
lms-locations-inv α T B SA $\wedge$
lms-unchanged-inv α T B $SA0$ SA $\wedge$
lms-inserted-inv LMS SA $LMSa$ $LMSb$ $\wedge$
lms-sorted-inv T LMS SA $\wedge$
strict-mono α $\wedge$
$\alpha\ (\mathit{Max}\ (\mathit{set}\ T)) < \mathit{length}\ B \wedge$
$\mathit{set}\ LMS \subseteq \{i.\ \mathit{abs\text{-}is\text{-}lms}\ T\ i\} \wedge$
$\mathit{length}\ SA0 = \mathit{length}\ T \wedge$
$\mathit{length}\ SA = \mathit{length}\ T \wedge$
$(\forall i < \mathit{length}\ T.\ SA0 ! i = \mathit{length}\ T)$

lemma *lms-invD*:
lms-inv α T B LMS $LMSa$ $LMSb$ $SA0$ $SA \Longrightarrow$ *lms-distinct-inv* T SA $LMSb$
lms-inv α T B LMS $LMSa$ $LMSb$ $SA0$ $SA \Longrightarrow$ *lms-bucket-ptr-inv* α T B SA

$lms\text{-}inv\ \alpha\ T\ B\ LMS\ LMSa\ LMSb\ SA0\ SA \Longrightarrow lms\text{-}unknowns\text{-}inv\ \alpha\ T\ B\ SA$
$lms\text{-}inv\ \alpha\ T\ B\ LMS\ LMSa\ LMSb\ SA0\ SA \Longrightarrow lms\text{-}locations\text{-}inv\ \alpha\ T\ B\ SA$
$lms\text{-}inv\ \alpha\ T\ B\ LMS\ LMSa\ LMSb\ SA0\ SA \Longrightarrow lms\text{-}unchanged\text{-}inv\ \alpha\ T\ B\ SA0\ SA$
$lms\text{-}inv\ \alpha\ T\ B\ LMS\ LMSa\ LMSb\ SA0\ SA \Longrightarrow lms\text{-}inserted\text{-}inv\ LMS\ SA\ LMSa\ LMSb$
$lms\text{-}inv\ \alpha\ T\ B\ LMS\ LMSa\ LMSb\ SA0\ SA \Longrightarrow lms\text{-}sorted\text{-}inv\ T\ LMS\ SA$
$lms\text{-}inv\ \alpha\ T\ B\ LMS\ LMSa\ LMSb\ SA0\ SA \Longrightarrow strict\text{-}mono\ \alpha$
$lms\text{-}inv\ \alpha\ T\ B\ LMS\ LMSa\ LMSb\ SA0\ SA \Longrightarrow \alpha\ (Max\ (set\ T)) < length\ B$
$lms\text{-}inv\ \alpha\ T\ B\ LMS\ LMSa\ LMSb\ SA0\ SA \Longrightarrow set\ LMS \subseteq \{i.\ abs\text{-}is\text{-}lms\ T\ i\}$
$lms\text{-}inv\ \alpha\ T\ B\ LMS\ LMSa\ LMSb\ SA0\ SA \Longrightarrow length\ SA0 = length\ T$
$lms\text{-}inv\ \alpha\ T\ B\ LMS\ LMSa\ LMSb\ SA0\ SA \Longrightarrow length\ SA = length\ T$
$lms\text{-}inv\ \alpha\ T\ B\ LMS\ LMSa\ LMSb\ SA0\ SA \Longrightarrow \forall\, i < length\ T.\ SA0\ !\ i = length\ T$
unfolding *lms-inv-def* **by** *blast+*

lemma *lms-inv-lms-helper*:
$lms\text{-}inv\ \alpha\ T\ B\ LMS\ LMSa\ LMSb\ SA0\ SA \Longrightarrow \forall\, x \in set\ LMS.\ abs\text{-}is\text{-}lms\ T\ x$
$lms\text{-}inv\ \alpha\ T\ B\ LMS\ LMSa\ LMSb\ SA0\ SA \Longrightarrow \forall\, x \in set\ LMSa.\ abs\text{-}is\text{-}lms\ T\ x$
$lms\text{-}inv\ \alpha\ T\ B\ LMS\ LMSa\ LMSb\ SA0\ SA \Longrightarrow \forall\, x \in set\ LMSb.\ abs\text{-}is\text{-}lms\ T\ x$
using *lms-invD*(*10*) *lms-inserted-invD*(*1*)[*OF lms-invD*(*6*)] **by** *fastforce+*

62.3 Helpers

lemma *lms-distinct-bucket-ptr-lower-bound*:
assumes $b = \alpha\ (T\ !\ x)$
and $lms\text{-}distinct\text{-}inv\ T\ SA\ (x\ \#\ LMS)$
and $lms\text{-}bucket\text{-}ptr\text{-}inv\ \alpha\ T\ B\ SA$
and $strict\text{-}mono\ \alpha$
and $\forall\, i \in set\ (x\ \#\ LMS).\ abs\text{-}is\text{-}lms\ T\ i$
shows $lms\text{-}bucket\text{-}start\ \alpha\ T\ b < B\ !\ b$
proof (*rule ccontr*)
assume $\neg\ lms\text{-}bucket\text{-}start\ \alpha\ T\ b < B\ !\ b$
hence $B\ !\ b \le lms\text{-}bucket\text{-}start\ \alpha\ T\ b$
by *simp*
moreover
from *lms-bucket-ptr-invD*[*OF assms*(*3*), *of b*] *assms*(*1*,*4*,*5*)
have $B\ !\ b + num\text{-}lms\text{-}types\ \alpha\ T\ SA\ b = bucket\text{-}end\ \alpha\ T\ b$
by (*simp add*: *abs-is-lms-imp-less-length strict-mono-leD*)
ultimately
have $lms\text{-}bucket\text{-}start\ \alpha\ T\ b + num\text{-}lms\text{-}types\ \alpha\ T\ SA\ b \ge bucket\text{-}end\ \alpha\ T\ b$
by *linarith*
with *lms-bucket-pl-size-eq-end*
have $num\text{-}lms\text{-}types\ \alpha\ T\ SA\ b \ge lms\text{-}bucket\text{-}size\ \alpha\ T\ b$
by (*metis add-le-cancel-left*)
with *card-seteq*[*OF finite-lms-bucket cur-lms-subset-lms-bucket*]
have $cur\text{-}lms\text{-}types\ \alpha\ T\ SA\ b = lms\text{-}bucket\ \alpha\ T\ b$
by (*simp add*: *lms-bucket-size-def num-lms-types-def*)
with *cur-lms-subset-SA*
have $lms\text{-}bucket\ \alpha\ T\ b \subseteq set\ SA$

by *blast*
moreover
from *assms*(*1*,*5*)
have $x \in$ *lms-bucket* α *T b*
by (*simp add*: *bucket-def abs-is-lms-imp-less-length lms-bucket-def*)
ultimately
have $x \in$ *set SA*
by *blast*
moreover
from *assms*(*2*,*5*)
have $x \notin$ *set SA*
by (*simp add*: *abs-is-lms-imp-less-length lms-distinct-inv-def*)
ultimately
show *False*
by *blast*
qed

lemma *lms-next-insert-at-unknown*:
assumes $b = \alpha$ (*T* ! *x*)
and $k =$ (*B* ! *b*) $-$ *Suc 0*
and *lms-distinct-inv T SA* (*x* # *LMS*)
and *lms-bucket-ptr-inv* α *T B SA*
and *lms-unknowns-inv* α *T B SA*
and *strict-mono* α
and *length SA* = *length T*
and $\forall$ *i* $\in$ *set* (*x* # *LMS*). *abs-is-lms T i*
shows *k* < *length SA* $\wedge$ *SA* ! *k* = *length T*
proof $-$

from *lms-distinct-bucket-ptr-lower-bound*[*OF assms*(*1*,*3*$-$*4*,*6*,*8*)]
have *lms-bucket-start* α *T b* < *B* ! *b*
by *assumption*
with *assms*(*2*)
have *lms-bucket-start* α *T b* $\leq$ *k k* < *B* ! *b*
by *linarith*+
with *lms-unknowns-invD*[*OF assms*(*5*), *of b k*] *assms*(*1*,*6*,*8*)
have *SA* ! *k* = *length T*
by (*simp add*: *abs-is-lms-imp-less-length strict-mono-less-eq*)
moreover
from ‹*k* < *B* ! *b*› *lms-bucket-ptr-invD*[*OF assms*(*4*), *of b*] *assms*(*1*,*6*,*8*)
have *k* < *bucket-end* α *T b*
by (*simp add*: *assms*(*7*) *abs-is-lms-imp-less-length strict-mono-less-eq*)
with *assms*(*7*)
have *k* < *length SA*
by (*metis bucket-end-le-length dual-order.strict-trans1*)
ultimately
show *?thesis*
by *blast*
qed

lemma *lms-distinct-slice*:
  assumes *lms-distinct-inv T SA LMS*
  and *lms-bucket-ptr-inv* α *T B SA*
  and *lms-locations-inv* α *T B SA*
  and *length SA* = *length T*
  and $b \leq \alpha$ (*Max* (*set T*))
shows *distinct* (*list-slice SA* (*B* ! *b*) (*bucket-end* α *T b*))
proof −
  from *assms*(*4*)
  have *bucket-end* α *T b* ≤ *length SA*
    by (*simp add*: *bucket-end-le-length*)

  from *lms-bucket-ptr-upper-bound*[*OF assms*(*2*,*5*)]
  have *B* ! *b* ≤ *bucket-end* α *T b* **.**

  from *lms-locations-invD*[*OF assms*(*3*,*5*)]
  have ∀ *i*. *B* ! *b* ≤ *i* ∧ *i* < *bucket-end* α *T b* ⟶ *SA* ! *i* ∈ *lms-bucket* α *T b*
    by *blast*
  hence ∀ *i*. *B* ! *b* ≤ *i* ∧ *i* < *bucket-end* α *T b* ⟶ *SA* ! *i* < *length T*
    by (*simp add*: *abs-is-lms-imp-less-length lms-bucket-def*)

  have ∀ *x* ∈ *set* (*list-slice SA* (*B* ! *b*) (*bucket-end* α *T b*)). *x* < *length T*
  proof
    fix *x*
    assume *A*: *x* ∈ *set* (*list-slice SA* (*B* ! *b*) (*bucket-end* α *T b*))
    from *in-set-conv-nth*[*THEN iffD1*, *OF A*]
    obtain *i* **where**
      i < *length* (*list-slice SA* (*B* ! *b*) (*bucket-end* α *T b*))
      (*list-slice SA* (*B* ! *b*) (*bucket-end* α *T b*)) ! *i* = *x*
      by *blast*
    with *nth-list-slice*
    have (*list-slice SA* (*B* ! *b*) (*bucket-end* α *T b*)) ! *i* = *SA* ! (*B* ! *b* + *i*)
      by *blast*
    moreover
    from ‹*i* < *length* (*list-slice SA* (*B* ! *b*) (*bucket-end* α *T b*))›
    have *B* ! *b* + *i* < *bucket-end* α *T b*
      by (*simp add*: ‹*bucket-end* α *T b* ≤ *length SA*›
                length-list-slice)
    with ‹∀ *i*. *B* ! *b* ≤ *i* ∧ *i* < *bucket-end* α *T b* ⟶ *SA* ! *i* < *length T*›
    have *SA* ! (*B* ! *b* + *i*) < *length T*
      by *simp*
    ultimately
    show *x* < *length T*
      using ‹(*list-slice SA* (*B* ! *b*) (*bucket-end* α *T b*)) ! *i* = *x*› **by** *simp*
  qed

  from *lms-inv-distinct-inv-helper*[*OF assms*(*1*)]
  have *distinct* (*filter* (λ*x*. *x* < *length T*) *SA*) *distinct LMS*

set (*filter* (*λx*. *x* < *length T*) *SA*) ∩ *set LMS* = {}
by *blast*+

have *SA* = *list-slice SA 0* (*length SA*)
by (*simp add*: *list-slice-0-length*)
hence *SA* = *list-slice SA 0* (*B* ! *b*) @ *list-slice SA* (*B* ! *b*) (*length SA*)
using *append-take-drop-id*
by (*simp add*: *list-slice.simps*)
moreover
from *list-slice-append*[*OF* ‹*B* ! *b* ≤ *bucket-end α T b*› ‹*bucket-end α T b* ≤ *length SA*›, *of SA*]
have *list-slice SA* (*B* ! *b*) (*length SA*)
= *list-slice SA* (*B* ! *b*) (*bucket-end α T b*) @ *list-slice SA* (*bucket-end α T b*) (*length SA*)
by *assumption*
ultimately
have *SA* = *list-slice SA 0* (*B* ! *b*) @ *list-slice SA* (*B* ! *b*) (*bucket-end α T b*) @
list-slice SA (*bucket-end α T b*) (*length SA*)
by *metis*
with ‹*distinct* (*filter* (*λx*. *x* < *length T*) *SA*)›
have *distinct* (*filter* (*λx*. *x* < *length T*) (*list-slice SA* (*B* ! *b*) (*bucket-end α T b*)))
by (*metis distinct-append filter-append*)
with ‹∀ *x* ∈ *set* (*list-slice SA* (*B* ! *b*) (*bucket-end α T b*)). *x* < *length T*›
show *distinct* (*list-slice SA* (*B* ! *b*) (*bucket-end α T b*))
by *simp*
qed

lemma *lms-slice-subset-lms-bucket*:
assumes *lms-locations-inv α T B SA*
and *length SA* = *length T*
and *b* ≤ *α* (*Max* (*set T*))
shows *set* (*list-slice SA* (*B* ! *b*) (*bucket-end α T b*)) ⊆ *lms-bucket α T b*
proof
fix *x*
assume *A*: *x* ∈ *set* (*list-slice SA* (*B* ! *b*) (*bucket-end α T b*))
from *in-set-conv-nth*[*THEN iffD1*, *OF A*]
obtain *i* **where**
i < *length* (*list-slice SA* (*B* ! *b*) (*bucket-end α T b*))
(*list-slice SA* (*B* ! *b*) (*bucket-end α T b*)) ! *i* = *x*
by *blast*
with *nth-list-slice*
have (*list-slice SA* (*B* ! *b*) (*bucket-end α T b*)) ! *i* = *SA* ! (*B* ! *b* + *i*)
by *blast*
moreover
from ‹*i* < *length* (*list-slice SA* (*B* ! *b*) (*bucket-end α T b*))›
have *B* ! *b* + *i* < *bucket-end α T b*
by (*simp add*: *assms*(*2*) *bucket-end-le-length length-list-slice*)
moreover

have $B\ !\ b \leq B\ !\ b + i$
by *simp*
ultimately
show $x \in$ *lms-bucket* α T b
using ‹*list-slice SA* $(B\ !\ b)$ (*bucket-end* α T b) ! $i = x$› *assms*(*1*,*3*) *lms-locations-invD*
by *fastforce*
qed

lemma *lms-val-location*:
assumes *lms-locations-inv* α T B SA
and *lms-unchanged-inv* α T B $SA0$ SA
and *strict-mono* α
and *length* SA = *length* T
and $\forall\, i <$ *length* T. $SA0\ !\ i$ = *length* T
and $i <$ *length* SA
and $SA\ !\ i <$ *length* T
shows $\exists\, b \leq \alpha$ (*Max* (*set* T)). $B\ !\ b \leq i \wedge i <$ *bucket-end* α T b
proof −
from *assms*
have $i <$ *length* T
by *simp*
with *index-in-bucket-interval-gen*[*OF* - *assms*(*3*)]
obtain b **where**
$b \leq \alpha$ (*Max* (*set* T))
bucket-start α T $b \leq i$
$i <$ *bucket-end* α T b
by *blast*
moreover
have $B\ !\ b \leq i$
proof (*rule ccontr*)
assume $\neg B\ !\ b \leq i$
hence $i < B\ !\ b$
by *simp*
with *lms-unchanged-invD*[*OF assms*(*2*) ‹$b \leq \alpha$ (*Max* (*set* T))› ‹*bucket-start* α T $b \leq i$›]
have $SA\ !\ i = SA0\ !\ i$
by *blast*
with *assms*(*5*) ‹$i <$ *length* T›
have $SA\ !\ i$ = *length* T
by *auto*
with *assms*(*7*)
show *False*
by *auto*
qed
ultimately
show *?thesis*
by *auto*
qed

lemma *lms-val-imp-abs-is-lms*:
  assumes *lms-locations-inv* α *T B SA*
  and *lms-unchanged-inv* α *T B SA0 SA*
  and *strict-mono* α
  and *length SA* = *length T*
  and $\forall$ *i* < *length T*. *SA0* ! *i* = *length T*
  and *i* < *length SA*
  and *SA* ! *i* < *length T*
shows *abs-is-lms T* (*SA* ! *i*)
proof −
  from *lms-val-location*[*OF assms*(*1*−*7*)]
  obtain *b* **where**
    b ≤ α (*Max* (*set T*))
    B ! *b* ≤ *i*
    i < *bucket-end* α *T b*
    by *blast*
  with *lms-locations-invD*[*OF assms*(*1*)]
  have *SA* ! *i* ∈ *lms-bucket* α *T b*
    by *blast*
  then show *abs-is-lms T* (*SA* ! *i*)
    using *lms-bucket-def* **by** *blast*
qed

lemma *lms-lms-prefix-sorted*:
  assumes *lms-bucket-ptr-inv* α *T B SA*
  and *lms-locations-inv* α *T B SA*
  and *lms-unchanged-inv* α *T B SA0 SA*
  and *strict-mono* α
  and *length SA* = *length T*
  and $\forall$ *i* < *length T*. *SA0* ! *i* = *length T*
  and *set LMS* = {*i*. *abs-is-lms T i*}
shows *ordlistns.sorted* (*map* (*lms-prefix T*) (*filter* (λ*x*. *x* < *length T*) *SA*))
proof (*intro sorted-wrt-mapI*)
  fix *i j*
  let *?fs* = *filter* (λ*x*. *x* < *length T*) *SA*
  let *?goal* = *list-less-eq-ns* (*lms-prefix T* (*?fs* ! *i*)) (*lms-prefix T* (*?fs* ! *j*))
  assume *i* < *j j* < *length ?fs*

  from *filter-nth-relative-2*[*OF* ‹*j* < *length ?fs*› ‹*i* < *j*›]
  obtain *i′ j′* **where**
    j′ < *length SA*
    i′ < *j′*
    ?fs ! *j* = *SA* ! *j′*
    ?fs ! *i* = *SA* ! *i′*
    by *blast*
  hence *i′* < *length SA*
    by *linarith*

  have *SA* ! *i′* < *length T*

by (*metis* ‹*?fs* ! *i* = *SA* ! *i*′› ‹*i* < *j*› ‹*j* < *length ?fs*› *filter-set member-filter nth-mem order.strict-trans*)
with *lms-val-location*[*OF assms*(*2*−*6*) ‹*i*′ < *length SA*›]
obtain *b* **where**
b ≤ *α* (*Max* (*set T*))
B ! *b* ≤ *i*′
i′ < *bucket-end α T b*
by *blast*
with *lms-locations-invD*[*OF assms*(*2*)]
have *SA* ! *i*′ ∈ *lms-bucket α T b*
by *blast*
hence *α* (*T* ! (*SA* ! *i*′)) = *b abs-is-lms T* (*SA* ! *i*′)
by (*simp add*: *bucket-def lms-bucket-def*)+

from *lms-lms-prefix*[*OF* ‹*abs-is-lms T* (*SA* ! *i*′)›] ‹*?fs* ! *i* = *SA* ! *i*′›
have *S1*: *lms-prefix T* (*?fs* ! *i*) = [*T* ! (*SA* ! *i*′)]
by *simp*

have *SA* ! *j*′ < *length T*
using ‹*?fs* ! *j* = *SA* ! *j*′› ‹*j* < *length ?fs*› *nth-mem* **by** *fastforce*
with *lms-val-location*[*OF assms*(*2*−*6*) ‹*j*′ < *length SA*›]
obtain *b*′ **where**
b′ ≤ *α* (*Max* (*set T*))
B ! *b*′ ≤ *j*′
j′ < *bucket-end α T b*′
by *blast*
with *lms-locations-invD*[*OF assms*(*2*)]
have *SA* ! *j*′ ∈ *lms-bucket α T b*′
by *blast*
hence *α* (*T* ! (*SA* ! *j*′)) = *b*′ *abs-is-lms T* (*SA* ! *j*′)
by (*simp add*: *bucket-def lms-bucket-def*)+

from *lms-lms-prefix*[*OF* ‹*abs-is-lms T* (*SA* ! *j*′)›] ‹*?fs* ! *j* = *SA* ! *j*′›
have *S2*: *lms-prefix T* (*?fs* ! *j*) = [*T* ! (*SA* ! *j*′)]
by *simp*

have *b* ≤ *b*′
proof (*rule ccontr*)
assume ¬*b* ≤ *b*′
hence *b*′ < *b*
by *simp*
hence *bucket-end α T b*′ ≤ *bucket-start α T b*
by (*simp add*: *less-bucket-end-le-start*)
hence *bucket-end α T b*′ ≤ *lms-bucket-start α T b*
by (*metis l-bucket-end-def l-bucket-end-le-lms-bucket-start le-add1 le-trans*)
with *lms-bucket-ptr-lower-bound*[*OF assms*(*1*) ‹*b* ≤ *α* (*Max* (*set T*))›]
have *bucket-end α T b*′ ≤ *B* ! *b*
by *linarith*
with ‹*j*′ < *bucket-end α T b*′› ‹*B* ! *b* ≤ *i*′› ‹*i*′ < *j*′›

show *False*
by *linarith*
qed
moreover
have $b < b' \Longrightarrow$ *?goal*
proof $-$
assume $b < b'$
with ‹α (T ! (SA ! i')) = b› ‹α (T ! (SA ! j')) = b'› *assms*(*4*)
have T ! (SA ! i') < T ! (SA ! j')
using *strict-mono-less* **by** *blast*
with *S1 S2*
show *?goal*
by (*simp add*: *list-less-eq-ns-cons*)
qed
moreover
have $b = b' \Longrightarrow$ *?goal*
proof $-$
assume $b = b'$
with ‹α (T ! (SA ! i')) = b› ‹α (T ! (SA ! j')) = b'› *assms*(*4*)
have T ! (SA ! i') = T ! (SA ! j')
by (*meson strict-mono-eq*)
with *S1 S2*
show *?goal*
by *simp*
qed
ultimately
show *?goal*
using *less-le* **by** *blast*
qed

lemma *lms-suffix-sorted*:
assumes *lms-bucket-ptr-inv* α *T B SA*
and *lms-locations-inv* α *T B SA*
and *lms-unchanged-inv* α *T B SA0 SA*
and *lms-sorted-inv T LMS SA*
and *strict-mono* α
and *length SA* = *length T*
and $\forall$ *i* < *length T*. *SA0* ! *i* = *length T*
and *set LMS* = {*i*. *abs-is-lms T i*}
and *ordlistns.sorted* (*map* (*suffix T*) (*rev LMS*))
shows *ordlistns.sorted* (*map* (*suffix T*) (*filter* (λx. *x* < *length T*) *SA*))
proof (*intro sorted-wrt-mapI*)
fix *i j*
let *?fs* = *filter* (λx. *x* < *length T*) *SA*
let *?goal* = *list-less-eq-ns* (*suffix T* (*?fs* ! *i*)) (*suffix T* (*?fs* ! *j*))
assume *i* < *j* *j* < *length ?fs*

from *filter-nth-relative-2*[*OF* ‹*j* < *length ?fs*› ‹*i* < *j*›]
obtain i' j' **where**

$j' <$ *length SA*
$i' < j'$
?fs ! $j =$ *SA* ! j'
?fs ! $i =$ *SA* ! i'
by *blast*
hence $i' <$ *length SA*
by *linarith*

have *SA* ! $i' <$ *length T*
by (*metis* ‹*?fs* ! $i =$ *SA* ! i'› ‹$i < j$› ‹$j <$ *length ?fs*› *filter-set member-filter*
nth-mem order.strict-trans)
with *lms-val-location*[*OF assms*(*2*,*3*,*5*−*7*) ‹$i' <$ *length SA*›]
obtain b **where**
$b \leq \alpha$ (*Max* (*set T*))
B ! $b \leq i'$
$i' <$ *bucket-end* α *T* b
by *blast*
with *lms-locations-invD*[*OF assms*(*2*)]
have *SA* ! $i' \in$ *lms-bucket* α *T* b
by *blast*
hence α (*T* ! (*SA* ! i')) $= b$ *abs-is-lms T* (*SA* ! i')
by (*simp add*: *bucket-def lms-bucket-def*)+
hence *SA* ! $i' \in$ *set LMS*
using *assms*(*8*)
by *blast*

have *SA* ! $j' <$ *length T*
using ‹*?fs* ! $j =$ *SA* ! j'› ‹$j <$ *length ?fs*› *nth-mem* **by** *fastforce*
with *lms-val-location*[*OF assms*(*2*,*3*,*5*−*7*) ‹$j' <$ *length SA*›]
obtain b' **where**
$b' \leq \alpha$ (*Max* (*set T*))
B ! $b' \leq j'$
$j' <$ *bucket-end* α *T* b'
by *blast*
with *lms-locations-invD*[*OF assms*(*2*)]
have *SA* ! $j' \in$ *lms-bucket* α *T* b'
by *blast*
hence α (*T* ! (*SA* ! j')) $= b'$ *abs-is-lms T* (*SA* ! j')
by (*simp add*: *bucket-def lms-bucket-def*)+
hence *SA* ! $j' \in$ *set LMS*
using *assms*(*8*)
by *blast*

have $b \leq b'$
proof (*rule ccontr*)
assume $\neg b \leq b'$
hence $b' < b$
by *simp*
hence *bucket-end* α *T* $b' \leq$ *bucket-start* α *T* b

by (*simp add*: *less-bucket-end-le-start*)
hence $bucket\text{-}end\ \alpha\ T\ b' \leq lms\text{-}bucket\text{-}start\ \alpha\ T\ b$
by (*metis l-bucket-end-def l-bucket-end-le-lms-bucket-start le-add1 le-trans*)
with $lms\text{-}bucket\text{-}ptr\text{-}lower\text{-}bound[OF\ assms(1)\ \langle b \leq \alpha\ (Max\ (set\ T))\rangle]$
have $bucket\text{-}end\ \alpha\ T\ b' \leq B\ !\ b$
by *linarith*
with $\langle j' < bucket\text{-}end\ \alpha\ T\ b'\rangle\ \langle B\ !\ b \leq i'\rangle\ \langle i' < j'\rangle$
show *False*
by *linarith*
qed
moreover
have $b < b' \Longrightarrow ?goal$
proof −
assume $b < b'$
with $\langle\alpha\ (T\ !\ (SA\ !\ i')) = b\rangle\ \langle\alpha\ (T\ !\ (SA\ !\ j')) = b'\rangle\ assms(5)$
have $T\ !\ (SA\ !\ i') < T\ !\ (SA\ !\ j')$
using *strict-mono-less* **by** *blast*
with $\langle ?fs\ !\ i = SA\ !\ i'\rangle\ \langle ?fs\ !\ j = SA\ !\ j'\rangle\ \langle SA\ !\ i' < length\ T\rangle\ \langle SA\ !\ j' < length\ T\rangle$
show *?goal*
by (*metis list-less-ns-cons-diff ordlistns.less-imp-le suffix-cons-ex*)
qed
moreover
have $b = b' \Longrightarrow ?goal$
proof −
assume $b = b'$
with $\langle\alpha\ (T\ !\ (SA\ !\ i')) = b\rangle\ \langle\alpha\ (T\ !\ (SA\ !\ j')) = b'\rangle\ assms(5)$
have $T\ !\ (SA\ !\ i') = T\ !\ (SA\ !\ j')$
by (*meson strict-mono-eq*)
with $lms\text{-}sorted\text{-}invD2[OF\ assms(4)\ \langle j' < length\ SA\rangle\ \langle i' < j'\rangle\ \langle SA\ !\ i' \in set\ LMS\rangle$
$\langle SA\ !\ j' \in set\ LMS\rangle]$
obtain $m\ n$ **where**
$n < length\ LMS$
$m < n$
$LMS\ !\ m = SA\ !\ j'$
$LMS\ !\ n = SA\ !\ i'$
by *blast*
hence $rev\ LMS\ !\ (length\ LMS - Suc\ m) = SA\ !\ j'\ rev\ LMS\ !\ (length\ LMS - Suc\ n) = SA\ !\ i'$
by (*metis length-rev order.strict-trans rev-nth rev-rev-ident*)+
moreover
from $\langle m < n\rangle\ \langle n < length\ LMS\rangle$
have $length\ LMS - Suc\ n \leq length\ LMS - Suc\ m$
by *linarith*
moreover
have $length\ LMS - Suc\ m < length\ (rev\ LMS)$
using $\langle n < length\ LMS\rangle$ **by** *auto*
ultimately

```
    have list-less-eq-ns (suffix T (SA ! i')) (suffix T (SA ! j'))
      using ordlistns.sorted-nth-mono[OF assms(9)]
      by fastforce
    with ‹?fs ! i = SA ! i'› ‹?fs ! j = SA ! j'›
    show ?goal
      by simp
  qed
  ultimately
  show ?goal
    using less-le by blast
qed

lemma next-index-outside:
  assumes b = α (T ! x)
  and     k = B ! b − Suc 0
  and     lms-distinct-inv T SA (x # LMS)
  and     lms-bucket-ptr-inv α T B SA
  and     strict-mono α
  and     ∀ a ∈ set (x # LMS). abs-is-lms T a
  and     b' ≤ α (Max (set T))
  and     b ≠ b'
shows k < bucket-start α T b' ∨ bucket-end α T b' ≤ k
proof −

  from lms-distinct-bucket-ptr-lower-bound[OF assms(1,3−6)]
  have lms-bucket-start α T b < B ! b .
  hence k < B ! b
    using assms(2) by linarith

  from ‹lms-bucket-start α T b < B ! b›
  have lms-bucket-start α T b ≤ k
    using assms(2) by linarith

  have x < length T
    by (simp add: assms(6) abs-is-lms-imp-less-length)
  hence b ≤ α (Max (set T))
    by (simp add: assms(1,5) strict-mono-leD)

  from assms(8)
  have b < b' ∨ b' < b
    by linarith
  moreover
  have b < b' ⟹ k < bucket-start α T b'
  proof −
    assume b < b'
    hence bucket-end α T b ≤ bucket-start α T b'
      by (simp add: less-bucket-end-le-start)
    with lms-bucket-ptr-upper-bound[OF assms(4) ‹b ≤ α (Max (set T))›]
    have B ! b ≤ bucket-start α T b'
```

by *linarith*
with ‹$k < B\ !\ b$›
show *?thesis*
by *linarith*
qed
moreover
have $b' < b \Longrightarrow$ *bucket-end* α T $b' \leq k$
proof $-$
assume $b' < b$
hence *bucket-end* α T $b' \leq$ *bucket-start* α T b
by (*simp add*: *less-bucket-end-le-start*)
hence *bucket-end* α T $b' \leq$ *lms-bucket-start* α T b
by (*metis l-bucket-end-def l-bucket-end-le-lms-bucket-start le-add1 le-trans*)
with ‹*lms-bucket-start* α T $b \leq k$›
show *?thesis*
using *le-trans* **by** *blast*
qed
ultimately show *?thesis*
by *blast*
qed

62.4 Establishment and Maintenance Steps

62.4.1 Distinctness

lemma *lms-distinct-inv-established*:
assumes *distinct LMS*
and $\forall i <$ *length SA*. *SA* ! $i =$ *length T*
shows *lms-distinct-inv T SA LMS*
proof $-$
from *assms*(*2*)
have *filter* ($\lambda x.\ x <$ *length T*) *SA* = []
by (*metis filter-False in-set-conv-nth nat-neq-iff*)
then show *?thesis*
unfolding *lms-distinct-inv-def*
using *distinct-append assms*(*1*)
by *simp*
qed

lemma *lms-distinct-inv-maintained-step*:
assumes *lms-distinct-inv T SA* (x # *LMS*)
shows *lms-distinct-inv T* (*SA*[$k := x$]) *LMS*
unfolding *lms-distinct-inv-def*
proof(*intro distinct-conv-nth*[*THEN iffD2*] *allI impI*)
let *?xs* = *filter* ($\lambda x.\ x <$ *length T*) *SA* **and**
?ys = *filter* ($\lambda x.\ x <$ *length T*) (*SA*[$k := x$])
fix i j
assume $i \neq j$ $i <$ *length* (*filter* ($\lambda x.\ x <$ *length T*) (*SA*[$k := x$]) @ *LMS*)
$j <$ *length* (*filter* ($\lambda x.\ x <$ *length T*) (*SA*[$k := x$]) @ *LMS*)
hence $i <$ *length ?ys* + *length LMS* $j <$ *length ?ys* + *length LMS*

by *simp-all*

let $?goal = (?ys\ @\ LMS)\ !\ i \neq (?ys\ @\ LMS)\ !\ j$

from *assms*(*1*) *distinct-append*
have *distinct LMS distinct ?xs* $x \notin$ *set ?xs* $x \notin$ *set LMS set ?xs* $\cap$ *set LMS* $= \{\}$
by (*simp add*: *lms-distinct-inv-def*)+

from ‹*distinct LMS*› ‹$i <$ *length ?ys* $+$ *length LMS*› ‹$j <$ *length ?ys* $+$ *length LMS*› ‹$i \neq j$›
have *R1*: $[\![$*length ?ys* $\leq i$; *length ?ys* $\leq j]\!] \Longrightarrow$ *?goal*
by (*metis le-Suc-ex nat-add-left-cancel-less nth-append-length-plus nth-eq-iff-index-eq*)

have *set ?ys* $\subseteq \{x\} \cup$ *set ?xs*
by (*meson filter-nth-update-subset*)

have *R2*:
$\bigwedge i\ j.\ [\![i <$ *length ?ys*; *length ?ys* $\leq j$; $j <$ *length ?ys* $+$ *length LMS*$]\!] \Longrightarrow$
$(?ys\ @\ LMS)\ !\ i \neq (?ys\ @\ LMS)\ !\ j$
proof $-$
fix *i j*
assume $i <$ *length ?ys length ?ys* $\leq j$ $j <$ *length ?ys* $+$ *length LMS*
hence *?ys* ! $i \in \{x\} \cup$ *set ?xs*
using ‹*set ?ys* $\subseteq \{x\} \cup$ *set ?xs*› *nth-mem* **by** *blast*
hence $(?ys\ @\ LMS)\ !\ i \in \{x\} \cup$ *set ?xs*
by (*simp add*: ‹$i <$ *length ?ys*› *nth-append*)
moreover
from ‹*length ?ys* $\leq j$› ‹$j <$ *length ?ys* $+$ *length LMS*›
have $(?ys\ @\ LMS)\ !\ j \in$ *set LMS*
by (*metis add.commute in-set-conv-nth leD less-diff-conv2 nth-append*)
moreover
from ‹*set ?xs* $\cap$ *set LMS* $= \{\}$› ‹$x \notin$ *set LMS*›
have $(\{x\} \cup$ *set ?xs*$) \cap$ *set LMS* $= \{\}$
by *blast*
ultimately
show $(?ys\ @\ LMS)\ !\ i \neq (?ys\ @\ LMS)\ !\ j$
by (*metis disjoint-iff-not-equal*)
qed

have *R3*: $[\![i <$ *length ?ys*; $j <$ *length ?ys*$]\!] \Longrightarrow$ *?goal*
proof $-$
assume $i <$ *length ?ys* $j <$ *length ?ys*
with *filter-nth-relative-neq-2*[*OF* - - ‹$i \neq j$›]
obtain i' j' **where**
$i' <$ *length* $(SA[k := x])$
$j' <$ *length* $(SA[k := x])$
$(SA[k := x])\ !\ i' = ?ys\ !\ i$
$(SA[k := x])\ !\ j' = ?ys\ !\ j$
$i' \neq j'$

by *blast*

have $?ys\ !\ i < length\ T$
  using ‹$i < length\ ?ys$› *nth-mem* **by** *fastforce*
hence $(SA[k := x])\ !\ i' < length\ T$
  using ‹$(SA[k := x])\ !\ i' = ?ys\ !\ i$› **by** *simp*

have $?ys\ !\ j < length\ T$
  using ‹$j < length\ ?ys$› *nth-mem* **by** *fastforce*
hence $(SA[k := x])\ !\ j' < length\ T$
  using ‹$(SA[k := x])\ !\ j' = ?ys\ !\ j$› **by** *simp*

have *R4*:
  $\bigwedge i\ j.\ [\![i \neq k;\ j = k;\ i < length\ (SA[k := x]);\ j < length\ (SA[k := x]);$
    $(SA[k := x])\ !\ i < length\ T]\!] \Longrightarrow$
    $(SA[k := x])\ !\ i \neq (SA[k := x])\ !\ j$
proof –
  fix $i\ j$
  assume $i \neq k\ j = k\ i < length\ (SA[k := x])\ j < length\ (SA[k := x])$
    $(SA[k := x])\ !\ i < length\ T$

  from ‹$j = k$› ‹$j < length\ (SA[k := x])$›
  have $(SA[k := x])\ !\ j = x$
    by *simp*
  moreover
  from ‹$i \neq k$› ‹$i < length\ (SA[k := x])$›
  have $(SA[k := x])\ !\ i = SA\ !\ i$
    by *simp*
  with ‹$(SA[k := x])\ !\ i < length\ T$›
  have $SA\ !\ i < length\ T$
    by *simp*
  with *filter-nth-1* [*of i SA* $\lambda x.\ x < length\ T$] ‹$i < length\ (SA[k := x])$›
  obtain i' **where**
    $i' < length\ ?xs$
    $?xs\ !\ i' = SA\ !\ i$
    by *auto*
  with ‹$(SA[k := x])\ !\ i = SA\ !\ i$›
  have $(SA[k := x])\ !\ i \in set\ ?xs$
    using *nth-mem* **by** *fastforce*
  ultimately
  show $(SA[k := x])\ !\ i \neq (SA[k := x])\ !\ j$
    using ‹$x \notin set\ ?xs$› **by** *auto*
qed

have $[\![i' \neq k;\ j' \neq k]\!] \Longrightarrow (SA[k := x])\ !\ i' \neq (SA[k := x])\ !\ j'$
proof –
  assume $i' \neq k\ j' \neq k$
  with ‹$(SA[k := x])\ !\ i' = ?ys\ !\ i$› ‹$(SA[k := x])\ !\ j' = ?ys\ !\ j$›
  have $?ys\ !\ i = SA\ !\ i'\ ?ys\ !\ j = SA\ !\ j'$

```
      by auto
    with ‹?ys ! i < length T› ‹?ys ! j < length T› ‹i' < length (SA[k := x])›
         ‹j' < length (SA[k := x])›
         filter-nth-relative-neq-1[of i' SA λx. x < length T j', OF - - - - ‹i' ≠ j'›]
    obtain i'' j'' where
      i'' < length ?xs
      j'' < length ?xs
      ?xs ! i'' = SA ! i'
      ?xs ! j'' = SA ! j'
      i'' ≠ j''
      by auto
    with ‹distinct ?xs›
    have SA ! i' ≠ SA ! j'
      by (metis distinct-Ex1 in-set-conv-nth)
    then show ?thesis
      using ‹SA[k := x] ! i' = ?ys ! i› ‹?ys ! i = SA ! i'› ‹j' ≠ k› by auto
  qed
  moreover
  from ‹i' ≠ j'›
  have ⟦i' = k; j' = k⟧ ⟹ False
    by blast
  ultimately
  have (SA[k := x]) ! i' ≠ (SA[k := x]) ! j'
    using R4[of i' j', OF - - ‹i' < length (SA[k := x])› ‹j' < length (SA[k := x])›
            ‹(SA[k := x]) ! i' < length T›]
          R4[of j' i', OF - - ‹j' < length (SA[k := x])› ‹i' < length (SA[k := x])›
            ‹(SA[k := x]) ! j' < length T›]
    by metis
  with ‹(SA[k := x]) ! i' = ?ys ! i› ‹i < length ?ys›
       ‹(SA[k := x]) ! j' = ?ys ! j› ‹j < length ?ys›
  show ?goal
    by (simp add: nth-append)
qed

from R1 R2[of i j, OF - - ‹j < length ?ys + length LMS›]
     R2[of j i, OF - - ‹i < length ?ys + length LMS›] R3
show ?goal
  by presburger
qed

lemma lms-distinct-inv-maintained:
  assumes lms-distinct-inv T SA LMS
  shows lms-distinct-inv T (abs-bucket-insert α T B SA LMS) []
  using assms
proof (induct rule: abs-bucket-insert.induct[of - α T B SA LMS])
  case (1 α T uu SA)
  then show ?case
    by simp
next
```

case (*2 α T B SA x xs*)
note *IH = this*
let *?b = α (T ! x)*
let *?k = B ! ?b − Suc 0*
from *IH(1)[OF - - - - lms-distinct-inv-maintained-step[OF IH(2), of ?k], of ?b ?k B[?b := ?k]]*
show *?case*
by (*metis (full-types) One-nat-def abs-bucket-insert.simps(2)*)
qed

lemma *abs-bucket-insert-lms-distinct-inv*:
assumes *distinct LMS*
and ∀ *i < length SA. SA ! i = length T*
shows *lms-distinct-inv T (abs-bucket-insert α T B SA LMS) []*
using *assms lms-distinct-inv-maintained lms-distinct-inv-established*
by *blast*

62.4.2 Bucket Ptr

lemma *lms-bucket-ptr-inv-established*:
assumes *lms-bucket-init α T B*
and ∀ *i < length SA. SA ! i = length T*
shows *lms-bucket-ptr-inv α T B SA*
unfolding *lms-bucket-ptr-inv-def*
proof (*intro allI impI*)
fix *b*
assume *b ≤ α (Max (set T))*
with *lms-bucket-initD[OF assms(1)]*
have *B ! b = bucket-end α T b*
by *simp*
moreover
from *assms(2)*
have ∀ *i ∈ set SA. ¬abs-is-lms T i*
by (*metis in-set-conv-nth abs-is-lms-imp-less-length less-not-refl2*)
hence *cur-lms-types α T SA b = {}*
by (*simp add: cur-lms-types-def lms-bucket-def*)
hence *num-lms-types α T SA b = 0*
by (*simp add: num-lms-types-def*)
ultimately
show *B ! b + num-lms-types α T SA b = bucket-end α T b*
by *simp*
qed

lemma *lms-bucket-ptr-inv-maintained-step*:
assumes *b = α (T ! x)*
and *k = B ! b − Suc 0*
and *lms-distinct-inv T SA (x # LMS)*
and *lms-bucket-ptr-inv α T B SA*
and *lms-unknowns-inv α T B SA*

and *strict-mono* α
and α (*Max* (*set* *T*)) < *length* *B*
and *length* *SA* = *length* *T*
and $\forall$ *a* $\in$ *set* (*x* # *LMS*). *abs-is-lms* *T* *a*
shows *lms-bucket-ptr-inv* α *T* (*B*[*b* := *k*]) (*SA*[*k* := *x*])
unfolding *lms-bucket-ptr-inv-def*
proof (*intro allI impI*)
fix b'
assume $b' \leq \alpha$ (*Max* (*set* *T*))

let *?goal* = *B*[*b* := *k*] ! b' + *num-lms-types* α *T* (*SA*[*k* := *x*]) b' = *bucket-end* α *T* b'

from *assms*(*1*,*9*)
have *x* $\in$ *lms-bucket* α *T* *b*
by (*simp add*: *bucket-def abs-is-lms-imp-less-length lms-bucket-def*)

from *lms-next-insert-at-unknown*[*OF assms*(*1*−*6*,*8*,*9*)]
have *k* < *length* *SA* *SA* ! *k* = *length* *T*
by *blast*+

have $b' \neq b \Longrightarrow$ *?goal*
proof −
assume $b' \neq b$
with ‹*x* $\in$ *lms-bucket* α *T* *b*›
have *x* $\notin$ *lms-bucket* α *T* b'
by (*simp add*: *bucket-def lms-bucket-def*)
with *cur-lms-subset-lms-bucket*
have *x* $\notin$ *cur-lms-types* α *T* *SA* b'
by *blast*

have *cur-lms-types* α *T* (*SA*[*k* := *x*]) b' = *cur-lms-types* α *T* *SA* b'
unfolding *cur-lms-types-def*
proof (*intro equalityI subsetI*)
fix *y*
assume *y* $\in$ {*i* |*i*. *i* $\in$ *set* (*SA*[*k* := *x*]) $\wedge$ *i* $\in$ *lms-bucket* α *T* b'}
hence *y* $\in$ *set* (*SA*[*k* := *x*]) *y* $\in$ *lms-bucket* α *T* b'
by *simp-all*
with ‹*x* $\notin$ *lms-bucket* α *T* b'›
have *y* $\in$ *set* *SA*
by (*metis in-set-conv-nth length-list-update nth-list-update-eq nth-list-update-neq*)
then show *y* $\in$ {*i* |*i*. *i* $\in$ *set* *SA* $\wedge$ *i* $\in$ *lms-bucket* α *T* b'}
by (*simp add*: ‹*y* $\in$ *lms-bucket* α *T* b'›)
next
fix *y*
assume *y* $\in$ {*i* |*i*. *i* $\in$ *set* *SA* $\wedge$ *i* $\in$ *lms-bucket* α *T* b'}
hence *y* $\in$ *set* *SA* *y* $\in$ *lms-bucket* α *T* b'
by *simp-all*
with ‹*x* $\notin$ *lms-bucket* α *T* b'› ‹*SA* ! *k* = *length* *T*›

    have $y \in set\ (SA[k := x])$
     using *in-set-list-update abs-is-lms-imp-less-length lms-bucket-def* **by** *fastforce*
    then show $y \in \{i\ |i.\ i \in set\ (SA[k := x]) \wedge i \in lms\text{-}bucket\ \alpha\ T\ b'\}$
      using ‹$y \in lms\text{-}bucket\ \alpha\ T\ b'$› **by** *blast*
  qed
  hence $num\text{-}lms\text{-}types\ \alpha\ T\ (SA[k := x])\ b' = num\text{-}lms\text{-}types\ \alpha\ T\ SA\ b'$
    by (*simp add*: *num-lms-types-def*)
  with *lms-bucket-ptr-invD*[*OF assms*(*4*) ‹$b' \leq \alpha\ (Max\ (set\ T))$›] ‹$b' \neq b$›
  show *?thesis*
    by *simp*
qed
moreover
have $b' = b \Longrightarrow$ *?goal*
proof −
  assume $b' = b$
  with ‹$b' \leq \alpha\ (Max\ (set\ T))$›
  have $b \leq \alpha\ (Max\ (set\ T))$
    by *simp*

  from *assms*(*3*,*9*)
  have $x \notin set\ SA$
    by (*simp add*: *abs-is-lms-imp-less-length lms-distinct-inv-def*)

  from ‹$k < length\ SA$›
  have $x \in set\ (SA[k := x])$
    by (*simp add*: *set-update-memI*)

  have $b < length\ B$
    using ‹$b \leq \alpha\ (Max\ (set\ T))$› *assms*(*7*) *dual-order.strict-trans2* **by** *blast*
  hence $B[b := k]\ !\ b = k$
    by *simp*

  have *finite* (*cur-lms-types* α *T SA b*)
    by (*meson List.finite-set cur-lms-subset-SA finite-subset*)
  moreover
  from ‹$x \notin set\ SA$›
  have $cur\text{-}lms\text{-}types\ \alpha\ T\ SA\ b - \{x\} = cur\text{-}lms\text{-}types\ \alpha\ T\ SA\ b$
    using *cur-lms-subset-SA* **by** *fastforce*
  moreover
  have $insert\ x\ (cur\text{-}lms\text{-}types\ \alpha\ T\ SA\ b) = cur\text{-}lms\text{-}types\ \alpha\ T\ (SA[k := x])\ b$
    unfolding *cur-lms-types-def*
  proof (*intro equalityI subsetI*)
    fix y
    assume $y \in insert\ x\ \{i\ |i.\ i \in set\ SA \wedge i \in lms\text{-}bucket\ \alpha\ T\ b\}$
    hence $y = x \vee y \in set\ SA \wedge y \in lms\text{-}bucket\ \alpha\ T\ b$
      by *blast*
    with ‹$SA\ !\ k = length\ T$› ‹$x \in lms\text{-}bucket\ \alpha\ T\ b$› ‹$x \in set\ (SA[k := x])$›
    have $y \in set\ (SA[k := x]) \wedge y \in lms\text{-}bucket\ \alpha\ T\ b$
        by (*metis* (*no-types*, *lifting*) *in-set-list-update abs-is-lms-imp-less-length*

less-irrefl-nat
lms-bucket-def mem-Collect-eq)
then show $y \in \{i \mid i.\ i \in set\ (SA[k := x]) \wedge i \in lms\text{-}bucket\ \alpha\ T\ b\}$
by *blast*
next
fix y
assume $y \in \{i \mid i.\ i \in set\ (SA[k := x]) \wedge i \in lms\text{-}bucket\ \alpha\ T\ b\}$
hence $y \in set\ (SA[k := x])$ $y \in lms\text{-}bucket\ \alpha\ T\ b$
by *simp-all*
moreover
have $y \in set\ SA \Longrightarrow y \in insert\ x\ \{i \mid i.\ i \in set\ SA \wedge i \in lms\text{-}bucket\ \alpha\ T\ b\}$
using *calculation*(*2*) **by** *blast*
moreover
from ‹$k < length\ SA \wedge SA\ !\ k = length\ T$›
have $y \notin set\ SA \Longrightarrow y \in insert\ x\ \{i \mid i.\ i \in set\ SA \wedge i \in lms\text{-}bucket\ \alpha\ T\ b\}$
by (*metis* (*no-types*, *lifting*) *calculation*(*1*) *in-set-conv-nth insert-iff length-list-update nth-list-update*)
ultimately show $y \in insert\ x\ \{i \mid i.\ i \in set\ SA \wedge i \in lms\text{-}bucket\ \alpha\ T\ b\}$
by *blast*
qed
ultimately
have $num\text{-}lms\text{-}types\ \alpha\ T\ (SA[k := x])\ b = Suc\ (num\text{-}lms\text{-}types\ \alpha\ T\ SA\ b)$
by (*metis num-lms-types-def card.insert-remove*)
with ‹$b' = b$› ‹$B[b := k]\ !\ b = k$› *assms*(*2*)
have $B[b := k]\ !\ b' + num\text{-}lms\text{-}types\ \alpha\ T\ (SA[k := x])\ b'$
$= B\ !\ b - Suc\ 0 + Suc\ (num\text{-}lms\text{-}types\ \alpha\ T\ SA\ b)$
by *simp*
hence $B[b := k]\ !\ b' + num\text{-}lms\text{-}types\ \alpha\ T\ (SA[k := x])\ b' = B\ !\ b + num\text{-}lms\text{-}types\ \alpha\ T\ SA\ b$
using *add-Suc-shift assms less-nat-zero-code lms-distinct-bucket-ptr-lower-bound neq0-conv*
by *fastforce*
with ‹$b' = b$›
have $B[b := k]\ !\ b' + num\text{-}lms\text{-}types\ \alpha\ T\ (SA[k := x])\ b' = B\ !\ b' + num\text{-}lms\text{-}types\ \alpha\ T\ SA\ b'$
by *simp*
with *lms-bucket-ptr-invD*[*OF assms*(*4*) ‹$b' \leq \alpha\ (Max\ (set\ T))$›]
show *?thesis*
by *simp*
qed
ultimately
show *?goal*
by *blast*
qed

62.4.3 Unknowns

lemma *lms-unknowns-inv-established*:
assumes *lms-bucket-init* $\alpha\ T\ B$

and $\forall\, i < \mathit{length}\ \mathit{SA}.\ \mathit{SA}\ !\ i = \mathit{length}\ T$
and $\mathit{length}\ \mathit{SA} = \mathit{length}\ T$
shows *lms-unknowns-inv* α *T B SA*
unfolding *lms-unknowns-inv-def*
proof (*intro allI impI*; *elim conjE*)
fix *b i*
assume $b \le \alpha\ (\mathit{Max}\ (\mathit{set}\ T))$ *lms-bucket-start* $\alpha\ T\ b \le i$ $i < B\ !\ b$
with *lms-bucket-initD*[*OF assms*(*1*)]
have $B\ !\ b = \mathit{bucket\text{-}end}\ \alpha\ T\ b$
by *simp*
with ‹$i < B\ !\ b$›
have $i < \mathit{length}\ T$
by (*simp add*: *bucket-end-le-length less-le-trans*)
with *assms*(*3*)
have $i < \mathit{length}\ \mathit{SA}$
by *simp*
with *assms*(*2*)
show $\mathit{SA}\ !\ i = \mathit{length}\ T$
by *simp*
qed

lemma *lms-unknowns-inv-maintained-step*:
assumes $b = \alpha\ (T\ !\ x)$
and $k = B\ !\ b - \mathit{Suc}\ 0$
and *lms-distinct-inv T SA* (x # *LMS*)
and *lms-bucket-ptr-inv* α *T B SA*
and *lms-unknowns-inv* α *T B SA*
and *strict-mono* α
and $\alpha\ (\mathit{Max}\ (\mathit{set}\ T)) < \mathit{length}\ B$
and $\forall\, a \in \mathit{set}\ (x\ \#\ \mathit{LMS}).\ \mathit{abs\text{-}is\text{-}lms}\ T\ a$
shows *lms-unknowns-inv* α T $(B[b := k])$ $(\mathit{SA}[k := x])$
unfolding *lms-unknowns-inv-def*
proof (*intro allI impI*; *elim conjE*)
fix b' *i*
assume $b' \le \alpha\ (\mathit{Max}\ (\mathit{set}\ T))$ *lms-bucket-start* $\alpha\ T\ b' \le i$ $i < B[b := k]\ !\ b'$

let $?goal = \mathit{SA}[k := x]\ !\ i = \mathit{length}\ T$

have $b' \ne b \Longrightarrow\ ?goal$
proof $-$
assume $b' \ne b$
with ‹$i < B[b := k]\ !\ b'$›
have $i < B\ !\ b'$
by *simp*
with *lms-unknowns-invD*[*OF assms*(*5*) ‹$b' \le \alpha\ (\mathit{Max}\ (\mathit{set}\ T))$› ‹*lms-bucket-start* $\alpha\ T\ b' \le i$›]
have $\mathit{SA}\ !\ i = \mathit{length}\ T$
by *simp*

from ‹$b' \neq b$›
have $b' < b \vee b < b'$
by *auto*
then show *?thesis*
proof
assume $b' < b$
from *lms-distinct-bucket-ptr-lower-bound*[*OF assms*(*1*,*3*,*4*,*6*,*8*)]
have *lms-bucket-start* α T $b < B$! b **.**
hence *bucket-start* α T $b < B$! b
by (*metis dual-order.strict-trans2 l-bucket-end-def l-bucket-end-le-lms-bucket-start le-add1*)
moreover
from *lms-bucket-ptr-upper-bound*[*OF assms*(*4*) ‹$b' \leq \alpha$ (*Max* (*set* T))›]
have B ! $b' \leq$ *bucket-end* α T b' **.**
ultimately
have B ! $b' < B$! b
by (*meson* ‹$b' < b$› *dual-order.strict-trans2 less-bucket-end-le-start*)
hence $i \neq k$
using *assms*(*2*,*3*) ‹$i < B$! b'›
by *linarith*
with ‹*SA* ! i = *length* T›
show *?thesis*
by *auto*
next
assume $b < b'$
from ‹*lms-bucket-start* α T $b' \leq i$›
have *bucket-start* α T $b' \leq i$
by (*metis l-bucket-end-def l-bucket-end-le-lms-bucket-start le-add1 le-trans*)
moreover
from *lms-bucket-ptr-upper-bound*[*OF assms*(*4*), *of b*] *assms*(*7*)
have B ! $b \leq$ *bucket-end* α T b
using ‹$b < b'$› ‹$b' \leq \alpha$ (*Max* (*set* T))› **by** *linarith*
hence $k <$ *bucket-end* α T b
using *assms lms-distinct-bucket-ptr-lower-bound* **by** *fastforce*
ultimately
have $i \neq k$
by (*meson* ‹$b < b'$› *leD le-trans less-bucket-end-le-start*)
with ‹*SA* ! i = *length* T›
show *?thesis*
by *auto*
qed
qed
moreover
have $b' = b \Longrightarrow$ *?goal*
proof −
assume $b' = b$
with ‹$b' \leq \alpha$ (*Max* (*set* T))› ‹*lms-bucket-start* α T $b' \leq i$› ‹$i < B[b := k]$! b'›
have $b \leq \alpha$ (*Max* (*set* T)) *lms-bucket-start* α T $b \leq i$ $i < B[b := k]$! b
by *simp-all*

hence $i < B \mathbin{!} b - \mathit{Suc}\ 0$
using *assms* **by** *auto*
with *lms-unknowns-invD*[*OF assms*(*5*) ‹$b \le \alpha$ (*Max* (*set T*))› ‹*lms-bucket-start* α *T* $b \le i$›]
have $SA \mathbin{!} i = \mathit{length}\ T$
by *linarith*
with ‹$i < B \mathbin{!} b - \mathit{Suc}\ 0$› *assms*(*2*)
show *?thesis*
by *simp*
qed
ultimately
show *?goal*
by *blast*
qed

62.4.4 Locations

lemma *lms-locations-inv-established*:
assumes *lms-bucket-init* α *T B*
shows *lms-locations-inv* α *T B SA*
unfolding *lms-locations-inv-def*
proof (*intro allI impI*; *elim conjE*)
fix *b i*
assume $b \le \alpha$ (*Max* (*set T*)) $B \mathbin{!} b \le i$ $i <$ *bucket-end* α *T b*
with *lms-bucket-initD*[*OF assms*(*1*), *of b*]
have *False*
by *linarith*
then show $SA \mathbin{!} i \in$ *lms-bucket* α *T b*
by *blast*
qed

lemma *lms-locations-inv-maintained-step*:
assumes $b = \alpha$ ($T \mathbin{!} x$)
and $k = (B \mathbin{!} b) - \mathit{Suc}\ 0$
and *lms-distinct-inv T SA* (x # *LMS*)
and *lms-bucket-ptr-inv* α *T B SA*
and *lms-locations-inv* α *T B SA*
and *strict-mono* α
and α (*Max* (*set T*)) $<$ *length B*
and *length SA* $=$ *length T*
and $\forall a \in$ *set* (x # *LMS*). *abs-is-lms T a*
shows *lms-locations-inv* α *T* ($B[b := k]$) ($SA[k := x]$)
unfolding *lms-locations-inv-def*
proof (*intro allI impI*; *elim conjE*)
fix b' *i*
assume $b' \le \alpha$ (*Max* (*set T*)) $B[b := k] \mathbin{!} b' \le i$ $i <$ *bucket-end* α *T* b'

let *?goal* $= SA[k := x] \mathbin{!} i \in$ *lms-bucket* α *T* b'

have *lms-bucket-start* α T b $<$ B ! b
  using *assms lms-distinct-bucket-ptr-lower-bound* **by** *blast*

have $b' \neq b \Longrightarrow$ *?goal*
proof $-$
  assume $b' \neq b$
  with ‹$B[b := k]$! $b' \leq i$›
  have B ! $b' \leq i$
    by *simp*

  from ‹$b' \neq b$›
  have $b' < b \vee b < b'$
    by *auto*
  then show *?thesis*
  proof
    assume $b' < b$
    from ‹*lms-bucket-start* α T b $<$ B ! b›
    have *bucket-start* α T b $<$ B ! b
    by (*metis dual-order.strict-trans2 l-bucket-end-def l-bucket-end-le-lms-bucket-start*
          le-add1)
    with ‹$i <$ *bucket-end* α T b'› ‹$b' < b$›
    have $i \neq k$
      using *assms*(*2*,*3*)
      by (*metis Suc-lessI Suc-pred dual-order.strict-trans1 less-bucket-end-le-start*
          less-nat-zero-code not-less-iff-gr-or-eq)
    hence $SA[k := x]$! i $=$ SA ! i
      by *auto*
    with *lms-locations-invD*[*OF assms*(*5*) ‹$b' \leq \alpha$ (*Max* (*set* T))› ‹B ! $b' \leq i$›
                ‹$i <$ *bucket-end* α T b'›]
    show *?thesis*
      by *simp*
  next
    assume $b < b'$
    from *lms-bucket-ptr-lower-bound*[*OF assms*(*4*) ‹$b' \leq \alpha$ (*Max* (*set* T))›]
    have *lms-bucket-start* α T $b' \leq B$! b' **.**
    hence *bucket-start* α T $b' \leq B$! b'
      by (*metis l-bucket-end-def l-bucket-end-le-lms-bucket-start le-add1 le-trans*)
    hence *bucket-start* α T $b' \leq i$
      using ‹B ! $b' \leq i$› *le-trans* **by** *blast*
    moreover
    from *lms-bucket-ptr-upper-bound*[*OF assms*(*4*), *of* b] *assms*(*7*)
    have B ! $b \leq$ *bucket-end* α T b
      using ‹$b < b'$› ‹$b' \leq \alpha$ (*Max* (*set* T))› **by** *linarith*
    with ‹*lms-bucket-start* α T b $<$ B ! b›
    have $k <$ *bucket-end* α T b
      using *assms*(*2*) **by** *linarith*
    ultimately
    have $i \neq k$
      by (*meson* ‹$b < b'$› *leD le-less-trans less-bucket-end-le-start*)

hence $SA[k := x] ! i = SA ! i$
by *simp*
with *lms-locations-invD*[*OF assms*(*5*)‹$b' \leq \alpha$ (*Max* (*set T*))› ‹$B ! b' \leq i$›
‹$i <$ *bucket-end* α *T* b'›]
show *?thesis*
by *simp*
qed
qed
moreover
have $b' = b \Longrightarrow$ *?goal*
proof −
assume $b' = b$
with ‹$b' \leq \alpha$ (*Max* (*set T*))› ‹$B[b := k] ! b' \leq i$› ‹$i <$ *bucket-end* α *T* b'›
have $b \leq \alpha$ (*Max* (*set T*)) $B[b := k] ! b \leq i$ $i <$ *bucket-end* α *T* b
by *simp-all*
hence $k \leq i$
by (*simp add*: *assms*(*7*) *order.strict-trans1*)
hence $k = i \vee B ! b \leq i$
using *assms*(*2*) **by** *linarith*
then show *?thesis*
proof
assume $k = i$
hence $SA[k := x] ! i = x$
using ‹$i <$ *bucket-end* α *T* b› *assms*(*8*) *bucket-end-le-length order.strict-trans2*
by *fastforce*
moreover
from *assms*(*1*,*9*)
have $x \in$ *lms-bucket* α *T* b
by (*simp add*: *bucket-def abs-is-lms-imp-less-length lms-bucket-def*)
ultimately
show *?thesis*
using ‹$b' = b$› **by** *simp*
next
assume $B ! b \leq i$
with *lms-locations-invD*[*OF assms*(*5*) ‹$b \leq \alpha$ (*Max* (*set T*))› - ‹$i <$ *bucket-end*
α *T* b›]
have $SA ! i \in$ *lms-bucket* α *T* b
by *blast*
moreover
from ‹$B ! b \leq i$› *assms*(*2*) ‹*lms-bucket-start* α *T* $b < B ! b$›
have $SA[k := x] ! i = SA ! i$
by *auto*
ultimately
show *?thesis*
using ‹$b' = b$› **by** *simp*
qed
qed
ultimately
show *?goal*

by *blast*
qed

62.4.5 Unchanged

lemma *lms-unchanged-inv-established*:
lms-unchanged-inv α *T B SA SA*
unfolding *lms-unchanged-inv-def*
by *blast*

lemma *lms-unchanged-inv-maintained-step*:
assumes $b = \alpha\ (T\ !\ x)$
and $k = (B\ !\ b) - Suc\ 0$
and *lms-distinct-inv T SA* $(x\ \#\ LMS)$
and *lms-bucket-ptr-inv* α *T B SA*
and *lms-unchanged-inv* α *T B SA0 SA*
and *strict-mono* α
and $\alpha\ (Max\ (set\ T)) < length\ B$
and $length\ SA = length\ T$
and $\forall\, a \in set\ (x\ \#\ LMS).\ abs\text{-}is\text{-}lms\ T\ a$
shows *lms-unchanged-inv* α $T\ (B[b := k])\ SA0\ (SA[k := x])$
unfolding *lms-unchanged-inv-def*
proof (*intro allI impI*; *elim conjE*)
fix $b'\ i$
assume $b' \leq \alpha\ (Max\ (set\ T))$ $bucket\text{-}start\ \alpha\ T\ b' \leq i$ $i < B[b := k]\ !\ b'$

let $?goal = SA[k := x]\ !\ i = SA0\ !\ i$

have $lms\text{-}bucket\text{-}start\ \alpha\ T\ b < B\ !\ b$
using *assms lms-distinct-bucket-ptr-lower-bound* **by** *blast*

have $b' \neq b \Longrightarrow ?goal$
proof −
assume $b' \neq b$
with ‹$i < B[b := k]\ !\ b'$›
have $i < B\ !\ b'$
by *simp*

from ‹$b' \neq b$›
have $b' < b \vee b < b'$
by *linarith*
then show *?thesis*
proof
assume $b' < b$
from ‹$lms\text{-}bucket\text{-}start\ \alpha\ T\ b < B\ !\ b$›
have $bucket\text{-}start\ \alpha\ T\ b < B\ !\ b$
by (*simp add*: *lms-bucket-start-def*)
with *assms*(*2*)
have $bucket\text{-}start\ \alpha\ T\ b \leq k$

by *linarith*
moreover
from *lms-bucket-ptr-upper-bound*[*OF assms*(*4*) ‹$b' \leq \alpha$ (*Max* (*set T*))›]
have B ! $b' \leq$ *bucket-end* α T b' **.**
with ‹$i < B$! b'›
have $i <$ *bucket-end* α T b'
using *less-le-trans* by *blast*
ultimately
have $i \neq k$
using ‹$b' < b$›
by (*meson dual-order.strict-trans2 less-bucket-end-le-start order.strict-implies-not-eq*)
hence *SA*[$k := x$] ! i = *SA* ! i
by *simp*
with *lms-unchanged-invD*[*OF assms*(*5*) ‹$b' \leq \alpha$ (*Max* (*set T*))› ‹*bucket-start*
α T $b' \leq i$›
‹$i < B$! b'›]
show *?thesis*
by *simp*
next
assume $b < b'$
from ‹*lms-bucket-start* α T $b < B$! b› *assms*(*2*)
have $k < B$! b
by *linarith*
with *lms-bucket-ptr-upper-bound*[*OF assms*(*4*), *of b*] ‹$b' \leq \alpha$ (*Max* (*set T*))›
‹$b < b'$› *assms*(*7*)
have $k <$ *bucket-end* α T b
by *linarith*
with ‹*bucket-start* α T $b' \leq i$› ‹$b < b'$›
have $i \neq k$
by (*meson dual-order.strict-trans1 leD less-bucket-end-le-start*)
hence *SA*[$k := x$] ! i = *SA* ! i
by *simp*
with *lms-unchanged-invD*[*OF assms*(*5*) ‹$b' \leq \alpha$ (*Max* (*set T*))› ‹*bucket-start*
α T $b' \leq i$›
‹$i < B$! b'›]
show *?thesis*
by *simp*
qed
qed
moreover
have $b' = b \Longrightarrow$ *?goal*
proof −
assume $b' = b$
with ‹$b' \leq \alpha$ (*Max* (*set T*))› ‹*bucket-start* α T $b' \leq i$› ‹$i < B[b := k]$! b'›
have $b \leq \alpha$ (*Max* (*set T*)) *bucket-start* α T $b \leq i$ $i < B[b := k]$! b
by *simp-all*
hence $i < k$
using *assms*(*7*) by *auto*
hence $i < B$! b

using *assms*(*2*) **by** *linarith*
with *lms-unchanged-invD*[*OF assms*(*5*) ‹$b \leq \alpha$ (*Max* (*set T*))› ‹*bucket-start* α *T b* $\leq$ *i*›]
have *SA* ! *i* = *SA0* ! *i*
by *simp*
with ‹$i < k$›
show *?thesis*
by *auto*
qed
ultimately
show *?goal*
by *blast*
qed

62.4.6 Inserted

lemma *lms-inserted-inv-established*:
shows *lms-inserted-inv LMS SA* [] *LMS*
unfolding *lms-inserted-inv-def*
by *simp*

lemma *lms-inserted-inv-maintained-step*:
assumes $b = \alpha$ (*T* ! *x*)
and k = (*B* ! *b*) − *Suc 0*
and *lms-distinct-inv T SA* (*x* # *LMSb*)
and *lms-bucket-ptr-inv* α *T B SA*
and *lms-unknowns-inv* α *T B SA*
and *lms-inserted-inv LMS SA LMSa* (*x* # *LMSb*)
and *strict-mono* α
and *length SA* = *length T*
and $\forall$ *a* $\in$ *set LMS*. *abs-is-lms T a*
shows *lms-inserted-inv LMS* (*SA*[*k* := *x*]) (*LMSa* @ [*x*]) *LMSb*
proof −

from *lms-inserted-invD*(*1*)[*OF assms*(*6*)] *assms*(*9*)
have $\forall$ *a* $\in$ *set* (*x* # *LMSb*). *abs-is-lms T a*
by *auto*
with *lms-next-insert-at-unknown*[*OF assms*(*1*−*5*,*7*,*8*)]
have *k* < *length SA SA* ! *k* = *length T*
by *blast+*

from *lms-inserted-invD*(*1*)[*OF assms*(*6*)] *assms*(*9*)
have $\forall$ *a* $\in$ *set LMSa*. *abs-is-lms T a*
by *auto*
hence $\forall$ *a* $\in$ *set LMSa*. *a* < *length T*
using *abs-is-lms-imp-less-length* **by** *blast*

have *set LMSa* $\subseteq$ *set* (*SA*[*k* := *x*])
proof (*intro subsetI*)

fix y
assume $y \in set\ LMSa$
with ‹$\forall a \in set\ LMSa.\ a < length\ T$›
have $y < length\ T$
by *blast*
with ‹$SA\ !\ k = length\ T$›
have $SA\ !\ k \neq y$
by *simp*
moreover
from *lms-inserted-invD*$(2)[OF\ assms(6)]$ ‹$y \in set\ LMSa$›
have $y \in set\ SA$
by *blast*
ultimately
show $y \in set\ (SA[k := x])$
using *in-set-list-update* **by** *fast*
qed
moreover
from ‹$k < length\ SA$›
have $x \in set\ (SA[k := x])$
by (*simp add*: *set-update-memI*)
ultimately
have $set\ (LMSa\ @\ [x]) \subseteq set\ (SA[k := x])$
by *simp*
with *lms-inserted-invD*$(1)[OF\ assms(6)]$
show *?thesis*
by (*simp add*: *lms-inserted-inv-def*)
qed

62.4.7 Sorted

lemma *lms-sorted-inv-established*:
assumes $\forall i < length\ SA.\ SA\ !\ i = length\ T$
and $\forall a \in set\ LMS.\ abs\text{-}is\text{-}lms\ T\ a$
shows *lms-sorted-inv T LMS SA*
unfolding *lms-sorted-inv-def*
proof (*intro allI impI*; *elim conjE*)
fix $i\ j$
assume A: $j < length\ SA\ \ i < j\ \ SA\ !\ i \in set\ LMS\ SA\ !\ j \in set\ LMS$
from $A(1)\ assms(1)$
have $SA\ !\ j = length\ T$
by *blast*
moreover
from $A(4)\ assms(2)$
have $SA\ !\ j < length\ T$
using *abs-is-lms-imp-less-length* **by** *blast*
ultimately
have *False*
by *auto*
then show

$(T ! (SA ! i) \neq T ! (SA ! j) \longrightarrow T ! (SA ! i) < T ! (SA ! j)) \wedge$
$(T ! (SA ! i) = T ! (SA ! j) \longrightarrow$
$(\exists j'<length\ LMS.\ \exists i'<j'.\ LMS ! i' = SA ! j \wedge LMS ! j' = SA ! i))$
by *blast*
qed

lemma *lms-sorted-inv-maintained-step*:
assumes $b = \alpha\ (T ! x)$
and $k = (B ! b) - Suc\ 0$
and *lms-distinct-inv T SA* $(x \# LMSb)$
and *lms-bucket-ptr-inv* α *T B SA*
and *lms-unknowns-inv* α *T B SA*
and *lms-locations-inv* α *T B SA*
and *lms-unchanged-inv* α *T B SA0 SA*
and *lms-inserted-inv LMS SA LMSa* $(x \# LMSb)$
and *lms-sorted-inv T LMS SA*
and *strict-mono* α
and $length\ SA = length\ T$
and $\forall i < length\ T.\ SA0 ! i = length\ T$
and $\forall a \in set\ LMS.\ abs\text{-}is\text{-}lms\ T\ a$
shows *lms-sorted-inv T LMS* $(SA[k := x])$
unfolding *lms-sorted-inv-def*
proof (*intro allI impI*; *elim conjE*)
fix *i j*
assume *A*: $j < length\ (SA[k := x])\ i < j\ SA[k := x] ! i \in set\ LMS$
$SA[k := x] ! j \in set\ LMS$
let $?goal1 = T ! (SA[k := x] ! i) \neq T ! (SA[k := x] ! j) \longrightarrow$
$T ! (SA[k := x] ! i) < T ! (SA[k := x] ! j)$
and $?goal2 = T ! (SA[k := x] ! i) = T ! (SA[k := x] ! j) \longrightarrow$
$(\exists j'<length\ LMS.\ \exists i'<j'.$
$LMS ! i' = SA[k := x] ! j \wedge LMS! j' = SA[k := x] ! i)$

let $?goal = ?goal1 \wedge ?goal2$

from *assms*(*13*) *lms-inserted-invD*[*OF assms*(*8*)]
have $\forall a\in set\ (x \# LMSb).\ abs\text{-}is\text{-}lms\ T\ a$
by *auto*
with *lms-next-insert-at-unknown*[*OF assms*(*1*−*5*,*10*,*11*)]
have $k < length\ SA\ SA ! k = length\ T$
by *blast*+

from *lms-distinct-bucket-ptr-lower-bound*[*OF assms*(*1*,*3*,*4*,*10*) ‹$\forall a\in set\ (x \# LMSb).$ *abs-is-lms T a*›]
have *lms-bucket-start* α $T\ b < B ! b$ **.**

from *assms*(*1*,*10*) ‹$\forall a\in set\ (x \# LMSb).\ abs\text{-}is\text{-}lms\ T\ a$›
have $b \leq \alpha\ (Max\ (set\ T))$
by (*simp add*: *abs-is-lms-imp-less-length strict-mono-less-eq*)

have $[\![k \neq i; k \neq j]\!] \Longrightarrow$ *?goal*
proof $-$
assume $k \neq i$ $k \neq j$
with *A*
have $j <$ *length SA SA* ! $i \in$ *set LMS SA* ! $j \in$ *set LMS*
by *simp-all*
with *lms-sorted-invD*[*OF assms*(*9*) - ‹$i < j$›] ‹$k \neq i$› ‹$k \neq j$›
show *?goal*
by *simp*
qed
moreover
have $[\![k = i; k \neq j]\!] \Longrightarrow$ *?goal*
proof $-$
assume $k = i$ $k \neq j$
with *A*(*1*,*4*)
have $j <$ *length SA SA* ! $j \in$ *set LMS SA*[$k := x$] ! $j =$ *SA* ! j
by *simp-all*

from ‹$k = i$› *assms*(*2*)
have $i = B$! $b -$ *Suc 0*
by *simp*

from ‹*SA* ! $j \in$ *set LMS*› *assms*(*13*)
have *SA* ! $j <$ *length T*
using *abs-is-lms-imp-less-length* **by** *blast*

from *index-in-bucket-interval-gen*[*of j T*, *OF* - *assms*(*10*)] *assms*(*11*) ‹$j <$ *length SA*›
obtain b' **where**
$b' \leq \alpha$ (*Max* (*set T*))
bucket-start α *T* $b' \leq j$
$j <$ *bucket-end* α *T* b'
by *auto*

have B ! $b' \leq j$
proof (*rule ccontr*)
assume $\neg$ B ! $b' \leq j$
hence $j < B$! b'
by *simp*
with *lms-unchanged-invD*[*OF assms*(*7*) ‹$b' \leq \alpha$ (*Max* (*set T*))› ‹*bucket-start* α *T* $b' \leq j$›]
assms(*11*,*12*) ‹$j <$ *length SA*›
have *SA* ! $j =$ *length T*
by *auto*
with ‹*SA* ! $j <$ *length T*›
show *False*
by *auto*
qed
with *lms-locations-invD*[*OF assms*(*6*) ‹$b' \leq \alpha$ (*Max* (*set T*))› - ‹$j <$ *bucket-end*

α *T b′*›]
  have *SA* ! *j* ∈ *lms-bucket* α *T b′*
    by *blast*
  hence α (*T* ! (*SA* ! *j*)) = *b′*
    by (*simp add*: *bucket-def lms-bucket-def*)

  from ‹*lms-bucket-start* α *T b* < *B* ! *b*› ‹*i* = *B* ! *b* − *Suc 0*›
  have *lms-bucket-start* α *T b* ≤ *i*
    by *linarith*
  hence *bucket-start* α *T b* ≤ *i*
    by (*metis l-bucket-end-def l-bucket-end-le-lms-bucket-start le-add1 le-trans*)

  have *b* ≤ *b′*
  proof (*rule ccontr*)
    assume ¬ *b* ≤ *b′*
    hence *b′* < *b*
      by *simp*
    hence *bucket-end* α *T b′* ≤ *bucket-start* α *T b*
      by (*simp add*: *less-bucket-end-le-start*)
    with ‹*j* < *bucket-end* α *T b′*›
    have *j* < *bucket-start* α *T b*
      by *linarith*
    with ‹*bucket-start* α *T b* ≤ *i*›
    have *j* < *i*
      by *linarith*
    with ‹*i* < *j*›
    show *False*
      by *linarith*
  qed

  from *assms*(*3*)[*simplified lms-distinct-inv-def distinct-append*] ‹*SA* ! *j* < *length*
T›
       ‹*j* < *length SA*›
  have *SA* ! *j* ∉ *set* (*x* # *LMSb*)
    by (*metis IntI empty-iff filter-set member-filter nth-mem*)
  with *lms-inserted-invD*[*OF assms*(*8*)] ‹*SA* ! *j* ∈ *set LMS*›
  have *SA* ! *j* ∈ *set LMSa*
    by *auto*
  hence ∃ *j′* < *length LMSa*. *LMSa* ! *j′* = *SA* ! *j*
    by (*simp add*: *in-set-conv-nth*)
  then obtain *j′* **where**
    j′ < *length LMSa*
    LMSa ! *j′* = *SA* ! *j*
    by *blast*
  with *lms-inserted-invD*(*1*)[*OF assms*(*8*)] ‹*SA*[*k* := *x*] ! *j* = *SA* ! *j*›
  have *LMS* ! *j′* = *SA*[*k* := *x*] ! *j*
    by (*simp add*: *nth-append*)

  from ‹α (*T* ! (*SA* ! *j*)) = *b′*› ‹*SA*[*k* := *x*] ! *j* = *SA* ! *j*›

have α $(T\ !\ (SA[k := x]\ !\ j)) = b'$
by *simp*

from *lms-inserted-invD*(*1*)[*OF assms*(*8*)]
have *length LMSa* $<$ *length LMS*
by *simp*

from *assms*(*3*)[*simplified lms-distinct-inv-def distinct-append*] *lms-inserted-invD*[*OF assms*(*8*)]
have $x \notin$ *set LMSa*
using ‹$\forall a \in$*set* $(x \# LMSb)$. *abs-is-lms T a*› *abs-is-lms-imp-less-length* **by** *fastforce*
with *lms-inserted-invD*(*1*)[*OF assms*(*8*)]
have *LMS* ! *length LMSa* $= x$
by *simp*
hence *LMS* ! *length LMSa* $= SA[k := x]\ !\ i$
using ‹$k <$ *length SA*› ‹$k = i$› **by** *auto*

from ‹$k = i$› *assms*(*1*) ‹$k <$ *length SA*›
have α $(T\ !\ (SA[k := x]\ !\ i)) = b$
by *auto*

have $b = b' \Longrightarrow$ *?goal2*
proof
from ‹*LMS* ! $j' = SA[k := x]\ !\ j$› ‹*LMS* ! *length LMSa* $= SA[k := x]\ !\ i$› ‹$j' <$ *length LMSa*›
‹*length LMSa* $<$ *length LMS*›
show $\exists j'<$*length LMS*. $\exists i'<j'$. *LMS* ! $i' = SA[k := x]\ !\ j \wedge$ *LMS* ! $j' = SA[k := x]\ !\ i$
by *blast*
qed
moreover
from ‹α $(T\ !\ (SA[k := x]\ !\ j)) = b'$› ‹$\alpha$ $(T\ !\ (SA[k := x]\ !\ i)) = b$› *assms*(*10*)
have $b = b' \Longrightarrow T\ !\ (SA[k := x]\ !\ i) = T\ !\ (SA[k := x]\ !\ j)$
using *strict-mono-eq* **by** *auto*
moreover
have $b < b' \Longrightarrow$ *?goal1*
proof
assume $b < b'$
with ‹α $(T\ !\ (SA[k := x]\ !\ i)) = b$› ‹$\alpha$ $(T\ !\ (SA[k := x]\ !\ j)) = b'$› *assms*(*10*)
show $T\ !\ (SA[k := x]\ !\ i) < T\ !\ (SA[k := x]\ !\ j)$
using *strict-mono-less* **by** *blast*
qed
moreover
from ‹α $(T\ !\ (SA[k := x]\ !\ j)) = b'$› ‹$\alpha$ $(T\ !\ (SA[k := x]\ !\ i)) = b$› *assms*(*10*)
have $b < b' \Longrightarrow T\ !\ (SA[k := x]\ !\ i) \neq T\ !\ (SA[k := x]\ !\ j)$
by *auto*
moreover
from ‹$b \leq b'$›

```
    have b = b′ ∨ b < b′
      by linarith
    ultimately
    show ?goal
      by blast
  qed
  moreover
  have ⟦k ≠ i; k = j⟧ ⟹ ?goal
  proof −
    assume k ≠ i k = j
    with ‹SA[k := x] ! i ∈ set LMS›
    have SA[k := x] ! i = SA ! i SA ! i ∈ set LMS
      by simp-all

    from ‹k = j› assms(2)
    have j = B ! b − Suc 0
      by simp

    from ‹j < length (SA[k := x])› ‹i < j› assms(11)
    have i < length T
      by simp
    with index-in-bucket-interval-gen[of i T, OF - assms(10)]
    obtain b′ where
      b′ ≤ α (Max (set T))
      bucket-start α T b′ ≤ i
      i < bucket-end α T b′
      by auto

    from ‹SA ! i ∈ set LMS› assms(13)
    have SA ! i < length T
      using abs-is-lms-imp-less-length by blast

    have B ! b′ ≤ i
    proof (rule ccontr)
      assume ¬B ! b′ ≤ i
      hence i < B ! b′
        by (simp add: not-le)
      with lms-unchanged-invD[OF assms(7) ‹b′ ≤ α (Max (set T))› ‹bucket-start
α T b′ ≤ i›]
            assms(12) ‹i < length T›
      have SA ! i = length T
        by simp
      with ‹SA ! i < length T›
      show False
        by linarith
    qed
    with lms-locations-invD[OF assms(6) ‹b′ ≤ α (Max (set T))› - ‹i < bucket-end
α T b′›]
    have SA ! i ∈ lms-bucket α T b′
```

by *blast*
hence α (T ! (SA ! i)) = b'
by (*simp add*: *bucket-def lms-bucket-def*)
with ‹$SA[k := x]$! i = SA ! i›
have α (T ! ($SA[k := x]$! i)) = b'
by *simp*

from ‹$i < j$› ‹$j = B$! b − *Suc 0*› ‹*lms-bucket-start* α T b < B ! b›
have $i < B$! b
using *diff-le-self dual-order.strict-trans1* **by** *blast*

have i < *bucket-start* α T b
proof (*rule ccontr*)
assume ¬i < *bucket-start* α T b
hence *bucket-start* α T $b \leq i$
by (*simp add*: *not-le*)
with *lms-unchanged-invD*[*OF assms*(*7*) ‹$b \leq \alpha$ (*Max* (*set* T))› - ‹$i < B$! b›]
assms(*12*) ‹i < *length* T›
have SA ! i = *length* T
by *auto*
with ‹SA ! i < *length* T›
show *False*
by *linarith*
qed
with ‹*bucket-start* α T $b' \leq i$›
have *bucket-start* α T b' < *bucket-start* α T b
by *linarith*
hence $b' < b$
by (*meson bucket-start-le leD leI*)

from *assms*(*1*) ‹$k = j$› ‹k < *length* SA›
have α (T ! ($SA[k := x]$! j)) = b
by *simp*
with ‹α (T ! ($SA[k := x]$! i)) = b'› ‹$b' < b$› *assms*(*10*)
have T ! ($SA[k := x]$! i) < T ! ($SA[k := x]$! j)
using *strict-mono-less* **by** *blast*
then show *?goal*
by *auto*
qed
moreover
from ‹$i < j$›
have $[\![k = i; k = j]\!] \Longrightarrow$ *?goal*
by *blast*
ultimately
show *?goal*
by *blast*
qed

62.5 Combined Establishment and Maintenance

lemma *lms-inv-established*:
  assumes $\forall\, i < length\ SA.\ SA\ !\ i = length\ T$
  and $\forall\, x \in set\ LMS.\ abs\text{-}is\text{-}lms\ T\ x$
  and *distinct LMS*
  and $lms\text{-}bucket\text{-}init\ \alpha\ T\ B$
  and $length\ SA = length\ T$
  and $strict\text{-}mono\ \alpha$
shows $lms\text{-}inv\ \alpha\ T\ B\ LMS\ []\ LMS\ SA\ SA$
  unfolding *lms-inv-def*
  using *lms-distinct-inv-established*[*OF assms*(*3*,*1*)]
    lms-bucket-ptr-inv-established[*OF assms*(*4*,*1*)]
    lms-unknowns-inv-established[*OF assms*(*4*,*1*,*5*)]
    lms-locations-inv-established[*OF assms*(*4*)]
    lms-unchanged-inv-established
    lms-inserted-inv-established
    lms-sorted-inv-established[*OF assms*(*1*,*2*)]
    lms-bucket-init-length[*OF assms*(*4*)]
    assms
  by *auto*

lemma *lms-inv-maintained-step*:
  assumes $lms\text{-}inv\ \alpha\ T\ B\ LMS\ LMSa\ (x\ \#\ LMSb)\ SA0\ SA$
  and $b = \alpha\ (T\ !\ x)$
  and $k = (B\ !\ b) - Suc\ 0$
shows $lms\text{-}inv\ \alpha\ T\ (B[b := k])\ LMS\ (LMSa\ @\ [x])\ LMSb\ SA0\ (SA[k := x])$
  unfolding *lms-inv-def*
  using *lms-distinct-inv-maintained-step*[*OF lms-invD*(*1*)[*OF assms*(*1*)]]
    lms-bucket-ptr-inv-maintained-step[*OF assms*(*2*−*3*)
      lms-invD(*1*−*3*,*8*,*9*,*12*)[*OF assms*(*1*)]
      lms-inv-lms-helper(*3*)[*OF assms*(*1*)]]
    lms-unknowns-inv-maintained-step[*OF assms*(*2*−*3*)
      lms-invD(*1*−*3*,*8*,*9*)[*OF assms*(*1*)]
      lms-inv-lms-helper(*3*)[*OF assms*(*1*)]]
    lms-locations-inv-maintained-step[*OF assms*(*2*−*3*)
      lms-invD(*1*−*2*,*4*,*8*,*9*,*12*)[*OF assms*(*1*)]
      lms-inv-lms-helper(*3*)[*OF assms*(*1*)]]
    lms-unchanged-inv-maintained-step[*OF assms*(*2*−*3*)
      lms-invD(*1*−*2*,*5*,*8*,*9*,*12*)[*OF assms*(*1*)]
      lms-inv-lms-helper(*3*)[*OF assms*(*1*)]]
    lms-inserted-inv-maintained-step[*OF assms*(*2*−*3*)
      lms-invD(*1*−*3*,*6*,*8*,*12*)[*OF assms*(*1*)]
      lms-inv-lms-helper(*1*)[*OF assms*(*1*)]]
    lms-sorted-inv-maintained-step[*OF assms*(*2*−*3*)
      lms-invD(*1*−*8*,*12*,*13*)[*OF assms*(*1*)]
      lms-inv-lms-helper(*1*)[*OF assms*(*1*)]]
  by (*metis assms*(*1*) *length-list-update lms-inv-def*)

lemma *lms-inv-maintained*:

assumes *bucket-insert-abs′ α T B SA gs xs* = (*SA′*, *B′*, *gs′*)
and *lms-inv α T B LMS gs xs SA0 SA*
shows *lms-inv α T B′ LMS gs′* [] *SA0 SA′*
using *assms*
proof (*induct arbitrary*: *SA′ B′ gs′ LMS SA0*
rule: *bucket-insert-abs′.induct*[*of* - *α T B SA gs xs*])
case (*1 α T B SA gs*)
note *BC* = *this*
from *BC*(*1*)[*simplified*]
have *B′* = *B SA′* = *SA gs′* = *gs*
by *auto*
with *BC*(*2*)
show *?case*
by *simp*
next
case (*2 α T B SA gs x xs*)
note *IH* = *this*
let *?b* = *α* (*T* ! *x*)
let *?k* = *B* ! *?b* − *Suc 0*
from *lms-inv-maintained-step*[*OF IH*(*3*), *of ?b ?k*]
have *R1*: *lms-inv α T* (*B*[*?b* := *?k*]) *LMS* (*gs* @ [*x*]) *xs SA0* (*SA*[*?k* := *x*])
by *simp*

from *IH*(*2*)[*simplified*, *simplified Let-def*]
have *R2*: *bucket-insert-abs′ α T* (*B*[*?b* := *?k*]) (*SA*[*?k* := *x*]) (*gs* @ [*x*]) *xs* = (*SA′*, *B′*, *gs′*) **.**

from *IH*(*1*)[*of ?b ?k SA*[*?k* := *x*] *B*[*?b* := *?k*] *gs* @ [*x*] *SA′ B′ gs′ LMS SA0*,
simplified,
OF R2 R1]
show *?case* **.**
qed

lemma *lms-inv-holds*:
assumes ∀ *i* < *length SA*. *SA* ! *i* = *length T*
and ∀ *x* ∈ *set LMS*. *abs-is-lms T x*
and *distinct LMS*
and *lms-bucket-init α T B*
and *length SA* = *length T*
and *strict-mono α*
and *bucket-insert-abs′ α T B SA* [] *LMS* = (*SA′*, *B′*, *gs′*)
shows *lms-inv α T B′ LMS gs′* [] *SA SA′*
using *lms-inv-maintained*[*OF assms*(*7*) *lms-inv-established*[*OF assms*(*1*−*6*)]] **.**

63 Exhaustiveness

definition *lms-type-exhaustive* :: (′*a* :: {*linorder*, *order-bot*}) *list* ⇒ *nat list* ⇒ *bool*
where
lms-type-exhaustive T SA = (∀ *i* < *length T*. *abs-is-lms T i* ⟶ *i* ∈ *set SA*)

lemma *lms-type-exhaustiveD*:
  $[\![$*lms-type-exhaustive T SA*; i < *length T*; *abs-is-lms T i*$]\!] \Longrightarrow i \in$ *set SA*
  using *lms-type-exhaustive-def* **by** *blast*

lemma *lms-all-inserted-imp-exhaustive*:
  assumes *lms-inserted-inv LMS SA LMS* []
  and *set LMS* = {*i*. *abs-is-lms T i*}
shows *lms-type-exhaustive T SA*
  unfolding *lms-type-exhaustive-def*
proof (*intro allI impI*)
  fix *i*
  assume *i* < *length T abs-is-lms T i*
  with *assms*(*2*)
  have $i \in$ *set LMS*
    by *blast*
  with *lms-inserted-invD*(*2*)[*OF assms*(*1*)]
  show $i \in$ *set SA*
    by *blast*
qed

lemma *lms-type-exhaustive-imp-lms-bucket-subset*:
  assumes *lms-type-exhaustive T SA*
  and $b \le \alpha$ (*Max* (*set T*))
shows *lms-bucket* α *T b* $\subseteq$ *set SA*
proof (*intro subsetI*)
  fix *x*
  assume $x \in$ *lms-bucket* α *T b*
  hence *x* < *length T*
    by (*simp add*: *abs-is-lms-imp-less-length lms-bucket-def*)

  from ‹$x \in$ *lms-bucket* α *T b*›
  have *abs-is-lms T x*
    by (*simp add*: *lms-bucket-def*)

  from *lms-type-exhaustiveD*[*OF assms*(*1*) ‹*x* < *length T*› ‹*abs-is-lms T x*›]
  show $x \in$ *set SA* **.**
qed

lemma *lms-B-val*:
  assumes $\forall$ *i* < *length SA*. *SA* ! *i* = *length T*
  and *distinct LMS*
  and *lms-bucket-init* α *T B*
  and *length SA* = *length T*
  and *strict-mono* α
  and *set LMS* = {*i*. *abs-is-lms T i*}
  and *bucket-insert-abs'* α *T B SA* [] *LMS* = (*SA'*, *B'*, *gs'*)
  and $b \le \alpha$ (*Max* (*set T*))

shows B' ! b = *lms-bucket-start* α T b
proof −
  from *assms*(*6*)
  have $\forall$ x $\in$ *set LMS*. *abs-is-lms T x*
    by *blast*
  with *lms-inv-holds*[*OF assms*(*1*) - *assms*(*2*−*5*,*7*)]
  have *lms-inv* α *T* B' *LMS* gs' [] *SA* SA' **.**
  hence *lms-inserted-inv LMS* SA' gs' []
    using *lms-inv-def* **by** *blast*
  hence gs' = *LMS*
    by (*simp add*: *lms-inserted-inv-def*)
  with ‹*lms-inserted-inv LMS* SA' gs' []› *lms-all-inserted-imp-exhaustive*[*OF* - *assms*(*6*), *of* SA']
  have *lms-type-exhaustive T* SA'
    by *simp*
  with *lms-type-exhaustive-imp-lms-bucket-subset assms*(*8*)
  have *lms-bucket* α *T b* $\subseteq$ *set* SA'
    by *blast*
  with *cur-lms-subset-lms-bucket*
  have *cur-lms-types* α *T* SA' *b* = *lms-bucket* α *T b*
    by (*simp add*: *cur-lms-types-def equalityI subset-eq*)
  hence *num-lms-types* α *T* SA' *b* = *card* (*lms-bucket* α *T b*)
    by (*simp add*: *num-lms-types-def*)
  moreover
  from ‹*lms-inv* α *T* B' *LMS* gs' [] *SA* SA'›
  have *lms-bucket-ptr-inv* α *T* B' SA'
    using *lms-inv-def* **by** *blast*
  with *lms-bucket-ptr-invD assms*(*8*)
  have B' ! *b* + *num-lms-types* α *T* SA' *b* = *bucket-end* α *T b*
    by *blast*
  ultimately
  have B' ! *b* + *card* (*lms-bucket* α *T b*) = *bucket-end* α *T b*
    by *simp*
  then show B' ! *b* = *lms-bucket-start* α *T b*
    by (*metis add-implies-diff lms-bucket-pl-size-eq-end lms-bucket-size-def*)
qed

64 Postconditions

definition *lms-vals-post* :: ($'a$ :: {*linorder*, *order-bot*} ⇒ *nat*) ⇒ $'a$ *list* ⇒ *nat list* ⇒ *bool*
where
lms-vals-post α *T SA* =
  ($\forall$ b $\leq$ α (*Max* (*set T*)).
    lms-bucket α *T b* = *set* (*list-slice SA* (*lms-bucket-start* α *T b*) (*bucket-end* α *T b*))
)

lemma *lms-vals-postD*:

$[\![$*lms-vals-post* α T *SA*; $b \leq \alpha$ (*Max* (*set* T))$]\!] \Longrightarrow$
lms-bucket α T b = *set* (*list-slice SA* (*lms-bucket-start* α T b) (*bucket-end* α T b))
using *lms-vals-post-def* **by** *blast*

definition
lms-pre :: ($'a$ :: {*linorder*, *order-bot*} $\Rightarrow$ *nat*) $\Rightarrow$ $'a$ *list* $\Rightarrow$ *nat list* $\Rightarrow$ *nat list* $\Rightarrow$ *nat list* $\Rightarrow$ *bool*
where
lms-pre α T B *SA LMS* $\equiv$
($\forall\, i <$ *length SA*. *SA* ! i = *length* T) $\wedge$
length SA = *length* T $\wedge$
lms-bucket-init α T B $\wedge$
strict-mono α $\wedge$
distinct LMS $\wedge$
set LMS = {i. *abs-is-lms* T i}

lemma *lms-pre-elims*:
lms-pre α T B *SA LMS* $\Longrightarrow$ $\forall\, i <$ *length SA*. *SA* ! i = *length* T
lms-pre α T B *SA LMS* $\Longrightarrow$ *length SA* = *length* T
lms-pre α T B *SA LMS* $\Longrightarrow$ *lms-bucket-init* α T B
lms-pre α T B *SA LMS* $\Longrightarrow$ *strict-mono* α
lms-pre α T B *SA LMS* $\Longrightarrow$ *distinct LMS*
lms-pre α T B *SA LMS* $\Longrightarrow$ *set LMS* = {i. *abs-is-lms* T i}
using *lms-pre-def* **by** *blast+*

lemma *lms-vals-post-holds*:
assumes $\forall\, i <$ *length SA*. *SA* ! i = *length* T
and *distinct LMS*
and *lms-bucket-init* α T B
and *length SA* = *length* T
and *strict-mono* α
and *set LMS* = {i. *abs-is-lms* T i}
and *bucket-insert-abs'* α T B *SA* [] *LMS* = (*SA'*, *B'*, *gs'*)
shows *lms-vals-post* α T *SA'*
unfolding *lms-vals-post-def*
proof(*intro allI impI*)
fix b
assume $b \leq \alpha$ (*Max* (*set* T))

from *assms*(*6*)
have $\forall\, x \in$ *set LMS*. *abs-is-lms* T x
by *blast*
with *lms-inv-holds*[*OF assms*(*1*) - *assms*(*2*−*5*,*7*)]
have *R0*:*lms-inv* α T *B' LMS gs'* [] *SA SA'* **.**

from *lms-B-val*[*OF assms*(*1*−*7*) ‹$b \leq \alpha$ (*Max* (*set* T))›]
have *R1*: *B'* ! b = *lms-bucket-start* α T b **.**

have *bucket-end* α *T b* $\leq$ *length SA'*
by (*metis R0 bucket-end-le-length lms-inv-def*)

have *finite* (*lms-bucket* α *T b*)
by (*simp add*: *finite-lms-bucket*)
moreover
from *lms-slice-subset-lms-bucket*[*OF lms-invD*(*4*,*12*)[*OF R0*] ‹*b* $\leq$ α (*Max* (*set T*))›]
have *set* (*list-slice SA'* (*B'* ! *b*) (*bucket-end* α *T b*)) $\subseteq$ *lms-bucket* α *T b* **.**
with *R1*
have *set* (*list-slice SA'* (*lms-bucket-start* α *T b*) (*bucket-end* α *T b*)) $\subseteq$ *lms-bucket* α *T b*
by *simp*
moreover
from *lms-distinct-slice*[*OF lms-invD*(*1*,*2*,*4*,*12*)[*OF R0*] ‹*b* $\leq$ α (*Max* (*set T*))›]
have *distinct* (*list-slice SA'* (*B'* ! *b*) (*bucket-end* α *T b*)) **.**
with *R1*
have *distinct* (*list-slice SA'* (*lms-bucket-start* α *T b*) (*bucket-end* α *T b*))
by *simp*
with *distinct-card*
have *card* (*set* (*list-slice SA'* (*lms-bucket-start* α *T b*) (*bucket-end* α *T b*)))
= *length* (*list-slice SA'* (*lms-bucket-start* α *T b*) (*bucket-end* α *T b*))
by *blast*
with ‹*bucket-end* α *T b* $\leq$ *length SA'*›
have *card* (*set* (*list-slice SA'* (*lms-bucket-start* α *T b*) (*bucket-end* α *T b*)))
= *lms-bucket-size* α *T b*
by (*metis add-diff-cancel-left' length-list-slice lms-bucket-pl-size-eq-end min.absorb-iff1*)
hence *card* (*set* (*list-slice SA'* (*lms-bucket-start* α *T b*) (*bucket-end* α *T b*)))
= *card* (*lms-bucket* α *T b*)
by (*simp add*: *lms-bucket-size-def*)
ultimately
show *lms-bucket* α *T b* = *set* (*list-slice SA'* (*lms-bucket-start* α *T b*) (*bucket-end* α *T b*))
using *card-subset-eq* **by** *blast*
qed

corollary *abs-bucket-insert-vals*:
assumes *lms-pre* α *T B SA LMS*
shows *lms-vals-post* α *T* (*abs-bucket-insert* α *T B SA LMS*)
proof −
have $\exists$ *SA' B' gs*. *bucket-insert-abs'* α *T B SA* [] *LMS* = (*SA'*, *B'*, *gs*)
by (*meson prod-cases3*)
then obtain *SA' B' gs* **where**
A: *bucket-insert-abs'* α *T B SA* [] *LMS* = (*SA'*, *B'*, *gs*)
by *blast*
hence *abs-bucket-insert* α *T B SA LMS* = *SA'*
by (*metis abs-bucket-insert-equiv fst-conv*)
with *lms-vals-post-holds*[*OF lms-pre-elims*(*1*,*5*,*3*,*2*,*4*,*6*)[*OF assms*] *A*]
show *?thesis*

by *simp*
qed

definition *lms-unknowns-post*
where
lms-unknowns-post α *T SA* =
($\forall b \leq \alpha$ (*Max* (*set T*)).
($\forall i$. *bucket-start* α *T b* $\leq i \wedge i <$ *lms-bucket-start* α *T b* $\longrightarrow$ *SA* ! i = *length* *T*)
)

lemma *lms-unknowns-postD*:
⟦*lms-unknowns-post* α *T SA*; $b \leq \alpha$ (*Max* (*set T*)); *bucket-start* α *T b* $\leq i$;
$i <$ *lms-bucket-start* α *T b*⟧ $\Longrightarrow$
SA ! i = *length T*
using *lms-unknowns-post-def* **by** *blast*

lemma *lms-unknowns-post-holds*:
assumes $\forall i <$ *length SA*. *SA* ! i = *length T*
and *distinct LMS*
and *lms-bucket-init* α *T B*
and *length SA* = *length T*
and *strict-mono* α
and *set LMS* = $\{i.$ *abs-is-lms T i*$\}$
and *bucket-insert-abs′* α *T B SA* [] *LMS* = (*SA′*, *B′*, *gs′*)
shows *lms-unknowns-post* α *T SA′*
unfolding *lms-unknowns-post-def*
proof(*intro allI impI*; *elim conjE*)
fix *b i*
assume $b \leq \alpha$ (*Max* (*set T*)) *bucket-start* α *T b* $\leq i$ $i <$ *lms-bucket-start* α *T b*

from *assms*(*6*)
have $\forall x \in$ *set LMS*. *abs-is-lms T x*
by *blast*
with *lms-inv-holds*[*OF assms*(*1*) - *assms*(*2*−*5*,*7*)]
have *R0*:*lms-inv* α *T B′ LMS gs′* [] *SA SA′* **.**

from ‹$i <$ *lms-bucket-start* α *T b*›
have $i <$ *length SA*
by (*metis assms*(*4*) *bucket-end-le-length dual-order.strict-trans1 lms-bucket-start-le-bucket-end*)
moreover
from *lms-B-val*[*OF assms*(*1*−*7*) ‹$b \leq \alpha$ (*Max* (*set T*))›]
have *B′* ! b = *lms-bucket-start* α *T b* **.**
with ‹$i <$ *lms-bucket-start* α *T b*›
have $i <$ *B′* ! b
by *simp*
with *lms-unchanged-invD*[*OF lms-invD*(*5*)[*OF R0*] ‹$b \leq \alpha$ (*Max* (*set T*))› ‹*bucket-start* α *T b* $\leq i$›]
have *SA′* ! i = *SA* ! i

by *blast*
ultimately
show SA' ! i = *length* T
using *assms*(*1*) **by** *auto*
qed

corollary *abs-bucket-insert-unknowns*:
assumes *lms-pre* α T B *SA* *LMS*
shows *lms-unknowns-post* α T (*abs-bucket-insert* α T B *SA* *LMS*)
proof −
have $\exists$ SA' B' *gs*. *bucket-insert-abs'* α T B *SA* [] *LMS* = (SA', B', *gs*)
by (*meson prod-cases3*)
then obtain SA' B' *gs* **where**
A: *bucket-insert-abs'* α T B *SA* [] *LMS* = (SA', B', *gs*)
by *blast*
hence *abs-bucket-insert* α T B *SA* *LMS* = SA'
by (*metis abs-bucket-insert-equiv fst-conv*)
with *lms-unknowns-post-holds*[*OF lms-pre-elims*(*1*,*5*,*3*,*2*,*4*,*6*)[*OF assms*] *A*]
show *?thesis*
by *simp*
qed

corollary *abs-bucket-insert-values*:
assumes *lms-pre* α T B *SA* *LMS*
shows $\forall$ $b \leq \alpha$ (*Max* (*set* T)).
($\forall$ i. *bucket-start* α T $b \leq i \wedge i <$ *lms-bucket-start* α T $b \longrightarrow$ (*abs-bucket-insert* α T B *SA* *LMS*) ! i = *length* T) $\wedge$
lms-bucket α T b = *set* (*list-slice* (*abs-bucket-insert* α T B *SA* *LMS*) (*lms-bucket-start* α T b) (*bucket-end* α T b))
by (*meson assms abs-bucket-insert-unknowns abs-bucket-insert-vals lms-unknowns-postD lms-vals-postD*)

lemma *lms-lms-prefix-sorted-holds*:
assumes $\forall$ $i <$ *length* *SA*. *SA* ! i = *length* T
and *distinct* *LMS*
and *lms-bucket-init* α T B
and *length* *SA* = *length* T
and *strict-mono* α
and *set* *LMS* = {i. *abs-is-lms* T i}
and *bucket-insert-abs'* α T B *SA* [] *LMS* = (SA', B', gs')
shows *ordlistns.sorted* (*map* (*lms-prefix* T) (*filter* (λx. $x <$ *length* T) SA'))
proof −
from *assms*(*6*)
have $\forall$ $x \in$ *set* *LMS*. *abs-is-lms* T x
by *blast*
with *lms-inv-holds*[*OF assms*(*1*) - *assms*(*2*−*5*,*7*)]
have *R0*:*lms-inv* α T B' *LMS* gs' [] *SA* SA' **.**

from *lms-lms-prefix-sorted*[*OF lms-invD*(*2*,*4*,*5*,*8*,*12*,*13*)[*OF R0*] *assms*(*6*)]

show *?thesis* **.**
qed

lemma *lms-suffix-sorted-holds*:
assumes $\forall\, i < \mathit{length}\ SA.\ SA\ !\ i = \mathit{length}\ T$
and *distinct LMS*
and *lms-bucket-init* α *T B*
and *length SA* = *length T*
and *strict-mono* α
and *set LMS* = $\{i.$ *abs-is-lms T i*$\}$
and *bucket-insert-abs'* α *T B SA* [] *LMS* = (*SA'*, *B'*, *gs'*)
and *ordlistns.sorted* (*map* (*suffix T*) (*rev LMS*))
shows *ordlistns.sorted* (*map* (*suffix T*) (*filter* ($\lambda x.\ x < \mathit{length}\ T$) *SA'*))
proof –
from *assms*(*6*)
have $\forall\, x \in \mathit{set}\ LMS.$ *abs-is-lms T x*
by *blast*
with *lms-inv-holds*[*OF assms*(*1*) - *assms*(*2*−*5*,*7*)]
have *R0*:*lms-inv* α *T B' LMS gs'* [] *SA SA'* **.**

from *lms-suffix-sorted*[*OF lms-invD*(*2*,*4*,*5*,*7*,*8*,*12*,*13*)[*OF R0*] *assms*(*6*) *assms*(*8*)]
show *?thesis* **.**
qed

lemma *lms-bot-is-first*:
assumes $\forall\, i < \mathit{length}\ SA.\ SA\ !\ i = \mathit{length}\ T$
and *distinct LMS*
and *lms-bucket-init* α *T B*
and *length SA* = *length T*
and *strict-mono* α
and *set LMS* = $\{i.$ *abs-is-lms T i*$\}$
and *bucket-insert-abs'* α *T B SA* [] *LMS* = (*SA'*, *B'*, *gs'*)
and *valid-list T*
and *length T* = *Suc* (*Suc n*)
and α *bot* = *0*
shows *SA'* ! *0* = *Suc n*
proof –
have *abs-is-lms T* (*Suc n*)
by (*simp add*: *assms*(*8*,*9*) *abs-is-lms-last*)
moreover
have α (*T* ! (*Suc n*)) = *0*
by (*metis assms*(*8*−*10*) *diff-Suc-1 last-conv-nth length-greater-0-conv valid-list-def*)
ultimately
have *Suc n* ∈ *lms-bucket* α *T 0*
by (*simp add*: *assms*(*5*,*8*−*10*) *bucket-0 lms-bucket-def*)

have *lms-bucket-size* α *T 0* = *Suc 0 l-bucket-size* α *T 0* = *0 pure-s-bucket-size* α *T 0* = *0*
using *assms*(*5*,*8*−*10*) *bucket-0-size2* **by** *blast+*

hence *lms-bucket* α *T 0* = {*Suc n*}
by (*metis One-nat-def add.commute assms*(*5*,*8*−*10*) *atLeastLessThan-singleton bucket-0*
lms-bucket-size-def lms-bucket-subset-bucket plus-1-eq-Suc subset-card-intvl-is-intvl)

from *assms*(*6*)
have ∀ *x* ∈ *set LMS*. *abs-is-lms T x*
by *blast*
with *lms-inv-holds*[*OF assms*(*1*) - *assms*(*2*−*5*,*7*)]
have *R0*:*lms-inv* α *T B′ LMS gs′* [] *SA SA′* **.**

from ‹*l-bucket-size* α *T 0* = *0*› ‹*pure-s-bucket-size* α *T 0* = *0*›
have *lms-bucket-start* α *T 0* = *0*
by (*simp add*: *bucket-start-0 lms-bucket-start-def*)
moreover
from *lms-B-val*[*OF assms*(*1*−*7*), *of 0*]
have *B′* ! *0* = *lms-bucket-start* α *T 0*
by *simp*
moreover
have *0* < *bucket-end* α *T 0*
by (*simp add*: *assms*(*5*,*8*,*10*) *valid-list-bucket-end-0*)
ultimately
show *?thesis*
using *lms-locations-invD*[*OF lms-invD*(*4*)[*OF R0*], *of 0 0*] ‹*lms-bucket* α *T 0* = {*Suc n*}›
by *auto*
qed

corollary *abs-bucket-insert-bot-first*:
assumes *lms-pre* α *T B SA LMS*
and *valid-list T*
and *length T* = *Suc* (*Suc n*)
and α *bot* = *0*
shows (*abs-bucket-insert* α *T B SA LMS*) ! *0* = *Suc n*
proof −
have ∃ *SA′ B′ gs*. *bucket-insert-abs′* α *T B SA* [] *LMS* = (*SA′*, *B′*, *gs*)
by (*meson prod-cases3*)
then obtain *SA′ B′ gs* **where**
A: *bucket-insert-abs′* α *T B SA* [] *LMS* = (*SA′*, *B′*, *gs*)
by *blast*
hence *abs-bucket-insert* α *T B SA LMS* = *SA′*
by (*metis abs-bucket-insert-equiv fst-conv*)
with *lms-bot-is-first*[*OF lms-pre-elims*(*1*,*5*,*3*,*2*,*4*,*6*)[*OF assms*(*1*)] *A assms*(*2*−)]
show *?thesis*
by *simp*
qed

— Used in SAIS algorithm as part of inducing the prefix ordering based on LMS

theorem *lms-prefix-sorted-bucket*:
  assumes *lms-pre* α *T B SA LMS*
  and $b \leq \alpha$ (*Max* (*set T*))
shows *ordlistns.sorted* (*map* (*lms-prefix T*)
  (*list-slice* (*abs-bucket-insert* α *T B SA LMS*) (*lms-bucket-start* α *T b*)
(*bucket-end* α *T b*)))
  (**is** *ordlistns.sorted* (*map ?f ?SA*))
proof −
  have $\exists$ *SA′ B′ gs. bucket-insert-abs′* α *T B SA* [] *LMS* = (*SA′*, *B′*, *gs*)
    by (*meson prod-cases3*)
  then obtain *SA′ B′ gs* **where**
    A: *bucket-insert-abs′* α *T B SA* [] *LMS* = (*SA′*, *B′*, *gs*)
    by *blast*
  hence *abs-bucket-insert* α *T B SA LMS* = *SA′*
    by (*metis abs-bucket-insert-equiv fst-conv*)

  from *lms-vals-postD*[*OF abs-bucket-insert-vals*[*OF assms*(*1*)] *assms*(*2*)]
  have *P*: $\forall$ *x* $\in$ *set ?SA* . *x* < *length T*
    using *abs-is-lms-imp-less-length lms-bucket-def* **by** *blast*

  from *lms-lms-prefix-sorted-holds*[*OF lms-pre-elims*(*1*,*5*,*3*,*2*,*4*,*6*)[*OF assms*(*1*)]
A]
  have *ordlistns.sorted* (*map* (*lms-prefix T*) (*filter* (λ*x*. *x* < *length T*) *SA′*)) **.**
  hence *ordlistns.sorted* (*map* (*lms-prefix T*) (*filter* (λ*x*. *x* < *length T*) *?SA*))
    using ‹*abs-bucket-insert* α *T B SA LMS* = *SA′*› *ordlistns.sorted-map-filter-list-slice*

    by *blast*
  then show *?thesis*
    by (*simp add*: *P*)
qed

— Used in SAIS algorithm as part of inducing the suffix ordering based on LMS
theorem *lms-suffix-sorted-bucket*:
  assumes *lms-pre* α *T B SA LMS*
  and *ordlistns.sorted* (*map* (*suffix T*) (*rev LMS*))
  and $b \leq \alpha$ (*Max* (*set T*))
shows *ordlistns.sorted* (*map* (*suffix T*)
  (*list-slice* (*abs-bucket-insert* α *T B SA LMS*) (*lms-bucket-start* α *T b*)
(*bucket-end* α *T b*)))
  (**is** *ordlistns.sorted* (*map ?f ?SA*))
proof −
  have $\exists$ *SA′ B′ gs. bucket-insert-abs′* α *T B SA* [] *LMS* = (*SA′*, *B′*, *gs*)
    by (*meson prod-cases3*)
  then obtain *SA′ B′ gs* **where**
    A: *bucket-insert-abs′* α *T B SA* [] *LMS* = (*SA′*, *B′*, *gs*)
    by *blast*
  hence *abs-bucket-insert* α *T B SA LMS* = *SA′*
    by (*metis abs-bucket-insert-equiv fst-conv*)

from *lms-vals-postD*[*OF abs-bucket-insert-vals*[*OF assms*(*1*)] *assms*(*3*)]
have *P*: $\forall x \in$ *set ?SA* . $x <$ *length T*
using *abs-is-lms-imp-less-length lms-bucket-def* **by** *blast*

from *lms-suffix-sorted-holds*[*OF lms-pre-elims*(*1*,*5*,*3*,*2*,*4*,*6*)[*OF assms*(*1*)] *A assms*(*2*)]
have *ordlistns.sorted* (*map* (*suffix T*) (*filter* ($\lambda x.$ $x <$ *length T*) *SA*$'$)) **.**
hence *ordlistns.sorted* (*map* (*suffix T*) (*filter* ($\lambda x.$ $x <$ *length T*) *?SA*))
using ‹*abs-bucket-insert* α *T B SA LMS* = *SA*$'$› *ordlistns.sorted-map-filter-list-slice*
by *blast*
then show *?thesis*
by (*simp add*: *P*)
qed

end
theory *Abs-Induce-L-Verification*
imports *../abs−def/Abs-SAIS*
begin

65 Abstract Induce L-types Simple Properties

lemma *abs-induce-l-step-ex*:
$\exists$ *B*$'$ *SA*$'$ *i*$'$. *abs-induce-l-step a b* = (*B*$'$, *SA*$'$, *i*$'$)
by (*cases a*; *cases b*; *clarsimp split*: *prod.splits nat.splits SL-types.splits simp*: *Let-def*)

lemma *abs-induce-l-step-B-length*:
abs-induce-l-step (*B*, *SA*, *i*) (α, *T*) = (*B*$'$, *SA*$'$, *i*$'$) $\Longrightarrow$ *length B*$'$ = *length B*
by (*clarsimp split*: *prod.splits nat.splits SL-types.splits if-splits simp*: *Let-def*)

lemma *abs-induce-l-step-SA-length*:
abs-induce-l-step (*B*, *SA*, *i*) (α, *T*) = (*B*$'$, *SA*$'$, *i*$'$) $\Longrightarrow$ *length SA*$'$ = *length SA*
by (*clarsimp split*: *prod.splits nat.splits SL-types.splits if-splits simp*: *Let-def*)

lemma *abs-induce-l-step-Suc*:
$\exists$ *B*$'$ *SA*$'$. *abs-induce-l-step* (*B*, *SA*, *i*) (α, *T*) = (*B*$'$, *SA*$'$, *Suc i*)
by (*clarsimp simp*: *Let-def split*: *prod.splits nat.splits SL-types.splits*)

lemma *abs-induce-l-step-B-val-1*:
⟦*length SA* $\leq$ *i*; *abs-induce-l-step* (*B*, *SA*, *i*) (α, *T*) = (*B*$'$, *SA*$'$, *i*$'$)⟧ $\Longrightarrow$
B$'$ = *B*
⟦*i* $<$ *length SA*; *length T* $\leq$ *SA* ! *i*; *abs-induce-l-step* (*B*, *SA*, *i*) (α, *T*) = (*B*$'$, *SA*$'$, *i*$'$)⟧ $\Longrightarrow$
B$'$ = *B*
⟦*i* $<$ *length SA*; *SA* ! *i* $<$ *length T*; *SA* ! *i* = *0*;
abs-induce-l-step (*B*, *SA*, *i*) (α, *T*) = (*B*$'$, *SA*$'$, *i*$'$)⟧ $\Longrightarrow$
B$'$ = *B*
⟦*i* $<$ *length SA*; *SA* ! *i* $<$ *length T*; *SA* ! *i* = *Suc j*; *suffix-type T j* = *S-type*;
abs-induce-l-step (*B*, *SA*, *i*) (α, *T*) = (*B*$'$, *SA*$'$, *i*$'$)⟧ $\Longrightarrow$
B$'$ = *B*

by (*clarsimp simp*: *Let-def split*: *prod.splits nat.splits SL-types.splits if-splits*)+

lemma *abs-induce-l-step-B-val-2*:
$[\![$*strict-mono* α;
α (*Max* (*set* T)) $<$ *length* B;
$i <$ *length* SA;
SA ! $i <$ *length* T;
SA ! $i =$ *Suc* j;
suffix-type T j = *L-type*;
abs-induce-l-step (B, SA, i) (α, T) = (B', SA', i')$]\!] \Longrightarrow$
$B' = B[\alpha\ (T\ !\ j) := Suc\ (B\ !\ \alpha\ (T\ !\ j))]$
by (*clarsimp simp*: *Let-def split*: *prod.splits nat.splits SL-types.splits if-splits*)

lemma *repeat-abs-induce-l-step-index*:
$\exists B'\ SA'$. *repeat n abs-induce-l-step* (B, SA, m) (α, T) = (B', SA', $n + m$)
proof (*induct n*)
case *0*
then show *?case*
by (*simp add*: *repeat-0*)
next
case (*Suc n*)
from *this*
obtain B' SA' **where**
A: *repeat n abs-induce-l-step* (B, SA, m) (α, T) = (B', SA', $n + m$)
by *blast*

from *repeat-step*[*of n abs-induce-l-step* (B, SA, m) (α, T)]
have *B*: *repeat* (*Suc n*) *abs-induce-l-step* (B, SA, m) (α, T) =
abs-induce-l-step (*repeat n abs-induce-l-step* (B, SA, m) (α, T)) (α, T)
by *assumption*

from *abs-induce-l-step-Suc*[*of* B' SA' $n + m$ α T]
obtain B'' SA'' **where**
abs-induce-l-step (B', SA', $n + m$) (α, T) = (B'', SA'', *Suc* ($n + m$))
by *blast*
with *A B*
show *?case*
by *simp*
qed

lemma *abs-induce-l-step-lengths*:
abs-induce-l-step (B, SA, i) (α, T) = (B', SA', i') $\Longrightarrow$
length B' = *length* B $\wedge$ *length* SA' = *length* SA
by (*clarsimp split*: *if-splits nat.splits SL-types.splits simp*: *Let-def*)

lemma *repeat-abs-induce-l-step-lengths*:
repeat n abs-induce-l-step (B, SA, i) (α, T) = (B', SA', i') $\Longrightarrow$
length B' = *length* B $\wedge$ *length* SA' = *length* SA
proof −

let *?P* = λ(*a*, *b*, *c*). *length a* = *length B* ∧ *length b* = *length SA*

from *abs-induce-l-step-lengths*
have *A*: ⋀*a*. *?P a* ⟹ *?P* (*abs-induce-l-step a* (α, *T*))
by (*clarsimp simp*: *Let-def split*: *prod.splits if-splits nat.splits SL-types.splits*)

assume *repeat n abs-induce-l-step* (*B*, *SA*, *i*) (α, *T*) = (*B'*, *SA'*, *i'*)

with *repeat-maintain-inv*[*of ?P abs-induce-l-step* (α, *T*) (*B*, *SA*, *i*) *n*, *OF A*]
show *?thesis*
by *auto*
qed

lemma *abs-induce-l-index*:
∃ *B' SA'*. *abs-induce-l-base* α *T B SA* = (*B'*, *SA'*, *length T*)
by (*metis add.right-neutral abs-induce-l-base-def repeat-abs-induce-l-step-index*)

lemma *abs-induce-l-length*:
length (*abs-induce-l* α *T B SA*) = *length SA*
unfolding *abs-induce-l-def abs-induce-l-base-def*
by (*rule repeat-maintain-inv*)
(*fastforce*
simp del: *abs-induce-l-step.simps*
split: *prod.splits*
dest: *abs-induce-l-step-SA-length*)+

66 Precondition Definitions

definition *lms-init* :: (*'a* :: {*linorder*,*order-bot*} ⇒ *nat*) ⇒ *'a list* ⇒ *nat list* ⇒ *bool*
where
lms-init α *T SA* =
(∀ *b* ≤ α (*Max* (*set T*)).
lms-bucket α *T b* =
set (*list-slice SA* (*lms-bucket-start* α *T b*) (*bucket-end* α *T b*))
)

lemma *lms-init-D*:
⟦*lms-init* α *T SA*; *b* ≤ α (*Max* (*set T*))⟧ ⟹
lms-bucket α *T b* = *set* (*list-slice SA* (*lms-bucket-start* α *T b*) (*bucket-end* α *T b*))
using *lms-init-def* **by** *blast*

lemma *lms-init-nth*:
⟦*lms-init* α *T SA*;
b ≤ α (*Max* (*set T*));
lms-bucket-start α *T b* ≤ *i*;
i < *bucket-end* α *T b*;

$length\ SA = length\ T\rrbracket \Longrightarrow$
$abs\text{-}is\text{-}lms\ T\ (SA\ !\ i) \wedge \alpha\ (T\ !\ (SA\ !\ i)) = b$
by (*fastforce*
dest: *lms-init-D list-slice-nth-mem*[**where** $xs = SA$]
simp: *bucket-end-le-length lms-bucket-def bucket-def*)

lemma *lms-init-imp-distinct-bucket*:
$\llbracket lms\text{-}init\ \alpha\ T\ SA;$
$b \le \alpha\ (Max\ (set\ T));$
$length\ SA = length\ T\rrbracket \Longrightarrow$
$distinct\ (list\text{-}slice\ SA\ (lms\text{-}bucket\text{-}start\ \alpha\ T\ b)\ (bucket\text{-}end\ \alpha\ T\ b))$
by (*metis bucket-end-def′ min.absorb1 diff-diff-add bucket-end-le-length*
l-pl-pure-s-pl-lms-size lms-bucket-start-def length-list-slice
diff-add-inverse lms-init-D lms-bucket-size-def card-distinct)

lemma *lms-init-imp-all-lms-in-SA*:
assumes $lms\text{-}init\ \alpha\ T\ SA$
and $strict\text{-}mono\ \alpha$
shows $\{k\ |k.\ abs\text{-}is\text{-}lms\ T\ k\} \subseteq set\ SA$
proof
fix x
assume $x \in \{k\ |k.\ abs\text{-}is\text{-}lms\ T\ k\}$
hence $x < length\ T$
using *abs-is-lms-gre-length le-less-linear* **by** *blast*

from ‹$x \in \{k\ |k.\ abs\text{-}is\text{-}lms\ T\ k\}$›
have $abs\text{-}is\text{-}lms\ T\ x$
by *blast*
with ‹$x < length\ T$›
have $x \in lms\text{-}bucket\ \alpha\ T\ (\alpha\ (T\ !\ x))$
unfolding *lms-bucket-def bucket-def*
by *blast*

from ‹$strict\text{-}mono\ \alpha$› ‹$x < length\ T$›
have $\alpha\ (T\ !\ x) \le \alpha\ (Max\ (set\ T))$
by (*simp add*: *strict-mono-leD*)

from *lms-init-D*[*OF* ‹$lms\text{-}init\ \alpha\ T\ SA$› ‹$\alpha\ (T\ !\ x) \le \alpha\ (Max\ (set\ T))$›]
‹$x \in lms\text{-}bucket\ \alpha\ T\ (\alpha\ (T\ !\ x))$›
show $x \in set\ SA$
using *list-slice-subset* **by** *force*
qed

definition *s-init* :: $('a :: \{linorder,order\text{-}bot\} \Rightarrow nat) \Rightarrow 'a\ list \Rightarrow nat\ list \Rightarrow bool$
where
$s\text{-}init\ \alpha\ T\ SA =$
$(\forall\, b \le \alpha\ (Max\ (set\ T)).$
$\forall\, i < length\ SA.\ l\text{-}bucket\text{-}end\ \alpha\ T\ b \le i \wedge i < lms\text{-}bucket\text{-}start\ \alpha\ T\ b \longrightarrow SA$

! i = *length T*
)

lemma *s-init-D*:
  ⟦*s-init* α *T SA*;
    $b \leq \alpha$ (*Max* (*set T*));
    $i <$ *length SA*;
    l-bucket-end α *T b* $\leq i$;
    $i <$ *lms-bucket-start* α *T b*⟧ $\Longrightarrow$
    SA ! i = *length T*
  using *s-init-def* **by** *blast*

definition *l-init* :: (*'a* :: {*linorder*,*order-bot*} ⇒ *nat*) ⇒ *'a list* ⇒ *nat list* ⇒ *bool*
  where
l-init α *T SA* =
  ($\forall b \leq \alpha$ (*Max* (*set T*)).
    $\forall i <$ *length SA*. *bucket-start* α *T b* $\leq i \wedge i <$ *l-bucket-end* α *T b* $\longrightarrow$ *SA* ! i =
length T
)

lemma *l-init-D*:
  ⟦*l-init* α *T SA*;
    $b \leq \alpha$ (*Max* (*set T*));
    $i <$ *length SA*;
    bucket-start α *T b* $\leq i$;
    $i <$ *l-bucket-end* α *T b*⟧ $\Longrightarrow$
    SA ! i = *length T*
  using *l-init-def* **by** *blast*

lemma *init-imp-lms-range*:
  assumes *lms-init* α *T SA*
  and *l-init* α *T SA*
  and *s-init* α *T SA*
  and *length SA* = *length T*
  and *strict-mono* α
  and $i <$ *length SA*
  and *SA* ! $i = j$
  and $j <$ *length T*
  shows *lms-bucket-start* α *T* (α (*T* ! j)) $\leq i \wedge i <$ *bucket-end* α *T* (α (*T* ! j))
proof −
  from ‹$i <$ *length SA*› ‹*length SA* = *length T*›
  have $i <$ *length T*
    by *simp*

  from *index-in-bucket-interval-gen*[*OF* ‹$i <$ *length T*› ‹*strict-mono* α›]
  obtain *b* **where**
    $b \leq \alpha$ (*Max* (*set T*))
    bucket-start α *T b* $\leq i$
    $i <$ *bucket-end* α *T b*

```
    by blast

  have i < l-bucket-end α T b ⟶ False
  proof
    assume i < l-bucket-end α T b
    with l-init-D[OF ‹l-init α T SA› ‹b ≤ α (Max (set T))› ‹i < length SA›]
         ‹bucket-start α T b ≤ i›
         ‹SA ! i = j›
         ‹j < length T›
    show False
      by simp
  qed

  have l-bucket-end α T b ≤ i ∧ i < lms-bucket-start α T b ⟶ False
  proof(intro impI; elim conjE)
    assume l-bucket-end α T b ≤ i i < lms-bucket-start α T b
    with s-init-D[OF ‹s-init α T SA› ‹b ≤ α (Max (set T))› ‹i < length SA›]
         ‹SA ! i = j›
         ‹j < length T›
    show False
      by simp
  qed
  with ‹i < l-bucket-end α T b ⟶ False›
       ‹b ≤ α (Max (set T))›
       ‹i < bucket-end α T b›
       ‹lms-init α T SA›
       ‹length SA = length T›
       ‹SA ! i = j›
  show ?thesis
    by (metis lms-init-nth not-less)
qed

lemma init-imp-only-lms-types:
  assumes lms-init α T SA
  and     l-init α T SA
  and     s-init α T SA
  and     length SA = length T
  and     strict-mono α
  shows ∀ i < length SA. SA ! i < length T ⟶ abs-is-lms T (SA ! i)
proof (intro allI impI)
  fix i
  assume i < length SA SA ! i < length T
  with init-imp-lms-range[OF ‹lms-init α T SA›
                             ‹l-init α T SA›
                             ‹s-init α T SA›
                             ‹length SA = length T›
                             ‹strict-mono α›
                             ‹i < length SA›]
  have lms-bucket-start α T (α (T ! (SA ! i))) ≤ i ∧ i < bucket-end α T (α (T !
```

$(SA ! i)))$
  by *simp*
 with ‹$SA ! i < length\ T$› ‹$lms\text{-}init\ \alpha\ T\ SA$› ‹$length\ SA = length\ T$› ‹$strict\text{-}mono\ \alpha$›
 show $abs\text{-}is\text{-}lms\ T\ (SA ! i)$
  by (*meson Max-greD lms-init-nth strict-mono-less-eq*)
qed

lemma *init-imp-only-s-types*:
 assumes $lms\text{-}init\ \alpha\ T\ SA$
 and $l\text{-}init\ \alpha\ T\ SA$
 and $s\text{-}init\ \alpha\ T\ SA$
 and $length\ SA = length\ T$
 and $strict\text{-}mono\ \alpha$
shows $\forall i < length\ SA.\ SA ! i < length\ T \longrightarrow suffix\text{-}type\ T\ (SA ! i) = S\text{-}type$
 using *assms init-imp-only-lms-types abs-is-lms-def* **by** *blast*

definition *lms-sorted-init* ::
 $('a :: \{linorder, order\text{-}bot\} \Rightarrow nat) \Rightarrow$
  $('a\ list \Rightarrow nat \Rightarrow 'a\ list) \Rightarrow$
  $'a\ list \Rightarrow$
  $nat\ list \Rightarrow$
  $bool$
 where
$lms\text{-}sorted\text{-}init\ \alpha\ f\ T\ SA =$
 $(\forall b \leq \alpha\ (Max\ (set\ T)).$
   $ordlistns.sorted\ (map\ (f\ T)\ (list\text{-}slice\ SA\ (lms\text{-}bucket\text{-}start\ \alpha\ T\ b)\ (bucket\text{-}end\ \alpha\ T\ b)))$
 $)$

lemma *lms-sorted-init-D*:
 $[\![lms\text{-}sorted\text{-}init\ \alpha\ f\ T\ SA;\ b \leq \alpha\ (Max\ (set\ T))]\!] \Longrightarrow$
   $ordlistns.sorted\ (map\ (f\ T)\ (list\text{-}slice\ SA\ (lms\text{-}bucket\text{-}start\ \alpha\ T\ b)\ (bucket\text{-}end\ \alpha\ T\ b)))$
 using *lms-sorted-init-def* **by** *blast*

definition *l-suffix-sorted-pre* ::
 $('a :: \{linorder, order\text{-}bot\} \Rightarrow nat) \Rightarrow 'a\ list \Rightarrow nat\ list \Rightarrow bool$
 where
$l\text{-}suffix\text{-}sorted\text{-}pre\ \alpha\ T\ SA =$
 $(\forall b \leq \alpha\ (Max\ (set\ T)).$
  $ordlistns.sorted\ (map\ (suffix\ T)\ (list\text{-}slice\ SA\ (lms\text{-}bucket\text{-}start\ \alpha\ T\ b)\ (bucket\text{-}end\ \alpha\ T\ b)))$
 $)$

lemma *l-suffix-sorted-preD*:
 $[\![l\text{-}suffix\text{-}sorted\text{-}pre\ \alpha\ T\ SA;\ b \leq \alpha\ (Max\ (set\ T))]\!] \Longrightarrow$
  $ordlistns.sorted\ (map\ (suffix\ T)\ (list\text{-}slice\ SA\ (lms\text{-}bucket\text{-}start\ \alpha\ T\ b)\ (bucket\text{-}end\ \alpha\ T\ b)))$

using *l-suffix-sorted-pre-def* **by** *blast*

definition *l-prefix-sorted-pre* ::
$('a :: \{linorder,\ order\text{-}bot\} \Rightarrow nat) \Rightarrow 'a\ list \Rightarrow nat\ list \Rightarrow bool$
where
l-prefix-sorted-pre α T SA =
$(\forall\, b \leq \alpha\ (Max\ (set\ T)).$
ordlistns.sorted (*map* (*lms-prefix* T) (*list-slice* SA (*lms-bucket-start* α T b) (*bucket-end* α T b)))
)

lemma *l-prefix-sorted-preD*:
$[\![$*l-prefix-sorted-pre* α T SA; $b \leq \alpha\ (Max\ (set\ T))]\!] \Longrightarrow$
ordlistns.sorted (*map* (*lms-prefix* T) (*list-slice* SA (*lms-bucket-start* α T b) (*bucket-end* α T b)))
using *l-prefix-sorted-pre-def* **by** *blast*

definition *l-perm-pre* ::
$('a :: \{linorder,\ order\text{-}bot\} \Rightarrow nat) \Rightarrow$
$'a\ list \Rightarrow$
$nat\ list \Rightarrow$
$nat\ list \Rightarrow$
$bool$
where
l-perm-pre α T B SA =
(*lms-init* α T SA $\wedge$
l-init α T SA $\wedge$
s-init α T SA $\wedge$
l-bucket-init α T B $\wedge$
$T \neq [] \wedge$
strict-mono α $\wedge$
length SA = *length* T $\wedge$
$\alpha\ (Max\ (set\ T)) <$ *length* B)

lemma *l-perm-pre-elims*:
l-perm-pre α T B $SA \Longrightarrow$ *lms-init* α T SA
l-perm-pre α T B $SA \Longrightarrow$ *l-init* α T SA
l-perm-pre α T B $SA \Longrightarrow$ *s-init* α T SA
l-perm-pre α T B $SA \Longrightarrow$ *l-bucket-init* α T B
l-perm-pre α T B $SA \Longrightarrow T \neq []$
l-perm-pre α T B $SA \Longrightarrow$ *strict-mono* α
l-perm-pre α T B $SA \Longrightarrow$ *length* SA = *length* T
l-perm-pre α T B $SA \Longrightarrow \alpha\ (Max\ (set\ T)) <$ *length* B
unfolding *l-perm-pre-def* **by** *blast+*

67 Invariant Definitions

This section contains all the various invariants that we need for the *abs-induce-l* subroutine.

67.1 Distinctness

definition *l-distinct-inv* :: ($'a$ :: {*linorder*, *order-bot*}) *list* ⇒ *nat list* ⇒ *bool*
  where
l-distinct-inv T SA = *distinct* (*filter* (λ*x*. *x* < *length T*) *SA*)

lemma *l-distinct-inv-D*:
  assumes *l-distinct-inv T SA*
  and i < *length SA*
  and j < *length SA*
  and $i \neq j$
  and *SA* ! i < *length T*
  and *SA* ! j < *length T*
  shows *SA* ! i ≠ *SA* ! j
proof −
  from *filter-nth-relative-neq-1*[**where** P = λ*x*. *x* < *length T*,
      OF ‹i < *length SA*›
        ‹*SA* ! i < *length T*›
        ‹j < *length SA*›
        ‹*SA* ! j < *length T*›
        ‹$i \neq j$›]
  obtain i' j' **where**
    i' < *length* (*filter* (λ*x*. *x* < *length T*) *SA*)
    j' < *length* (*filter* (λ*x*. *x* < *length T*) *SA*)
    filter (λ*x*. *x* < *length T*) *SA* ! i' = *SA* ! i
    filter (λ*x*. *x* < *length T*) *SA* ! j' = *SA* ! j
    $i' \neq j'$
    by *blast*

  from *distinct-conv-nth*[*THEN iffD1*,
      OF l-distinct-inv-def[*THEN iffD1*],
      OF ‹*l-distinct-inv T SA*›]
    ‹*filter* (λ*x*. *x* < *length T*) *SA* ! i' = *SA* ! i›
    ‹*filter* (λ*x*. *x* < *length T*) *SA* ! j' = *SA* ! j›
    ‹i' < *length* (*filter* (λ*x*. *x* < *length T*) *SA*)›
    ‹$i' \neq j'$›
    ‹j' < *length* (*filter* (λ*x*. *x* < *length T*) *SA*)›
  show *?thesis*
    by *fastforce*
qed

67.2 Predecessor

definition *l-pred-inv* :: ($'a$ :: {*linorder*, *order-bot*}) *list* ⇒ *nat list* ⇒ *nat* ⇒ *bool*

where
$l\text{-}pred\text{-}inv\ T\ SA\ k =$
$(\forall i < length\ SA.\ SA\ !\ i < length\ T \wedge suffix\text{-}type\ T\ (SA\ !\ i) = L\text{-}type \longrightarrow$
$(\exists j < length\ SA.\ SA\ !\ j = Suc\ (SA\ !\ i) \wedge j < i \wedge j < k))$

lemma *l-pred-inv-D*:
$[\![l\text{-}pred\text{-}inv\ T\ SA\ k;\ i < length\ SA;\ SA\ !\ i < length\ T;\ suffix\text{-}type\ T\ (SA\ !\ i) = L\text{-}type]\!] \Longrightarrow$
$\exists j < length\ SA.\ SA\ !\ j = Suc\ (SA\ !\ i) \wedge SA\ !\ j < length\ T \wedge j < i \wedge j < k$
by (*metis SL-types.simps*(*2*) *Suc-lessI l-pred-inv-def suffix-type-last*)

67.3 L Bucket Ptr

We prove that the pointer for each bucket is related to the number of L-types currently in SA. That is, if we subtract the original pointer with the current, we should have the number of L-types currently in SA for each symbol.

definition *cur-l-types* ::
$('a :: \{linorder,\ order\text{-}bot\} \Rightarrow nat) \Rightarrow 'a\ list \Rightarrow nat\ list \Rightarrow nat \Rightarrow nat\ set$
where
$cur\text{-}l\text{-}types\ \alpha\ T\ SA\ b = \{i|i.\ i \in set\ SA \wedge i \in l\text{-}bucket\ \alpha\ T\ b\ \}$

definition *num-l-types* ::
$('a :: \{linorder,\ order\text{-}bot\} \Rightarrow nat) \Rightarrow 'a\ list \Rightarrow nat\ list \Rightarrow nat \Rightarrow nat$
where
$num\text{-}l\text{-}types\ \alpha\ T\ SA\ b = card\ (cur\text{-}l\text{-}types\ \alpha\ T\ SA\ b)$

definition *l-bucket-ptr-inv* ::
$('a :: \{linorder,\ order\text{-}bot\} \Rightarrow nat) \Rightarrow 'a\ list \Rightarrow nat\ list \Rightarrow nat\ list \Rightarrow bool$
where
$l\text{-}bucket\text{-}ptr\text{-}inv\ \alpha\ T\ B\ SA \equiv$
$(\forall b \leq \alpha\ (Max\ (set\ T)).\ B\ !\ b = bucket\text{-}start\ \alpha\ T\ b + num\text{-}l\text{-}types\ \alpha\ T\ SA\ b)$

lemma *l-bucket-ptr-inv-D*:
$[\![l\text{-}bucket\text{-}ptr\text{-}inv\ \alpha\ T\ B\ SA;\ b \leq \alpha\ (Max\ (set\ T))]\!] \Longrightarrow$
$B\ !\ b = bucket\text{-}start\ \alpha\ T\ b + num\text{-}l\text{-}types\ \alpha\ T\ SA\ b$
using *l-bucket-ptr-inv-def* **by** *blast*

67.4 Unknowns

definition *l-unknowns-inv* ::
$('a :: \{linorder,\ order\text{-}bot\} \Rightarrow nat) \Rightarrow 'a\ list \Rightarrow nat\ list \Rightarrow nat\ list \Rightarrow bool$
where
$l\text{-}unknowns\text{-}inv\ \alpha\ T\ B\ SA \equiv$
$(\forall a \leq \alpha\ (Max\ (set\ T)).\ \forall k.\ B\ !\ a \leq k \wedge k < l\text{-}bucket\text{-}end\ \alpha\ T\ a \longrightarrow SA\ !\ k = length\ T)$

lemma *l-unknowns-inv-D*:
$[\![l\text{-}unknowns\text{-}inv\ \alpha\ T\ B\ SA;\ b \leq \alpha\ (Max\ (set\ T));\ B\ !\ b \leq k;\ k < l\text{-}bucket\text{-}end\ \alpha\ T\ b]\!] \Longrightarrow$

$SA ! k = length\ T$
using *l-unknowns-inv-def* **by** *blast*

67.5 Indexes

definition *l-index-inv* ::
$('a :: \{linorder,\ order\text{-}bot\} \Rightarrow nat) \Rightarrow 'a\ list \Rightarrow nat\ list \Rightarrow nat\ list \Rightarrow bool$
where
$l\text{-}index\text{-}inv\ \alpha\ T\ B\ SA \equiv$
$(\forall i < length\ SA.$
$(\forall j.\ SA ! i = Suc\ j \wedge Suc\ j < length\ T \wedge suffix\text{-}type\ T\ j = L\text{-}type \longrightarrow$
$i < B ! (\alpha\ (T ! j))$
$)$
$)$

lemma *l-index-inv-D*:
$[\![l\text{-}index\text{-}inv\ \alpha\ T\ B\ SA;\ i < length\ SA;\ SA ! i = Suc\ j;\ Suc\ j < length\ T;\ suffix\text{-}type\ T\ j = L\text{-}type]\!] \Longrightarrow$
$i < B ! (\alpha\ (T ! j))$
using *l-index-inv-def* **by** *blast*

67.6 Unchanged

definition *l-unchanged-inv* ::
$('a :: \{linorder,order\text{-}bot\} \Rightarrow nat) \Rightarrow 'a\ list \Rightarrow nat\ list \Rightarrow nat\ list \Rightarrow bool$
where
$l\text{-}unchanged\text{-}inv\ \alpha\ T\ SA\ SA' =$
$((length\ SA' = length\ SA) \wedge$
$(\forall b \leq \alpha\ (Max\ (set\ T)).$
$(\forall i < length\ SA.\ l\text{-}bucket\text{-}end\ \alpha\ T\ b \leq i \wedge i < bucket\text{-}end\ \alpha\ T\ b \longrightarrow SA ! i = SA' ! i)$
$))$

lemma *l-unchanged-inv-trans*:
$[\![l\text{-}unchanged\text{-}inv\ \alpha\ T\ SA0\ SA1;\ l\text{-}unchanged\text{-}inv\ \alpha\ T\ SA1\ SA2]\!] \Longrightarrow$
$l\text{-}unchanged\text{-}inv\ \alpha\ T\ SA0\ SA2$
by (*simp add*: *l-unchanged-inv-def*)

lemma *l-unchanged-inv-D*:
$[\![l\text{-}unchanged\text{-}inv\ \alpha\ T\ SA\ SA';\ length\ SA' = length\ SA;\ b \leq \alpha\ (Max\ (set\ T));$
$i < length\ SA;\ l\text{-}bucket\text{-}end\ \alpha\ T\ b \leq i;\ i < bucket\text{-}end\ \alpha\ T\ b]\!] \Longrightarrow$
$SA ! i = SA' ! i$
using *l-unchanged-inv-def* **by** *blast*

67.7 L Locations

definition *l-locations-inv* ::
$('a :: \{linorder,order\text{-}bot\} \Rightarrow nat) \Rightarrow 'a\ list \Rightarrow nat\ list \Rightarrow nat\ list \Rightarrow bool$
where
$l\text{-}locations\text{-}inv\ \alpha\ T\ B\ SA =$

$(\forall b \leq \alpha\ (Max\ (set\ T)).$
$\quad (\forall i < length\ SA.\ bucket\text{-}start\ \alpha\ T\ b \leq i \wedge i < B\ !\ b \longrightarrow$
$\quad\quad SA\ !\ i < length\ T \wedge suffix\text{-}type\ T\ (SA\ !\ i) = L\text{-}type \wedge \alpha\ (T\ !\ (SA\ !\ i)) = b$
$\quad)$
$)$

lemma *l-locations-inv-D*:
$[\![l\text{-}locations\text{-}inv\ \alpha\ T\ B\ SA;$
$\quad b \leq \alpha\ (Max\ (set\ T));$
$\quad i < length\ SA;$
$\quad bucket\text{-}start\ \alpha\ T\ b \leq i;$
$\quad i < B\ !\ b]\!] \Longrightarrow$
$\quad SA\ !\ i < length\ T \wedge suffix\text{-}type\ T\ (SA\ !\ i) = L\text{-}type \wedge \alpha\ (T\ !\ (SA\ !\ i)) = b$
using *l-locations-inv-def* **by** *blast*

lemma *l-locations-list-slice*:
assumes $l\text{-}locations\text{-}inv\ \alpha\ T\ B\ SA$
and $b \leq \alpha\ (Max\ (set\ T))$
shows $set\ (list\text{-}slice\ SA\ (bucket\text{-}start\ \alpha\ T\ b)\ (B\ !\ b)) \subseteq l\text{-}bucket\ \alpha\ T\ b$
(**is** $set\ ?xs \subseteq l\text{-}bucket\ \alpha\ T\ b$)
proof
fix x
assume $x \in set\ ?xs$

from *nth-mem-list-slice*[*OF* ‹$x \in set\ ?xs$›]
obtain i **where**
$i < length\ SA$
$bucket\text{-}start\ \alpha\ T\ b \leq i$
$i < B\ !\ b$
$SA\ !\ i = x$
by *blast*
with *l-locations-inv-D*[*OF assms*, *of i*]
have $x < length\ T\ suffix\text{-}type\ T\ x = L\text{-}type\ \alpha\ (T\ !\ x) = b$
by *blast+*
then show $x \in l\text{-}bucket\ \alpha\ T\ b$
by (*simp add*: *bucket-def l-bucket-def*)
qed

67.8 Seen

In this section, we prove that the seen invariant is maintained. In English, this invariant states for all L-type suffixes, excluding the one that starts at position 0, in the suffix array (SA) and that are less than the current index, their left neighbour is also in SA.

definition $l\text{-}seen\text{-}inv :: ('a :: \{linorder, order\text{-}bot\})\ list \Rightarrow nat\ list \Rightarrow nat \Rightarrow bool$
where
$l\text{-}seen\text{-}inv\ T\ SA\ n \equiv \forall i < n.\ i < length\ SA \wedge SA\ !\ i < length\ T \longrightarrow$
$\quad (\forall j.\ SA\ !\ i = Suc\ j \wedge suffix\text{-}type\ T\ j = L\text{-}type \longrightarrow$
$\quad\quad (\exists k < length\ SA.\ SA\ !\ k = j))$

lemma *l-seen-inv-nth-ex*:
  $[\![$*l-seen-inv T SA n*; $i < n$; $i <$ *length SA*; $SA\ !\ i <$ *length T*; $SA\ !\ i =$ *Suc j*;
    suffix-type T j $=$ *L-type*$]\!] \Longrightarrow$
  $\exists\, k <$ *length SA*. $SA\ !\ k = j$
  using *l-seen-inv-def* **by** *blast*

67.9 Sortedness

definition *abs-induce-l-sorted* ::
  $(('a :: \{linorder,order\text{-}bot\})\ list \Rightarrow nat \Rightarrow 'a\ list) \Rightarrow 'a\ list \Rightarrow nat\ list \Rightarrow bool$
  where
abs-induce-l-sorted f T SA $=$ *ordlistns.sorted* (*map* (*f T*) (*filter* ($\lambda x.\ x <$ *length T*) *SA*))

lemma *abs-induce-l-sorted-nth*:
  assumes *abs-induce-l-sorted f T SA*
  and $i < j$
  and $j <$ *length SA*
  and $SA\ !\ i <$ *length T*
  and $SA\ !\ j <$ *length T*
  shows *list-less-eq-ns* (*f T* ($SA\ !\ i$)) (*f T* ($SA\ !\ j$))
proof −
  let *?SA* $=$ *filter* ($\lambda x.\ x <$ *length T*) *SA* **and**
    ?le $=$ ($\lambda x\ y.$ *list-less-eq-ns* (*f T x*) (*f T y*))
  from *filter-nth-relative-1*[**where** $P = (\lambda x.\ x <$ *length T*$)$,
                  OF ‹$j <$ *length SA*›
                    ‹$SA\ !\ j <$ *length T*›
                    ‹$i < j$›
                    ‹$SA\ !\ i <$ *length T*›]
  obtain i' j' **where**
    $j' <$ *length ?SA*
    $i' < j'$
    ?SA $!\ j' = SA\ !\ j$
    ?SA $!\ i' = SA\ !\ i$
    by *blast*

  from *ordlistns.sorted-map*
    ‹*abs-induce-l-sorted f T SA*›
  have *sorted-wrt ?le ?SA*
    unfolding *abs-induce-l-sorted-def*
    by *blast*

  from *sorted-wrt-nth-less*[*OF* ‹*sorted-wrt ?le ?SA*›
                  ‹$i' < j'$›
                  ‹$j' <$ *length ?SA*›]
  have *list-less-eq-ns* (*f T* (*?SA* $!\ i'$)) (*f T* (*?SA* $!\ j'$))
    by *assumption*
  with ‹*?SA* $!\ i' = SA\ !\ i$› ‹*?SA* $!\ j' = SA\ !\ j$›

show *?thesis*
  by *simp*
qed

definition *l-suffix-sorted-inv* ::
  $(('a :: \{linorder,order\text{-}bot\}) \Rightarrow nat) \Rightarrow 'a\ list \Rightarrow nat\ list \Rightarrow nat\ list \Rightarrow bool$
  where
$l\text{-}suffix\text{-}sorted\text{-}inv\ \alpha\ T\ B\ SA =$
  $(\forall b \leq \alpha\ (Max\ (set\ T)).$
    $ordlistns.sorted\ (map\ (suffix\ T)\ (list\text{-}slice\ SA\ (bucket\text{-}start\ \alpha\ T\ b)\ (B\ !\ b))))$

lemma *l-suffix-sorted-invD*:
  $[\![l\text{-}suffix\text{-}sorted\text{-}inv\ \alpha\ T\ B\ SA;\ b \leq \alpha\ (Max\ (set\ T))]\!] \Longrightarrow$
    $ordlistns.sorted\ (map\ (suffix\ T)\ (list\text{-}slice\ SA\ (bucket\text{-}start\ \alpha\ T\ b)\ (B\ !\ b)))$
  using *l-suffix-sorted-inv-def* **by** *blast*

definition *l-prefix-sorted-inv* ::
  $(('a :: \{linorder,order\text{-}bot\}) \Rightarrow nat) \Rightarrow 'a\ list \Rightarrow nat\ list \Rightarrow nat\ list \Rightarrow bool$
  where
$l\text{-}prefix\text{-}sorted\text{-}inv\ \alpha\ T\ B\ SA =$
  $(\forall b \leq \alpha\ (Max\ (set\ T)).$
    $ordlistns.sorted\ (map\ (lms\text{-}prefix\ T)\ (list\text{-}slice\ SA\ (bucket\text{-}start\ \alpha\ T\ b)\ (B\ !\ b))))$

lemma *l-prefix-sorted-invD*:
  $[\![l\text{-}prefix\text{-}sorted\text{-}inv\ \alpha\ T\ B\ SA;\ b \leq \alpha\ (Max\ (set\ T))]\!] \Longrightarrow$
    $ordlistns.sorted\ (map\ (lms\text{-}prefix\ T)\ (list\text{-}slice\ SA\ (bucket\text{-}start\ \alpha\ T\ b)\ (B\ !\ b)))$
  using *l-prefix-sorted-inv-def* **by** *blast*

67.10 Permutation

definition *l-perm-inv* ::
  $('a :: \{linorder,\ order\text{-}bot\} \Rightarrow nat) \Rightarrow$
   $'a\ list \Rightarrow$
   $nat\ list \Rightarrow$
   $nat\ list \Rightarrow$
   $nat\ list \Rightarrow$
   $nat \Rightarrow$
   $bool$
  where
$l\text{-}perm\text{-}inv\ \alpha\ T\ B\ SA\ SA'\ i \equiv$
  $\alpha\ (Max\ (set\ T)) < length\ B\ \wedge$
  $length\ SA = length\ T\ \wedge$
  $length\ SA' = length\ SA\ \wedge$
  $l\text{-}distinct\text{-}inv\ T\ SA'\ \wedge$
  $l\text{-}unknowns\text{-}inv\ \alpha\ T\ B\ SA'\ \wedge$
  $l\text{-}bucket\text{-}ptr\text{-}inv\ \alpha\ T\ B\ SA'\ \wedge$
  $l\text{-}index\text{-}inv\ \alpha\ T\ B\ SA'\ \wedge$
  $l\text{-}unchanged\text{-}inv\ \alpha\ T\ SA\ SA'\ \wedge$
  $l\text{-}locations\text{-}inv\ \alpha\ T\ B\ SA'\ \wedge$

$l\text{-}pred\text{-}inv\ T\ SA'\ i \wedge$
$l\text{-}seen\text{-}inv\ T\ SA'\ i \wedge$
$strict\text{-}mono\ \alpha \wedge$
$T \neq [] \wedge$
$lms\text{-}init\ \alpha\ T\ SA \wedge$
$s\text{-}init\ \alpha\ T\ SA$

lemma *l-perm-inv-elims*:
$l\text{-}perm\text{-}inv\ \alpha\ T\ B\ SA\ SA'\ i \Longrightarrow \alpha\ (Max\ (set\ T)) < length\ B$
$l\text{-}perm\text{-}inv\ \alpha\ T\ B\ SA\ SA'\ i \Longrightarrow length\ SA = length\ T$
$l\text{-}perm\text{-}inv\ \alpha\ T\ B\ SA\ SA'\ i \Longrightarrow length\ SA' = length\ SA$
$l\text{-}perm\text{-}inv\ \alpha\ T\ B\ SA\ SA'\ i \Longrightarrow l\text{-}distinct\text{-}inv\ T\ SA'$
$l\text{-}perm\text{-}inv\ \alpha\ T\ B\ SA\ SA'\ i \Longrightarrow l\text{-}unknowns\text{-}inv\ \alpha\ T\ B\ SA'$
$l\text{-}perm\text{-}inv\ \alpha\ T\ B\ SA\ SA'\ i \Longrightarrow l\text{-}bucket\text{-}ptr\text{-}inv\ \alpha\ T\ B\ SA'$
$l\text{-}perm\text{-}inv\ \alpha\ T\ B\ SA\ SA'\ i \Longrightarrow l\text{-}index\text{-}inv\ \alpha\ T\ B\ SA'$
$l\text{-}perm\text{-}inv\ \alpha\ T\ B\ SA\ SA'\ i \Longrightarrow l\text{-}unchanged\text{-}inv\ \alpha\ T\ SA\ SA'$
$l\text{-}perm\text{-}inv\ \alpha\ T\ B\ SA\ SA'\ i \Longrightarrow l\text{-}locations\text{-}inv\ \alpha\ T\ B\ SA'$
$l\text{-}perm\text{-}inv\ \alpha\ T\ B\ SA\ SA'\ i \Longrightarrow l\text{-}pred\text{-}inv\ T\ SA'\ i$
$l\text{-}perm\text{-}inv\ \alpha\ T\ B\ SA\ SA'\ i \Longrightarrow l\text{-}seen\text{-}inv\ T\ SA'\ i$
$l\text{-}perm\text{-}inv\ \alpha\ T\ B\ SA\ SA'\ i \Longrightarrow strict\text{-}mono\ \alpha$
$l\text{-}perm\text{-}inv\ \alpha\ T\ B\ SA\ SA'\ i \Longrightarrow T \neq []$
$l\text{-}perm\text{-}inv\ \alpha\ T\ B\ SA\ SA'\ i \Longrightarrow lms\text{-}init\ \alpha\ T\ SA$
$l\text{-}perm\text{-}inv\ \alpha\ T\ B\ SA\ SA'\ i \Longrightarrow s\text{-}init\ \alpha\ T\ SA$
by (*simp add*: *l-perm-inv-def*)+

68 Invariant Helpers

68.1 Distinctness of New Insert

We prove that the next item to be inserted cannot already be in the suffix array.

lemma *l-distinct-pred-inv-helper*:
assumes $i < length\ SA$
and $SA\ !\ i = Suc\ j$
and $Suc\ j < length\ T$
and $suffix\text{-}type\ T\ j = L\text{-}type$
and $l\text{-}distinct\text{-}inv\ T\ SA$
and $l\text{-}pred\text{-}inv\ T\ SA\ i$
shows $j \notin set\ SA$
proof
assume $j \in set\ SA$
then obtain l **where**
$l < length\ SA$
$SA\ !\ l = j$
by (*meson in-set-conv-nth*)

from *l-pred-inv-D*[*OF* ‹*l-pred-inv T SA i*› ‹$l < length\ SA$›]
‹$SA\ !\ l = j$›

```
      ‹SA ! i = Suc j›
      ‹Suc j < length T›
      ‹suffix-type T j = L-type›
  obtain i' where
    i' < length SA
    SA ! i' = Suc j
    i' < l
    i' < i
    by auto

  from ‹SA ! i = Suc j› ‹SA ! i' = Suc j›
  have SA ! i = SA ! i'
    by simp

  from ‹SA ! i' = Suc j› ‹Suc j < length T›
  have SA ! i' < length T
    by simp

  from ‹i' < i›
  have i ≠ i'
    by simp

  from ‹SA ! i = Suc j› ‹Suc j < length T›
  have SA ! i < length T
    by simp

  from l-distinct-inv-D[OF ‹l-distinct-inv T SA›
                           ‹i < length SA›
                           ‹i' < length SA›
                           ‹i ≠ i'›
                           ‹SA ! i < length T›
                           ‹SA ! i' < length T›]
      ‹SA ! i = SA ! i'›
  show False
    by simp
qed

lemma l-distinct-slice:
  assumes l-distinct-inv T SA
  and     l-locations-inv α T B SA
  and     length SA = length T
  and     b ≤ α (Max (set T))
shows distinct (list-slice SA (bucket-start α T b) (B ! b))
     (is distinct ?xs)
proof (intro distinct-conv-nth[THEN iffD2] allI impI)
  fix i j
  assume i < length ?xs j < length ?xs i ≠ j

  let ?i = bucket-start α T b + i
```

and *?j* = *bucket-start* α *T b* + *j*

have *SA* ! *?i* = *?xs* ! *i*
  using ‹*i* < *length ?xs*› *nth-list-slice* **by** *force*

have *SA* ! *?j* = *?xs* ! *j*
  using ‹*j* < *length ?xs*› *nth-list-slice* **by** *force*

from ‹*i* $\neq$ *j*›
have *?i* $\neq$ *?j*
  by *auto*
moreover
from ‹*i* < *length ?xs*›
have *?i* < *length SA ?i* < *B* ! *b*
  by *auto*
with *l-locations-inv-D*[*OF assms*(*2*,*4*), *of ?i*]
have *SA* ! *?i* < *length T*
  using *le-add1* **by** *blast*
moreover
from ‹*j* < *length ?xs*›
have *?j* < *length SA ?j* < *B* ! *b*
  by *auto*
with *l-locations-inv-D*[*OF assms*(*2*,*4*), *of ?j*]
have *SA* ! *?j* < *length T*
  using *le-add1* **by** *blast*
ultimately have *SA* ! *?i* $\neq$ *SA* ! *?j*
  using *l-distinct-inv-D*[*OF assms*(*1*) ‹*?i* < *length SA*› ‹*?j* < *length SA*›]
  by *blast*
with ‹*SA* ! *?i* = *?xs* ! *i*› ‹*SA* ! *?j* = *?xs* ! *j*›
show *?xs* ! *i* $\neq$ *?xs* ! *j*
  by *simp*
qed

68.2 Bucket Ranges

lemma *num-l-types-le-l-bucket-size*:
  num-l-types α *T SA b* $\leq$ *l-bucket-size* α *T b*
  by (*metis* (*no-types*, *lifting*) *card-mono cur-l-types-def finite-l-bucket l-bucket-size-def mem-Collect-eq num-l-types-def subsetI*)

lemma *num-l-types-less-l-bucket-size*:
  $[\![$*j* $\notin$ *set SA*; *suffix-type T j* = *L-type*; α (*T* ! *j*) = *b*; *j* < *length T*$]\!] \Longrightarrow$
  num-l-types α *T SA b* < *l-bucket-size* α *T b*
  apply (*clarsimp simp*: *num-l-types-def cur-l-types-def l-bucket-size-def*)
  apply (*rule psubset-card-mono*)
   apply (*simp add*: *finite-l-bucket*)
  apply (*rule psubsetI*)
   apply (*rule subsetI*)
   apply *blast*

apply *clarsimp*
apply (*drule equalityD2*)
apply (*drule subsetD*[**where** $c = j$])
apply (*simp add*: *bucket-def l-bucket-def*)
apply *blast*
done

lemma *l-bucket-ptr-inv-imp-le-l-bucket-end*:
$[\![$*l-bucket-ptr-inv* α T B SA; $b \leq \alpha$ (*Max* (*set* T))$]\!] \Longrightarrow$
B ! $b \leq$ *l-bucket-end* α T b
apply (*drule* (*1*) *l-bucket-ptr-inv-D*)
by (*simp add*: *l-bucket-end-def num-l-types-le-l-bucket-size*)

lemma *l-bucket-ptr-inv-imp-less-l-bucket-end*:
$[\![$*l-bucket-ptr-inv* α T B SA; $j <$ *length* T; *suffix-type* T j = *L-type*; $j \notin$ *set* SA; *strict-mono* $\alpha]\!] \Longrightarrow$
B ! (α (T ! j)) $<$ *l-bucket-end* α T (α (T ! j))
apply (*frule num-l-types-less-l-bucket-size*[**where** $\alpha = \alpha$ **and** $b = \alpha$ (T ! j)]; *assumption?*)
apply *simp*
apply (*drule l-bucket-ptr-inv-D*[**where** $b = \alpha$ (T ! j)])
apply (*simp add*: *strict-mono-leD*)
by (*simp add*: *l-bucket-end-def*)

lemma *bucket-size-imp-less-length*:
$[\![$*l-bucket-ptr-inv* α T B SA; $j <$ *length* T; *suffix-type* T j = *L-type*; $j \notin$ *set* SA; *strict-mono* $\alpha]\!] \Longrightarrow$
B ! (α (T ! j)) $<$ *length* T
apply (*drule* (*4*) *l-bucket-ptr-inv-imp-less-l-bucket-end*)
apply (*erule order.strict-trans2*)
apply (*rule order.trans*[*OF l-bucket-end-le-bucket-end bucket-end-le-length*])
done

lemma *l-bucket-ptr-inv-imp-ge-bucket-start*:
$[\![$*l-bucket-ptr-inv* α T B SA; $b \leq \alpha$ (*Max* (*set* T))$]\!] \Longrightarrow$
bucket-start α T $b \leq B$! b
by (*simp add*: *l-bucket-ptr-inv-D*)

lemma *l-bucket-ptr-inv-le-bucket-pointers*:
$[\![$*l-bucket-ptr-inv* α T B SA; $a < b$; $b \leq \alpha$ (*Max* (*set* T))$]\!] \Longrightarrow$
B ! $a \leq B$! b
apply (*frule l-bucket-ptr-inv-imp-le-l-bucket-end*[**where** $b = a$])
apply *arith*
apply (*frule l-bucket-ptr-inv-D*[**where** $b = b$])
apply *assumption*
apply *simp*
apply (*erule order.trans*)
apply (*rule order.trans*[**where** b = *bucket-start* α T b])
apply (*erule order.trans*[*OF l-bucket-end-le-bucket-end less-bucket-end-le-start*])

apply *simp*
done

68.3 No Overwrite

We prove that the next location is set as unknown.

lemma *l-unknowns-l-bucket-ptr-inv-helper*:
$[\![$*l-unknowns-inv* α *T B SA*;
l-bucket-ptr-inv α *T B SA*;
$j <$ *length T*;
suffix-type T j = *L-type*;
$j \notin$ *set SA*;
strict-mono α;
$k = \alpha\ (T\ !\ j)$;
$l = B\ !\ k]\!] \Longrightarrow$
SA ! l = *length T*
apply (*drule* (*4*) *l-bucket-ptr-inv-imp-less-l-bucket-end*)
apply (*drule l-unknowns-inv-D*[**where** $b = k$ **and** $k = l$]; *simp add*: *strict-mono-leD*)
done

lemma *unchanged-slice*:
assumes *l-unchanged-inv* α *T SA0 SA*
and *length SA* = *length SA0*
and *length SA* = *length T*
and $b \leq \alpha$ (*Max* (*set T*))
and *l-bucket-end* α *T b* $\leq i$
and $j \leq$ *bucket-end* α *T b*
shows *list-slice SA0 i j* = *list-slice SA i j*
proof (*intro list-eq-iff-nth-eq*[*THEN iffD2*] *allI impI conjI*)
from ‹*length SA* = *length SA0*›
length-list-slice
show *length* (*list-slice SA0 i j*) = *length* (*list-slice SA i j*)
by *simp*
next
fix k
assume $k <$ *length* (*list-slice SA0 i j*)
with ‹*l-bucket-end* α *T b* $\leq i$›
have *l-bucket-end* α *T b* $\leq i + k$
by *simp*

from ‹*length SA* = *length T*›
have *bucket-end* α *T b* $\leq$ *length SA*
by (*simp add*: *bucket-end-le-length*)
with ‹$j \leq$ *bucket-end* α *T b*›
have $j \leq$ *length SA*
by *simp*
with ‹*length SA* = *length SA0*›
have *length* (*list-slice SA0 i j*) = $j - i$

by *simp*
with ‹k < *length* (*list-slice SA0 i j*)›
have $i < j$
by *linarith*

from ‹$j \leq$ *length SA*›
‹k < *length* (*list-slice SA0 i j*)›
‹*length* (*list-slice SA0 i j*) = $j - i$›
‹*length SA* = *length SA0*›
have $i + k$ < *length SA0*
by *linarith*

from ‹$j \leq$ *bucket-end* α *T b*›
‹k < *length* (*list-slice SA0 i j*)›
‹*length* (*list-slice SA0 i j*) = $j - i$›
have $i + k$ < *bucket-end* α *T b*
by *linarith*

from *l-unchanged-inv-D*[*OF* ‹*l-unchanged-inv* α *T SA0 SA*›
‹*length SA* = *length SA0*›
‹$b \leq \alpha$ (*Max* (*set T*))›
‹$i + k$ < *length SA0*›
‹*l-bucket-end* α *T b* $\leq i + k$›
‹$i + k$ < *bucket-end* α *T b*›]
‹k < *length* (*list-slice SA0 i j*)›
‹*length SA* = *length SA0*›
show *list-slice SA0 i j* ! k = *list-slice SA i j* ! k
by (*metis length-list-slice nth-list-slice*)
qed

lemma *lms-init-unchanged*:
assumes *l-unchanged-inv* α *T SA0 SA*
and *length SA* = *length SA0*
and *length SA* = *length T*
and *lms-init* α *T SA0*
shows *lms-init* α *T SA*
unfolding *lms-init-def*
proof (*intro allI impI*)
fix *b*
assume $b \leq \alpha$ (*Max* (*set T*))

have *l-bucket-end* α *T b* $\leq$ *lms-bucket-start* α *T b*
by (*simp add*: *l-bucket-end-le-lms-bucket-start*)

from *unchanged-slice*[*OF* ‹*l-unchanged-inv* α *T SA0 SA*›
‹*length SA* = *length SA0*›
‹*length SA* = *length T*›
‹$b \leq \alpha$ (*Max* (*set T*))›
‹*l-bucket-end* α *T b* $\leq$ *lms-bucket-start* α *T b*›

order.refl]
lms-init-D[*OF* ‹*lms-init α T SA0*› ‹*b ≤ α* (*Max* (*set T*))›]
show *lms-bucket α T b* = *set* (*list-slice SA* (*lms-bucket-start α T b*) (*bucket-end α T b*))
by *simp*
qed

lemma *s-init-unchanged*:
assumes *l-unchanged-inv α T SA0 SA*
and *length SA* = *length SA0*
and *length SA* = *length T*
and *s-init α T SA0*
shows *s-init α T SA*
unfolding *s-init-def*
proof (*intro allI impI*; *elim conjE*)
fix *b i*
assume *b ≤ α* (*Max* (*set T*))
i < *length SA*
l-bucket-end α T b ≤ i
i < *lms-bucket-start α T b*

have *lms-bucket-start α T b ≤ bucket-end α T b*
by (*simp add*: *bucket-end-def′ l-pl-pure-s-pl-lms-size lms-bucket-start-def*)
with ‹*i* < *lms-bucket-start α T b*›
have *i* < *bucket-end α T b*
by *simp*

from ‹*i* < *length SA*› ‹*length SA* = *length SA0*›
have *i* < *length SA0*
by *simp*

from *l-unchanged-inv-D*[*OF* ‹*l-unchanged-inv α T SA0 SA*›
‹*length SA* = *length SA0*›
‹*b ≤ α* (*Max* (*set T*))›
‹*i* < *length SA0*›
‹*l-bucket-end α T b ≤ i*›
‹*i* < *bucket-end α T b*›]
have *SA0* ! *i* = *SA* ! *i*
by *assumption*

from *s-init-D*[*OF* ‹*s-init α T SA0*›
‹*b ≤ α* (*Max* (*set T*))›
‹*i* < *length SA0*›
‹*l-bucket-end α T b ≤ i*›
‹*i* < *lms-bucket-start α T b*›]
have *SA0* ! *i* = *length T*
by *simp*
with ‹*SA0* ! *i* = *SA* ! *i*›
show *SA* ! *i* = *length T*

```
    by simp
qed

lemma l-suffix-sorted-pre-maintained:
  assumes l-unchanged-inv α T SA0 SA
  and     length SA = length SA0
  and     length SA = length T
  and     l-suffix-sorted-pre α T SA0
shows l-suffix-sorted-pre α T SA
  unfolding l-suffix-sorted-pre-def
proof (safe)
  fix b
  assume b ≤ α (Max (set T))
  let ?xs = list-slice SA0 (lms-bucket-start α T b) (bucket-end α T b) and
      ?ys = list-slice SA (lms-bucket-start α T b) (bucket-end α T b)

  have l-bucket-end α T b ≤ lms-bucket-start α T b
    using l-bucket-end-le-lms-bucket-start by auto
  with unchanged-slice[OF assms(1−3) ‹b ≤ -›]
  have ?xs = ?ys
    by blast
  then show ordlistns.sorted (map (suffix T) ?ys)
    by (metis ‹b ≤ α (Max (set T))› assms(4) l-suffix-sorted-pre-def)
qed

lemma l-prefix-sorted-pre-maintained:
  assumes l-unchanged-inv α T SA0 SA
  and     length SA = length SA0
  and     length SA = length T
  and     l-prefix-sorted-pre α T SA0
shows l-prefix-sorted-pre α T SA
  unfolding l-prefix-sorted-pre-def
proof (safe)
  fix b
  assume b ≤ α (Max (set T))
  let ?xs = list-slice SA0 (lms-bucket-start α T b) (bucket-end α T b) and
      ?ys = list-slice SA (lms-bucket-start α T b) (bucket-end α T b)

  have l-bucket-end α T b ≤ lms-bucket-start α T b
    using l-bucket-end-le-lms-bucket-start by auto
  with unchanged-slice[OF assms(1−3) ‹b ≤ -›]
  have ?xs = ?ys
    by blast
  then show ordlistns.sorted (map (lms-prefix T) ?ys)
    by (metis ‹b ≤ α (Max (set T))› assms(4) l-prefix-sorted-pre-def)
qed

lemma unknown-range-values:
  assumes l-unchanged-inv α T SA0 SA
```

and *l-unknowns-inv* α *T B SA*
and *length SA* = *length SA0*
and *length SA* = *length T*
and *lms-init* α *T SA0*
and *s-init* α *T SA0*
and $b \leq \alpha$ (*Max* (*set T*))
and *B* ! $b \leq i$
and $i <$ *lms-bucket-start* α *T b*
shows *SA* ! i = *length T*
proof −
have $i <$ *length T*
by (*meson assms*(*9*) *bucket-end-le-length leD le-less-linear le-less-trans lms-bucket-start-le-bucket-end*)
hence $i <$ *length SA*
by (*simp add*: *assms*(*4*))

have $i <$ *l-bucket-end* α *T b* $\vee$ *l-bucket-end* α *T* $b \leq i$
using *not-less* **by** *blast*
moreover
have $i <$ *l-bucket-end* α *T b* $\Longrightarrow$ *?thesis*
proof −
assume $i <$ *l-bucket-end* α *T b*
with *l-unknowns-inv-D*[*OF assms*(*2*,*7*,*8*)]
show *?thesis* **.**
qed
moreover
have *l-bucket-end* α *T* $b \leq i \Longrightarrow$ *?thesis*
proof −
assume *l-bucket-end* α *T* $b \leq i$
with *s-init-D*[*OF s-init-unchanged*[*OF assms*(*1*,*3*−*4*,*6*)] *assms*(*7*) ‹$i <$ *length SA*› - *assms*(*9*)]
show *?thesis* **.**
qed
ultimately show *?thesis*
by *blast*
qed

68.4 Bucket Values

lemma *same-bucket-same-hd*:
assumes *l-unchanged-inv* α *T SA0 SA*
and *l-locations-inv* α *T B SA*
and *l-bucket-ptr-inv* α *T B SA*
and *l-unknowns-inv* α *T B SA*
and *length SA* = *length T*
and *length SA* = *length SA0*
and *lms-init* α *T SA0*
and *s-init* α *T SA0*
and $b \leq \alpha$ (*Max* (*set T*))
and $i <$ *length SA*

and $SA\ !\ i < length\ T$
and $bucket\text{-}start\ \alpha\ T\ b \leq i$
and $i < bucket\text{-}end\ \alpha\ T\ b$
shows $\alpha\ (T\ !\ (SA\ !\ i)) = b$
proof −
have $i < B\ !\ b \lor B\ !\ b \leq i$
by *linarith*
then show *?thesis*
proof
assume $i < B\ !\ b$
from *l-locations-inv-D*[*OF* ‹*l-locations-inv* α *T B SA*›
‹$b \leq \alpha\ (Max\ (set\ T))$›
‹$i < length\ SA$›
‹$bucket\text{-}start\ \alpha\ T\ b \leq i$›
‹$i < B\ !\ b$›]
show *?thesis*
by *blast*
next
assume $B\ !\ b \leq i$

from ‹$i < length\ SA$› ‹$length\ SA = length\ SA0$›
have $i < length\ SA0$
by *simp*

have $lms\text{-}bucket\text{-}start\ \alpha\ T\ b \leq i$
proof −
have $i < l\text{-}bucket\text{-}end\ \alpha\ T\ b \longrightarrow SA\ !\ i = length\ T$
proof
assume $i < l\text{-}bucket\text{-}end\ \alpha\ T\ b$
from *l-unknowns-inv-D*[*OF* ‹*l-unknowns-inv* α *T B SA*›
‹$b \leq \alpha\ (Max\ (set\ T))$›
‹$B\ !\ b \leq i$›
‹$i < l\text{-}bucket\text{-}end\ \alpha\ T\ b$›]
show $SA\ !\ i = length\ T$
by *assumption*
qed

have $l\text{-}bucket\text{-}end\ \alpha\ T\ b \leq i \land i < lms\text{-}bucket\text{-}start\ \alpha\ T\ b \longrightarrow SA\ !\ i = length$
T
proof (*intro impI*; *elim conjE*)
assume $i < lms\text{-}bucket\text{-}start\ \alpha\ T\ b$
$l\text{-}bucket\text{-}end\ \alpha\ T\ b \leq i$

from ‹*s-init* α *T SA0*›
‹$l\text{-}bucket\text{-}end\ \alpha\ T\ b \leq i$›
‹$i < lms\text{-}bucket\text{-}start\ \alpha\ T\ b$›
‹$b \leq \alpha\ (Max\ (set\ T))$›
‹$i < length\ SA0$›
have $SA0\ !\ i = length\ T$

by (*metis s-init-def*)
with *l-unchanged-inv-D*[*OF* ‹*l-unchanged-inv* α *T SA0 SA*›
‹*length SA* = *length SA0*›
‹$b \leq \alpha$ (*Max* (*set T*))›
‹$i <$ *length SA0*›
‹*l-bucket-end* α *T b* $\leq i$›
‹$i <$ *bucket-end* α *T b*›]
show *SA* ! i = *length T*
by *simp*
qed

from ‹$i <$ *l-bucket-end* α *T b* $\longrightarrow$ *SA* ! i = *length T*›
‹*l-bucket-end* α *T b* $\leq i \wedge i <$ *lms-bucket-start* α *T b* $\longrightarrow$ *SA* ! i = *length*
T›
‹*SA* ! $i <$ *length T*›
show *lms-bucket-start* α *T b* $\leq i$
by *auto*
qed
hence *l-bucket-end* α *T b* $\leq i$
using *l-bucket-end-le-lms-bucket-start le-trans* **by** *blast*

from ‹*length SA* = *length T*›
have *bucket-end* α *T b* $\leq$ *length SA*
by (*simp add*: *bucket-end-le-length*)
with ‹*lms-init* α *T SA0*›
‹*lms-bucket-start* α *T b* $\leq i$›
‹$i <$ *bucket-end* α *T b*›
‹$b \leq \alpha$ (*Max* (*set T*))›
‹*length SA* = *length SA0*›
have *SA0* ! $i \in$ *lms-bucket* α *T b*
by (*metis list-slice-nth-mem lms-init-def*)
with *l-unchanged-inv-D*[*OF* ‹*l-unchanged-inv* α *T SA0 SA*›
‹*length SA* = *length SA0*›
‹$b \leq \alpha$ (*Max* (*set T*))›
‹$i <$ *length SA0*›
‹*l-bucket-end* α *T b* $\leq i$›
‹$i <$ *bucket-end* α *T b*›]
have *SA* ! $i \in$ *lms-bucket* α *T b*
by *simp*
then show *?thesis*
by (*simp add*: *bucket-def lms-bucket-def*)
qed
qed

lemma *same-hd-same-bucket*:
assumes *l-unchanged-inv* α *T SA0 SA*
and *l-locations-inv* α *T B SA*
and *l-bucket-ptr-inv* α *T B SA*
and *l-unknowns-inv* α *T B SA*

and *strict-mono* α
and *length SA* = *length T*
and *length SA* = *length SA0*
and *lms-init* α *T SA0*
and *s-init* α *T SA0*
and i < *length SA*
and *SA* ! i < *length T*
and $b = \alpha$ (*T* ! (*SA* ! *i*))
shows *bucket-start* α *T b* $\leq i \wedge i <$ *bucket-end* α *T b*
proof –
from ‹*length SA* = *length T*› ‹*i* < *length SA*›
have i < *length T*
by *simp*
from *index-in-bucket-interval-gen*[*OF* ‹*i* < *length T*› ‹*strict-mono* α›]
obtain b' **where**
$b' \leq \alpha$ (*Max* (*set T*))
bucket-start α *T* $b' \leq i$
i < *bucket-end* α *T* b'
by *blast*

from *same-bucket-same-hd*[*OF* ‹*l-unchanged-inv* α *T SA0 SA*›
‹*l-locations-inv* α *T B SA*›
‹*l-bucket-ptr-inv* α *T B SA*›
‹*l-unknowns-inv* α *T B SA*›
‹*length SA* = *length T*›
‹*length SA* = *length SA0*›
‹*lms-init* α *T SA0*›
‹*s-init* α *T SA0*›
‹$b' \leq \alpha$ (*Max* (*set T*))›
‹*i* < *length SA*›
‹*SA* ! *i* < *length T*›
‹*bucket-start* α *T* $b' \leq i$›
‹*i* < *bucket-end* α *T* b'›]
‹$b = \alpha$ (*T* ! (*SA* ! *i*))›
‹*bucket-start* α *T* $b' \leq i$›
‹*i* < *bucket-end* α *T* b'›
show *?thesis*
by *blast*
qed

lemma *less-bucket-less-hd*:
assumes *l-unchanged-inv* α *T SA0 SA*
and *l-locations-inv* α *T B SA*
and *l-bucket-ptr-inv* α *T B SA*
and *l-unknowns-inv* α *T B SA*
and *strict-mono* α
and *length SA* = *length T*
and *length SA* = *length SA0*

and *lms-init* α *T SA0*
and *s-init* α *T SA0*
and $i <$ *length SA*
and *SA* ! $i <$ *length T*
and $i <$ *bucket-start* α *T b*
shows α (*T* ! (*SA* ! *i*)) $< b$
proof −
from *same-hd-same-bucket*[*OF* ‹*l-unchanged-inv* α *T SA0 SA*›
‹*l-locations-inv* α *T B SA*›
‹*l-bucket-ptr-inv* α *T B SA*›
‹*l-unknowns-inv* α *T B SA*›
‹*strict-mono* α›
‹*length SA* = *length T*›
‹*length SA* = *length SA0*›
‹*lms-init* α *T SA0*›
‹*s-init* α *T SA0*›
‹$i <$ *length SA*›
‹*SA* ! $i <$ *length T*›,
of α (*T* ! (*SA* ! *i*))]
have *bucket-start* α *T* (α (*T* ! (*SA* ! *i*))) $\leq i \wedge i <$ *bucket-end* α *T* (α (*T* ! (*SA* ! *i*)))
by *simp*
then show *?thesis*
by (*meson* ‹$i <$ *bucket-start* α *T b*› *bucket-start-le leD le-less-linear le-trans*)
qed

lemma *gr-bucket-gr-hd*:
assumes *l-unchanged-inv* α *T SA0 SA*
and *l-locations-inv* α *T B SA*
and *l-bucket-ptr-inv* α *T B SA*
and *l-unknowns-inv* α *T B SA*
and *strict-mono* α
and *length SA* = *length T*
and *length SA* = *length SA0*
and *lms-init* α *T SA0*
and *s-init* α *T SA0*
and $i <$ *length SA*
and *SA* ! $i <$ *length T*
and *bucket-end* α *T b* $\leq i$
shows $b < \alpha$ (*T* ! (*SA* ! *i*))
proof −
from *same-hd-same-bucket*[*OF* ‹*l-unchanged-inv* α *T SA0 SA*›
‹*l-locations-inv* α *T B SA*›
‹*l-bucket-ptr-inv* α *T B SA*›
‹*l-unknowns-inv* α *T B SA*›
‹*strict-mono* α›
‹*length SA* = *length T*›
‹*length SA* = *length SA0*›
‹*lms-init* α *T SA0*›

$\langle$*s-init* α *T SA0*$\rangle$
$\langle i <$ *length SA*$\rangle$
$\langle$*SA* ! $i <$ *length T*$\rangle$,
of α (*T* ! (*SA* ! *i*))]
have *bucket-start* α *T* (α (*T* ! (*SA* ! *i*))) $\leq i \wedge i <$ *bucket-end* α *T* (α (*T* ! (*SA* ! *i*)))
by *simp*
then show *?thesis*
by (*meson* $\langle$*bucket-end* α *T b* $\leq i\rangle$ *bucket-end-le leD le-less-linear less-le-trans*)
qed

68.5 Seen

We have two helper lemmas in the case of updating the suffix array SA, and in the case when the current index is incremented. The two lemmas are used in conjunction in the case that the SA is updated and the current index is incremented.

lemma *l-seen-inv-upd*:
assumes *l-seen-inv T SA n n* $\leq$ *k SA* ! $k =$ *length T*
shows *l-seen-inv T* (*SA*[$k := x$]) *n*
unfolding *l-seen-inv-def*
proof *safe*
fix *i j*
assume *A*: $i < n$ $i <$ *length* (*SA*[$k := x$]) *SA*[$k := x$] ! $i <$ *length T SA*[$k := x$] ! $i =$ *Suc j*
suffix-type T j = *L-type*
hence $i \neq k$
using *assms*(*2*) *leD* **by** *blast*
hence *B*: $i <$ *length SA SA* ! $i <$ *length T SA* ! $i =$ *Suc j*
using *A* **by** *auto*
with *l-seen-inv-nth-ex*[*OF assms*(*1*) *A*(*1*) *B A*(*5*)]
have $\exists k'<$*length SA*. *SA* ! $k' = j$
by *blast*
then obtain k' **where**
$k' <$ *length SA SA* ! $k' = j$
by *blast*
then show $\exists k'<$*length* (*SA*[$k := x$]). *SA*[$k := x$] ! $k' = j$
by (*metis B*(*2*,*3*) *Suc-lessD assms*(*3*) *length-list-update less-irrefl-nat nth-list-update-neq*)
qed

lemma *l-seen-inv-Suc*:
assumes *l-seen-inv T SA n SA* ! $n =$ *Suc j* $k <$ *length SA SA* ! $k = j$
shows *l-seen-inv T SA* (*Suc n*)
unfolding *l-seen-inv-def*
proof *safe*
fix *i j'*
assume *A*: $i <$ *Suc n* $i <$ *length SA SA* ! $i <$ *length T SA* ! $i =$ *Suc j'*
suffix-type T j' = *L-type*
have $i < n \vee \neg\, i < n$

by *blast*
then show $\exists k<length\ SA.\ SA\ !\ k = j'$
proof
assume $i < n$
with *l-seen-inv-nth-ex*[*OF assms*(*1*) - *A*(*2*−)]
show $\exists k<length\ SA.\ SA\ !\ k = j'$
by *blast*
next
assume $\neg\ i < n$
then show $\exists k<length\ SA.\ SA\ !\ k = j'$
using *A*(*1*) *A*(*4*) *assms*(*2*−) *not-less-less-Suc-eq* **by** *force*
qed
qed

69 Distinctness

lemma *distinct-app3*:
distinct (*xs* @ *ys* @ *zs*) $\longleftrightarrow$
distinct xs ∧ *distinct ys* ∧ *distinct zs* ∧
set xs ∩ *set ys* = {} ∧ *set xs* ∩ *set zs* = {} ∧ *set ys* ∩ *set zs* = {}
by *auto*

69.1 Establishment

lemma *abs-is-lms-imp-in-lms-bucket*:
abs-is-lms T i $\Longrightarrow$ *i* ∈ *lms-bucket* α *T* (α (*T* ! *i*))
apply (*clarsimp simp*: *lms-bucket-def bucket-def*)
by (*simp add*: *abs-is-lms-def suffix-type-s-bound*)

lemma *l-distinct-inv-established*:
assumes *lms-init* α *T SA*
and *l-init* α *T SA*
and *s-init* α *T SA*
and *length SA* = *length T*
and *strict-mono* α
and *l-bucket-init* α *T B*
shows *l-distinct-inv T SA*
unfolding *l-distinct-inv-def*
proof (*intro distinct-conv-nth*[*THEN iffD2*] *allI impI*)
let *?P* = (λ*x*. *x* < *length T*)
let *?SA* = *filter* (λ*x*. *x* < *length T*) *SA*

fix *i j*
assume *i* < *length ?SA j* < *length ?SA i* ≠ *j*

from ‹*i* < *length ?SA*›
have *?SA* ! *i* < *length T*
using *mem-Collect-eq nth-mem* **by** *fastforce*

from ‹$j < length\ ?SA$›
have $?SA\ !\ j < length\ T$
  using *mem-Collect-eq nth-mem* **by** *fastforce*

from *filter-nth-relative-neq-2*[*OF* ‹$i < length\ ?SA$› ‹$j < length\ ?SA$› ‹$i \neq j$›]
obtain i' j' **where**
  $i' < length\ SA$
  $j' < length\ SA$
  $SA\ !\ i' = ?SA\ !\ i$
  $SA\ !\ j' = ?SA\ !\ j$
  $i' \neq j'$
  by *blast*

from ‹$?SA\ !\ i < length\ T$› ‹$SA\ !\ i' = ?SA\ !\ i$›
have $SA\ !\ i' < length\ T$
  by *simp*

from ‹$?SA\ !\ j < length\ T$› ‹$SA\ !\ j' = ?SA\ !\ j$›
have $SA\ !\ j' < length\ T$
  by *simp*

from *init-imp-lms-range assms* ‹$i' < length\ SA$› ‹$SA\ !\ i' < length\ T$›
have $lms\text{-}bucket\text{-}start\ \alpha\ T\ (\alpha\ (T\ !\ (SA\ !\ i'))) \leq i'$ $i' < bucket\text{-}end\ \alpha\ T\ (\alpha\ (T\ !\ (SA\ !\ i')))$
  by *blast+*

from *init-imp-lms-range assms* ‹$j' < length\ SA$› ‹$SA\ !\ j' < length\ T$›
have $lms\text{-}bucket\text{-}start\ \alpha\ T\ (\alpha\ (T\ !\ (SA\ !\ j'))) \leq j'$ $j' < bucket\text{-}end\ \alpha\ T\ (\alpha\ (T\ !\ (SA\ !\ j')))$
  by *blast+*

have $\alpha\ (T\ !\ (SA\ !\ i')) = \alpha\ (T\ !\ (SA\ !\ j')) \lor \alpha\ (T\ !\ (SA\ !\ i')) \neq \alpha\ (T\ !\ (SA\ !\ j'))$
  by *simp*
then show $?SA\ !\ i \neq ?SA\ !\ j$
proof
  assume $\alpha\ (T\ !\ (SA\ !\ i')) = \alpha\ (T\ !\ (SA\ !\ j'))$
  with ‹$lms\text{-}bucket\text{-}start\ \alpha\ T\ (\alpha\ (T\ !\ (SA\ !\ i'))) \leq i'$›
  have $lms\text{-}bucket\text{-}start\ \alpha\ T\ (\alpha\ (T\ !\ (SA\ !\ j'))) \leq i'$
    by *simp*

  from ‹$\alpha\ (T\ !\ (SA\ !\ i')) = \alpha\ (T\ !\ (SA\ !\ j'))$›
    ‹$i' < bucket\text{-}end\ \alpha\ T\ (\alpha\ (T\ !\ (SA\ !\ i')))$›
  have $i' < bucket\text{-}end\ \alpha\ T\ (\alpha\ (T\ !\ (SA\ !\ j')))$
    by *simp*

  with *list-slice-nth-eq-iff-index-eq*[*OF lms-init-imp-distinct-bucket*,
                          OF ‹*lms-init* α *T SA*›

$\langle$*length SA* = *length T*$\rangle$
-
$\langle$*lms-bucket-start* α *T* (α (*T* ! (*SA* ! j'))) $\leq i'$$\rangle$
-
$\langle$*lms-bucket-start* α *T* (α (*T* ! (*SA* ! j'))) $\leq j'$$\rangle$
$\langle j' <$ *bucket-end* α *T* (α (*T* ! (*SA* ! j')))$\rangle$]
$\langle i' \neq j'\rangle$
$\langle$*length SA* = *length T*$\rangle$
$\langle$*SA* ! j' < *length T*$\rangle$
$\langle$*strict-mono* $\alpha\rangle$
have *SA* ! $i' \neq$ *SA* ! j'
by (*simp add*: *bucket-end-le-length strict-mono-less-eq*)
with $\langle$*SA* ! i' = *?SA* ! $i\rangle$ $\langle$*SA* ! j' = *?SA* ! $j\rangle$
show *?thesis*
by *simp*
next
assume α (*T* ! (*SA* ! i')) $\neq$ α (*T* ! (*SA* ! j'))
with $\langle$*SA* ! i' = *?SA* ! $i\rangle$ $\langle$*SA* ! j' = *?SA* ! $j\rangle$
show *?thesis*
by *auto*
qed
qed

corollary *l-distinct-inv-perm-established*:
assumes *l-perm-pre* α *T B SA*
shows *l-distinct-inv T SA*
using *assms l-distinct-inv-established l-perm-pre-def* **by** *blast*

69.2 Maintenance

lemma *l-distinct-inv-maintained*:
assumes *i* < *length SA*
and *SA* ! *i* = *Suc j*
and *Suc j* < *length T*
and *suffix-type T j* = *L-type*
and *l-distinct-inv T SA*
and *l-pred-inv T SA i*
shows *l-distinct-inv T* (*SA*[*l* := *j*])
proof −
let *?P* = (λx. *x* < *length T*)

from $\langle$*Suc j* < *length T*$\rangle$
have *j* < *length T*
by *simp*

— We case on whether the update occurs on an index within bounds
have *l* < *length SA* $\vee$ *l* $\geq$ *length SA*
by *arith*
then show *?thesis*

proof
assume $l < length\ SA$

from *l-distinct-pred-inv-helper*[*OF* ‹$i < length\ SA$›
‹$SA\ !\ i = Suc\ j$›
‹$Suc\ j < length\ T$›
‹$suffix\text{-}type\ T\ j = L\text{-}type$›
‹$l\text{-}distinct\text{-}inv\ T\ SA$›
‹$l\text{-}pred\text{-}inv\ T\ SA\ i$›]
have $j \notin set\ SA$
by *assumption*

let $?xs = filter\ ?P\ (take\ l\ SA)$ **and**
$?ys = filter\ ?P\ (drop\ (Suc\ l)\ SA)$
— We have that $j \notin set\ SA$ and show that if we now add j to SA, it will maintain distinctness. This is straightforward but does require some massaging, i.e. casing on whether $SA\ !\ l < length\ T$ or not, due to the use of *filter* in the *l-distinct-inv* $?T\ ?SA = distinct\ (filter\ (\lambda x.\ x < length\ ?T)\ ?SA)$ definition.

from ‹$j < length\ T$› ‹$l < length\ SA$›
have *f-SA′*: $filter\ ?P\ (SA[l := j]) = ?xs\ @\ [j]\ @\ ?ys$
by (*simp add*: *filter-update-nth-success*)

from ‹$j \notin set\ SA$›
have $set\ ?xs \cap set\ [j] = \{\}$
using *in-set-takeD* **by** *fastforce*

from ‹$j \notin set\ SA$›
have $set\ [j] \cap set\ ?ys = \{\}$
using *in-set-dropD* **by** *force*

have $SA\ !\ l < length\ T \lor SA\ !\ l \geq length\ T$
by *arith*
then show *?thesis*
proof
assume $SA\ !\ l < length\ T$
with ‹$l < length\ SA$›
have *f-SA*: $filter\ ?P\ SA = ?xs\ @\ [SA\ !\ l]\ @\ ?ys$
by (*meson filter-take-nth-drop-success*)

from *f-SA* ‹$l\text{-}distinct\text{-}inv\ T\ SA$› *distinct-app3*[*of* $?xs\ [SA\ !\ l]\ ?ys$]
have $distinct\ ?xs$
$distinct\ ?ys$
$set\ ?xs \cap set\ ?ys = \{\}$
unfolding *l-distinct-inv-def*
by *auto*

with ‹$set\ ?xs \cap set\ [j] = \{\}$›
‹$set\ [j] \cap set\ ?ys = \{\}$›

f-SA'
show *?thesis*
unfolding *l-distinct-inv-def*
by *auto*
next
assume $SA \; ! \; l \geq$ *length T*
with ‹$l <$ *length SA*›
have *f-SA*: *filter ?P SA* $=$ *?xs @ ?ys*
by (*simp add*: *filter-take-nth-drop-fail*)

from *f-SA* ‹*l-distinct-inv T SA*› *distinct-append*[*of ?xs ?ys*]
have *distinct ?xs*
distinct ?ys
set ?xs $\cap$ *set ?ys* $= \{\}$
unfolding *l-distinct-inv-def*
by *auto*
with ‹*set ?xs* $\cap$ *set* $[j] = \{\}$›
‹*set* $[j] \cap$ *set ?ys* $= \{\}$›
f-SA'
show *?thesis*
unfolding *l-distinct-inv-def*
by *auto*
qed
next
— We now handle the case *length SA* $\leq l$, which is straightforward. In the actual *abs-induce-l* subroutine, l will always be less than *length SA*, but for this proof, we make no such assumption, nor do we need to prove it.
assume $l \geq$ *length SA*
hence $SA[l := j] = SA$
by *simp*
with ‹*l-distinct-inv T SA*›
show *?thesis*
by *simp*
qed
qed

corollary *l-distinct-inv-perm-maintained*:
assumes *l-perm-inv* α *T B SA0 SA i*
and $i <$ *length SA*
and $SA \; ! \; i = Suc \; j$
and $Suc \; j <$ *length T*
and *suffix-type T j* $=$ *L-type*
shows *l-distinct-inv T* ($SA[l := j]$)
by (*meson assms l-distinct-inv-maintained l-perm-inv-elims*(*4*,*10*))

70 Unknowns

70.1 Establishment

lemma *l-unknowns-inv-established*:
  assumes *l-init* α *T SA*
    l-bucket-init α *T B*
    length SA = *length T*
  shows *l-unknowns-inv* α *T B SA*
  unfolding *l-unknowns-inv-def*
proof (*intro allI impI*; *elim conjE*)
  fix *a k*
  assume $a \leq \alpha$ (*Max* (*set T*))
    B ! $a \leq k$
    $k <$ *l-bucket-end* α *T a*

  from ‹*B* ! $a \leq k$› *l-bucket-initD*[*OF* ‹*l-bucket-init* α *T B*› ‹$a \leq \alpha$ (*Max* (*set T*))›]
  have *bucket-start* α *T a* $\leq k$
    by *simp*

  from ‹*length SA* = *length T*› ‹$k <$ *l-bucket-end* α *T a*›
  have $k <$ *length SA*
    by (*metis bucket-end-le-length l-bucket-end-le-bucket-end less-le-trans*)

  from *l-init-D*[*OF* ‹*l-init* α *T SA*›
      ‹$a \leq \alpha$ (*Max* (*set T*))›
      ‹$k <$ *length SA*›
      ‹*bucket-start* α *T a* $\leq k$›
      ‹$k <$ *l-bucket-end* α *T a*›]
  show *SA* ! k = *length T*
    by *assumption*
qed

corollary *l-unknowns-inv-perm-established*:
  assumes *l-perm-pre* α *T B SA*
  shows *l-unknowns-inv* α *T B SA*
  using *assms l-perm-pre-elims*(*2*,*4*,*7*) *l-unknowns-inv-established* **by** *blast*

70.2 Maintenance

lemma *l-unknowns-inv-maintained*:
  assumes *l-unknowns-inv* α *T B SA*
  and *length B* $> \alpha$ (*Max* (*set T*))
  and $i <$ *length SA*
  and *SA* ! i = *Suc j*
  and *Suc j* $<$ *length T*
  and *suffix-type T j* = *L-type*
  and $k = \alpha$ (*T* ! *j*)
  and l = *B* ! k
  and *strict-mono* α

and *l-distinct-inv T SA*
and *l-pred-inv T SA i*
and *l-bucket-ptr-inv* α *T B SA*
shows *l-unknowns-inv* α T $(B[k := Suc\ (B\ !\ k)])$ $(SA[l := j])$
unfolding *l-unknowns-inv-def*
proof (*intro allI impI*; *elim conjE*)
fix $a\ k'$
assume $a \leq \alpha\ (Max\ (set\ T))$
$B[k := Suc\ (B\ !\ k)]\ !\ a \leq k'$
$k' < l\text{-}bucket\text{-}end\ \alpha\ T\ a$

from *l-distinct-pred-inv-helper*[*OF* ‹$i < length\ SA$›
‹$SA\ !\ i = Suc\ j$›
‹$Suc\ j < length\ T$›
‹$suffix\text{-}type\ T\ j = L\text{-}type$›
‹*l-distinct-inv T SA*›
‹*l-pred-inv T SA i*›]
have $j \notin set\ SA$
by *assumption*

from ‹$Suc\ j < length\ T$›
have $j < length\ T$
by *simp*

from *l-unknowns-l-bucket-ptr-inv-helper*[*OF* ‹*l-unknowns-inv* α *T B SA*›
‹*l-bucket-ptr-inv* α *T B SA*›
‹$j < length\ T$›
‹$suffix\text{-}type\ T\ j = L\text{-}type$›
‹$j \notin set\ SA$›
‹*strict-mono* α›
‹$k = \alpha\ (T\ !\ j)$›
‹$l = B\ !\ k$›]
have $SA\ !\ l = length\ T$
by *assumption*

from ‹*strict-mono* α› ‹$k = \alpha\ (T\ !\ j)$› ‹$j < length\ T$›
have $k \leq \alpha\ (Max\ (set\ T))$
by (*simp add*: *strict-mono-leD*)

from *l-unknowns-inv-D*[*OF* ‹*l-unknowns-inv* α *T B SA*›
‹$a \leq \alpha\ (Max\ (set\ T))$›
\-
‹$k' < l\text{-}bucket\text{-}end\ \alpha\ T\ a$›]
‹$B[k := Suc\ (B\ !\ k)]\ !\ a \leq k'$›
‹$a \leq \alpha\ (Max\ (set\ T))$›
‹$\alpha\ (Max\ (set\ T)) < length\ B$›
have $SA\ !\ k' = length\ T$
by (*metis Suc-le-mono le-SucI le-less-trans nth-list-update nth-list-update-neq*)

from ‹$j <$ *length* T› ‹*strict-mono* α› ‹*l-bucket-ptr-inv* α T B SA› ‹$k = \alpha$ (T ! j)› ‹$l = B$! k›
have *bucket-start* α T $k \leq l$
using *Max-greD l-bucket-ptr-inv-imp-ge-bucket-start strict-mono-less-eq* **by** *blast*

from ‹$j <$ *length* T›
‹$j \notin$ *set* SA›
‹*strict-mono* α›
‹*l-bucket-ptr-inv* α T B SA›
‹*suffix-type* T j = *L-type*›
‹$k = \alpha$ (T ! j)›
‹$l = B$! k›
have $l <$ *l-bucket-end* α T k
using *l-bucket-ptr-inv-imp-less-l-bucket-end* **by** *blast*

have $a = k \vee a \neq k$
by *simp*
then show $SA[l := j]$! $k' =$ *length* T
proof
assume $a = k$
hence $k' \neq l$
using ‹$B[k :=$ *Suc* (B ! k)] ! $a \leq k'$› ‹$a \leq \alpha$ (*Max* (*set* T))› ‹α (*Max* (*set* T)) $<$ *length* B›
‹$l = B$! k› **by** *auto*
then show *?thesis*
using ‹SA ! $k' =$ *length* T› **by** *auto*
next
assume $a \neq k$
hence $a < k \vee a > k$
by *arith*
then show *?thesis*
proof
assume $a < k$

from *less-bucket-end-le-start*[*OF* ‹$a < k$›]
have *bucket-end* α T $a \leq$ *bucket-start* α T k
by *blast*
with ‹*bucket-start* α T $k \leq l$›
have *bucket-end* α T $a \leq l$
by *simp*
with *l-bucket-end-le-bucket-end*
have *l-bucket-end* α T $a \leq l$
using *le-trans* **by** *blast*
with ‹$k' <$ *l-bucket-end* α T a›
have $k' < l$
using *less-le-trans* **by** *blast*
then show *?thesis*
using ‹SA ! $k' =$ *length* T› **by** *auto*
next

assume $a > k$

from ‹$B[k := Suc\ (B\ !\ k)]\ !\ a \leq k'$›
‹$a \leq \alpha\ (Max\ (set\ T))$›
‹$a \neq k$›
‹*l-bucket-ptr-inv* α *T B SA*›
l-bucket-ptr-inv-imp-ge-bucket-start
have *bucket-start* α $T\ a \leq k'$
by *force*

from *less-bucket-end-le-start*[*OF* ‹$k < a$›]
have *bucket-end* α $T\ k \leq$ *bucket-start* α $T\ a$
by *blast*
with ‹*bucket-start* α $T\ a \leq k'$›
have *bucket-end* α $T\ k \leq k'$
by *simp*
with *l-bucket-end-le-bucket-end*
have *l-bucket-end* α $T\ k \leq k'$
using *le-trans* **by** *blast*
with ‹$l <$ *l-bucket-end* α $T\ k$›
have $l < k'$
using *less-le-trans* **by** *blast*
then show *?thesis*
using ‹$SA\ !\ k' = length\ T$› **by** *auto*
qed
qed
qed

corollary *l-unknowns-inv-perm-maintained*:
assumes *l-perm-inv* α *T B SA0 SA i*
and $i < length\ SA$
and $SA\ !\ i = Suc\ j$
and $Suc\ j < length\ T$
and *suffix-type* $T\ j = L\text{-}type$
and $k = \alpha\ (T\ !\ j)$
and $l = B\ !\ k$
shows *l-unknowns-inv* α $T\ (B[k := Suc\ (B\ !\ k)])\ (SA[l := j])$
by (*metis assms l-perm-inv-elims*(*1*,*4*−*6*,*10*,*12*) *l-unknowns-inv-maintained*)

71 Number of L-types

71.1 Establishment

We first prove that this invariant is established from the precondition, i.e., that initially, there are only LMS-types, which are just a special type of S-types, and that the initial pointer is the start of the bucket.

lemma *l-bucket-ptr-inv-established*:
assumes *lms-init* α *T SA*

```
  and     l-init α T SA
  and     s-init α T SA
  and     length SA = length T
  and     strict-mono α
  and     l-bucket-init α T B
  shows l-bucket-ptr-inv α T B SA
proof −
  have ∀ b ≤ α (Max (set T)). cur-l-types α T SA b = {}
    unfolding cur-l-types-def
  proof (intro allI impI equalityI subsetI)
    fix b x
    assume x ∈ {i |i. i ∈ set SA ∧ i ∈ l-bucket α T b}
    hence x ∈ set SA x ∈ l-bucket α T b
      by blast+

    have x < length T ∨ x ≥ length T
      using not-less by blast
    then show x ∈ {}
    proof
      assume x < length T
      with ‹x ∈ set SA› init-imp-only-s-types assms
      have suffix-type T x = S-type
        by (metis in-set-conv-nth)
      hence x ∉ l-bucket α T b
        by (simp add: l-bucket-def)
      with ‹x ∈ l-bucket α T b›
      show ?thesis
        by blast
    next
      assume  length T ≤ x
      hence x ∉ l-bucket α T b
        by (simp add: bucket-def l-bucket-def)
      with ‹x ∈ l-bucket α T b›
      show ?thesis
        by blast
    qed
  next
    fix b :: nat
    fix x :: nat
    assume x ∈ {}
    then show x ∈ {i |i. i ∈ set SA ∧ i ∈ l-bucket α T b}
      by blast
  qed
  hence ∀ b ≤ α (Max (set T)). num-l-types α T SA b = 0
    by (simp add: num-l-types-def)
  with ‹l-bucket-init α T B›
  show ?thesis
    by (simp add: l-bucket-ptr-inv-def l-bucket-init-def)
qed
```

corollary *l-bucket-ptr-inv-perm-established*:
assumes *l-perm-pre* α *T B SA*
shows *l-bucket-ptr-inv* α *T B SA*
using *assms l-bucket-ptr-inv-established l-perm-pre-def* **by** *blast*

71.2 Maintenance

We now prove that the invariant is maintained.

lemma *set-update-mem-neqI*:
$[\![x \in set\ xs;\ xs\ !\ i \neq x]\!] \Longrightarrow x \in set\ (xs[i := y])$
by (*metis in-set-conv-nth length-list-update nth-list-update-neq*)

lemma *cur-l-types-update-1*:
$[\![SA\ !\ l = length\ T;\ l < length\ SA;\ j \notin set\ SA;\ suffix\text{-}type\ T\ j = L\text{-}type;\ j < length\ T;$
$\alpha\ (T\ !\ j) = b]\!] \Longrightarrow$
$cur\text{-}l\text{-}types\ \alpha\ T\ (SA[l := j])\ b = insert\ j\ (cur\text{-}l\text{-}types\ \alpha\ T\ SA\ b)$
apply (*intro equalityI subsetI*)
apply (*metis* (*no-types, lifting*) *cur-l-types-def in-set-conv-nth insertCI length-list-update mem-Collect-eq nth-list-update nth-list-update-neq*)
by (*metis* (*mono-tags, lifting*) *bucket-def cur-l-types-def insert-iff l-bucket-def less-irrefl-nat mem-Collect-eq set-update-memI set-update-mem-neqI*)

lemma *cur-l-types-update-2*:
assumes $SA\ !\ l = length\ T\ \ \alpha\ (T\ !\ j) \neq b$
shows $cur\text{-}l\text{-}types\ \alpha\ T\ (SA[l := j])\ b = cur\text{-}l\text{-}types\ \alpha\ T\ SA\ b$
proof (*cases* $l < length\ SA$)
assume $l < length\ SA$
show *?thesis*
proof *safe*
fix x
assume $x \in cur\text{-}l\text{-}types\ \alpha\ T\ (SA[l := j])\ b$
show $x \in cur\text{-}l\text{-}types\ \alpha\ T\ SA\ b$
proof (*cases* $x = j$)
assume $x = j$
then show *?thesis*
using ‹$x \in cur\text{-}l\text{-}types\ \alpha\ T\ (SA[l := j])\ b$› *assms*(*2*) *bucket-def cur-l-types-def l-bucket-def*
by *fastforce*
next
assume $x \neq j$
then show *?thesis*
by (*metis* (*no-types, lifting*) ‹$x \in cur\text{-}l\text{-}types\ \alpha\ T\ (SA[l := j])\ b$› *cur-l-types-def in-set-conv-nth length-list-update mem-Collect-eq nth-list-update nth-list-update-neq*)
qed
next

fix x
assume $x \in$ *cur-l-types* α T *SA* b
then show $x \in$ *cur-l-types* α T $(SA[l := j])$ b
by (*simp add*: *assms*(*1*) *bucket-def cur-l-types-def l-bucket-def set-update-mem-neqI*)
qed
next
assume $\neg$ $l <$ *length SA*
then show *?thesis*
by *simp*
qed

lemma *num-l-types-update-1*:
$[\![SA ! l = length\ T; l < length\ SA; j \notin set\ SA; suffix\text{-}type\ T\ j = L\text{-}type; j < length\ T;$
$\alpha\ (T ! j) = b]\!] \Longrightarrow$
$num\text{-}l\text{-}types\ \alpha\ T\ (SA[l := j])\ b = Suc\ (num\text{-}l\text{-}types\ \alpha\ T\ SA\ b)$
apply (*clarsimp simp*: *num-l-types-def*)
apply (*subst cur-l-types-update-1*; *simp?*)
apply (*subst card-insert-disjoint*)
apply (*metis* (*no-types*, *lifting*) *List.finite-set cur-l-types-def finite-nat-set-iff-bounded mem-Collect-eq*)
apply (*simp add*: *cur-l-types-def*)
apply *simp*
done

lemma *num-l-types-update-2*:
$[\![SA ! l = length\ T; \alpha\ (T ! j) \neq b]\!] \Longrightarrow$
$num\text{-}l\text{-}types\ \alpha\ T\ (SA[l := j])\ b = num\text{-}l\text{-}types\ \alpha\ T\ SA\ b$
apply (*cases* $l <$ *length SA*; *clarsimp?*)
apply (*clarsimp simp*: *num-l-types-def*)
apply (*intro arg-cong*[**where** f = *card*])
by (*erule* (*1*) *cur-l-types-update-2*)

lemma *l-bucket-ptr-inv-maintained*:
assumes *l-bucket-ptr-inv* α T B *SA*
and *length SA* = *length T*
and *length B* $> \alpha$ (*Max* (*set T*))
and $i <$ *length SA*
and *SA* ! i = *Suc* j
and *Suc* $j <$ *length T*
and *suffix-type T j* = *L-type*
and $k = \alpha\ (T ! j)$
and $l = B ! k$
and *strict-mono* α
and *l-distinct-inv T SA*
and *l-pred-inv T SA i*
and *l-unknowns-inv* α T B *SA*
shows *l-bucket-ptr-inv* α T $(B[k := Suc\ (B ! k)])\ (SA[l := j])$
unfolding *l-bucket-ptr-inv-def*

proof *safe*
  fix b
  assume $b \leq \alpha\ (Max\ (set\ T))$

  from *l-distinct-pred-inv-helper*[*OF* ‹$i < length\ SA$›
                                    ‹$SA\ !\ i = Suc\ j$›
                                    ‹$Suc\ j < length\ T$›
                                    ‹$suffix\text{-}type\ T\ j = L\text{-}type$›
                                    ‹$l\text{-}distinct\text{-}inv\ T\ SA$›
                                    ‹$l\text{-}pred\text{-}inv\ T\ SA\ i$›]
  have $j \notin set\ SA$
    by *assumption*

  from ‹$Suc\ j < length\ T$›
  have $j < length\ T$
    by *arith*

  from *l-unknowns-l-bucket-ptr-inv-helper*[*OF* ‹$l\text{-}unknowns\text{-}inv\ \alpha\ T\ B\ SA$›
                                    ‹$l\text{-}bucket\text{-}ptr\text{-}inv\ \alpha\ T\ B\ SA$›
                                    ‹$j < length\ T$›
                                    ‹$suffix\text{-}type\ T\ j = L\text{-}type$›
                                    ‹$j \notin set\ SA$›
                                    ‹$strict\text{-}mono\ \alpha$›
                                    ‹$k = \alpha\ (T\ !\ j)$›
                                    ‹$l = B\ !\ k$›]
  have $SA\ !\ l = length\ T$
    by *assumption*

  let $?G = B[k := Suc\ (B\ !\ k)]\ !\ b = bucket\text{-}start\ \alpha\ T\ b + num\text{-}l\text{-}types\ \alpha\ T\ (SA[l$
$:= j])\ b$
  have $b = k \lor b \neq k$
    by *blast*
  then show $?G$
  proof
    assume $b = k$
    hence $B[k := Suc\ (B\ !\ k)]\ !\ b = Suc\ (B\ !\ k)$
      using ‹$b \leq \alpha\ (Max\ (set\ T))$› ‹$length\ B > \alpha\ (Max\ (set\ T))$› **by** *auto*

    from *num-l-types-less-l-bucket-size*[*OF* ‹$j \notin set\ SA$› ‹$suffix\text{-}type\ T\ j = L\text{-}type$›]
        ‹$Suc\ j < length\ T$›
        ‹$b = k$›
        ‹$k = \alpha\ (T\ !\ j)$›
    have $num\text{-}l\text{-}types\ \alpha\ T\ SA\ b < l\text{-}bucket\text{-}size\ \alpha\ T\ b$
      by *simp*

    from ‹$l\text{-}bucket\text{-}ptr\text{-}inv\ \alpha\ T\ B\ SA$› ‹$b \leq \alpha\ (Max\ (set\ T))$›
    have $B\ !\ b = bucket\text{-}start\ \alpha\ T\ b + num\text{-}l\text{-}types\ \alpha\ T\ SA\ b$
      by (*metis l-bucket-ptr-inv-def*)
    with ‹$num\text{-}l\text{-}types\ \alpha\ T\ SA\ b < l\text{-}bucket\text{-}size\ \alpha\ T\ b$›

bucket-end-le-length l-bucket-le-bucket-size bucket-end-def′
have $B\ !\ b < \mathit{length}\ T$
by (*metis add-less-cancel-left less-le-trans*)
with ‹$k = \alpha\ (T\ !\ j)$› ‹$b = k$› ‹$l = B\ !\ k$› ‹$\mathit{length}\ SA = \mathit{length}\ T$›
have $l < \mathit{length}\ SA$
by *simp*

from ‹$SA\ !\ l = \mathit{length}\ T$›
‹$b = k$›
‹$b \leq \alpha\ (\mathit{Max}\ (\mathit{set}\ T))$›
‹$j \notin \mathit{set}\ SA$›
‹$l < \mathit{length}\ SA$›
‹$k = \alpha\ (T\ !\ j)$›
num-l-types-update-1
‹*suffix-type* $T\ j$ = *L-type*›
‹$\mathit{Suc}\ j < \mathit{length}\ T$›
Suc-lessD
have $\mathit{num\text{-}l\text{-}types}\ \alpha\ T\ (SA[l := j])\ b = \mathit{Suc}\ (\mathit{num\text{-}l\text{-}types}\ \alpha\ T\ SA\ b)$
by *blast*
from ‹$B[k := \mathit{Suc}\ (B\ !\ k)]\ !\ b = \mathit{Suc}\ (B\ !\ k)$›
‹$b = k$›
‹$b \leq \alpha\ (\mathit{Max}\ (\mathit{set}\ T))$›
‹$\mathit{num\text{-}l\text{-}types}\ \alpha\ T\ (SA[l := j])\ b = \mathit{Suc}\ (\mathit{num\text{-}l\text{-}types}\ \alpha\ T\ SA\ b)$›
‹*l-bucket-ptr-inv* $\alpha\ T\ B\ SA$›
l-bucket-ptr-inv-def
show *?thesis*
by *fastforce*
next
assume $b \neq k$
hence $B[k := \mathit{Suc}\ (B\ !\ k)]\ !\ b = B\ !\ b$
by *auto*

from *num-l-types-update-2*[*OF* ‹$SA\ !\ l = \mathit{length}\ T$›]
‹$b \leq \alpha\ (\mathit{Max}\ (\mathit{set}\ T))$›
‹$b \neq k$›
‹$k = \alpha\ (T\ !\ j)$›
have $\mathit{num\text{-}l\text{-}types}\ \alpha\ T\ (SA[l := j])\ b = \mathit{num\text{-}l\text{-}types}\ \alpha\ T\ SA\ b$
by *simp*

from ‹$B[k := \mathit{Suc}\ (B\ !\ k)]\ !\ b = B\ !\ b$›
‹$b \leq \alpha\ (\mathit{Max}\ (\mathit{set}\ T))$›
‹$\mathit{num\text{-}l\text{-}types}\ \alpha\ T\ (SA[l := j])\ b = \mathit{num\text{-}l\text{-}types}\ \alpha\ T\ SA\ b$›
‹*l-bucket-ptr-inv* $\alpha\ T\ B\ SA$›
l-bucket-ptr-inv-def
show *?thesis*
by *fastforce*
qed
qed

corollary *l-bucket-ptr-inv-perm-maintained*:
assumes *l-perm-inv* α *T B SA0 SA i*
and $i <$ *length SA*
and *SA* ! $i =$ *Suc j*
and *Suc j* $<$ *length T*
and *suffix-type T j* $=$ *L-type*
and $k = \alpha$ $(T$! $j)$
and $l = B$! k
shows *l-bucket-ptr-inv* α *T* $(B[k := Suc\ (B\ !\ k)])$ $(SA[l := j])$
by (*metis assms l-bucket-ptr-inv-maintained l-perm-inv-elims*(*1*,*2*,*3*−*6*,*10*,*12*))

72 L Locations

72.1 Establishment

lemma *l-locations-inv-established*:
assumes *l-bucket-init* α *T B*
shows *l-locations-inv* α *T B SA*
using *assms l-bucket-initD l-locations-inv-def* **by** *fastforce*

corollary *l-locations-inv-perm-established*:
assumes *l-perm-pre* α *T B SA*
shows *l-locations-inv* α *T B SA*
using *assms l-locations-inv-established l-perm-pre-elims*(*4*) **by** *blast*

72.2 Maintenance

lemma *l-locations-inv-maintained*:
assumes *l-locations-inv* α *T B SA*
and *length B* $> \alpha$ (*Max* (*set T*))
and $i <$ *length SA*
and *SA* ! $i =$ *Suc j*
and *Suc j* $<$ *length T*
and *suffix-type T j* $=$ *L-type*
and $k = \alpha$ $(T$! $j)$
and $l = B$! k
and *strict-mono* α
and *l-distinct-inv T SA*
and *l-pred-inv T SA i*
and *l-bucket-ptr-inv* α *T B SA*
shows *l-locations-inv* α *T* $(B[k := Suc\ (B\ !\ k)])$ $(SA[l := j])$
unfolding *l-locations-inv-def*
proof (*intro allI impI*; *elim conjE*)
fix b i'
assume $b \le \alpha$ (*Max* (*set T*))
$i' <$ *length* $(SA[l := j])$
bucket-start α *T* $b \le i'$
$i' < B[k := Suc\ (B\ !\ k)]\ !\ b$
hence $i' <$ *length SA*

by *simp*

have $b = k \vee b \neq k$
by *blast*
then
show $SA[l := j] ! i' < length\ T \wedge suffix\text{-}type\ T\ (SA[l := j] ! i') = L\text{-}type \wedge \alpha\ (T ! (SA[l := j] ! i')) = b$
proof
assume $b = k$
with ‹$bucket\text{-}start\ \alpha\ T\ b \leq i'$›
have $bucket\text{-}start\ \alpha\ T\ k \leq i'$
by *simp*

from ‹$b = k$› ‹$i' < B[k := Suc\ (B ! k)] ! b$›
have $i' < Suc\ (B ! k)$
using ‹$b \leq \alpha\ (Max\ (set\ T))$› ‹$length\ B > \alpha\ (Max\ (set\ T))$› by *auto*
hence $i' < B ! k \vee i' = B ! k$
by (*simp add*: *less-Suc-eq*)
then show *?thesis*
proof
assume $i' < B ! k$
with ‹$l = B ! k$› ‹$b = k$› ‹$b \leq \alpha\ (Max\ (set\ T))$› ‹$bucket\text{-}start\ \alpha\ T\ k \leq i'$› ‹$i' < length\ SA$›
show *?thesis*
using ‹$l\text{-}locations\text{-}inv\ \alpha\ T\ B\ SA$› *l-locations-inv-def* by *fastforce*
next
assume $i' = B ! k$
with ‹$l = B ! k$›
have $i' = l$
by *simp*

from ‹$i' < length\ SA$› ‹$i' = l$›
have $SA[l := j] ! i' = j$
by *simp*
with ‹$suffix\text{-}type\ T\ j = L\text{-}type$› ‹$Suc\ j < length\ T$› ‹$i' < length\ SA$› ‹$b = k$› ‹$k = \alpha\ (T ! j)$›
show *?thesis*
by *auto*
qed
next
assume $b \neq k$
hence $B[k := Suc\ (B ! k)] ! b = B ! b$
by *simp*

from *l-bucket-ptr-inv-imp-le-l-bucket-end*[*OF* ‹$l\text{-}bucket\text{-}ptr\text{-}inv\ \alpha\ T\ B\ SA$› ‹$b \leq \alpha\ (Max\ (set\ T))$›]
have $B ! b \leq l\text{-}bucket\text{-}end\ \alpha\ T\ b$
by *simp*

from ‹*Suc j* < *length T*›
have j < *length T*
  by *simp*

from *l-distinct-pred-inv-helper*[*OF* ‹*i* < *length SA*›
  ‹*SA* ! *i* = *Suc j*›
  ‹*Suc j* < *length T*›
  ‹*suffix-type T j* = *L-type*›
  ‹*l-distinct-inv T SA*›
  ‹*l-pred-inv T SA i*›]
have $j \notin$ *set SA*
  by *assumption*

from *l-bucket-ptr-inv-imp-less-l-bucket-end*[*OF* ‹*l-bucket-ptr-inv* α *T B SA*›
  ‹j < *length T*›
  ‹*suffix-type T j* = *L-type*›
  ‹$j \notin$ *set SA*›
  ‹*strict-mono* α›]
  ‹$k = \alpha$ (*T* ! *j*)›
  ‹l = *B* ! *k*›
have l < *l-bucket-end* α *T k*
  by *simp*

from ‹$k = \alpha$ (*T* ! *j*)› ‹j < *length T*› ‹*strict-mono* α›
have $k \leq \alpha$ (*Max* (*set T*))
  by (*simp add*: *strict-mono-less-eq*)

from *l-bucket-ptr-inv-imp-ge-bucket-start*[*OF* ‹*l-bucket-ptr-inv* α *T B SA*›
  ‹$k \leq \alpha$ (*Max* (*set T*))›]
  ‹l = *B* ! *k*›
have *bucket-start* α *T k* $\leq l$
  by *simp*

from ‹*B*[*k* := *Suc* (*B* ! *k*)] ! *b* = *B* ! *b*›
  ‹i' < *B*[*k* := *Suc* (*B* ! *k*)] ! *b*›
  l-bucket-ptr-inv-imp-le-l-bucket-end[*OF* ‹*l-bucket-ptr-inv* α *T B SA*›
  ‹$b \leq \alpha$ (*Max* (*set T*))›]
  l-bucket-end-le-bucket-end
have i' < *bucket-end* α *T b*
  by (*metis less-le-trans*)

from ‹$b \neq k$›
have $b < k \vee k < b$
  by *linarith*
hence $i' \neq l$
proof
  assume $b < k$
  hence *bucket-end* α *T b* $\leq$ *bucket-start* α *T k*
    by (*simp add*: *less-bucket-end-le-start*)

with ‹$i' <$ *bucket-end* α *T b*› ‹*bucket-start* α *T k* $\leq l$›
have $i' < l$
by *linarith*
then show *?thesis*
by *simp*
next
assume $k < b$
hence *bucket-end* α *T k* $\leq$ *bucket-start* α *T b*
by (*simp add*: *less-bucket-end-le-start*)
hence *l-bucket-end* α *T k* $\leq$ *bucket-start* α *T b*
using *l-bucket-end-le-bucket-end le-trans* by *blast*
with ‹*bucket-start* α *T b* $\leq i'$› ‹$l <$ *l-bucket-end* α *T k*›
have $l < i'$
by *simp*
then show *?thesis*
by *simp*
qed
with ‹$B[k := Suc\ (B\ !\ k)]\ !\ b = B\ !\ b$›
‹$b \leq \alpha\ (Max\ (set\ T))$›
‹*bucket-start* α *T b* $\leq i'$›
‹$i' < B[k := Suc\ (B\ !\ k)]\ !\ b$›
‹$i' <$ *length SA*›
‹*l-locations-inv* α *T B SA*›
show *?thesis*
using *l-locations-inv-D* by *fastforce*
qed
qed

corollary *l-locations-inv-perm-maintained*:
assumes *l-perm-inv* α *T B SA0 SA i*
and $i <$ *length SA*
and *SA* ! $i = Suc\ j$
and $Suc\ j <$ *length T*
and *suffix-type T j* = *L-type*
and $k = \alpha\ (T\ !\ j)$
and $l = B\ !\ k$
shows *l-locations-inv* α *T* $(B[k := Suc\ (B\ !\ k)])\ (SA[l := j])$
by (*metis assms l-locations-inv-maintained l-perm-inv-elims*(*1*,*4*,*6*,*9*,*10*,*12*))

73 Unchanged

73.1 Establishment

lemma *l-unchanged-inv-established*:
l-unchanged-inv α *T SA SA*
using *l-unchanged-inv-def* **by** *blast*

73.2 Maintenance

lemma *l-unchanged-inv-maintained*:
  assumes *l-unchanged-inv* α *T SA0 SA*
  and *length B* $>$ α (*Max* (*set T*))
  and *i* $<$ *length SA*
  and *SA* ! *i* = *Suc j*
  and *Suc j* $<$ *length T*
  and *suffix-type T j* = *L-type*
  and *k* = α (*T* ! *j*)
  and *l* = *B* ! *k*
  and *strict-mono* α
  and *l-distinct-inv T SA*
  and *l-pred-inv T SA i*
  and *l-bucket-ptr-inv* α *T B SA*
  shows *l-unchanged-inv* α *T SA0* (*SA*[*l* := *j*])
proof −
  have *l-unchanged-inv* α *T SA* (*SA*[*l* := *j*])
    unfolding *l-unchanged-inv-def*
  proof *safe*
    show *length* (*SA*[*l* := *j*]) = *length SA*
      by *simp*
  next
    fix *b i'*
    assume *b* $\leq$ α (*Max* (*set T*))
        i' $<$ *length SA*
        l-bucket-end α *T b* $\leq$ *i'*
        i' $<$ *bucket-end* α *T b*

    from ‹*strict-mono* α›
        ‹*l-bucket-ptr-inv* α *T B SA*›
        ‹*Suc j* $<$ *length T*›
        ‹*k* = α (*T* ! *j*)›
        ‹*l* = *B* ! *k*›
    have *bucket-start* α *T k* $\leq$ *l*
    by (*metis Max-greD Suc-lessD l-bucket-ptr-inv-imp-ge-bucket-start strict-mono-leD*)

    from ‹*l-distinct-inv T SA*›
        ‹*l-pred-inv T SA i*›
        ‹*i* $<$ *length SA*›
        ‹*SA* ! *i* = *Suc j*›
        ‹*Suc j* $<$ *length T*›
        ‹*suffix-type T j* = *L-type*›
    have *j* $\notin$ *set SA*
      using *l-distinct-pred-inv-helper* **by** *blast*

    from ‹*j* $\notin$ *set SA*›
        ‹*strict-mono* α›
        ‹*l-bucket-ptr-inv* α *T B SA*›
        ‹*Suc j* $<$ *length T*›

⟨*suffix-type T j = L-type*⟩
⟨$k = \alpha$ (*T* ! *j*)⟩
⟨*l* = *B* ! *k*⟩
have *l* < *l-bucket-end* α *T k*
using *Suc-lessD l-bucket-ptr-inv-imp-less-l-bucket-end* **by** *blast*

have $b = k \vee b \neq k$
by *blast*
then show *SA* ! *i'* = *SA*[*l* := *j*] ! *i'*
proof
assume $b = k$
then show *?thesis*
using ⟨*l* < *l-bucket-end* α *T k*⟩ ⟨*l-bucket-end* α *T b* ≤ *i'*⟩ **by** *auto*
next
assume $b \neq k$
hence $b < k \vee k < b$
using *less-linear* **by** *blast*
then show *?thesis*
proof
assume $b < k$
hence *bucket-end* α *T b* ≤ *bucket-start* α *T k*
by (*simp add*: *less-bucket-end-le-start*)
with ⟨*i'* < *bucket-end* α *T b*⟩ ⟨*bucket-start* α *T k* ≤ *l*⟩
have *i'* < *l*
by *linarith*
then show *?thesis*
by *simp*
next
assume $k < b$
hence *bucket-end* α *T k* ≤ *bucket-start* α *T b*
by (*simp add*: *less-bucket-end-le-start*)
hence *l-bucket-end* α *T k* ≤ *bucket-start* α *T b*
using *l-bucket-end-le-bucket-end le-trans* **by** *blast*
hence *l-bucket-end* α *T k* ≤ *l-bucket-end* α *T b*
by (*simp add*: *l-bucket-end-def*)
with ⟨*l* < *l-bucket-end* α *T k*⟩ ⟨*l-bucket-end* α *T b* ≤ *i'*⟩
have *l* < *i'*
by *linarith*
then show *?thesis*
by *simp*
qed
qed
qed
with ⟨*l-unchanged-inv* α *T SA0 SA*⟩
show *?thesis*
using *l-unchanged-inv-trans* **by** *blast*
qed

corollary *l-unchanged-inv-perm-maintained*:

assumes *l-perm-inv* α *T B SA0 SA i*
and $i <$ *length SA*
and *SA* ! $i =$ *Suc j*
and *Suc j* $<$ *length T*
and *suffix-type T j* $=$ *L-type*
and $k = \alpha$ (*T* ! *j*)
and $l = B$! k
shows *l-unchanged-inv* α *T SA0* (*SA*[$l := j$])
by (*metis assms l-perm-inv-elims*(*1*,*4*,*6*,*8*,*10*,*12*) *l-unchanged-inv-maintained*)

74 Invariant about the Current Index

74.1 Establishment

The first invariant is that current index is always less than the index where the update will occur.

lemma *l-index-inv-established*:
assumes *lms-init* α *T SA*
and *l-init* α *T SA*
and *s-init* α *T SA*
and *length SA* $=$ *length T*
and *strict-mono* α
and *l-bucket-init* α *T B*
shows *l-index-inv* α *T B SA*
unfolding *l-index-inv-def*
proof (*intro allI impI*; *elim conjE*)
fix *i j*
assume $i <$ *length SA SA* ! $i =$ *Suc j Suc j* $<$ *length T suffix-type T j* $=$ *L-type*
with *init-imp-only-s-types*[*OF* ‹*lms-init* α *T SA*›
‹*l-init* α *T SA*›
‹*s-init* α *T SA*›
‹*length SA* $=$ *length T*›
‹*strict-mono* α›,
THEN spec, *of i*]
have *suffix-type T* (*Suc j*) $=$ *S-type*
by *simp*
with ‹*suffix-type T j* $=$ *L-type*› ‹*Suc j* $<$ *length T*›
have *T* ! $j \neq$ *T* ! *Suc j*
by (*simp add*: *suffix-type-neq*)
with ‹*suffix-type T j* $=$ *L-type*› ‹*Suc j* $<$ *length T*›
have *T* ! *Suc j* $<$ *T* ! *j*
using *nth-less-imp-s-type* **by** *fastforce*
from *same-hd-same-bucket*[*OF l-unchanged-inv-established*
l-locations-inv-established[*OF* ‹*l-bucket-init* α *T B*›]
l-bucket-ptr-inv-established[*OF* ‹*lms-init* α *T SA*›
‹*l-init* α *T SA*›
‹*s-init* α *T SA*›
‹*length SA* $=$ *length T*›

$\langle$*strict-mono* $\alpha\rangle$]
l-unknowns-inv-established
- - - - -
$\langle i <$ *length SA*$\rangle$]
assms
$\langle SA\ !\ i = Suc\ j\rangle$
$\langle Suc\ j <$ *length* $T\rangle$
have *bucket-start* α T (α (T ! *Suc* j)) $\leq i$ $i <$ *bucket-end* α T (α (T ! *Suc* j))
by *simp+*
with $\langle T\ !\ Suc\ j < T\ !\ j\rangle$ $\langle$*strict-mono* $\alpha\rangle$
have $i <$ *bucket-start* α T (α (T ! j))
by (*meson less-bucket-end-le-start less-le-trans strict-mono-less*)

from $\langle$*l-bucket-init* α T $B\rangle$ $\langle Suc\ j <$ *length* $T\rangle$ $\langle$*strict-mono* $\alpha\rangle$
have B ! α (T ! j) = *bucket-start* α T (α (T ! j))
by (*simp add*: *Suc-lessD l-bucket-initD strict-mono-leD*)
with $\langle i <$ *bucket-start* α T (α (T ! j))$\rangle$
show $i < B$! α (T ! j)
by *simp*
qed

corollary *l-index-inv-perm-established*:
assumes *l-perm-pre* α T B SA
shows *l-index-inv* α T B SA
using *assms l-index-inv-established l-perm-pre-def* **by** *blast*

74.2 Maintenance

lemma *l-index-inv-maintained*:
assumes *l-index-inv* α T B SA
and *length* $B > \alpha$ (*Max* (*set* T))
and $i <$ *length* SA
and SA ! i = *Suc* j
and *Suc* $j <$ *length* T
and *suffix-type* T j = *L-type*
and $k = \alpha$ (T ! j)
and $l = B$! k
and *strict-mono* α
and *l-distinct-inv* T SA
and *l-pred-inv* T SA i
and *l-bucket-ptr-inv* α T B SA
and *l-unknowns-inv* α T B SA
shows *l-index-inv* α T (B[k := *Suc* (B ! k)]) (SA[l := j])
unfolding *l-index-inv-def*
proof(*intro impI allI*; *elim conjE*)
fix l' j'
assume $l' <$ *length* (SA[l := j])
SA[l := j] ! l' = *Suc* j'
Suc $j' <$ *length* T

```
      suffix-type T j' = L-type

from ‹l' < length (SA[l := j])›
have l' < length SA
  by simp

have l' = l ∨ l' ≠ l
  by simp
then show l' < B[k := Suc (B ! k)] ! α (T ! j')
proof
  assume l' = l
  with ‹SA[l := j] ! l' = Suc j'› ‹l' < length SA›
  have j = Suc j'
    by simp
  with ‹suffix-type T j' = L-type› ‹Suc j' < length T›
  have T ! j ≤ T ! j'
    by (simp add: Suc-lessD l-type-letter-gre-Suc)
  with ‹strict-mono α›
  have α (T ! j) ≤ α (T ! j')
    using strict-mono-less-eq by blast
  hence α (T ! j) = α (T ! j') ∨ α (T ! j) < α (T ! j')
    using le-imp-less-or-eq by blast
  then show ?thesis
  proof
    assume α (T ! j) = α (T ! j')
    with ‹k = α (T ! j)›
         ‹Suc j' < length T›
         ‹j = Suc j'›
         ‹strict-mono α›
         ‹length B > α (Max (set T))›
    have B[k := Suc (B ! k)] ! α (T ! j') = Suc (B ! k)
     by (metis Max-greD less-le-trans not-less nth-list-update-eq strict-mono-less)
    with ‹l = B ! k› ‹l' = l›
    show ?thesis
      by simp
  next
    assume α (T ! j) < α (T ! j')

    from ‹α (T ! j) < α (T ! j')› ‹k = α (T ! j)›
    have B[k := Suc (B ! k)] ! α (T ! j') = B ! α (T ! j')
      by simp

    from l-distinct-pred-inv-helper[OF ‹i < length SA›
                                      ‹SA ! i = Suc j›
                                      ‹Suc j < length T›
                                      ‹suffix-type T j = L-type›
                                      ‹l-distinct-inv T SA›
                                      ‹l-pred-inv T SA i›]
    have j ∉ set SA
```

by *assumption*

from *Suc-lessD*[*OF* ‹*Suc j* < *length T*›]
have $j <$ *length T*
by *assumption*

from *l-bucket-ptr-inv-imp-less-l-bucket-end*[*OF* ‹*l-bucket-ptr-inv* α *T B SA*›
‹$j <$ *length T*›
‹*suffix-type T j* = *L-type*›
‹$j \notin$ *set SA*›
‹*strict-mono* α›]
have $B ! \alpha (T ! j) <$ *l-bucket-end* α T $(\alpha (T ! j))$
by *assumption*
with ‹$l' = l$› ‹$k = \alpha (T ! j)$› ‹$l = B ! k$›
have $l' <$ *l-bucket-end* α T $(\alpha (T ! j))$
by *simp*
hence $l' <$ *bucket-end* α T $(\alpha (T ! j))$
using *l-bucket-end-le-bucket-end less-le-trans* by *blast*
with *less-bucket-end-le-start*[*OF* ‹$\alpha (T ! j) < \alpha (T ! j')$›]
have $l' <$ *bucket-start* α T $(\alpha (T ! j'))$
using *less-le-trans* by *blast*
with *l-bucket-ptr-inv-imp-ge-bucket-start*[*OF* ‹*l-bucket-ptr-inv* α *T B SA*›]
‹*Suc* $j' <$ *length T*›
‹*strict-mono* α›
have $l' < B ! \alpha (T ! j')$
by (*meson Max-greD Suc-lessD less-le-trans strict-mono-less-eq*)
with ‹$B[k :=$ *Suc* $(B ! k)] ! \alpha (T ! j') = B ! \alpha (T ! j')$›
show *?thesis*
by *simp*
qed
next
assume $l' \neq l$
with ‹$SA[l := j] ! l' =$ *Suc* j'›
have $SA ! l' =$ *Suc* j'
by *auto*

from *l-index-inv-D*[*OF* ‹*l-index-inv* α *T B SA*›
‹$l' <$ *length SA*›
‹$SA ! l' =$ *Suc* j'›
‹*Suc* $j' <$ *length T*›
‹*suffix-type T* j' = *L-type*›]

show *?thesis*
by (*metis Suc-leI Suc-le-lessD Suc-lessD list-update-beyond not-less nth-list-update-eq nth-list-update-neq*)
qed
qed

corollary *l-index-inv-perm-maintained*:

assumes *l-perm-inv* α *T B SA0 SA i*
and $i <$ *length SA*
and *SA* ! $i =$ *Suc j*
and *Suc j* $<$ *length T*
and *suffix-type T j* $=$ *L-type*
and $k = \alpha$ $(T$! $j)$
and $l = B$! k
shows *l-index-inv* α *T* $(B[k := Suc\ (B\ !\ k)])$ $(SA[l := j])$
using *assms l-index-inv-maintained l-perm-inv-def* **by** *blast*

75 Predecessor Invariant

75.1 Establishment

The proof for the establishment is simple because initially, SA contains no L-types.

lemma *l-pred-inv-established*:
assumes *lms-init* α *T SA*
and *l-init* α *T SA*
and *s-init* α *T SA*
and *length SA* $=$ *length T*
and *strict-mono* α
shows *l-pred-inv T SA 0*
using *assms init-imp-only-s-types l-pred-inv-def* **by** *fastforce*

corollary *l-pred-inv-perm-established*:
assumes *l-perm-pre* α *T B SA*
shows *l-pred-inv T SA 0*
using *assms l-perm-pre-def l-pred-inv-established* **by** *blast*

75.2 Maintenance

In this section, we prove that the predecessor invariant *l-pred-inv ?T ?SA ?k* $= (\forall i{<}length\ ?SA.\ ?SA\ !\ i < length\ ?T \wedge suffix\text{-}type\ ?T\ (?SA\ !\ i) = L\text{-}type \longrightarrow (\exists j{<}length\ ?SA.\ ?SA\ !\ j = Suc\ (?SA\ !\ i) \wedge j < i \wedge j < ?k))$is maintained. In English, this invariant states that for all L-type suffixes in the suffix array (SA), their right neighbour is in SA and occurs before them.

We now prove that the invariant is maintained for each branch of the *abs-induce-l-step*

lemma *l-pred-inv-maintained-no-update*:
assumes *l-pred-inv T SA i*
shows *l-pred-inv T SA* $(Suc\ i)$
using *assms*
unfolding *l-pred-inv-def*
using *less-Suc-eq* **by** *auto*

lemma *l-pred-inv-maintained*:

assumes *l-pred-inv* T SA i
and $i < \mathit{length}\ SA$
and $SA\ !\ i = \mathit{Suc}\ j$
and $\mathit{Suc}\ j < \mathit{length}\ T$
and $\mathit{suffix\text{-}type}\ T\ j = L\text{-}\mathit{type}$
and $k = \alpha\ (T\ !\ j)$
and $l = B\ !\ k$
and $\mathit{strict\text{-}mono}\ \alpha$
and $\mathit{l\text{-}distinct\text{-}inv}\ T\ SA$
and $\mathit{l\text{-}bucket\text{-}ptr\text{-}inv}\ \alpha\ T\ B\ SA$
and $\mathit{l\text{-}unknowns\text{-}inv}\ \alpha\ T\ B\ SA$
and $\mathit{l\text{-}index\text{-}inv}\ \alpha\ T\ B\ SA$
shows $\mathit{l\text{-}pred\text{-}inv}\ T\ (SA[l := j])\ (\mathit{Suc}\ i)$
proof $-$

from *l-distinct-pred-inv-helper*[*OF* ‹$i < \mathit{length}\ SA$›
‹$SA\ !\ i = \mathit{Suc}\ j$›
‹$\mathit{Suc}\ j < \mathit{length}\ T$›
‹$\mathit{suffix\text{-}type}\ T\ j = L\text{-}\mathit{type}$›
‹$\mathit{l\text{-}distinct\text{-}inv}\ T\ SA$›
‹$\mathit{l\text{-}pred\text{-}inv}\ T\ SA\ i$›]
have $j \notin \mathit{set}\ SA$
by *assumption*

from ‹$\mathit{Suc}\ j < \mathit{length}\ T$›
have $j < \mathit{length}\ T$
by *simp*

from *l-unknowns-l-bucket-ptr-inv-helper*[*OF* ‹$\mathit{l\text{-}unknowns\text{-}inv}\ \alpha\ T\ B\ SA$›
‹$\mathit{l\text{-}bucket\text{-}ptr\text{-}inv}\ \alpha\ T\ B\ SA$›
‹$j < \mathit{length}\ T$›
‹$\mathit{suffix\text{-}type}\ T\ j = L\text{-}\mathit{type}$›
‹$j \notin \mathit{set}\ SA$›
‹$\mathit{strict\text{-}mono}\ \alpha$›
‹$k = \alpha\ (T\ !\ j)$›
‹$l = B\ !\ k$›]
have $SA\ !\ l = \mathit{length}\ T$
by *assumption*

from *l-index-inv-D*[*OF* ‹$\mathit{l\text{-}index\text{-}inv}\ \alpha\ T\ B\ SA$›
‹$i < \mathit{length}\ SA$›
‹$SA\ !\ i = \mathit{Suc}\ j$›
‹$\mathit{Suc}\ j < \mathit{length}\ T$›
‹$\mathit{suffix\text{-}type}\ T\ j = L\text{-}\mathit{type}$›]
‹$k = \alpha\ (T\ !\ j)$›
‹$l = B\ !\ k$›
have $l > i$
by *simp*

show *?thesis*
unfolding *l-pred-inv-def*
proof (*intro impI allI*; *elim conjE*)
fix i'
assume $i' < \mathit{length}\ (SA[l := j])$
$SA[l := j]\ !\ i' < \mathit{length}\ T$
$\mathit{suffix\text{-}type}\ T\ (SA[l := j]\ !\ i') = L\text{-}type$
have $i' = l \lor i' \neq l$
by *blast*
then show
$\exists ja<\mathit{length}\ (SA[l := j]).\ SA[l := j]\ !\ ja = Suc\ (SA[l := j]\ !\ i') \land ja < i' \land ja < Suc\ i$
proof
assume $i' = l$
with ‹$i' < \mathit{length}\ (SA[l := j])$›
have $SA[l := j]\ !\ i' = j$
by *simp*

from ‹$l > i$› ‹$SA\ !\ i = Suc\ j$›
have $SA[l := j]\ !\ i = Suc\ j$
by *simp*
with ‹$l > i$› ‹$i' = l$› ‹$SA[l := j]\ !\ i' = j$› ‹$i < \mathit{length}\ SA$›
show *?thesis*
by *auto*
next
assume $i' \neq l$
with ‹$i' < \mathit{length}\ (SA[l := j])$›
have $i' < \mathit{length}\ SA$
by *simp*

from ‹$i' \neq l$› ‹$SA[l := j]\ !\ i' < \mathit{length}\ T$›
have $SA\ !\ i' < \mathit{length}\ T$
by *simp*

from ‹$i' \neq l$› ‹$\mathit{suffix\text{-}type}\ T\ (SA[l := j]\ !\ i') = L\text{-}type$›
have $\mathit{suffix\text{-}type}\ T\ (SA\ !\ i') = L\text{-}type$
by *simp*

from ‹$i' < \mathit{length}\ SA$› ‹$SA\ !\ i' < \mathit{length}\ T$› ‹$\mathit{suffix\text{-}type}\ T\ (SA\ !\ i') = L\text{-}type$›
‹*l-pred-inv T SA i*›[*simplified l-pred-inv-def*, *THEN spec*, *of* i']
obtain j' **where**
$j' < \mathit{length}\ SA$
$SA\ !\ j' = Suc\ (SA\ !\ i')$
$j' < i'$
$j' < i$
by *blast*

with ‹$SA\ !\ l = \mathit{length}\ T$› ‹$i' \neq l$› ‹$\mathit{suffix\text{-}type}\ T\ (SA\ !\ i') = L\text{-}type$› ‹$i < l$›
show *?thesis*

by *auto*
qed
qed
qed

corollary *l-pred-inv-perm-maintained*:
assumes *l-perm-inv* α *T B SA0 SA i*
and $i <$ *length SA*
and *SA* ! $i =$ *Suc j*
and *Suc j* $<$ *length T*
and *suffix-type T j* $=$ *L-type*
and $k = \alpha$ (*T* ! *j*)
and $l = B$! k
shows *l-pred-inv T* (*SA*[$l := j$]) (*Suc i*)
by (*metis assms l-perm-inv-elims*(*4*−*7*,*10*,*12*) *l-pred-inv-maintained*)

76 Seen Invariant

76.1 Establishment

We first show that the invariant is initially true, i.e. *l-seen-inv T SA 0*.

lemma *l-seen-inv-established*:
l-seen-inv T SA 0
by (*simp add*: *l-seen-inv-def*)

76.2 Maintenance

We now show that the invariant is maintained after each call of *abs-induce-l-step*.

lemma *l-seen-inv-maintained-no-update*:
⟦*l-seen-inv T SA i*; *length T* $\leq$ *SA* ! *i*⟧ $\Longrightarrow$ *l-seen-inv T SA* (*Suc i*)
⟦*l-seen-inv T SA i*; *length SA* $\leq$ *i*⟧ $\Longrightarrow$ *l-seen-inv T SA* (*Suc i*)
⟦*l-seen-inv T SA i*; *SA* ! $i <$ *length T*; *SA* ! $i = 0$⟧ $\Longrightarrow$ *l-seen-inv T SA* (*Suc i*)
⟦*l-seen-inv T SA i*; *SA* ! $i <$ *length T*; *SA* ! $i =$ *Suc j*; *suffix-type T j* $=$ *S-type*⟧
$\Longrightarrow$
l-seen-inv T SA (*Suc i*)
unfolding *l-seen-inv-def*
using *less-Suc-eq* **by** *auto*

lemma *l-seen-inv-maintained*:
assumes *l-seen-inv T SA i*
and $i <$ *length SA*
and *SA* ! $i =$ *Suc j*
and *Suc j* $<$ *length T*
and *suffix-type T j* $=$ *L-type*
and $k = \alpha$ (*T* ! *j*)
and $l = B$! k
and *length SA* $=$ *length T*
and *strict-mono* α

and *l-distinct-inv T SA*
and *l-pred-inv T SA i*
and *l-unknowns-inv* α *T B SA*
and *l-bucket-ptr-inv* α *T B SA*
and *l-index-inv* α *T B SA*
shows *l-seen-inv T* $(SA[l := j])$ $(Suc\ i)$
proof −
from *l-distinct-pred-inv-helper*[*OF* ‹*i* < *length SA*›
‹*SA* ! *i* = *Suc j*›
‹*Suc j* < *length T*›
‹*suffix-type T j* = *L-type*›
‹*l-distinct-inv T SA*›
‹*l-pred-inv T SA i*›]
have $j \notin$ *set SA*
by *assumption*

from ‹*Suc j* < *length T*›
have *j* < *length T*
by *simp*

from *bucket-size-imp-less-length*[*OF* ‹*l-bucket-ptr-inv* α *T B SA*›
‹*j* < *length T*›
‹*suffix-type T j* = *L-type*›
‹$j \notin$ *set SA*›
‹*strict-mono* α›]
‹$k = \alpha$ (*T* ! *j*)›
‹*l* = *B* ! *k*›
have *l* < *length T*
by *simp*

from *l-index-inv-D*[*OF* ‹*l-index-inv* α *T B SA*›
‹*i* < *length SA*›
‹*SA* ! *i* = *Suc j*›
‹*Suc j* < *length T*›
‹*suffix-type T j* = *L-type*›]
‹$k = \alpha$ (*T* ! *j*)›
‹*l* = *B* ! *k*›
have *l* > *i*
by *simp*

from *l-unknowns-l-bucket-ptr-inv-helper*[*OF* ‹*l-unknowns-inv* α *T B SA*›
‹*l-bucket-ptr-inv* α *T B SA*›
‹*j* < *length T*›
‹*suffix-type T j* = *L-type*›
‹$j \notin$ *set SA*›
‹*strict-mono* α›
‹$k = \alpha$ (*T* ! *j*)›
‹*l* = *B* ! *k*›]
have *SA* ! *l* = *length T*

by *assumption*

with ‹SA ! i = Suc j› ‹Suc j < $length$ T› ‹i < l›
have (SA[l := j]) ! i < $length$ T
by *auto*

with ‹SA ! i = Suc j› ‹i < l›
have (SA[l := j]) ! i = Suc j
by *auto*

from *l-seen-inv-upd*[*OF* ‹*l-seen-inv* T SA i›]
‹l > i›
‹SA ! l = $length$ T›
‹l < $length$ T›
‹$length$ SA = $length$ T›
have *l-seen-inv* T (SA[l := j]) i
by *auto*
with *l-seen-inv-Suc*[*OF* - ‹(SA[l := j]) ! i = Suc j›]
‹l < $length$ T›
‹$length$ SA = $length$ T›
show *?thesis*
by (*metis length-list-update nth-list-update-eq*)
qed

corollary *l-seen-inv-perm-maintained*:
assumes *l-perm-inv* α T B $SA0$ SA i
and i < $length$ SA
and SA ! i = Suc j
and Suc j < $length$ T
and *suffix-type* T j = *L-type*
and k = α (T ! j)
and l = B ! k
shows *l-seen-inv* T (SA[l := j]) (Suc i)
by (*metis assms l-perm-inv-elims*(*2*,*3*−*7*,*10*−*12*) *l-seen-inv-maintained*)

77 Permutation

77.1 Establishment

lemma *l-perm-inv-established*:
assumes *l-perm-pre* α T B SA
shows *l-perm-inv* α T B SA SA *0*
unfolding *l-perm-inv-def*
by (*simp add*: *l-perm-pre-elims*[*OF assms*] *l-distinct-inv-perm-established*[*OF assms*]
l-unknowns-inv-perm-established[*OF assms*] *l-bucket-ptr-inv-perm-established*[*OF assms*]
l-index-inv-perm-established[*OF assms*] *l-unchanged-inv-established*
l-locations-inv-perm-established[*OF assms*] *l-pred-inv-perm-established*[*OF assms*]

l-seen-inv-established)

77.2 Maintenance

lemma *l-perm-inv-maintained*:
  assumes *l-perm-inv* α *T B SA0 SA i*
  and $i < length\ SA$
  and $SA\ !\ i = Suc\ j$
  and $Suc\ j < length\ T$
  and $suffix\text{-}type\ T\ j = L\text{-}type$
  and $k = \alpha\ (T\ !\ j)$
  and $l = B\ !\ k$
shows *l-perm-inv* α T $(B[k := Suc\ (B\ !\ k)])$ *SA0* $(SA[l := j])$ $(Suc\ i)$
  unfolding *l-perm-inv-def*
  by (*simp add*: *l-perm-inv-elims*[*OF assms*(*1*)] *l-distinct-inv-perm-maintained*[*OF assms*(*1−5*)]
    l-unknowns-inv-perm-maintained[*OF assms*] *l-bucket-ptr-inv-perm-maintained*[*OF assms*]
    l-index-inv-perm-maintained[*OF assms*] *l-unchanged-inv-perm-maintained*[*OF assms*]
    l-locations-inv-perm-maintained[*OF assms*] *l-pred-inv-perm-maintained*[*OF assms*]
      l-seen-inv-perm-maintained[*OF assms*])

lemma *l-perm-inv-maintained-no-upd-1*:
  assumes *l-perm-inv* α *T B SA0 SA i*
  and $length\ SA \leq i$
shows *l-perm-inv* α *T B SA0 SA* (*Suc i*)
  unfolding *l-perm-inv-def*
  by (*simp add*: *l-perm-inv-elims*[*OF assms*(*1*)] *l-pred-inv-maintained-no-update*
      l-seen-inv-maintained-no-update(*2*)[*OF l-perm-inv-elims*(*11*)[*OF assms*(*1*)] *assms*(*2*)]])

lemma *l-perm-inv-maintained-no-upd-2*:
  assumes *l-perm-inv* α *T B SA0 SA i*
  and $length\ T \leq SA\ !\ i$
shows *l-perm-inv* α *T B SA0 SA* (*Suc i*)
  unfolding *l-perm-inv-def*
  by (*simp add*: *l-perm-inv-elims*[*OF assms*(*1*)] *l-pred-inv-maintained-no-update*
      l-seen-inv-maintained-no-update(*1*)[*OF l-perm-inv-elims*(*11*)[*OF assms*(*1*)] *assms*(*2*)]])

lemma *l-perm-inv-maintained-no-upd-3*:
  assumes *l-perm-inv* α *T B SA0 SA i*
  and $SA\ !\ i < length\ T$
  and $SA\ !\ i = 0$
shows *l-perm-inv* α *T B SA0 SA* (*Suc i*)
  unfolding *l-perm-inv-def*
  by (*simp add*: *l-perm-inv-elims*[*OF assms*(*1*)] *l-pred-inv-maintained-no-update*

l-seen-inv-maintained-no-update(*3*)[*OF* *l-perm-inv-elims*(*11*)[*OF* *assms*(*1*)] *assms*(*2*−)])

lemma *l-perm-inv-maintained-no-upd-4*:
  assumes *l-perm-inv* α *T B SA0 SA i*
  and *SA* ! $i <$ *length T*
  and *SA* ! $i =$ *Suc j*
  and *suffix-type T j* = *S-type*
shows *l-perm-inv* α *T B SA0 SA* (*Suc i*)
  unfolding *l-perm-inv-def*
  by (*simp add*: *l-perm-inv-elims*[*OF assms*(*1*)] *l-pred-inv-maintained-no-update*
  l-seen-inv-maintained-no-update(*4*)[*OF* *l-perm-inv-elims*(*11*)[*OF* *assms*(*1*)] *assms*(*2*−)])

lemmas *l-perm-inv-maintained-no-update* =
 l-perm-inv-maintained-no-upd-1 l-perm-inv-maintained-no-upd-2 l-perm-inv-maintained-no-upd-3
 l-perm-inv-maintained-no-upd-4

lemma *abs-induce-l-perm-step*:
  assumes *l-perm-inv* α *T B SA0 SA i*
  and *abs-induce-l-step* (*B*, *SA*, *i*) (α, *T*) = (B', SA', i')
shows *l-perm-inv* α *T* B' *SA0* SA' i'
proof (*cases* $i <$ *length SA* $\wedge$ *SA* ! $i <$ *length T*)
  assume *A*: $i <$ *length SA* $\wedge$ *SA* ! $i <$ *length T*
  show *?thesis*
  proof (*cases SA* ! *i*)
    case *0*
    then show *?thesis*
      using *A l-perm-inv-maintained-no-update*(*3*)[*OF assms*(*1*)] *assms*(*2*)
      by *force*
  next
    case (*Suc j*)
    assume *B*: *SA* ! $i =$ *Suc j*
    show *?thesis*
    proof (*cases suffix-type T j*)
      case *S-type*
      then show *?thesis*
        using *A B l-perm-inv-maintained-no-update*(*4*)[*OF assms*(*1*)] *assms*(*2*)
        by *force*
    next
      case *L-type*
      then show *?thesis*
        using *A B l-perm-inv-maintained*[*OF assms*(*1*)] *assms*(*2*)
        by (*clarsimp simp*: *Let-def*)
    qed
  qed
next
  assume *A*: $\neg$($i <$ *length SA* $\wedge$ *SA* ! $i <$ *length T*)

show *?thesis*
  using *l-perm-inv-maintained-no-update*(*1*,*2*)[*OF assms*(*1*)] *A assms*(*2*)
  by *force*
qed

lemma *abs-induce-l-base-perm-inv-maintained*:
  assumes *l-perm-inv* α *T B SA0 SA 0*
  and *abs-induce-l-base* α *T B SA* = (*B′*, *SA′*, *i*)
shows *l-perm-inv* α *T B′ SA0 SA′ i*
proof −
  let *?P* = λ(*B*, *SA*, *i*). *l-perm-inv* α *T B SA0 SA i*

  from *assms*(*2*)
  have *repeat* (*length T*) *abs-induce-l-step* (*B*, *SA*, *0*) (α, *T*) = (*B′*, *SA′*, *i*)
    by (*simp add*: *abs-induce-l-base-def*)
  moreover
  have $\bigwedge$*a*. *?P a* $\Longrightarrow$ *?P* (*abs-induce-l-step a* (α, *T*))
    using *abs-induce-l-perm-step* **by** *blast*
  ultimately show *?thesis*
    using *repeat-maintain-inv*[*of ?P abs-induce-l-step* (α, *T*) (*B*, *SA*, *0*) *length T*]
    using *assms*(*1*) **by** *auto*
qed

78 Sorted

lemma *l-suffix-sorted-inv-established*:
  assumes *l-bucket-init* α *T B*
  shows *l-suffix-sorted-inv* α *T B SA*
  unfolding *l-suffix-sorted-inv-def*
proof(*intro allI impI*)
  fix *b*
  assume *b* $\leq$ α (*Max* (*set T*))
  with *l-bucket-initD*[*OF assms*, *of b*]
  have *B* ! *b* = *bucket-start* α *T b* **.**
  then
  show *ordlistns.sorted*
      (*map* (*suffix T*)
      (*list-slice SA* (*bucket-start* α *T b*) (*B* ! *b*)))
    by *simp*
qed

lemma *l-prefix-sorted-inv-established*:
  assumes *l-bucket-init* α *T B*
  shows *l-prefix-sorted-inv* α *T B SA*
  unfolding *l-prefix-sorted-inv-def*
proof(*intro allI impI*)
  fix *b*
  assume *b* $\leq$ α (*Max* (*set T*))
  with *l-bucket-initD*[*OF assms*, *of b*]

have $B\ !\ b = \textit{bucket-start}\ \alpha\ T\ b$ **.**
then show *ordlistns.sorted* (*map* (*lms-prefix* T) (*list-slice SA* (*bucket-start* α T b) (B ! b)))
by *simp*
qed

lemma *l-sorted-inv-maintained-step*:
assumes *l-perm-inv* α *T B SA0 SA i*
and $i < \textit{length SA}$
and $SA\ !\ i = \textit{Suc}\ j$
and $\textit{Suc}\ j < \textit{length}\ T$
and *suffix-type T j = L-type*
and $k = \alpha\ (T\ !\ j)$
and $l = B\ !\ k$
and $b \leq \alpha\ (\textit{Max}\ (\textit{set}\ T))$
and $b \neq k$
and *ordlistns.sorted* (*map f* (*list-slice SA* (*bucket-start* α *T b*) (*B* ! *b*)))
shows *ordlistns.sorted* (*map f* (*list-slice* (*SA*[*l* := *j*]) (*bucket-start* α *T b*) (*B*[*k* := *Suc l*] ! *b*)))
proof −
let *?xs* = *list-slice* (*SA*[*l* := *j*]) (*bucket-start* α *T b*) (*B*[*k* := *Suc l*] ! *b*) **and**
?ys = *list-slice SA* (*bucket-start* α *T b*) (*B* ! *b*)

have $i < \textit{length}\ T$
by (*metis assms*(*1*,*2*) *l-perm-inv-def*)
hence $k \leq \alpha\ (\textit{Max}\ (\textit{set}\ T))$
using *assms*(*1*,*4*,*6*) *l-perm-inv-def strict-mono-less-eq* **by** *fastforce*

from ‹*Suc j* < *length T*›
have $j < \textit{length}\ T$
by *simp*

from *l-distinct-pred-inv-helper*[*OF* ‹*i* < *length SA*›
‹*SA* ! *i* = *Suc j*›
‹*Suc j* < *length T*›
‹*suffix-type T j* = *L-type*›
l-perm-inv-elims(*4*,*10*)[*OF assms*(*1*)]]
have $j \notin \textit{set SA}$
by *assumption*

from *l-bucket-ptr-inv-imp-less-l-bucket-end*[*OF l-perm-inv-elims*(*6*)[*OF assms*(*1*)]
‹*j* < *length T*›
‹*suffix-type T j* = *L-type*›
‹$j \notin$ *set SA*›
l-perm-inv-elims(*12*)[*OF assms*(*1*)]]
‹$k = \alpha\ (T\ !\ j)$›
‹$l = B\ !\ k$›
have $l < \textit{l-bucket-end}\ \alpha\ T\ k$
by *simp*

hence $l < length\ SA$
by $(metis\ assms(1)\ bucket\text{-}end\text{-}le\text{-}length\ dual\text{-}order.strict\text{-}trans1\ l\text{-}bucket\text{-}end\text{-}le\text{-}bucket\text{-}end\ l\text{-}perm\text{-}inv\text{-}def)$

have $B[k := Suc\ l]\ !\ b = B\ !\ b$
using $assms(9)$ **by** $auto$

have $l < bucket\text{-}start\ \alpha\ T\ b \lor B\ !\ b \le l$
proof $-$
have $b < k \lor k < b$
using $‹b \neq k›$ $less\text{-}linear$ **by** $blast$
moreover
have $b < k \Longrightarrow ?thesis$
proof $-$
assume $b < k$
hence $bucket\text{-}end\ \alpha\ T\ b \le bucket\text{-}start\ \alpha\ T\ k$
by $(simp\ add: less\text{-}bucket\text{-}end\text{-}le\text{-}start)$
hence $l\text{-}bucket\text{-}end\ \alpha\ T\ b \le bucket\text{-}start\ \alpha\ T\ k$
using $l\text{-}bucket\text{-}end\text{-}le\text{-}bucket\text{-}end\ le\text{-}trans$ **by** $blast$
with $l\text{-}bucket\text{-}ptr\text{-}inv\text{-}imp\text{-}le\text{-}l\text{-}bucket\text{-}end[OF\ l\text{-}perm\text{-}inv\text{-}elims(6)[OF\ assms(1)]\ ‹b \le \text{-}›]$
have $B\ !\ b \le bucket\text{-}start\ \alpha\ T\ k$
by $linarith$
with $l\text{-}bucket\text{-}ptr\text{-}inv\text{-}imp\text{-}ge\text{-}bucket\text{-}start[OF\ l\text{-}perm\text{-}inv\text{-}elims(6)[OF\ assms(1)]\ ‹k \le \text{-}›]$
show $?thesis$
using $assms(7)\ le\text{-}trans$ **by** $blast$
qed
moreover
have $k < b \Longrightarrow ?thesis$
proof $-$
assume $k < b$
hence $bucket\text{-}end\ \alpha\ T\ k \le bucket\text{-}start\ \alpha\ T\ b$
by $(simp\ add: less\text{-}bucket\text{-}end\text{-}le\text{-}start)$
hence $l\text{-}bucket\text{-}end\ \alpha\ T\ k \le bucket\text{-}start\ \alpha\ T\ b$
using $l\text{-}bucket\text{-}end\text{-}le\text{-}bucket\text{-}end\ le\text{-}trans$ **by** $blast$
with $‹l < l\text{-}bucket\text{-}end\ \alpha\ T\ k›$
show $?thesis$
using $less\text{-}le\text{-}trans$ **by** $blast$
qed
ultimately show $?thesis$
by $blast$
qed
with $‹B[k := Suc\ l]\ !\ b = B\ !\ b›$
$list\text{-}slice\text{-}update\text{-}unchanged\text{-}1$
$list\text{-}slice\text{-}update\text{-}unchanged\text{-}2$
have $?xs = ?ys$
by $auto$
then show $?thesis$

    using *assms*(*10*) **by** *auto*
qed

lemma *l-suffix-sorted-inv-maintained-step*:
  assumes *l-perm-inv* α *T B SA0 SA i*
  and *l-suffix-sorted-pre* α *T SA0*
  and *l-suffix-sorted-inv* α *T B SA*
  and $i <$ *length SA*
  and *SA* ! $i =$ *Suc j*
  and *Suc j* $<$ *length T*
  and *suffix-type T j* $=$ *L-type*
  and $k = \alpha$ (*T* ! *j*)
  and $l = B$! k
shows *l-suffix-sorted-inv* α *T* (*B*[k := *Suc l*]) (*SA*[l := j])
  unfolding *l-suffix-sorted-inv-def*
proof (*safe*)
  fix *b*
  assume $b \leq \alpha$ (*Max* (*set T*))
  let *?xs* = *list-slice* (*SA*[l := j]) (*bucket-start* α *T b*) (*B*[k := *Suc l*] ! *b*) **and**
      ?ys = *list-slice SA* (*bucket-start* α *T b*) (*B* ! *b*)

  have $i <$ *length T*
    by (*metis assms*(*1*,*4*) *l-perm-inv-def*)
  hence $k \leq \alpha$ (*Max* (*set T*))
    using *assms*(*1*,*6*,*8*) *l-perm-inv-def strict-mono-less-eq* **by** *fastforce*

   from ‹*Suc j* $<$ *length T*›
    have $j <$ *length T*
      by *simp*

    from *l-distinct-pred-inv-helper*[*OF* ‹$i <$ *length SA*›
                                ‹*SA* ! $i =$ *Suc j*›
                                ‹*Suc j* $<$ *length T*›
                                ‹*suffix-type T j* $=$ *L-type*›
                                l-perm-inv-elims(*4*,*10*)[*OF assms*(*1*)]]
    have $j \notin$ *set SA*
      by *assumption*

   from *l-bucket-ptr-inv-imp-less-l-bucket-end*[*OF l-perm-inv-elims*(*6*)[*OF assms*(*1*)]
                                ‹$j <$ *length T*›
                                ‹*suffix-type T j* $=$ *L-type*›
                                ‹$j \notin$ *set SA*›
                                l-perm-inv-elims(*12*)[*OF assms*(*1*)]]
        ‹$k = \alpha$ (*T* ! *j*)›
        ‹$l = B$! k›
    have $l <$ *l-bucket-end* α *T k*
      by *simp*
    hence $l <$ *length SA*
     by (*metis assms*(*1*) *bucket-end-le-length dual-order.strict-trans1 l-bucket-end-le-bucket-end*

l-perm-inv-def)

have $b = k \vee b \neq k$
by *simp*
moreover
have $b \neq k \Longrightarrow$ *ordlistns.sorted* (*map* (*suffix* T) *?xs*)
using ‹$b \leq \alpha$ (*Max* (*set* T))› *assms l-sorted-inv-maintained-step l-suffix-sorted-inv-def*
by *blast*
moreover
have $b = k \Longrightarrow$ *ordlistns.sorted* (*map* (*suffix* T) *?xs*)
proof –
assume $b = k$
hence $B[k := Suc\ l]\ !\ b = Suc\ l$
using ‹$b \leq \alpha$ (*Max* (*set* T))› *assms*(*1*) *l-perm-inv-elims*(*1*) **by** *fastforce*

have $SA[l := j]\ !\ l = j$
by (*simp add*: ‹$l <$ *length SA*›)

from *list-slice-update-unchanged-2*[*of B* ! *b j* - *bucket-start* α *T b*]
have *list-slice* (*SA*[$l := j$]) (*bucket-start* α *T b*) (*B* ! *b*) = *?ys*
using ‹$b = k$› *assms*(*9*)
by (*simp add*: *list-slice-update-unchanged-2*)
hence *?xs* = *?ys* @ *list-slice* (*SA*[$l := j$]) (*B* ! *b*) (*B*[$k :=$ *Suc l*] ! *k*)
by (*metis Suc-n-not-le-n* ‹*B*[$k :=$ *Suc l*] ! b = *Suc l*› ‹$b = k$› ‹$k \leq \alpha$ (*Max* (*set* T))› *assms*(*1*,*9*)
l-bucket-ptr-inv-imp-ge-bucket-start l-perm-inv-elims(*6*) *linear list-slice-append*)
moreover
have *list-slice* (*SA*[$l := j$]) (*B* ! *b*) (*B*[$k :=$ *Suc l*] ! *k*) = [*j*]
by (*metis* ‹*B*[$k :=$ *Suc l*] ! b = *Suc l*› ‹*SA*[$l := j$] ! $l = j$›
‹$b = k$› ‹$l <$ *length SA*› *assms*(*9*)
length-list-update lessI list-slice-Suc list-slice-n-n)
ultimately have *?xs* = *?ys* @ [*j*]
by *simp*
hence *map* (*suffix* T) *?xs* = (*map* (*suffix* T) *?ys*) @ [*suffix T j*]
by *simp*
moreover
have *ordlistns.sorted* ((*map* (*suffix* T) *?ys*) @ [*suffix T j*])
proof –
from *l-suffix-sorted-invD*[*OF assms*(*3*) ‹$b \leq$ -›]
have *ordlistns.sorted* (*map* (*suffix* T) *?ys*) **.**
moreover
have *ordlistns.sorted* [*suffix T j*]
by *simp*
moreover
have $\forall x \in$ *set* (*map* (*suffix* T) *?ys*).
$\forall y \in$ *set* [*suffix T j*].
list-less-eq-ns x y
proof(*safe*)

fix *x y*
assume
  $x \in set\ (map\ (suffix\ T)\ ?ys)$
  $y \in set\ [suffix\ T\ j]$
hence $y = suffix\ T\ j$
  by *simp*

have *A*: $length\ ?ys = B\ !\ b - bucket\text{-}start\ \alpha\ T\ b$
  using ‹$b = k$› ‹$l < length\ SA$› *assms*(*9*) *min-def* **by** *auto*

from *in-set-conv-nth*[*THEN iffD1*, *OF* ‹$x \in set\ (map\ (suffix\ T)\ ?ys)$›]
have $\exists\, i'.\ x = suffix\ T\ (SA\ !\ i') \wedge$
    $bucket\text{-}start\ \alpha\ T\ b \leq i' \wedge i' < B\ !\ b$
  by (*metis A add.commute le-add1 length-map*
     less-diff-conv nth-list-slice nth-map)
then obtain j' **where** j'*-assms*:
  $x = suffix\ T\ (SA\ !\ j')$
  $bucket\text{-}start\ \alpha\ T\ b \leq j'$
  $j' < B\ !\ b$
  by *blast*
hence j'*-less*: $j' < length\ SA$
  using ‹$b = k$› ‹$l < length\ SA$› *assms*(*9*) *dual-order.strict-trans*
  by *blast*
with *l-locations-inv-D*
    [*OF l-perm-inv-elims*(*9*)[*OF assms*(*1*)] ‹$b \leq$ -› - j'*-assms*(*2*,*3*)]
have $SA\ !\ j' < length\ T$
   $suffix\text{-}type\ T\ (SA\ !\ j') = L\text{-}type$
   $\alpha\ (T\ !\ (SA\ !\ j')) = b$
  by *blast+*
with *l-pred-inv-D*[*OF l-perm-inv-elims*(*10*)[*OF assms*(*1*)] j'*-less*]
have $\exists j{<}length\ SA.$
    $SA\ !\ j = Suc\ (SA\ !\ j') \wedge$
    $SA\ !\ j < length\ T \wedge$
    $j < j' \wedge$
    $j < i$
  by *blast*
then obtain i' **where** i'*-assms*:
  $i' < length\ SA$
  $SA\ !\ i' = Suc\ (SA\ !\ j')$
  $SA\ !\ i' < length\ T$
  $i' < j'$
  $i' < i$
  by *blast*

have $\alpha\ (T\ !\ j) = b$
  using ‹$b = k$› *assms*(*8*) **by** *blast*
with ‹$\alpha\ (T\ !\ (SA\ !\ j')) = b$›
have $T\ !\ (SA\ !\ j') = T\ !\ j$
  by (*metis* (*no-types*, *lifting*) *assms*(*1*) *l-perm-inv-elims*(*12*))

less-le not-le strict-mono-less-eq)
moreover
have $x = T ! (SA ! j') \# suffix\ T\ (SA ! i')$
using ‹$SA ! i' = Suc\ (SA ! j')$›
‹$SA ! j' < length\ T$›
‹$x = suffix\ T\ (SA ! j')$›
suffix-cons-Suc
by *auto*
moreover
have $y = T ! j \# suffix\ T\ (SA ! i)$
using ‹$j < length\ T$› ‹$y = suffix\ T\ j$›
suffix-cons-Suc
‹$SA ! i = Suc\ j$›
by *auto*
ultimately
have *list-less-eq-ns x y* =
list-less-eq-ns (*suffix* T ($SA ! i'$)) (*suffix* T ($SA ! i$))
using *list-less-eq-ns-cons*
[*of* $T ! (SA ! j')$
suffix T ($SA ! i'$)
$T ! j$
suffix T ($SA ! i$)]
by *simp*
moreover
have *list-less-eq-ns* (*suffix* T ($SA ! i'$)) (*suffix* T ($SA ! i$))
proof −
have $i' < length\ T$
using ‹$i < length\ T$› ‹$i' < i$› *order.strict-trans* **by** *blast*
with *index-in-bucket-interval-gen*
[*of* i' T α,
OF - *l-perm-inv-elims*(*12*)[*OF assms*(*1*)]]
obtain *b0* **where** *b0-assms*:
$b0 \leq \alpha\ (Max\ (set\ T))$
bucket-start α T *b0* $\leq i'$
$i' <$ *bucket-end* α T *b0*
by *blast*
with *same-bucket-same-hd*
[*OF l-perm-inv-elims*(*8*,*9*,*6*,*5*)[*OF assms*(*1*)],
of b0 i']
l-perm-inv-elims(*2*,*3*,*14*,*15*)[*OF assms*(*1*)]
have $\alpha\ (T ! (SA ! i')) = b0$
using ‹$SA ! i' < length\ T$› ‹$i' < length\ SA$› **by** *auto*

from *index-in-bucket-interval-gen*[*OF* ‹$i < length\ T$› *l-perm-inv-elims*(*12*)[*OF assms*(*1*)]]
obtain *b1* **where** *b1-assms*:
$b1 \leq \alpha\ (Max\ (set\ T))$
bucket-start α T *b1* $\leq i$
$i <$ *bucket-end* α T *b1*

by *blast*
with *same-bucket-same-hd*
[*OF l-perm-inv-elims*(*8*,*9*,*6*,*5*)[*OF assms*(*1*)],
of b1 i]
l-perm-inv-elims(*2*,*3*,*14*,*15*)[*OF assms*(*1*)]
have α (*T* ! (*SA* ! *i*)) = *b1*
using *assms*(*4*−*6*) **by** *auto*

have *b0* ≤ *b1*
proof (*rule ccontr*)
assume ¬ *b0* ≤ *b1*
hence *b1* < *b0*
by *auto*
hence *bucket-end* α *T b1* ≤ *bucket-start* α *T b0*
by (*simp add*: *less-bucket-end-le-start*)
with ‹*i* < *bucket-end* α *T b1*› ‹*bucket-start* α *T b0* ≤ *i'*› ‹*i'* < *i*›
show *False*
by *linarith*
qed
hence *b0* < *b1* ∨ *b0* = *b1*
by *linarith*
moreover
have *b0* < *b1* ⟹ *?thesis*
proof −
assume *b0* < *b1*
with ‹α (*T* ! (*SA* ! *i'*)) = *b0*›
‹α (*T* ! (*SA* ! *i*)) = *b1*›
have *T* ! (*SA* ! *i'*) < *T* ! (*SA* ! *i*)
using *assms*(*1*) *l-perm-inv-elims*(*12*) *strict-mono-less* **by** *blast*
moreover
have ∃ *as*. *suffix T* (*SA* ! *i'*) = *T* ! (*SA* ! *i'*) # *as*
using ‹*SA* ! *i'* < *length T*› *suffix-cons-Suc* **by** *blast*
then obtain *as* **where** *as-assm*:
suffix T (*SA* ! *i'*) = *T* ! (*SA* ! *i'*) # *as*
by *blast*
moreover
have ∃ *bs*. *suffix T* (*SA* ! *i*) = *T* ! (*SA* ! *i*) # *bs*
by (*metis Cons-nth-drop-Suc assms*(*5*,*6*))
then obtain *bs* **where** *bs-assm*:
suffix T (*SA* ! *i*) = *T* ! (*SA* ! *i*) # *bs*
by *blast*
ultimately show *?thesis*
by (*simp add*: *less-le list-less-eq-ns-cons*)
qed
moreover
have *b0* = *b1* ⟹ *?thesis*
proof −
assume *b0* = *b1*
hence α (*T* ! (*SA* ! *i'*)) = α (*T* ! (*SA* ! *i*))

by (*simp add*: ‹α (*T* ! (*SA* ! *i'*)) = *b0*› ‹α (*T* ! (*SA* ! *i*)) = *b1*›)
hence *T* ! (*SA* ! *i'*) = *T* ! (*SA* ! *i*)
by (*metis* (*no-types*, *opaque-lifting*) *assms*(*1*) *strict-mono-less*
l-perm-inv-elims(*12*) *not-less-iff-gr-or-eq*)

have *i* < *bucket-end* α *T b0*
by (*simp add*: ‹*b0* = *b1*› ‹*i* < *bucket-end* α *T b1*›)

from *unknown-range-values*[*OF l-perm-inv-elims*(*8*,*5*)[*OF assms*(*1*)] - -
l-perm-inv-elims(*14*,*15*)[*OF assms*(*1*)]
‹*b0* ≤ α -›]
l-perm-inv-elims(*2*,*3*)[*OF assms*(*1*)]
have *i* < *B* ! *b0* ∨ *lms-bucket-start* α *T b0* ≤ *i*
using *assms*(*5*) *assms*(*6*) *not-le* **by** *force*

from *unknown-range-values*[*OF l-perm-inv-elims*(*8*,*5*)[*OF assms*(*1*)] - -
l-perm-inv-elims(*14*,*15*)[*OF assms*(*1*)]
‹*b0* ≤ α -›]
l-perm-inv-elims(*2*,*3*)[*OF assms*(*1*)]
‹*SA* ! *i'* < *length T*›
have *i'* < *B* ! *b0* ∨
lms-bucket-start α *T b0* ≤ *i'*
using *not-le* **by** *force*
moreover
have *i'* < *B* ! *b0* ⟹ *?thesis*
proof −
assume *i'* < *B* ! *b0*
have *i* < *B* ! *b0* ⟹ *?thesis*
proof −
assume *i-b0-assm*: *i* < *B* ! *b0*
let *?xs* = *list-slice SA* (*bucket-start* α *T b0*) (*B* ! *b0*)

have *i'* ≤ *i*
by (*simp add*: ‹*i'* < *i*› *less-imp-le-nat*)
from *l-suffix-sorted-invD*[*OF assms*(*3*) ‹*b0* ≤ α -›]
have *ordlistns.sorted* (*map* (*suffix T*) *?xs*) **.**
with *ordlistns.list-slice-sorted-nth-mono*
[*OF* - *b0-assms*(*2*) ‹*i'* ≤ *i*› *i-b0-assm*,
of map (*suffix T*) *SA*]
show *?thesis*
by (*metis i'-assms*(*1*) *assms*(*4*) *length-map*
map-list-slice nth-map)
qed
moreover
have *lms-bucket-start* α *T b0* ≤ *i* ⟹ *?thesis*
proof −
assume *lms-bucket-start* α *T b0* ≤ *i*
with *lms-init-D*
[*OF lms-init-unchanged*

```
              [OF l-perm-inv-elims(8)[OF assms(1)]],
            OF - - l-perm-inv-elims(14)[OF assms(1)]
              ‹b0 ≤ α -›]
          l-perm-inv-elims(2,3)[OF assms(1)]
          ‹i < bucket-end α T b0›
    have SA ! i ∈ lms-bucket α T b0
      by (metis bucket-end-le-length list-slice-nth-mem)
    hence suffix-type T (SA ! i) = S-type
      by (metis ‹b0 ≤ α (Max (set T))›
                ‹i < bucket-end α T b0›
                ‹length SA = length SA0›
                ‹length SA0 = length T›
                ‹lms-bucket-start α T b0 ≤ i›
                ‹lms-init α T SA0›
                assms(1) abs-is-lms-def l-perm-inv-elims(8)
                lms-init-nth lms-init-unchanged)
    moreover
    from l-locations-inv-D
            [OF l-perm-inv-elims(9)[OF assms(1)]
                ‹b0 ≤ α -›
                i′-assms(1)
                b0-assms(2)
                ‹i′ < B ! b0›]
    have suffix-type T (SA ! i′) = L-type SA ! i′ < length T
      by blast+
    ultimately show ?thesis
      using l-less-than-s-type-suffix[of SA ! i T SA ! i′]
      by (simp add: ordlistns.less-imp-le ‹α (T ! (SA ! i′)) = b0›
                  ‹T ! (SA ! i′) = T ! (SA ! i)› assms(5,6))
  qed
  ultimately
  show ?thesis
    using ‹i < B ! b0 ∨ lms-bucket-start α T b0 ≤ i›
    by blast
qed
moreover
have lms-bucket-start α T b0 ≤ i′ ⟹ ?thesis
proof −
  assume lms-bucket-start α T b0 ≤ i′

  let ?xs =
    list-slice SA (lms-bucket-start α T b0) (bucket-end α T b0)

  from l-suffix-sorted-pre-maintained
        [OF l-perm-inv-elims(8)[OF assms(1)]]
       l-perm-inv-elims(2,3)[OF assms(1)]
       l-suffix-sorted-preD[of α T SA b0, OF - ‹b0 ≤ α -›]
  have ordlistns.sorted (map (suffix T) ?xs)
    by (simp add: assms(2))
```

```
              with ordlistns.list-slice-sorted-nth-mono
                      [of map (suffix T) SA
                          lms-bucket-start α T b0
                          bucket-end α T b0 i′ i]
                    ‹i < bucket-end α T b0›
                    ‹i′ < i›
                    i′-assms(1)
                    ‹lms-bucket-start α T b0 ≤ i′›
              show ?thesis
                by (metis assms(4) length-map map-list-slice
                          not-less not-less-iff-gr-or-eq nth-map)
            qed
            ultimately show ?thesis
              by blast
          qed
          ultimately show ?thesis
            by blast
        qed
        ultimately show list-less-eq-ns x y
          by simp
      qed
      ultimately show ?thesis
        using ordlistns.sorted-append[of map (suffix T) ?ys [suffix T j]]
        by blast
    qed
    ultimately show ?thesis
      by simp
  qed
  ultimately show ordlistns.sorted (map (suffix T) ?xs)
    by blast
qed

lemma l-prefix-sorted-inv-maintained-step:
  assumes l-perm-inv α T B SA0 SA i
  and     l-prefix-sorted-pre α T SA0
  and     l-prefix-sorted-inv α T B SA
  and     i < length SA
  and     SA ! i = Suc j
  and     Suc j < length T
  and     suffix-type T j = L-type
  and     k = α (T ! j)
  and     l = B ! k
shows l-prefix-sorted-inv α T (B[k := Suc l]) (SA[l := j])
  unfolding l-prefix-sorted-inv-def
proof (safe)
  fix b
  assume b ≤ α (Max (set T))
  let ?xs = list-slice (SA[l := j]) (bucket-start α T b) (B[k := Suc l] ! b)
  and ?ys = list-slice SA (bucket-start α T b) (B ! b)
```

have $i < length\ T$
  by $(metis\ assms(1,4)\ l\text{-}perm\text{-}inv\text{-}def)$
hence $k \leq \alpha\ (Max\ (set\ T))$
  using $assms(1,6,8)\ l\text{-}perm\text{-}inv\text{-}def\ strict\text{-}mono\text{-}less\text{-}eq$ **by** $fastforce$

 from $\langle Suc\ j < length\ T\rangle$
  have $j < length\ T$
    by $simp$

  from $l\text{-}distinct\text{-}pred\text{-}inv\text{-}helper$
        $[OF\ \langle i < length\ SA\rangle$
            $\langle SA\ !\ i = Suc\ j\rangle$
            $\langle Suc\ j < length\ T\rangle$
            $\langle suffix\text{-}type\ T\ j = L\text{-}type\rangle$
            $l\text{-}perm\text{-}inv\text{-}elims(4,10)[OF\ assms(1)]]$
  have $j \notin set\ SA$
    by $assumption$

  from $l\text{-}bucket\text{-}ptr\text{-}inv\text{-}imp\text{-}less\text{-}l\text{-}bucket\text{-}end$
        $[OF\ l\text{-}perm\text{-}inv\text{-}elims(6)[OF\ assms(1)]$
            $\langle j < length\ T\rangle$
            $\langle suffix\text{-}type\ T\ j = L\text{-}type\rangle$
            $\langle j \notin set\ SA\rangle$
            $l\text{-}perm\text{-}inv\text{-}elims(12)[OF\ assms(1)]]$
      $\langle k = \alpha\ (T\ !\ j)\rangle$
      $\langle l = B\ !\ k\rangle$
  have $l < l\text{-}bucket\text{-}end\ \alpha\ T\ k$
    by $simp$
  hence $l < length\ SA$
    by $(metis\ bucket\text{-}end\text{-}le\text{-}length\ dual\text{-}order.strict\text{-}trans1$
          $assms(1)\ l\text{-}bucket\text{-}end\text{-}le\text{-}bucket\text{-}end\ l\text{-}perm\text{-}inv\text{-}def)$

have $b = k \vee b \neq k$
  by $simp$
moreover
have $b \neq k \Longrightarrow$
      $ordlistns.sorted\ (map\ (lms\text{-}prefix\ T)\ ?xs)$
  using $\langle b \leq \alpha\ (Max\ (set\ T))\rangle$
        $assms\ l\text{-}sorted\text{-}inv\text{-}maintained\text{-}step\ l\text{-}prefix\text{-}sorted\text{-}inv\text{-}def$
  by $blast$
moreover
have $b = k \Longrightarrow$
      $ordlistns.sorted\ (map\ (lms\text{-}prefix\ T)\ ?xs)$
proof $-$
  assume $b = k$
  hence $B[k := Suc\ l]\ !\ b = Suc\ l$
    using $\langle b \leq \alpha\ (Max\ (set\ T))\rangle\ assms(1)\ l\text{-}perm\text{-}inv\text{-}elims(1)$
    by $fastforce$

```
have SA[l := j] ! l = j
  by (simp add: ‹l < length SA›)

from list-slice-update-unchanged-2
have list-slice (SA[l := j]) (bucket-start α T b) (B ! b) = ?ys
  using ‹b = k› assms(9)
  by fast
hence ?xs = ?ys @ list-slice (SA[l := j]) (B ! b) (B[k := Suc l] ! k)
  by (metis ‹B[k := Suc l] ! b = Suc l› ‹b = k›
        ‹k ≤ α (Max (set T))›
        assms(1,9) Suc-n-not-le-n list-slice-append linear
        l-bucket-ptr-inv-imp-ge-bucket-start l-perm-inv-elims(6))
moreover
have list-slice (SA[l := j]) (B ! b) (B[k := Suc l] ! k) = [j]
  by (metis ‹B[k := Suc l] ! b = Suc l›
        ‹SA[l := j] ! l = j›
        ‹b = k›
        ‹l < length SA›
        assms(9) length-list-update lessI list-slice-Suc list-slice-n-n)
ultimately
have ?xs = ?ys @ [j]
  by simp
hence map (lms-prefix T) ?xs = (map (lms-prefix T) ?ys) @ [lms-prefix T j]
  by simp
moreover
have ordlistns.sorted ((map (lms-prefix T) ?ys) @ [lms-prefix T j])
proof −
  from l-prefix-sorted-invD[OF assms(3) ‹b ≤ -›]
  have ordlistns.sorted (map (lms-prefix T) ?ys) .
  moreover
  have ordlistns.sorted [lms-prefix T j]
    by simp
  moreover
  have ∀ x ∈ set (map (lms-prefix T) ?ys). ∀ y ∈ set [lms-prefix T j].
        list-less-eq-ns x y
  proof(safe)
    fix x y
    assume x ∈ set (map (lms-prefix T) ?ys) y ∈ set [lms-prefix T j]
    hence y = lms-prefix T j
      by simp

    have A: length ?ys = B ! b − bucket-start α T b
      using ‹b = k› ‹l < length SA› assms(9) min-def by auto

    from in-set-conv-nth[THEN iffD1, OF ‹x ∈ set (map (lms-prefix T) ?ys)›]
    have ∃ i'. x = lms-prefix T (SA ! i') ∧
            bucket-start α T b ≤ i' ∧
            i' < B ! b
```

by (*metis A add.commute le-add1 length-map less-diff-conv*
nth-list-slice nth-map)
then obtain j' **where** j'*-assms*:
$x = $ *lms-prefix* T $(SA ! j')$
bucket-start α T $b \leq j'$
$j' < B ! b$
by *blast*
hence $j' <$ *length SA*
using ‹$b = k$› ‹$l <$ *length SA*›
assms(*9*) *dual-order.strict-trans* **by** *blast*
with *l-locations-inv-D*
[*OF l-perm-inv-elims*(*9*)[*OF assms*(*1*)]
‹$b \leq$ -› -
j'*-assms*(*2*,*3*)]
have $SA ! j' <$ *length* T
suffix-type T $(SA ! j') = $ *L-type*
α $(T ! (SA ! j')) = b$
by *blast*+
with *l-pred-inv-D*[*OF l-perm-inv-elims*(*10*)[*OF assms*(*1*)] ‹$j' <$ *length SA*›]
have $\exists j<$*length SA*. $SA ! j = $ *Suc* $(SA ! j') \wedge SA ! j <$ *length* $T \wedge j < j' \wedge$
$j < i$
by *blast*
then obtain i' **where**
$i' <$ *length SA*
$SA ! i' = $ *Suc* $(SA ! j')$
$SA ! i' <$ *length* T
$i' < j'$
$i' < i$
by *blast*

have α $(T ! j) = b$
using ‹$b = k$› *assms*(*8*) **by** *blast*
with ‹α $(T ! (SA ! j')) = b$›
have $T ! (SA ! j') = T ! j$
by (*metis* (*no-types*, *lifting*) *assms*(*1*) *l-perm-inv-elims*(*12*) *less-le not-le*
strict-mono-less-eq)
moreover
have $x = T ! (SA ! j')$ # *lms-prefix* T $(SA ! i')$
by (*simp add*: ‹$SA ! i' = $ *Suc* $(SA ! j')$› ‹$SA ! j' <$ *length* T›
‹*suffix-type* T $(SA ! j') = $ *L-type*› ‹$x = $ *lms-prefix* T $(SA ! j')$›
l-type-lms-prefix-cons)
moreover
have $y = T ! j$ # *lms-prefix* T $(SA ! i)$
by (*simp add*: ‹$j <$ *length* T› ‹$y = $ *lms-prefix* T j› *assms*(*5*,*7*)
l-type-lms-prefix-cons)
ultimately have
list-less-eq-ns x y =
list-less-eq-ns (*lms-prefix* T $(SA ! i')$) (*lms-prefix* T $(SA ! i)$)
using *list-less-eq-ns-cons*[*of* $T ! (SA ! j')$ *lms-prefix* T $(SA ! i')$ $T ! j$

lms-prefix T (*SA* ! *i*)]
by *simp*
moreover
have *list-less-eq-ns* (*lms-prefix T* (*SA* ! *i'*)) (*lms-prefix T* (*SA* ! *i*))
proof −
have $i' <$ *length T*
using ‹$i <$ *length T*› ‹$i' < i$› *order.strict-trans* **by** *blast*
with *index-in-bucket-interval-gen*[*of i' T α*, *OF* - *l-perm-inv-elims*(*12*)[*OF* *assms*(*1*)]]
obtain *b0* **where**
b0 ≤ *α* (*Max* (*set T*))
bucket-start α T b0 ≤ *i'*
i' < *bucket-end α T b0*
by *blast*
with *same-bucket-same-hd*[*OF l-perm-inv-elims*(*8*,*9*,*6*,*5*)[*OF assms*(*1*)], *of b0 i'*]
l-perm-inv-elims(*2*,*3*,*14*,*15*)[*OF assms*(*1*)]
have *α* (*T* ! (*SA* ! *i'*)) = *b0*
using ‹*SA* ! *i'* < *length T*› ‹*i'* < *length SA*› **by** *auto*

from *index-in-bucket-interval-gen*[*OF* ‹*i* < *length T*› *l-perm-inv-elims*(*12*)[*OF* *assms*(*1*)]]
obtain *b1* **where**
b1 ≤ *α* (*Max* (*set T*))
bucket-start α T b1 ≤ *i*
i < *bucket-end α T b1*
by *blast*
with *same-bucket-same-hd*[*OF l-perm-inv-elims*(*8*,*9*,*6*,*5*)[*OF assms*(*1*)], *of b1 i*]
l-perm-inv-elims(*2*,*3*,*14*,*15*)[*OF assms*(*1*)]
have *α* (*T* ! (*SA* ! *i*)) = *b1*
using *assms*(*4*−*6*) **by** *auto*

have *b0* ≤ *b1*
proof (*rule ccontr*)
assume ¬*b0* ≤ *b1*
hence *b1* < *b0*
by *auto*
hence *bucket-end α T b1* ≤ *bucket-start α T b0*
by (*simp add*: *less-bucket-end-le-start*)
with ‹*i* < *bucket-end α T b1*› ‹*bucket-start α T b0* ≤ *i'*› ‹*i'* < *i*›
show *False*
by *linarith*
qed
hence *b0* < *b1* ∨ *b0* = *b1*
by *linarith*
moreover
have *b0* < *b1* ⟹ *?thesis*
proof −

assume *b0* < *b1*
with ‹α (*T* ! (*SA* ! *i*′)) = *b0*› ‹α (*T* ! (*SA* ! *i*)) = *b1*›
have *T* ! (*SA* ! *i*′) < *T* ! (*SA* ! *i*)
using *assms*(*1*) *l-perm-inv-elims*(*12*) *strict-mono-less* **by** *blast*
moreover
have ∃ *as*. *lms-prefix T* (*SA* ! *i*′) = *T* ! (*SA* ! *i*′) # *as*
by (*metis* ‹*SA* ! *i*′ < *length T*› *lms-slice-hd*
lms-lms-prefix not-lms-imp-next-eq-lms-prefix)
then obtain *as* **where**
lms-prefix T (*SA* ! *i*′) = *T* ! (*SA* ! *i*′) # *as*
by *blast*
moreover
have ∃ *bs*. *lms-prefix T* (*SA* ! *i*) = *T* ! (*SA* ! *i*) # *bs*
by (*metis* (*full-types*) *SL-types.exhaust assms*(*5*−*7*) *abs-is-lms-def*
l-type-lms-prefix-cons lms-lms-prefix)
then obtain *bs* **where**
lms-prefix T (*SA* ! *i*) = *T* ! (*SA* ! *i*) # *bs*
by *blast*
ultimately show *?thesis*
by (*simp add*: *less-le list-less-eq-ns-cons*)
qed
moreover
have *b0* = *b1* ⟹ *?thesis*
proof −
assume *b0* = *b1*
hence α (*T* ! (*SA* ! *i*′)) = α (*T* ! (*SA* ! *i*))
by (*simp add*: ‹α (*T* ! (*SA* ! *i*′)) = *b0*› ‹α (*T* ! (*SA* ! *i*)) = *b1*›)
hence *T* ! (*SA* ! *i*′) = *T* ! (*SA* ! *i*)
by (*metis* (*no-types*, *opaque-lifting*) *assms*(*1*) *strict-mono-less*
l-perm-inv-elims(*12*) *not-less-iff-gr-or-eq*)

have *i* < *bucket-end* α *T b0*
by (*simp add*: ‹*b0* = *b1*› ‹*i* < *bucket-end* α *T b1*›)

from *unknown-range-values*
[*OF l-perm-inv-elims*(*8*,*5*)[*OF assms*(*1*)] - -
l-perm-inv-elims(*14*,*15*)[*OF assms*(*1*)] ‹*b0* ≤ α -›]
l-perm-inv-elims(*2*,*3*)[*OF assms*(*1*)]
have *i* < *B* ! *b0* ∨ *lms-bucket-start* α *T b0* ≤ *i*
using *assms*(*5*) *assms*(*6*) *not-le* **by** *force*

from *unknown-range-values*
[*OF l-perm-inv-elims*(*8*,*5*)[*OF assms*(*1*)] - -
l-perm-inv-elims(*14*,*15*)[*OF assms*(*1*)] ‹*b0* ≤ α -›]
l-perm-inv-elims(*2*,*3*)[*OF assms*(*1*)]
‹*SA* ! *i*′ < *length T*›
have *i*′ < *B* ! *b0* ∨ *lms-bucket-start* α *T b0* ≤ *i*′
using *not-le* **by** *force*
moreover

have $i' < B\ !\ b0 \Longrightarrow$ *?thesis*
proof $-$
  assume $i' < B\ !\ b0$
  have $i < B\ !\ b0 \Longrightarrow$ *?thesis*
  proof $-$
    assume $i < B\ !\ b0$
    let *?xs* = *list-slice SA* (*bucket-start* α *T b0*) (*B* ! *b0*)

    have $i' \leq i$
      by (*simp add*: ‹$i' < i$› *less-imp-le-nat*)
    from *l-prefix-sorted-invD*[*OF assms*(*3*) ‹$b0 \leq \alpha$ -›]
    have *ordlistns.sorted* (*map* (*lms-prefix T*) *?xs*) **.**
    with *ordlistns.list-slice-sorted-nth-mono*
        [*OF* - ‹*bucket-start* α *T b0* $\leq i'$› ‹$i' \leq i$›
          ‹$i < B\ !\ b0$›,
        of map (*lms-prefix T*) *SA*]
    show *?thesis*
        by (*metis* ‹$i' <$ *length SA*› *assms*(*4*) *length-map map-list-slice*
nth-map)
  qed
  moreover
  have *lms-bucket-start* α *T b0* $\leq i \Longrightarrow$ *?thesis*
  proof $-$
    assume *lms-bucket-start* α *T b0* $\leq i$
    with *lms-init-D*[*OF lms-init-unchanged*
            [*OF l-perm-inv-elims*(*8*)[*OF assms*(*1*)]],
          OF - - *l-perm-inv-elims*(*14*)[*OF assms*(*1*)] ‹$b0 \leq \alpha$ -›]
      l-perm-inv-elims(*2*,*3*)[*OF assms*(*1*)]
      ‹$i <$ *bucket-end* α *T b0*›
    have *SA* ! $i \in$ *lms-bucket* α *T b0*
      by (*metis bucket-end-le-length list-slice-nth-mem*)
    hence *suffix-type T* (*SA* ! *i*) = *S-type*
      by (*metis* ‹$b0 \leq \alpha$ (*Max* (*set T*))›
          ‹$i <$ *bucket-end* α *T b0*›
          ‹*length SA* = *length SA0*›
          ‹*length SA0* = *length T*›
          ‹*lms-bucket-start* α *T b0* $\leq i$›
          ‹*lms-init* α *T SA0*› *assms*(*1*)
          abs-is-lms-def l-perm-inv-elims(*8*) *lms-init-nth*
          lms-init-unchanged)
    moreover
    from *l-locations-inv-D*[*OF l-perm-inv-elims*(*9*)[*OF assms*(*1*)] ‹$b0 \leq$
α -›
          ‹$i' <$ *length SA*› ‹*bucket-start* α *T b0* $\leq i'$›
          ‹$i' < B\ !\ b0$›]
    have *suffix-type T* (*SA* ! i') = *L-type SA* ! $i' <$ *length T*
      by *blast+*
    ultimately show *?thesis*
      using *lms-prefix-l-less-than-s-type*[*of SA* ! *i T SA* ! i']

```
        by (simp add: ‹T ! (SA ! i') = T ! (SA ! i)› assms(5−6)
                      lms-prefix-l-less-than-s-type
                      ordlistns.dual-order.strict-implies-order)
      qed
      ultimately show ?thesis
        using ‹i < B ! b0 ∨ lms-bucket-start α T b0 ≤ i› by blast
    qed
    moreover
    have lms-bucket-start α T b0 ≤ i' ⟹ ?thesis
    proof −
      assume lms-bucket-start α T b0 ≤ i'

      let ?xs = list-slice SA (lms-bucket-start α T b0) (bucket-end α T b0)

  from l-prefix-sorted-pre-maintained[OF l-perm-inv-elims(8)[OF assms(1)]]
          l-perm-inv-elims(2,3)[OF assms(1)]
          l-prefix-sorted-preD[of α T SA b0, OF - ‹b0 ≤ α -›]
      have ordlistns.sorted (map (lms-prefix T) ?xs)
        by (simp add: assms(2))
      with ordlistns.list-slice-sorted-nth-mono
                   [of map (lms-prefix T) SA
                       lms-bucket-start α T b0
                       bucket-end α T b0 i' i]
           ‹i < bucket-end α T b0› ‹i' < i›
           ‹i' < length SA›
           ‹lms-bucket-start α T b0 ≤ i'›
      show ?thesis
        by (metis assms(4) length-map map-list-slice
                 not-less not-less-iff-gr-or-eq nth-map)
    qed
    ultimately show ?thesis
      by blast
  qed
  ultimately show ?thesis
    by blast
  qed
  ultimately show list-less-eq-ns x y
    by simp
 qed
 ultimately show ?thesis
   using ordlistns.sorted-append
                [of map (lms-prefix T) ?ys
                    [lms-prefix T j]]
   by blast
qed
ultimately show ?thesis
  by simp
qed
ultimately show ordlistns.sorted (map (lms-prefix T) ?xs)
```

by *blast*
qed

lemma *abs-induce-l-suffix-sorted-step*:
assumes *l-perm-inv* α *T B SA0 SA i*
and *l-suffix-sorted-pre* α *T SA0*
and *l-suffix-sorted-inv* α *T B SA*
and *abs-induce-l-step* (B, SA, i) $(\alpha, T) = (B', SA', i')$
shows *l-suffix-sorted-inv* α *T* B' SA'
proof (*cases* $i <$ *length SA* $\wedge$ *SA* ! $i <$ *length T*)
assume $\neg$ ($i <$ *length SA* $\wedge$ *SA* ! $i <$ *length T*)
then show *?thesis*
using *assms*(*3*,*4*) **by** *force*
next
assume *A*: $i <$ *length SA* $\wedge$ *SA* ! $i <$ *length T*
show *?thesis*
proof (*cases SA* ! *i*)
case *0*
then show *?thesis*
using *assms*(*3*,*4*) *A* **by** *force*
next
case (*Suc j*)
assume *B*: *SA* ! i = *Suc j*
show *?thesis*
proof (*cases suffix-type T j*)
case *S-type*
then show *?thesis*
using *assms*(*3*,*4*) *A B* **by** *force*
next
case *L-type*
then show *?thesis*
using *assms*(*3*,*4*) *A B*
l-suffix-sorted-inv-maintained-step
[*OF assms*(*1*−*3*), *of j* α (*T* ! *j*) *B* ! α (*T* ! *j*)]
by (*clarsimp simp*: *Let-def*)
qed
qed
qed

lemma *abs-induce-l-prefix-sorted-step*:
assumes *l-perm-inv* α *T B SA0 SA i*
and *l-prefix-sorted-pre* α *T SA0*
and *l-prefix-sorted-inv* α *T B SA*
and *abs-induce-l-step* (B, SA, i) $(\alpha, T) = (B', SA', i')$
shows *l-prefix-sorted-inv* α *T* B' SA'
proof (*cases* $i <$ *length SA* $\wedge$ *SA* ! $i <$ *length T*)
assume $\neg$ ($i <$ *length SA* $\wedge$ *SA* ! $i <$ *length T*)
then show *?thesis*
using *assms*(*3*,*4*) **by** *force*

next
  assume A: i < *length SA* ∧ *SA* ! i < *length T*
  show *?thesis*
  proof (*cases SA* ! *i*)
    case *0*
    then show *?thesis*
      using *assms*(*3*,*4*) *A* **by** *force*
  next
    case (*Suc j*)
    assume *B*: *SA* ! *i* = *Suc j*
    show *?thesis*
    proof (*cases suffix-type T j*)
      case *S-type*
      then show *?thesis*
        using *assms*(*3*,*4*) *A B* **by** *force*
    next
      case *L-type*
      then show *?thesis*
        using *assms*(*3*,*4*) *A B*
              l-prefix-sorted-inv-maintained-step[*OF assms*(*1*−*3*), *of j* α (*T* ! *j*) *B* !
α (*T* ! *j*)]
        by (*clarsimp simp*: *Let-def*)
    qed
  qed
qed

lemma *abs-induce-l-base-suffix-sorted-inv-maintained*:
  assumes *l-perm-inv* α *T B SA0 SA 0*
  and   *l-suffix-sorted-pre* α *T SA0*
  and   *l-suffix-sorted-inv* α *T B SA*
  and   *abs-induce-l-base* α *T B SA* = (*B*′, *SA*′, *i*)
  shows *l-suffix-sorted-inv* α *T B*′ *SA*′
proof −
  let *?P* = λ(*B*, *SA*, *i*). *l-perm-inv* α *T B SA0 SA i* ∧ *l-suffix-sorted-inv* α *T B*
SA

  from *assms*(*4*)
  have *repeat* (*length T*) *abs-induce-l-step* (*B*, *SA*, *0*) (α, *T*) = (*B*′, *SA*′, *i*)
    by (*simp add*: *abs-induce-l-base-def*)
  moreover
  have ⋀*a*. *?P a* ⟹ *?P* (*abs-induce-l-step a* (α, *T*))
    using *abs-induce-l-perm-step abs-induce-l-suffix-sorted-step assms*(*2*) **by** *blast*
  ultimately show *?thesis*
    using *repeat-maintain-inv*[*of ?P abs-induce-l-step* (α, *T*) (*B*, *SA*, *0*) *length T*]
    using *assms*(*1*,*2*,*3*) **by** *auto*
qed

lemma *abs-induce-l-base-prefix-sorted-inv-maintained*:
  assumes *l-perm-inv* α *T B SA0 SA 0*

and *l-prefix-sorted-pre* α *T SA0*
and *l-prefix-sorted-inv* α *T B SA*
and *abs-induce-l-base* α *T B SA* $=$ (B', SA', i)
shows *l-prefix-sorted-inv* α *T B' SA'*
proof $-$
let $?P = \lambda(B, SA, i)$. *l-perm-inv* α *T B SA0 SA i* $\wedge$ *l-prefix-sorted-inv* α *T B SA*

from *assms*(*4*)
have *repeat* (*length T*) *abs-induce-l-step* (*B*, *SA*, *0*) (α, *T*) $=$ (B', SA', *i*)
by (*simp add*: *abs-induce-l-base-def*)
moreover
have $\bigwedge a.\ ?P\ a \Longrightarrow\ ?P$ (*abs-induce-l-step a* (α, *T*))
using *abs-induce-l-perm-step abs-induce-l-prefix-sorted-step assms*(*2*) **by** *blast*
ultimately show *?thesis*
using *repeat-maintain-inv*[*of ?P abs-induce-l-step* (α, *T*) (*B*, *SA*, *0*) *length T*]
using *assms*(*1*,*2*,*3*) **by** *auto*
qed

79 L-type Exhaustiveness

The *abs-induce-l* function is exhaustive if it has inserted all the L-types

definition *l-type-exhaustive* :: (*'a* :: {*linorder*, *order-bot*}) *list* $\Rightarrow$ *nat list* $\Rightarrow$ *bool*
where
l-type-exhaustive T SA $=$ ($\forall$ *i* $<$ *length T*. *suffix-type T i* $=$ *L-type* $\longrightarrow$ *i* $\in$ *set SA*)

There two cases when the *abs-induce-l* function is not exhaustive: when there is an L-type that is not in SA but its successor (right neighbour) is in SA, and the other is when there is an L-type that is not in SA and its successor is also not in SA. We will show that both cases will be False.

lemma *not-l-type-exhaustive-imp-ex*:
$\neg$*l-type-exhaustive T SA* $\Longrightarrow$
($\exists$ *i* $<$ *length T*. *suffix-type T i* $=$ *L-type* $\wedge$ *i* $\notin$ *set SA* $\wedge$ *Suc i* $\in$ *set SA*) $\vee$
(($\exists$ *i* $<$ *length T*. *suffix-type T i* $=$ *L-type* $\wedge$ *i* $\notin$ *set SA*) $\wedge$
$\neg$($\exists$ *i*. *i* $<$ *length T* $\wedge$ *suffix-type T i* $=$ *L-type* $\wedge$ *i* $\notin$ *set SA* $\wedge$ *Suc i* $\in$ *set SA*))
using *l-type-exhaustive-def*
by *blast*

lemma *l-type-exhaustive-imp-l-bucket*:
$[\![$*strict-mono* α; *l-type-exhaustive T SA*; *b* $\leq$ α (*Max* (*set T*))$]\!]$ $\Longrightarrow$
{*i*. *i* $\in$ *set SA* $\wedge$ *i* $\in$ *l-bucket* α *T b*} $=$ *l-bucket* α *T b*
by (*intro equalityI subsetI*; *clarsimp simp add*: *bucket-def l-bucket-def l-type-exhaustive-def*)

lemma *l-type-exhaustive-imp-all-l-types*:
l-type-exhaustive T SA $\Longrightarrow$
{*i*. *i* $\in$ *set SA* $\wedge$ *i* $\in$ *l-bucket* α *T* (α (*T* ! *i*))} $=$ {*i*. *i* $<$ *length T* $\wedge$ *suffix-type T i* $=$ *L-type*}
apply (*intro equalityI subsetI*; *clarsimp*)

apply (*simp add*: *bucket-def l-bucket-def*)
by (*simp add*: *l-type-exhaustive-def bucket-def l-bucket-def*)

79.1 Case 1

In the case 1, we have that $\exists\, k{<}length\ T.\ suffix\text{-}type\ T\ k = L\text{-}type \wedge k \notin set\ SA \wedge Suc\ k \in set\ SA$. From this, we know that $\exists\, j{<}length\ SA.\ SA\ !\ j = Suc\ k$

lemma
$Suc\ k \in set\ SA \Longrightarrow \exists\, j < length\ SA.\ SA\ !\ j = Suc\ k$
by (*simp add*: *in-set-conv-nth*)

After executing the *abs-induce-l* function, we know that we have seen

79.2 Case 2

In the case 2, we have that $\exists\, k{<}length\ T.\ suffix\text{-}type\ T\ k = L\text{-}type \wedge k \notin set\ SA \wedge Suc\ k \notin set\ SA$.

lemma *finite-and-Suc-imp-False*:
assumes *finite-A*: *finite A*
and *not-empty*: $A \neq \{\}$
and *Suc-A*: $\forall\, a \in A.\ Suc\ a \in A$
shows *False*
proof −
from *Max-in*[*OF finite-A not-empty*]
have $Max\ A \in A$ **by** *assumption*

with *Suc-A bspec*
have $Suc\ (Max\ A) \in A$ **by** *blast*

with ‹$Max\ A \in A$› *finite-A*
show *?thesis*
using *Max-ge Suc-n-not-le-n* **by** *blast*
qed

lemma *not-exhaustive-neighbour-is-l-type*:
assumes *A*: $A = \{k\ |k.\ suffix\text{-}type\ T\ k = L\text{-}type \wedge k \notin B \wedge Suc\ k \notin B \wedge k < length\ T\}$
and *subset-B*: $\{k\ |k.\ abs\text{-}is\text{-}lms\ T\ k\} \subseteq B$
and $k \in A$
shows $suffix\text{-}type\ T\ (Suc\ k) = L\text{-}type$
proof −
from *A* ‹$k \in A$›
have $Suc\ k \notin B$
by *blast*
with *subset-B*
have $\neg abs\text{-}is\text{-}lms\ T\ (Suc\ k)$
by *blast*

from A ‹$k \in A$›
have *suffix-type* T k = *L-type*
by *blast*

with ‹$\sim$*abs-is-lms* T (*Suc* k)›
show *?thesis*
by (*meson abs-is-lms-def suffix-type-def*)
qed

lemma *no-exhausted-neighbour*:
assumes A: $A = \{k \mid k.$ *suffix-type* T k = *L-type* $\wedge k \notin B \wedge$ *Suc* $k \notin B \wedge k <$ *length* $T\}$
and B: $\{k \mid k.$ *abs-is-lms* T $k\} \subseteq B$
and C: $\neg(\exists k.\ k <$ *length* $T \wedge$ *suffix-type* T k = *L-type* $\wedge k \notin B \wedge$ *Suc* $k \in B)$
and D: *suffix-type* T i = *L-type*
and E: $i \notin B$
and F: $i <$ *length* T
shows $i \in A$
proof −
from C[*simplified*] D E F
have *Suc* $i \notin B$
by *blast*

with A D E F
show *?thesis*
by *blast*
qed

lemma *l-type-less-length-imp-neightbour-less-length*:
⟦*suffix-type* T i = *L-type*; $i <$ *length* T⟧ $\Longrightarrow$ *Suc* $i <$ *length* T
by (*metis SL-types.simps*(*2*) *Suc-lessI suffix-type-last*)

lemma *no-exhausted-neighbour-imp-False*:
assumes A: $A = \{k \mid k.$ *suffix-type* T k = *L-type* $\wedge k \notin B \wedge$ *Suc* $k \notin B \wedge k <$ *length* $T\}$
and B: $\{k \mid k.$ *abs-is-lms* T $k\} \subseteq B$
and C: $\neg(\exists k.\ k <$ *length* $T \wedge$ *suffix-type* T k = *L-type* $\wedge k \notin B \wedge$ *Suc* $k \in B)$
and *nempty*: $A \neq \{\}$
shows *False*
proof −

from A
have *finite* A
by *auto*

have $\forall a \in A.$ *Suc* $a \in A$
proof
fix a

assume $a \in A$
with *not-exhaustive-neighbour-is-l-type*[*OF A B*]
have *suffix-type T* (*Suc a*) = *L-type*
by *blast*

from ‹$a \in A$› *A*
have *Suc a* < *length T*
by (*simp add*: *l-type-less-length-imp-neightbour-less-length*)

from ‹$a \in A$› *A*
have *Suc a* $\notin$ *B*
by *blast*

from *no-exhausted-neighbour*[*OF A B C* ‹*suffix-type T* (*Suc a*) = *L-type*› ‹*Suc a* $\notin$ *B*› ‹*Suc a* < *length T*›]
show *Suc a* $\in$ *A*
by *blast*
qed

with ‹*finite A*› *nempty*
show *?thesis*
using *finite-and-Suc-imp-False* **by** *blast*
qed

79.3 Exhaustiveness Proof

lemma *abs-induce-l-exhaustive*:
assumes *l-seen-inv T SA* (*length SA*)
and *lms-init* α *T SA0*
and *length SA* = *length SA0*
and *length SA* = *length T*
and *strict-mono* α
and *l-unchanged-inv* α *T SA0 SA*
shows *l-type-exhaustive T SA*
proof(*rule ccontr*)

let *?P* = $\exists$ *i*<*length T*. *suffix-type T i* = *L-type* $\wedge$ *i* $\notin$ *set SA* $\wedge$ *Suc i* $\in$ *set SA* **and**
?Q1 = $\exists$ *i*<*length T*. *suffix-type T i* = *L-type* $\wedge$ *i* $\notin$ *set SA* **and**
?Q2 = $\neg$($\exists$ *i*. *i* < *length T* $\wedge$ *suffix-type T i* = *L-type* $\wedge$ *i* $\notin$ *set SA* $\wedge$ *Suc i* $\in$ *set SA*)

assume $\neg$*l-type-exhaustive T SA*
with *not-l-type-exhaustive-imp-ex*
have *?P* $\vee$ (*?Q1* $\wedge$ *?Q2*)
by *blast*
then show *False*
proof
assume *?P*

then obtain i **where**
$i < \mathit{length}\ T$
$\mathit{suffix\text{-}type}\ T\ i = \mathit{L\text{-}type}$
$i \notin \mathit{set}\ SA$
$\mathit{Suc}\ i \in \mathit{set}\ SA$
by *blast*

from ‹$\mathit{Suc}\ i \in \mathit{set}\ SA$›
have $\exists\, k < \mathit{length}\ SA.\ SA\ !\ k = \mathit{Suc}\ i$
by (*simp add*: *in-set-conv-nth*)
then obtain k **where**
$k < \mathit{length}\ SA$
$SA\ !\ k = \mathit{Suc}\ i$
by *blast*

from *l-type-less-length-imp-neighbour-less-length*[*OF* ‹$\mathit{suffix\text{-}type}\ T\ i = \mathit{L\text{-}type}$› ‹$i < \mathit{length}\ T$›]
have $\mathit{Suc}\ i < \mathit{length}\ T$
by *assumption*

from ‹$\mathit{l\text{-}seen\text{-}inv}\ T\ SA\ (\mathit{length}\ SA)$›
have $\forall\, i < \mathit{length}\ SA.\ SA\ !\ i < \mathit{length}\ T \longrightarrow$
$(\forall\, j.\ SA\ !\ i = \mathit{Suc}\ j \wedge \mathit{suffix\text{-}type}\ T\ j = \mathit{L\text{-}type} \longrightarrow$
$(\exists\, k < \mathit{length}\ SA.\ SA\ !\ k = j))$
using *l-seen-inv-def* **by** *blast*
with ‹$k < \mathit{length}\ SA$› ‹$SA\ !\ k = \mathit{Suc}\ i$› ‹$\mathit{Suc}\ i < \mathit{length}\ T$›
have $\forall\, j.\ SA\ !\ k = \mathit{Suc}\ j \wedge \mathit{suffix\text{-}type}\ T\ j = \mathit{L\text{-}type} \longrightarrow (\exists\, k{<}\mathit{length}\ SA.\ SA\ !\ k = j)$
by *auto*
with ‹$SA\ !\ k = \mathit{Suc}\ i$› ‹$\mathit{suffix\text{-}type}\ T\ i = \mathit{L\text{-}type}$›
have $\exists\, k{<}\mathit{length}\ SA.\ SA\ !\ k = i$
by *blast*
with ‹$i \notin \mathit{set}\ SA$›
show *?thesis*
using *nth-mem* **by** *blast*
next
assume $?Q1 \wedge ?Q2$
then have $?Q1\ ?Q2$
by *blast+*
then have $\exists\, A.\ A = \{k\ |k.\ \mathit{suffix\text{-}type}\ T\ k = \mathit{L\text{-}type} \wedge k \notin \mathit{set}\ SA \wedge \mathit{Suc}\ k \notin \mathit{set}\ SA \wedge k < \mathit{length}\ T\}$
by *blast*
then obtain A **where**
A-eq: $A = \{k\ |k.\ \mathit{suffix\text{-}type}\ T\ k = \mathit{L\text{-}type} \wedge k \notin \mathit{set}\ SA \wedge \mathit{Suc}\ k \notin \mathit{set}\ SA \wedge k < \mathit{length}\ T\}$
by *blast*

from ‹$?Q1$› ‹$?Q2$› *A-eq*
have $A \neq \{\}$

by *blast*

from *lms-init-unchanged*[*OF* ‹*l-unchanged-inv* α *T SA0 SA*›
‹*length SA* = *length SA0*›
‹*length SA* = *length T*›
‹*lms-init* α *T SA0*›]
lms-init-imp-all-lms-in-SA[*OF* - ‹*strict-mono* α›]
have *lms-subset-SA* : $\{k \mid k.$ *abs-is-lms T k*$\} \subseteq$ *set SA*
by *blast*

from *no-exhausted-neighbour-imp-False*[*OF A-eq lms-subset-SA* ‹*?Q2*› ‹$A \neq \{\}$›]
show *?thesis*
by *assumption*
qed
qed

80 Correctness and Exhaustiveness

lemma *abs-induce-l-perm-inv-imp-exhaustiveness*:
assumes *abs-induce-l-base* α *T B SA* = (B', SA', i)
and *l-perm-inv* α *T* B' *SA* SA' *i*
shows *l-type-exhaustive T* SA'
proof −
from *abs-induce-l-index*[*of* α *T B SA*] *assms*(*1*)
have *i* = *length T*
by *simp*
hence *i* = *length* SA'
using *assms*(*2*) *l-perm-inv-elims*(*2*,*3*) **by** *fastforce*
hence *P*: *l-perm-inv* α *T* B' *SA* SA' (*length* SA')
using *assms*(*2*) **by** *blast*

have *length* SA' = *length T*
using ‹*i* = *length* SA'› ‹*i* = *length T*› **by** *blast*
with *abs-induce-l-exhaustive*[*OF l-perm-inv-elims*(*11*,*14*,*3*)[*OF P*] - *l-perm-inv-elims*(*12*,*8*)[*OF P*]]
show *?thesis* **.**
qed

lemma *abs-induce-l-perm-inv-B-val*:
assumes *abs-induce-l-base* α *T B SA* = (B', SA', i)
and *l-perm-inv* α *T* B' *SA* SA' *i*
and $b \leq \alpha$ (*Max* (*set T*))
shows B' ! *b* = *l-bucket-end* α *T b*
proof −
from *abs-induce-l-perm-inv-imp-exhaustiveness*[*OF assms*(*1*−*2*)]
have *l-type-exhaustive T* SA'
by *assumption*

have *strict-mono* α
  using *assms*(*2*) *l-perm-inv-elims*(*12*) **by** *blast*

from *l-bucket-ptr-inv-D*[*OF l-perm-inv-elims*(*6*)[*OF assms*(*2*)] *assms*(*3*)]
have B' ! b = *bucket-start* α *T b* + *num-l-types* α *T SA'* *b*
  by *blast*
moreover
from *l-type-exhaustive-imp-l-bucket*[*OF* ‹*strict-mono* α› ‹*l-type-exhaustive T SA'*› *assms*(*3*)]
have *cur-l-types* α *T SA'* *b* = *l-bucket* α *T b*
  unfolding *cur-l-types-def*
  by *blast*
hence *num-l-types* α *T SA'* *b* = *l-bucket-size* α *T b*
  by (*simp add*: *l-bucket-size-def num-l-types-def*)
ultimately
show *?thesis*
  by (*simp add*: *l-bucket-end-def*)
qed

theorem *abs-induce-l-distinct-l-bucket*:
assumes *l-perm-pre* α *T B SA*
and $b \leq \alpha$ (*Max* (*set T*))
shows *distinct* (*list-slice* (*abs-induce-l* α *T B SA*) (*bucket-start* α *T b*) (*l-bucket-end* α *T b*))
proof −
from *abs-induce-l-index*[*of* α *T B SA*]
obtain B' SA' **where**
  A: *abs-induce-l-base* α *T B SA* = (B', SA', *length T*)
  by *blast*
with *abs-induce-l-base-perm-inv-maintained*[*OF l-perm-inv-established*[*OF assms*(*1*)], *of B' SA' length T*]
have *B*: *l-perm-inv* α *T B' SA SA'* (*length T*) **.**
with *abs-induce-l-perm-inv-B-val*[*OF A - assms*(*2*)]
have B' ! b = *l-bucket-end* α *T b* **.**
with *l-distinct-slice*[*OF l-perm-inv-elims*(*4*,*9*)[*OF B*] - *assms*(*2*)] *l-perm-inv-elims*(*2*,*3*)[*OF B*]
have *distinct* (*list-slice SA'* (*bucket-start* α *T b*) (*l-bucket-end* α *T b*))
  by *simp*
then show *?thesis*
  by (*simp add*: *A abs-induce-l-def*)
qed

theorem *abs-induce-l-list-slice-l-bucket*:
assumes *l-perm-pre* α *T B SA*
and $b \leq \alpha$ (*Max* (*set T*))
shows *set* (*list-slice* (*abs-induce-l* α *T B SA*) (*bucket-start* α *T b*) (*l-bucket-end* α *T b*)) = *l-bucket* α *T b*
  (**is** *set ?xs* = *l-bucket* α *T b*)
proof −

from *abs-induce-l-index*[*of* α *T B SA*]
obtain *B′ SA′* **where**
A: *abs-induce-l-base* α *T B SA* = (*B′*, *SA′*, *length T*)
by *blast*
with *abs-induce-l-base-perm-inv-maintained*[*OF l-perm-inv-established*[*OF assms*(*1*)],
of B′ SA′ length T]
have *B*: *l-perm-inv* α *T B′ SA SA′* (*length T*) **.**
with *abs-induce-l-perm-inv-B-val*[*OF A - assms*(*2*)]
have *B′* ! *b* = *l-bucket-end* α *T b* **.**
with *l-locations-list-slice*[*OF l-perm-inv-elims*(*9*)[*OF B*] *assms*(*2*)]
have *set* (*list-slice SA′* (*bucket-start* α *T b*) (*l-bucket-end* α *T b*)) $\subseteq$ *l-bucket* α
T b
by *simp*
hence *set ?xs* $\subseteq$ *l-bucket* α *T b*
by (*simp add*: *A abs-induce-l-def*)
moreover
from *distinct-card*[*OF abs-induce-l-distinct-l-bucket*[*OF assms*]]
have *card* (*set ?xs*) = *length ?xs* **.**
hence *card* (*set ?xs*) = *l-bucket-end* α *T b* − *bucket-start* α *T b*
by (*metis B bucket-end-le-length abs-induce-l-length l-bucket-end-le-bucket-end*
l-perm-inv-elims(*2*)
le-trans length-list-slice min-def)
hence *card* (*set ?xs*) = *card* (*l-bucket* α *T b*)
by (*simp add*: *l-bucket-end-def l-bucket-size-def*)
ultimately show *?thesis*
using *card-subset-eq*[*OF finite-l-bucket*]
by *blast*
qed

lemma *abs-induce-l-unchanged*:
assumes *l-perm-pre* α *T B SA*
and $b \leq \alpha$ (*Max* (*set T*))
and *s-bucket-start* α *T b* $\leq i$
and $i <$ *bucket-end* α *T b*
shows (*abs-induce-l* α *T B SA*) ! *i* = *SA* ! *i*
proof −
from *abs-induce-l-index*[*of* α *T B SA*]
obtain *B′ SA′* **where**
A: *abs-induce-l-base* α *T B SA* = (*B′*, *SA′*, *length T*)
by *blast*
with *abs-induce-l-base-perm-inv-maintained*[*OF l-perm-inv-established*[*OF assms*(*1*)],
of B′ SA′ length T]
have *B*: *l-perm-inv* α *T B′ SA SA′* (*length T*) **.**

have $i <$ *length SA*
by (*metis B assms*(*4*) *bucket-end-le-length l-perm-inv-elims*(*2*) *le-less-trans*
not-less-eq)
moreover
have *l-bucket-end* α *T b* $\leq i$

by (*metis assms*(*3*) *s-bucket-start-eq-l-bucket-end*)
ultimately have SA' ! i = SA ! i
using *l-unchanged-inv-D*[*OF l-perm-inv-elims*(*8*,*3*)[*OF B*] *assms*(*2*) - - *assms*(*4*)]
by *auto*
then show *?thesis*
by (*simp add*: *A abs-induce-l-def*)
qed

— Used in SAIS algorithm as part of inducing the suffix ordering based on LMS
theorem *abs-induce-l-suffix-sorted-l-bucket*:
assumes *l-perm-pre* α *T B SA*
and *l-suffix-sorted-pre* α *T SA*
and $b \leq \alpha$ (*Max* (*set T*))
shows *ordlistns.sorted* (*map* (*suffix T*)
(*list-slice* (*abs-induce-l* α *T B SA*) (*bucket-start* α *T b*) (*l-bucket-end* α *T b*)))
proof −

from *l-perm-inv-established*[*OF assms*(*1*)]
have *A*: *l-perm-inv* α *T B SA SA 0* **.**

from *l-suffix-sorted-inv-established*[*OF l-perm-pre-elims*(*4*)[*OF assms*(*1*)], *of SA*]
have *D*: *l-suffix-sorted-inv* α *T B SA* **.**

from *abs-induce-l-index*[*of* α *T B SA*]
obtain B' SA' **where**
B: *abs-induce-l-base* α *T B SA* = (B', SA', *length T*)
by *blast*
with *abs-induce-l-base-perm-inv-maintained*[*OF A*, *of* B' SA' *length T*]
have *C*: *l-perm-inv* α *T* B' *SA* SA' (*length T*) **.**
with *abs-induce-l-perm-inv-B-val*[*OF B* - *assms*(*3*)]
have B' ! b = *l-bucket-end* α *T b* **.**
moreover
from *abs-induce-l-base-suffix-sorted-inv-maintained*[*OF A assms*(*2*) *D B*]
have *l-suffix-sorted-inv* α *T* B' SA' **.**
ultimately show *?thesis*
using *l-suffix-sorted-invD*[*of* α *T* B' SA', *OF* - *assms*(*3*)]
by (*simp add*: *B abs-induce-l-def*)
qed

— Used in SAIS algorithm as part of inducing the prefix ordering based on LMS
theorem *abs-induce-l-prefix-sorted-l-bucket*:
assumes *l-perm-pre* α *T B SA*
and *l-prefix-sorted-pre* α *T SA*
and $b \leq \alpha$ (*Max* (*set T*))
shows *ordlistns.sorted* (*map* (*lms-prefix T*)
(*list-slice* (*abs-induce-l* α *T B SA*) (*bucket-start* α *T b*) (*l-bucket-end* α *T b*)))
proof −

from *l-perm-inv-established*[*OF assms*(*1*)]
have *A*: *l-perm-inv* α *T B SA SA 0* **.**

from *l-prefix-sorted-inv-established*[*OF l-perm-pre-elims*(*4*)[*OF assms*(*1*)], *of SA*]
have *D*: *l-prefix-sorted-inv* α *T B SA* **.**

from *abs-induce-l-index*[*of* α *T B SA*]
obtain B' SA' **where**
B: *abs-induce-l-base* α *T B SA* = (B', SA', *length T*)
by *blast*
with *abs-induce-l-base-perm-inv-maintained*[*OF A*, *of* B' SA' *length T*]
have *C*: *l-perm-inv* α *T* B' *SA* SA' (*length T*) **.**
with *abs-induce-l-perm-inv-B-val*[*OF B - assms*(*3*)]
have B' ! *b* = *l-bucket-end* α *T b* **.**
moreover
from *abs-induce-l-base-prefix-sorted-inv-maintained*[*OF A assms*(*2*) *D B*]
have *l-prefix-sorted-inv* α *T* B' SA' **.**
ultimately show *?thesis*
using *l-prefix-sorted-invD*[*of* α *T* B' SA', *OF - assms*(*3*)]
by (*simp add*: *B abs-induce-l-def*)
qed

end
theory *Abs-Induce-S-Verification*
imports *../abs−def/Abs-SAIS*
begin

81 Abstract Induce S Simple Properties

lemma *abs-induce-s-step-ex*:
$\exists$ B' SA' i'. *abs-induce-s-step a b* = (B', SA', i')
by (*meson prod-cases3*)

lemma *abs-induce-s-step-B-length*:
abs-induce-s-step (*B*, *SA*, *i*) (α, *T*) = (B', SA', i') $\Longrightarrow$ *length* B' = *length B*
by (*clarsimp split*: *prod.splits if-splits nat.splits SL-types.splits simp*: *Let-def*)

lemma *abs-induce-s-step-SA-length*:
abs-induce-s-step (*B*, *SA*, *i*) (α, *T*) = (B', SA', i') $\Longrightarrow$ *length* SA' = *length SA*
by (*clarsimp split*: *prod.splits if-splits nat.splits SL-types.splits simp*: *Let-def*)

lemma *abs-induce-s-step-Suc*:
abs-induce-s-step (*B*, *SA*, *Suc i*) (α, *T*) = (B', SA', i') $\Longrightarrow$ i' = *i*
by (*clarsimp split*: *prod.splits if-splits nat.splits SL-types.splits simp*: *Let-def*)

lemma *abs-induce-s-step-0*:
abs-induce-s-step (*B*, *SA*, *0*) (α, *T*) = (*B*, *SA*, *0*)
by (*clarsimp split*: *prod.splits if-splits nat.splits SL-types.splits simp*: *Let-def*)

corollary *abs-induce-s-step-0-alt*:
  assumes *abs-induce-s-step* (B, SA, i) $(\alpha, T) = (B', SA', i')$
  and $i = 0$
shows $B = B' \wedge SA = SA' \wedge i' = 0$
  using *assms*(*1*) *assms*(*2*) **by** *auto*

lemma *repeat-abs-induce-s-step-index*:
  $\exists B'\ SA'$. *repeat n abs-induce-s-step* (B, SA, m) $(\alpha, T) = (B', SA', m - n) \wedge$
    length SA' = *length* SA $\wedge$ *length* B' = *length* B
proof(*induct n arbitrary*: *m*)
  case *0*
  then show *?case* **by** (*clarsimp simp*: *repeat-0*)
next
  case (*Suc n*)
  note *IH* = *this*

  from *IH*[*of m*]
  obtain B' SA' **where**
    repeat n abs-induce-s-step (B, SA, m) $(\alpha, T) = (B', SA', m - n)$
    length SA' = *length* SA
    length B' = *length* B
    by *blast*
  with *repeat-step*[*of n abs-induce-s-step* (B, SA, m) (α, T)]
  have *repeat* (*Suc n*) *abs-induce-s-step* (B, SA, m) (α, T) = *abs-induce-s-step* $(B', SA', m - n)$ (α, T)
    by *simp*
  moreover
  have $\exists B''\ SA''$. *abs-induce-s-step* $(B', SA', m - n)$ $(\alpha, T) = (B'', SA'', m -$ *Suc n*$) \wedge$
      length SA'' = *length* SA' $\wedge$ *length* B'' = *length* B'
  proof (*cases* $n < m$)
    assume $n < m$
    hence $\exists k$. $m - n$ = *Suc k*
      by *presburger*
    then obtain *k* **where**
      $m - n$ = *Suc k*
      by *blast*

    from *abs-induce-s-step-ex*[*of* $(B', SA', m - n)$ (α, T)]
    obtain B'' SA'' i' **where**
      P: *abs-induce-s-step* $(B', SA', m - n)$ $(\alpha, T) = (B'', SA'', i')$
      by *blast*
    with ‹$m - n$ = *Suc k*› *abs-induce-s-step-Suc*[*of* B' SA' *k* α *T* B'' SA'' i']
    have $i' = m -$ *Suc n*
      by *auto*
    with ‹*abs-induce-s-step* $(B', SA', m - n)$ $(\alpha, T) = (B'', SA'', i')$›
        abs-induce-s-step-SA-length[*OF P*]
        abs-induce-s-step-B-length[*OF P*]

show *?thesis*
by *simp*
next
assume $\neg n < m$
hence $m \leq n$
by *simp*
hence $m - n = 0$
by *simp*
with *abs-induce-s-step-0*[*of* B' SA' α T]
show *?thesis*
by *simp*
qed
ultimately show *?case*
by (*simp add*: ‹*length* SA' = *length* *SA*› ‹*length* B' = *length* *B*›)
qed

lemma *abs-induce-s-base-index*:
$\exists B'\ SA'$. *abs-induce-s-base* α *T B SA* = (B', SA', *0*)
unfolding *abs-induce-s-base-def*
by (*metis cancel-comm-monoid-add-class.diff-cancel repeat-abs-induce-s-step-index*)

lemma *abs-induce-s-length*:
length (*abs-induce-s* α *T B SA*) = *length SA*
unfolding *abs-induce-s-def abs-induce-s-base-def*
apply (*rule repeat-maintain-inv*; *simp add*: *Let-def*)
apply (*clarsimp split*: *prod.splits simp del*: *abs-induce-s-step.simps*)
apply (*drule abs-induce-s-step-SA-length*; *simp*)
done

82 Preconditions

definition *l-types-init*
where
l-types-init α *T SA* $\equiv$
($\forall b \leq \alpha$ (*Max* (*set T*)).
set (*list-slice SA* (*bucket-start* α *T b*) (*l-bucket-end* α *T b*)) = *l-bucket* α *T b* $\wedge$
distinct (*list-slice SA* (*bucket-start* α *T b*) (*l-bucket-end* α *T b*))
)

lemma *l-types-initD*:
⟦*l-types-init* α *T SA*; $b \leq \alpha$ (*Max* (*set T*))⟧ $\Longrightarrow$
set (*list-slice SA* (*bucket-start* α *T b*) (*l-bucket-end* α *T b*)) = *l-bucket* α *T b*
⟦*l-types-init* α *T SA*; $b \leq \alpha$ (*Max* (*set T*))⟧ $\Longrightarrow$
distinct (*list-slice SA* (*bucket-start* α *T b*) (*l-bucket-end* α *T b*))
using *l-types-init-def* **by** *blast+*

lemma *l-types-init-nth*:
assumes *length SA* = *length T*
and *l-types-init* α *T SA*

and $b \le \alpha\ (Max\ (set\ T))$
and $bucket\text{-}start\ \alpha\ T\ b \le i$
and $i < l\text{-}bucket\text{-}end\ \alpha\ T\ b$
shows $SA\ !\ i \in l\text{-}bucket\ \alpha\ T\ b$
proof −
have $l\text{-}bucket\text{-}end\ \alpha\ T\ b \le length\ SA$
by (*metis assms*(*1*) *bucket-end-le-length dual-order.order-iff-strict l-bucket-end-le-bucket-end less-le-trans*)
with *l-types-initD*(*1*)[*OF assms*(*2*,*3*)] *list-slice-nth-mem*[*OF assms*(*4*,*5*)]
show *?thesis*
by *blast*
qed

definition *s-type-init*
where
$s\text{-}type\text{-}init\ T\ SA \equiv (\exists n.\ length\ T = Suc\ n \wedge SA\ !\ 0 = n)$

definition *s-perm-pre*
where
$s\text{-}perm\text{-}pre\ \alpha\ T\ B\ SA\ n \equiv$
$s\text{-}bucket\text{-}init\ \alpha\ T\ B \wedge$
$s\text{-}type\text{-}init\ T\ SA \wedge$
$strict\text{-}mono\ \alpha \wedge$
$\alpha\ (Max\ (set\ T)) < length\ B \wedge$
$length\ SA = length\ T \wedge$
$l\text{-}types\text{-}init\ \alpha\ T\ SA \wedge$
$valid\text{-}list\ T \wedge$
$\alpha\ bot = 0 \wedge$
$Suc\ 0 < length\ T \wedge$
$length\ T \le n$

definition *s-sorted-pre*
where
$s\text{-}sorted\text{-}pre\ \alpha\ T\ SA \equiv$
$(\forall b \le \alpha\ (Max\ (set\ T)).$
ordlistns.sorted (*map* (*suffix T*) (*list-slice SA* (*bucket-start* α *T b*) (*l-bucket-end* α *T b*)))
)

lemma *s-sorted-preD*:
$[\![s\text{-}sorted\text{-}pre\ \alpha\ T\ SA;\ b \le \alpha\ (Max\ (set\ T))]\!] \Longrightarrow$
ordlistns.sorted (*map* (*suffix T*) (*list-slice SA* (*bucket-start* α *T b*) (*l-bucket-end* α *T b*)))
using *s-sorted-pre-def* **by** *blast*

definition *s-prefix-sorted-pre*
where
$s\text{-}prefix\text{-}sorted\text{-}pre\ \alpha\ T\ SA \equiv$
$(\forall b \le \alpha\ (Max\ (set\ T)).$

ordlistns.sorted (*map* (*lms-slice* T) (*list-slice* SA (*bucket-start* α T b) (*l-bucket-end* α T b)))
)

lemma *s-prefix-sorted-preD*:
$[\![$*s-prefix-sorted-pre* α T SA; $b \leq \alpha$ (*Max* (*set* T))$]\!] \Longrightarrow$
ordlistns.sorted (*map* (*lms-slice* T) (*list-slice* SA (*bucket-start* α T b) (*l-bucket-end* α T b)))
using *s-prefix-sorted-pre-def* **by** *blast*

83 Invariants

83.1 Definitions

83.1.1 Distinctness

definition *s-distinct-inv*
where
s-distinct-inv α T B SA $\equiv$
($\forall\, b \leq \alpha$ (*Max* (*set* T)). *distinct* (*list-slice* SA (B ! b) (*bucket-end* α T b)))

lemma *s-distinct-invD*:
$[\![$*s-distinct-inv* α T B SA; $b \leq \alpha$ (*Max* (*set* T))$]\!] \Longrightarrow$
distinct (*list-slice* SA (B ! b) (*bucket-end* α T b))
using *s-distinct-inv-def* **by** *blast*

83.1.2 S Bucket Ptr

definition *s-bucket-ptr-inv* ::
($'a$ :: {*linorder*, *order-bot*} $\Rightarrow$ *nat*) $\Rightarrow$ $'a$ *list* $\Rightarrow$ *nat list* $\Rightarrow$ *bool*
where
s-bucket-ptr-inv α T B $\equiv$
($\forall\, b \leq \alpha$ (*Max* (*set* T)).
s-bucket-start α T $b \leq B$! b $\wedge$
B ! $b \leq$ *bucket-end* α T b $\wedge$
($b = 0 \longrightarrow B$! $b = 0$))

lemma *s-bucket-ptr-lower-bound*:
assumes *s-bucket-ptr-inv* α T B
and $b \leq \alpha$ (*Max* (*set* T))
shows *s-bucket-start* α T $b \leq B$! b
using *assms*(*1*) *assms*(*2*) *s-bucket-ptr-inv-def* **by** *blast*

lemma *s-bucket-ptr-upper-bound*:
assumes *s-bucket-ptr-inv* α T B
and $b \leq \alpha$ (*Max* (*set* T))
shows B ! $b \leq$ *bucket-end* α T b
by (*metis assms s-bucket-ptr-inv-def*)

lemma *s-bucket-ptr-0*:

assumes *s-bucket-ptr-inv* α T B
and $b = 0$
shows B ! $b = 0$
using *assms s-bucket-ptr-inv-def* **by** *auto*

83.1.3 Locations

definition *s-locations-inv* ::
$('a :: \{linorder, order\text{-}bot\} \Rightarrow nat) \Rightarrow 'a\ list \Rightarrow nat\ list \Rightarrow nat\ list \Rightarrow bool$
where
s-locations-inv α T B $SA \equiv$
$(\forall\, b \le \alpha\ (Max\ (set\ T)).$
$(\forall\, i.\ B\ !\ b \le i \wedge i < bucket\text{-}end\ \alpha\ T\ b \longrightarrow SA\ !\ i \in s\text{-}bucket\ \alpha\ T\ b))$

lemma *s-locations-invD*:
$[\![s\text{-}locations\text{-}inv\ \alpha\ T\ B\ SA;\ b \le \alpha\ (Max\ (set\ T));\ B\ !\ b \le i;\ i < bucket\text{-}end\ \alpha\ T\ b]\!] \Longrightarrow$
$SA\ !\ i \in s\text{-}bucket\ \alpha\ T\ b$
using *s-locations-inv-def* **by** *blast*

lemma *s-locations-inv-in-list-slice*:
assumes *s-locations-inv* α T B SA
and $b \le \alpha\ (Max\ (set\ T))$
and $x \in set\ (list\text{-}slice\ SA\ (B\ !\ b)\ (bucket\text{-}end\ \alpha\ T\ b))$
shows $x \in s\text{-}bucket\ \alpha\ T\ b$
proof −
from *in-set-conv-nth*[*THEN iffD1*, *OF assms*(*3*)]
obtain i **where**
$i < length\ (list\text{-}slice\ SA\ (B\ !\ b)\ (bucket\text{-}end\ \alpha\ T\ b))$
$list\text{-}slice\ SA\ (B\ !\ b)\ (bucket\text{-}end\ \alpha\ T\ b)\ !\ i = x$
by *blast*
with *nth-list-slice*
have $SA\ !\ (B\ !\ b + i) = x$
by *fastforce*
moreover
have $B\ !\ b + i < bucket\text{-}end\ \alpha\ T\ b$
using ‹$i < length\ (list\text{-}slice\ SA\ (B\ !\ b)\ (bucket\text{-}end\ \alpha\ T\ b))$› **by** *auto*
ultimately
show *?thesis*
using *assms*(*1*,*2*) *le-add1 s-locations-invD* **by** *blast*
qed

lemma *s-locations-inv-subset-s-bucket*:
assumes *s-locations-inv* α T B SA
and $b \le \alpha\ (Max\ (set\ T))$
shows $set\ (list\text{-}slice\ SA\ (B\ !\ b)\ (bucket\text{-}end\ \alpha\ T\ b)) \subseteq s\text{-}bucket\ \alpha\ T\ b$
using *assms s-locations-inv-in-list-slice* **by** *blast*

83.1.4 Unchanged

definition *s-unchanged-inv* ::
$('a :: \{linorder, order\text{-}bot\} \Rightarrow nat) \Rightarrow 'a\ list \Rightarrow nat\ list \Rightarrow nat\ list \Rightarrow nat\ list \Rightarrow bool$
where
$s\text{-}unchanged\text{-}inv\ \alpha\ T\ B\ SA\ SA' \equiv$
$(\forall b \le \alpha\ (Max\ (set\ T)).\ (\forall i.\ bucket\text{-}start\ \alpha\ T\ b \le i \wedge i < B\ !\ b \longrightarrow\ SA'\ !\ i = SA\ !\ i))$

lemma *s-unchanged-invD*:
$[\![s\text{-}unchanged\text{-}inv\ \alpha\ T\ B\ SA\ SA';\ b \le \alpha\ (Max\ (set\ T));\ bucket\text{-}start\ \alpha\ T\ b \le i;\ i < B\ !\ b]\!] \Longrightarrow$
$SA'\ !\ i = SA\ !\ i$
using *s-unchanged-inv-def* **by** *blast*

83.1.5 Seen

definition *s-seen-inv* ::
$('a :: \{linorder, order\text{-}bot\} \Rightarrow nat) \Rightarrow 'a\ list \Rightarrow nat\ list \Rightarrow nat\ list \Rightarrow nat \Rightarrow bool$
where
$s\text{-}seen\text{-}inv\ \alpha\ T\ B\ SA\ n \equiv$
$\forall i < length\ SA.\ n \le i \longrightarrow$
$(suffix\text{-}type\ T\ (SA\ !\ i) = S\text{-}type \longrightarrow in\text{-}s\text{-}current\text{-}bucket\ \alpha\ T\ B\ (\alpha\ (T\ !\ (SA\ !\ i)))\ i) \wedge$
$(suffix\text{-}type\ T\ (SA\ !\ i) = L\text{-}type \longrightarrow in\text{-}l\text{-}bucket\ \alpha\ T\ (\alpha\ (T\ !\ (SA\ !\ i)))\ i) \wedge$
$SA\ !\ i < length\ T$

lemma *s-seen-invD*:
$[\![s\text{-}seen\text{-}inv\ \alpha\ T\ B\ SA\ n;\ i < length\ SA;\ n \le i]\!] \Longrightarrow SA\ !\ i < length\ T$
$[\![s\text{-}seen\text{-}inv\ \alpha\ T\ B\ SA\ n;\ i < length\ SA;\ n \le i;\ suffix\text{-}type\ T\ (SA\ !\ i) = L\text{-}type]\!] \Longrightarrow$
$in\text{-}l\text{-}bucket\ \alpha\ T\ (\alpha\ (T\ !\ (SA\ !\ i)))\ i$
$[\![s\text{-}seen\text{-}inv\ \alpha\ T\ B\ SA\ n;\ i < length\ SA;\ n \le i;\ suffix\text{-}type\ T\ (SA\ !\ i) = S\text{-}type]\!] \Longrightarrow$
$in\text{-}s\text{-}current\text{-}bucket\ \alpha\ T\ B\ (\alpha\ (T\ !\ (SA\ !\ i)))\ i$
unfolding *s-seen-inv-def* **by** *blast+*

83.1.6 Predecessor

definition *s-pred-inv* ::
$(('a :: \{linorder, order\text{-}bot\}) \Rightarrow nat) \Rightarrow 'a\ list \Rightarrow nat\ list \Rightarrow nat\ list \Rightarrow nat \Rightarrow bool$
where
$s\text{-}pred\text{-}inv\ \alpha\ T\ B\ SA\ n =$
$(\forall b\ i.\ in\text{-}s\text{-}current\text{-}bucket\ \alpha\ T\ B\ b\ i \wedge b \ne 0 \longrightarrow$
$(\exists j < length\ SA.\ SA\ !\ j = Suc\ (SA\ !\ i) \wedge i < j \wedge n < j)$
$)$

lemma *s-pred-invD*:

$[\![s\text{-}pred\text{-}inv\ \alpha\ T\ B\ SA\ k;\ in\text{-}s\text{-}current\text{-}bucket\ \alpha\ T\ B\ b\ i;\ b \neq 0]\!] \Longrightarrow$
$\exists j < length\ SA.\ SA\ !\ j = Suc\ (SA\ !\ i) \wedge i < j \wedge k < j$
using *s-pred-inv-def* **by** *blast*

83.1.7 Successor

definition *s-suc-inv* ::
$('a :: \{linorder,\ order\text{-}bot\} \Rightarrow nat) \Rightarrow 'a\ list \Rightarrow nat\ list \Rightarrow nat\ list \Rightarrow nat \Rightarrow bool$
where
$s\text{-}suc\text{-}inv\ \alpha\ T\ B\ SA\ n \equiv$
$\forall i < length\ SA.\ n < i \longrightarrow$
$(\forall j.\ SA\ !\ i = Suc\ j \wedge suffix\text{-}type\ T\ j = S\text{-}type \longrightarrow$
$(\exists k.\ in\text{-}s\text{-}current\text{-}bucket\ \alpha\ T\ B\ (\alpha\ (T\ !\ j))\ k \wedge SA\ !\ k = j \wedge k < i))$

lemma *s-suc-invD*:
$[\![s\text{-}suc\text{-}inv\ \alpha\ T\ B\ SA\ n;\ i < length\ SA;\ n < i;\ SA\ !\ i = Suc\ j;\ suffix\text{-}type\ T\ j = S\text{-}type]\!] \Longrightarrow$
$\exists k.\ in\text{-}s\text{-}current\text{-}bucket\ \alpha\ T\ B\ (\alpha\ (T\ !\ j))\ k \wedge SA\ !\ k = j \wedge k < i$
using *s-suc-inv-def* **by** *blast*

83.1.8 Combined Permutation Invariant

definition *s-perm-inv* ::
$('a :: \{linorder,\ order\text{-}bot\} \Rightarrow nat) \Rightarrow 'a\ list \Rightarrow nat\ list \Rightarrow nat\ list \Rightarrow nat\ list \Rightarrow nat \Rightarrow bool$
where
$s\text{-}perm\text{-}inv\ \alpha\ T\ B\ SA\ SA'\ n \equiv$
$s\text{-}distinct\text{-}inv\ \alpha\ T\ B\ SA' \wedge$
$s\text{-}bucket\text{-}ptr\text{-}inv\ \alpha\ T\ B \wedge$
$s\text{-}locations\text{-}inv\ \alpha\ T\ B\ SA' \wedge$
$s\text{-}unchanged\text{-}inv\ \alpha\ T\ B\ SA\ SA' \wedge$
$s\text{-}seen\text{-}inv\ \alpha\ T\ B\ SA'\ n \wedge$
$s\text{-}pred\text{-}inv\ \alpha\ T\ B\ SA'\ n \wedge$
$s\text{-}suc\text{-}inv\ \alpha\ T\ B\ SA'\ n \wedge$
$strict\text{-}mono\ \alpha \wedge$
$\alpha\ (Max\ (set\ T)) < length\ B \wedge$
$length\ SA = length\ T \wedge$
$length\ SA' = length\ T \wedge$
$l\text{-}types\text{-}init\ \alpha\ T\ SA \wedge$
$valid\text{-}list\ T \wedge$
$\alpha\ bot = 0 \wedge$
$Suc\ 0 < length\ T$

lemma *s-perm-inv-elims*:
$s\text{-}perm\text{-}inv\ \alpha\ T\ B\ SA\ SA'\ n \Longrightarrow s\text{-}distinct\text{-}inv\ \alpha\ T\ B\ SA'$
$s\text{-}perm\text{-}inv\ \alpha\ T\ B\ SA\ SA'\ n \Longrightarrow s\text{-}bucket\text{-}ptr\text{-}inv\ \alpha\ T\ B$
$s\text{-}perm\text{-}inv\ \alpha\ T\ B\ SA\ SA'\ n \Longrightarrow s\text{-}locations\text{-}inv\ \alpha\ T\ B\ SA'$
$s\text{-}perm\text{-}inv\ \alpha\ T\ B\ SA\ SA'\ n \Longrightarrow s\text{-}unchanged\text{-}inv\ \alpha\ T\ B\ SA\ SA'$
$s\text{-}perm\text{-}inv\ \alpha\ T\ B\ SA\ SA'\ n \Longrightarrow s\text{-}seen\text{-}inv\ \alpha\ T\ B\ SA'\ n$
$s\text{-}perm\text{-}inv\ \alpha\ T\ B\ SA\ SA'\ n \Longrightarrow s\text{-}pred\text{-}inv\ \alpha\ T\ B\ SA'\ n$

$s\text{-}perm\text{-}inv\ \alpha\ T\ B\ SA\ SA'\ n \Longrightarrow s\text{-}suc\text{-}inv\ \alpha\ T\ B\ SA'\ n$
$s\text{-}perm\text{-}inv\ \alpha\ T\ B\ SA\ SA'\ n \Longrightarrow strict\text{-}mono\ \alpha$
$s\text{-}perm\text{-}inv\ \alpha\ T\ B\ SA\ SA'\ n \Longrightarrow \alpha\ (Max\ (set\ T)) < length\ B$
$s\text{-}perm\text{-}inv\ \alpha\ T\ B\ SA\ SA'\ n \Longrightarrow length\ SA = length\ T$
$s\text{-}perm\text{-}inv\ \alpha\ T\ B\ SA\ SA'\ n \Longrightarrow length\ SA' = length\ T$
$s\text{-}perm\text{-}inv\ \alpha\ T\ B\ SA\ SA'\ n \Longrightarrow l\text{-}types\text{-}init\ \alpha\ T\ SA$
$s\text{-}perm\text{-}inv\ \alpha\ T\ B\ SA\ SA'\ n \Longrightarrow valid\text{-}list\ T$
$s\text{-}perm\text{-}inv\ \alpha\ T\ B\ SA\ SA'\ n \Longrightarrow \alpha\ bot = 0$
$s\text{-}perm\text{-}inv\ \alpha\ T\ B\ SA\ SA'\ n \Longrightarrow Suc\ 0 < length\ T$
by (*simp-all add*: *s-perm-inv-def*)

fun *s-perm-inv-alt* ::
$('a :: \{linorder,\ order\text{-}bot\} \Rightarrow nat) \Rightarrow 'a\ list \Rightarrow nat\ list \Rightarrow nat\ list \times nat\ list \times nat \Rightarrow bool$
where
$s\text{-}perm\text{-}inv\text{-}alt\ \alpha\ T\ SA\ (B,\ SA',\ n) = s\text{-}perm\text{-}inv\ \alpha\ T\ B\ SA\ SA'\ n$

83.1.9 Sorted

definition *s-sorted-inv*
where
$s\text{-}sorted\text{-}inv\ \alpha\ T\ B\ SA \equiv$
$(\forall\, b \le \alpha\ (Max\ (set\ T)).$
$ordlistns.sorted\ (map\ (suffix\ T)\ (list\text{-}slice\ SA\ (B\ !\ b)\ (bucket\text{-}end\ \alpha\ T\ b)))$
$)$

lemma *s-sorted-invD*:
$[\![s\text{-}sorted\text{-}inv\ \alpha\ T\ B\ SA;\ b \le \alpha\ (Max\ (set\ T))]\!] \Longrightarrow$
$ordlistns.sorted\ (map\ (suffix\ T)\ (list\text{-}slice\ SA\ (B\ !\ b)\ (bucket\text{-}end\ \alpha\ T\ b)))$
using *s-sorted-inv-def* **by** *blast*

fun *s-sorted-inv-alt* ::
$('a :: \{linorder,\ order\text{-}bot\} \Rightarrow nat) \Rightarrow 'a\ list \Rightarrow nat\ list \Rightarrow nat\ list \times nat\ list \times nat \Rightarrow bool$
where
$s\text{-}sorted\text{-}inv\text{-}alt\ \alpha\ T\ SA\ (B,\ SA',\ n) =$
$(s\text{-}perm\text{-}inv\ \alpha\ T\ B\ SA\ SA'\ n \wedge s\text{-}sorted\text{-}pre\ \alpha\ T\ SA \wedge s\text{-}sorted\text{-}inv\ \alpha\ T\ B\ SA')$

definition *s-prefix-sorted-inv*
where
$s\text{-}prefix\text{-}sorted\text{-}inv\ \alpha\ T\ B\ SA \equiv$
$(\forall\, b \le \alpha\ (Max\ (set\ T)).$
$ordlistns.sorted\ (map\ (lms\text{-}slice\ T)\ (list\text{-}slice\ SA\ (B\ !\ b)\ (bucket\text{-}end\ \alpha\ T\ b)))$
$)$

lemma *s-prefix-sorted-invD*:
$[\![s\text{-}prefix\text{-}sorted\text{-}inv\ \alpha\ T\ B\ SA;\ b \le \alpha\ (Max\ (set\ T))]\!] \Longrightarrow$
$ordlistns.sorted\ (map\ (lms\text{-}slice\ T)\ (list\text{-}slice\ SA\ (B\ !\ b)\ (bucket\text{-}end\ \alpha\ T\ b)))$
using *s-prefix-sorted-inv-def* **by** *blast*

fun *s-prefix-sorted-inv-alt* ::
  $('a :: \{linorder, order\text{-}bot\} \Rightarrow nat) \Rightarrow 'a\ list \Rightarrow nat\ list \Rightarrow nat\ list \times nat\ list \times nat \Rightarrow bool$
  where
s-prefix-sorted-inv-alt α *T SA* (*B*, *SA'*, *n*) =
  (*s-perm-inv* α *T B SA SA' n* $\wedge$ *s-prefix-sorted-pre* α *T SA* $\wedge$ *s-prefix-sorted-inv* α *T B SA'*)

83.2 Helpers

lemma *s-current-bucket-pairwise-distinct*:
  assumes *s-distinct-inv* α *T B SA*
  and *s-locations-inv* α *T B SA*
  and $b \leq \alpha\ (Max\ (set\ T))$
  and $b' \leq \alpha\ (Max\ (set\ T))$
  and $b \neq b'$
shows *distinct* (*list-slice SA* (*B* ! *b*) (*bucket-end* α *T b*) @ *list-slice SA* (*B* ! *b'*) (*bucket-end* α *T b'*))
proof (*intro distinct-append*[*THEN iffD2*] *conjI disjointI'*)
  from *s-distinct-invD*[*OF assms*(*1*,*3*)]
  show *distinct* (*list-slice SA* (*B* ! *b*) (*bucket-end* α *T b*)) **.**
next
  from *s-distinct-invD*[*OF assms*(*1*,*4*)]
  show *distinct* (*list-slice SA* (*B* ! *b'*) (*bucket-end* α *T b'*)) **.**
next
  fix *x y*
  assume *A*: $x \in$ *set* (*list-slice SA* (*B* ! *b*) (*bucket-end* α *T b*))
    $y \in$ *set* (*list-slice SA* (*B* ! *b'*) (*bucket-end* α *T b'*))

  from *s-locations-inv-in-list-slice*[*OF assms*(*2*,*3*) *A*(*1*)]
  have $x \in$ *s-bucket* α *T b* **.**
  moreover
  from *s-locations-inv-in-list-slice*[*OF assms*(*2*,*4*) *A*(*2*)]
  have $y \in$ *s-bucket* α *T b'* **.**
  ultimately
  show $x \neq y$
    using *assms*(*5*)
    by (*metis* (*mono-tags*, *lifting*) *bucket-def s-bucket-def mem-Collect-eq*)
qed

lemma *s-unchanged-list-slice*:
  assumes *s-unchanged-inv* α *T B SA0 SA*
  and *length SA0* = *length T*
  and *length SA* = *length T*
  and $b \leq \alpha\ (Max\ (set\ T))$
  and *bucket-start* α *T b* $\leq i$
  and $j \leq B\ !\ b$
shows *list-slice SA i j* = *list-slice SA0 i j*

proof (*intro list-eq-iff-nth-eq*[*THEN iffD2*] *conjI allI impI*)
  show *length* (*list-slice SA i j*) = *length* (*list-slice SA0 i j*)
    by (*simp add*: *assms*)
next
  fix *k*
  assume *k* < *length* (*list-slice SA i j*)
  hence *k* < *length* (*list-slice SA0 i j*)
    by (*simp add*: *assms*)

  from *nth-list-slice*[*OF* ‹*k* < *length* (*list-slice SA i j*)›]
  have *list-slice SA i j* ! *k* = *SA* ! (*i* + *k*) **.**
  moreover
  from *nth-list-slice*[*OF* ‹*k* < *length* (*list-slice SA0 i j*)›]
  have *list-slice SA0 i j* ! *k* = *SA0* ! (*i* + *k*) **.**
  moreover
  {
    have *bucket-start* α *T b* $\leq$ *i* + *k*
      by (*simp add*: *assms*(*5*) *trans-le-add1*)
    moreover
    have *i* + *k* < *j*
      using ‹*k* < *length* (*list-slice SA i j*)› **by** *auto*
    hence *i* + *k* < *B* ! *b*
      using *assms*(*6*) *order.strict-trans2* **by** *blast*
    ultimately
    have *SA* ! (*i* + *k*) = *SA0* ! (*i* + *k*)
      using *s-unchanged-invD*[*OF assms*(*1*,*4*)]
      by *blast*
  }
  ultimately show *list-slice SA i j* ! *k* = *list-slice SA0 i j* ! *k*
    by *simp*
qed

lemma *l-types-init-maintained*:
  assumes *s-bucket-ptr-inv* α *T B*
  and *s-unchanged-inv* α *T B SA0 SA*
  and *length SA0* = *length T*
  and *length SA* = *length T*
  and *l-types-init* α *T SA0*
shows *l-types-init* α *T SA*
  unfolding *l-types-init-def*
proof(*intro allI impI*)
  fix *b*
  let *?xs* = *list-slice SA* (*bucket-start* α *T b*) (*l-bucket-end* α *T b*)
  let *?ys* = *list-slice SA0* (*bucket-start* α *T b*) (*l-bucket-end* α *T b*)
  assume *b* $\leq$ α (*Max* (*set T*))
  with *s-bucket-ptr-lower-bound*[*OF assms*(*1*), *of b*]
  have *l-bucket-end* α *T b* $\leq$ *B* ! *b*
    by (*simp add*: *s-bucket-start-eq-l-bucket-end*)
  with *s-unchanged-list-slice*[*OF assms*(*2*−*4*) ‹*b* $\leq$ α (*Max* (*set T*))›,

of bucket-start α *T b l-bucket-end* α *T b*]
have *?xs* = *?ys*
by *blast*
with *assms*(*5*)[*simplified l-types-init-def*, *THEN spec*, *THEN mp*, *OF* ‹$b \leq \alpha$ (*Max* (*set T*))›]
show *set ?xs* = *l-bucket* α *T b* ∧ *distinct ?xs*
by *simp*
qed

lemma *s-sorted-pre-maintained*:
assumes *s-bucket-ptr-inv* α *T B*
and *s-unchanged-inv* α *T B SA0 SA*
and *length SA0* = *length T*
and *length SA* = *length T*
and *s-sorted-pre* α *T SA0*
shows *s-sorted-pre* α *T SA*
unfolding *s-sorted-pre-def*
proof(*intro allI impI*)
fix *b*
let *?xs* = *list-slice SA* (*bucket-start* α *T b*) (*l-bucket-end* α *T b*)
let *?ys* = *list-slice SA0* (*bucket-start* α *T b*) (*l-bucket-end* α *T b*)
assume $b \leq \alpha$ (*Max* (*set T*))
with *s-bucket-ptr-lower-bound*[*OF assms*(*1*), *of b*]
have *l-bucket-end* α *T b* ≤ *B* ! *b*
by (*simp add*: *s-bucket-start-eq-l-bucket-end*)
with *s-unchanged-list-slice*[*OF assms*(*2*–*4*) ‹$b \leq \alpha$ (*Max* (*set T*))›,
of bucket-start α *T b l-bucket-end* α *T b*]
have *?xs* = *?ys*
by *blast*
then show *ordlistns.sorted* (*map* (*suffix T*) *?xs*)
using ‹$b \leq \alpha$ (*Max* (*set T*))› *assms*(*5*) *s-sorted-pre-def* **by** *auto*
qed

lemma *s-prefix-sorted-pre-maintained*:
assumes *s-bucket-ptr-inv* α *T B*
and *s-unchanged-inv* α *T B SA0 SA*
and *length SA0* = *length T*
and *length SA* = *length T*
and *s-prefix-sorted-pre* α *T SA0*
shows *s-prefix-sorted-pre* α *T SA*
unfolding *s-prefix-sorted-pre-def*
proof(*intro allI impI*)
fix *b*
let *?xs* = *list-slice SA* (*bucket-start* α *T b*) (*l-bucket-end* α *T b*)
let *?ys* = *list-slice SA0* (*bucket-start* α *T b*) (*l-bucket-end* α *T b*)
assume $b \leq \alpha$ (*Max* (*set T*))
with *s-bucket-ptr-lower-bound*[*OF assms*(*1*), *of b*]
have *l-bucket-end* α *T b* ≤ *B* ! *b*
by (*simp add*: *s-bucket-start-eq-l-bucket-end*)

with *s-unchanged-list-slice*[*OF assms*(*2*−*4*) ‹$b \leq \alpha$ (*Max* (*set T*))›,
  of bucket-start α *T b l-bucket-end* α *T b*]
have *?xs* = *?ys*
  by *blast*
then show *ordlistns.sorted* (*map* (*lms-slice T*) *?xs*)
  using ‹$b \leq \alpha$ (*Max* (*set T*))› *assms*(*5*) *s-prefix-sorted-pre-def* **by** *auto*
qed

lemma *s-next-item-not-seen*:
assumes *s-distinct-inv* α *T B SA*
and *s-bucket-ptr-inv* α *T B*
and *s-locations-inv* α *T B SA*
and *s-unchanged-inv* α *T B SA0 SA*
and *s-seen-inv* α *T B SA i*
and *s-pred-inv* α *T B SA i*
and *strict-mono* α
and *length SA0* = *length T*
and *length SA* = *length T*
and *l-types-init* α *T SA0*
and *valid-list T*
and α *bot* = *0*
and *i* = *Suc n*
and *Suc n* < *length SA*
and *SA* ! *Suc n* = *Suc j*
and *suffix-type T j* = *S-type*
and *b* = α (*T* ! *j*)
shows $j \notin$ *set* (*list-slice SA* (*B* ! *b*) (*bucket-end* α *T b*))
proof
let *?xs* = *list-slice SA* (*B* ! *b*) (*bucket-end* α *T b*)
let *?b* = α (*T* ! (*Suc j*))
assume $j \in$ *set ?xs*

from *s-seen-invD*(*1*)[*OF assms*(*5*,*14*)] *assms*(*13*,*15*)
have *Suc j* < *length T*
  by *simp*
hence *?b* $\leq \alpha$ (*Max* (*set T*))
  using *assms*(*7*)
  by (*simp add*: *strict-mono-leD*)

have *bucket-end* α *T ?b* $\leq$ *length SA*
  by (*simp add*: *assms*(*9*) *bucket-end-le-length*)
hence *l-bucket-end* α *T ?b* $\leq$ *length SA*
  by (*metis dual-order.trans l-bucket-end-le-bucket-end*)

have *j* < *length T*
  by (*simp add*: *assms*(*16*) *suffix-type-s-bound*)

from *valid-list-length-ex*[*OF assms*(*11*)]
obtain *n'* **where**

length T = *Suc n'*
  by *blast*
with ‹*Suc j* < *length T*›
have *j* < *n'*
  by *linarith*
hence *T* ! *j* ≠ *bot*
  by (*metis* ‹*length T* = *Suc n'*› *assms*(*11*) *diff-Suc-1 valid-list-def*)
hence *b* ≠ *0*
  using *assms*(*7*,*12*,*17*) *strict-mono-eq* **by** *fastforce*

with *in-set-conv-nth*[*THEN iffD1*, *OF* ‹*j* ∈ *set ?xs*›]
obtain *a* **where**
  a < *length ?xs*
  ?xs ! *a* = *j*
  by *blast*
hence *SA* ! (*B* ! *b* + *a*) = *j*
  using *nth-list-slice* **by** *fastforce*

from *assms*(*9*)
have *bucket-end α T b* ≤ *length SA*
  by (*simp add*: *bucket-end-le-length*)
with ‹*a* < *length ?xs*›
have *B* ! *b* + *a* < *bucket-end α T b*
  by *auto*
with *assms*(*7*,*17*) ‹*j* < *length T*›
have *in-s-current-bucket α T B b* (*B* ! *b* + *a*)
  unfolding *in-s-current-bucket-def*
  by (*simp add*: *strict-mono-less-eq*)
with *s-pred-invD*[*OF assms*(*6*) - ‹*b* ≠ *0*›, *of B* ! *b* + *a*] ‹*SA* ! (*B* ! *b* + *a*) = *j*›
obtain *m* **where**
  m < *length SA*
  SA ! *m* = *Suc j*
  B ! *b* + *a* < *m*
  i < *m*
  by *blast*
hence *i* ≠ *m*
  by *blast*

have *suffix-type T* (*Suc j*) = *L-type* ⟹ *False*
proof −
  assume *suffix-type T* (*Suc j*) = *L-type*
  with *s-seen-invD*(*2*)[*OF assms*(*5*) ‹*m* < *length SA*›] ‹*i* < *m*› ‹*SA* ! *m* = *Suc j*›
  have *in-l-bucket α T ?b m*
    by *simp*
  hence *bucket-start α T ?b* ≤ *m m* < *l-bucket-end α T ?b*
    using *in-l-bucket-def* **by** *blast+*
  moreover
  from ‹*suffix-type T* (*Suc j*) = *L-type*› *assms*(*13*,*15*)
      s-seen-invD(*2*)[*OF assms*(*5*) ‹*Suc n* < *length SA*›]

have *in-l-bucket* α *T ?b i*
by *simp*
hence *bucket-start* α *T ?b* $\leq$ *i i* $<$ *l-bucket-end* α *T ?b*
using *in-l-bucket-def* **by** *blast+*
ultimately
show *False*
using *list-slice-nth-eq-iff-index-eq*[
OF l-types-initD(2)[*OF l-types-init-maintained*[*OF assms(2,4,8−10)*]
‹*?b* $\leq$ α *-*›]
‹*l-bucket-end* α *T ?b* $\leq$ *length SA*›,
of i m]
‹*i* $\neq$ *m*› *assms(13,15)* ‹*SA ! m = Suc j*›
by *simp*
qed
moreover
have *suffix-type T (Suc j) = S-type* $\Longrightarrow$ *False*
proof −
assume *suffix-type T (Suc j) = S-type*
with *s-seen-invD(3)*[*OF assms(5)* ‹*m < length SA*›] ‹*i < m*› ‹*SA ! m = Suc j*›
have *in-s-current-bucket* α *T B ?b m*
by *simp*
hence *B ! ?b* $\leq$ *m m* $<$ *bucket-end* α *T ?b*
using *in-s-current-bucket-def* **by** *blast+*
moreover
from ‹*suffix-type T (Suc j) = S-type*› *assms(13,15)*
s-seen-invD(3)[*OF assms(5)* ‹*Suc n < length SA*›]
have *in-s-current-bucket* α *T B ?b i*
by *simp*
hence *B ! ?b* $\leq$ *i i* $<$ *bucket-end* α *T ?b*
using *in-s-current-bucket-def* **by** *blast+*
ultimately
show *False*
using *list-slice-nth-eq-iff-index-eq*[
OF s-distinct-invD[*OF assms(1)* ‹*?b* $\leq$ α *-*›] ‹*bucket-end* α *T ?b* $\leq$
length SA›,
of i m]
‹*i* $\neq$ *m*› *assms(13,15)* ‹*SA ! m = Suc j*›
by *simp*
qed
ultimately
show *False*
using *SL-types.exhaust* **by** *blast*
qed

lemma *s-bucket-ptr-strict-lower-bound*:
assumes *s-distinct-inv* α *T B SA*
and *s-bucket-ptr-inv* α *T B*
and *s-locations-inv* α *T B SA*
and *s-unchanged-inv* α *T B SA0 SA*

and $s\text{-}seen\text{-}inv\ \alpha\ T\ B\ SA\ i$
and $s\text{-}pred\text{-}inv\ \alpha\ T\ B\ SA\ i$
and $strict\text{-}mono\ \alpha$
and $length\ SA0 = length\ T$
and $length\ SA = length\ T$
and $l\text{-}types\text{-}init\ \alpha\ T\ SA0$
and $valid\text{-}list\ T$
and $\alpha\ bot = 0$
and $i = Suc\ n$
and $Suc\ n < length\ SA$
and $SA\ !\ Suc\ n = Suc\ j$
and $suffix\text{-}type\ T\ j = S\text{-}type$
and $b = \alpha\ (T\ !\ j)$
shows $s\text{-}bucket\text{-}start\ \alpha\ T\ b < B\ !\ b$
proof $-$
have $j < length\ T$
by $(simp\ add{:}\ assms(16)\ suffix\text{-}type\text{-}s\text{-}bound)$
hence $b \le \alpha\ (Max\ (set\ T))$
by $(simp\ add{:}\ assms(7{,}17)\ strict\text{-}mono\text{-}leD)$

let $?xs = list\text{-}slice\ SA\ (B\ !\ b)\ (bucket\text{-}end\ \alpha\ T\ b)$

have $bucket\text{-}end\ \alpha\ T\ b \le length\ SA$
by $(simp\ add{:}\ assms(9)\ bucket\text{-}end\text{-}le\text{-}length)$
hence $length\ ?xs = bucket\text{-}end\ \alpha\ T\ b - B\ !\ b$
by $auto$

from $s\text{-}next\text{-}item\text{-}not\text{-}seen[OF\ assms(1{-}17)]$
have $j \notin set\ ?xs$ **.**
moreover
have $j \in s\text{-}bucket\ \alpha\ T\ b$
by $(simp\ add{:}\ assms(16{,}17)\ bucket\text{-}def\ s\text{-}bucket\text{-}def\ suffix\text{-}type\text{-}s\text{-}bound)$
ultimately
have $set\ ?xs \subset s\text{-}bucket\ \alpha\ T\ b$
using $s\text{-}locations\text{-}inv\text{-}subset\text{-}s\text{-}bucket[OF\ assms(3)\ \langle b \le \text{-}\rangle]$
by $blast$
hence $card\ (set\ ?xs) < s\text{-}bucket\text{-}size\ \alpha\ T\ b$
using $psubset\text{-}card\text{-}mono[OF\ finite\text{-}s\text{-}bucket,\ simplified\ s\text{-}bucket\text{-}size\text{-}def[symmetric]]$
by $blast$
moreover
from $s\text{-}distinct\text{-}invD[OF\ assms(1)\ \langle b \le \text{-}\rangle]$
have $distinct\ ?xs$ **.**
ultimately
have $length\ ?xs < s\text{-}bucket\text{-}size\ \alpha\ T\ b$
by $(simp\ add{:}\ distinct\text{-}card)$
with $\langle length\ ?xs = \text{-}\rangle$
have $bucket\text{-}end\ \alpha\ T\ b - B\ !\ b < s\text{-}bucket\text{-}size\ \alpha\ T\ b$
by $simp$
hence $bucket\text{-}end\ \alpha\ T\ b < B\ !\ b + s\text{-}bucket\text{-}size\ \alpha\ T\ b$

by *linarith*
hence
$s\text{-}bucket\text{-}start\ \alpha\ T\ b + bucket\text{-}end\ \alpha\ T\ b < B\ !\ b + s\text{-}bucket\text{-}start\ \alpha\ T\ b + s\text{-}bucket\text{-}size\ \alpha\ T\ b$
by *simp*
then show $s\text{-}bucket\text{-}start\ \alpha\ T\ b < B\ !\ b$
by (*simp add*: *bucket-end-eq-s-start-pl-size*)
qed

lemma *outside-another-bucket*:
assumes $b \neq b'$
and $bucket\text{-}start\ \alpha\ T\ b \leq i$
and $i < bucket\text{-}end\ \alpha\ T\ b$
shows $\neg(bucket\text{-}start\ \alpha\ T\ b' \leq i \wedge i < bucket\text{-}end\ \alpha\ T\ b')$
by (*meson assms dual-order.antisym less-bucket-end-le-start not-le order.strict-trans1*)

lemma *s-B-val*:
assumes *s-distinct-inv* α *T B SA*
and *s-bucket-ptr-inv* α *T B*
and *s-locations-inv* α *T B SA*
and *s-unchanged-inv* α *T B SA0 SA*
and *s-seen-inv* α *T B SA i*
and *s-pred-inv* α *T B SA i*
and *strict-mono* α
and *length SA0* = *length T*
and *length SA* = *length T*
and *l-types-init* α *T SA0*
and *valid-list T*
and *length T* > *Suc 0*
and $b \leq \alpha\ (Max\ (set\ T))$
and $i < B\ !\ b$
shows $B\ !\ b = s\text{-}bucket\text{-}start\ \alpha\ T\ b$
proof (*rule ccontr*; *drule neq-iff*[*THEN iffD1*]; *elim disjE*)
assume $B\ !\ b < s\text{-}bucket\text{-}start\ \alpha\ T\ b$
with *s-bucket-ptr-lower-bound*[*OF assms*(*2*,*13*)]
show *False*
by *linarith*
next
let $?k = B\ !\ b - Suc\ 0$
assume $s\text{-}bucket\text{-}start\ \alpha\ T\ b < B\ !\ b$
hence $s\text{-}bucket\text{-}start\ \alpha\ T\ b \leq ?k$
by *linarith*
hence $bucket\text{-}start\ \alpha\ T\ b \leq ?k$
using *bucket-start-le-s-bucket-start le-trans* **by** *blast*

have $?k < B\ !\ b$
using ‹$s\text{-}bucket\text{-}start\ \alpha\ T\ b < B\ !\ b$› **by** *auto*

from *assms*(*14*)

```
have i ≤ ?k
  by linarith

from s-bucket-ptr-upper-bound[OF assms(2,13)]
have B ! b ≤ bucket-end α T b .
hence ?k < bucket-end α T b
  using ‹s-bucket-start α T b < B ! b› by linarith

have bucket-end α T b ≤ length SA
  by (simp add: assms(9) bucket-end-le-length)
moreover
have bucket-end α T b = length SA ⟹ False
proof −
  assume bucket-end α T b = length SA
  with ‹s-bucket-start α T b < B ! b› ‹B ! b ≤ bucket-end α T b› assms(7,9,13)
  have b = α (Max (set T))
    using bucket-end-eq-length by fastforce
  hence s-bucket α T b = {}
    by (simp add: assms(7,11,12) s-bucket-Max)
  hence s-bucket-size α T b = 0
    by (simp add: s-bucket-size-def)
  hence s-bucket-start α T b = bucket-end α T b
    by (simp add: bucket-end-eq-s-start-pl-size)
  with ‹B ! b ≤ bucket-end α T b› ‹s-bucket-start α T b < B ! b›
  show False
    by simp
qed
moreover
have bucket-end α T b < length SA ⟹ False
proof −
  assume bucket-end α T b < length SA
  hence B ! b < length SA
    using ‹B ! b ≤ bucket-end α T b›
    by linarith
  hence ?k < length SA
    using less-imp-diff-less by blast
  with s-seen-invD[OF assms(5) - ‹i ≤ ?k›]
  have SA ! ?k < length T
    by blast

  let ?b = α (T ! (SA ! ?k))

  from ‹SA ! ?k < length T› assms(7)
  have ?b ≤ α (Max (set T))
    using Max-greD strict-mono-leD by blast
  with s-bucket-ptr-lower-bound[OF assms(2)]
  have s-bucket-start α T ?b ≤ B ! ?b
    by blast
  hence bucket-start α T ?b ≤ B ! ?b
```

using *bucket-start-le-s-bucket-start le-trans* **by** *blast*

have *suffix-type* T $(SA ! ?k) = L\text{-}type \Longrightarrow False$
proof −
assume *suffix-type* T $(SA ! ?k) = L\text{-}type$
with *s-seen-invD(2)[OF assms(5)* ‹$?k <$ *length SA*› ‹$i \leq ?k$›]
have *in-l-bucket* α T $?b$ $?k$
by *blast*
hence *bucket-start* α T $?b \leq ?k$ $?k <$ *l-bucket-end* α T $?b$
using *in-l-bucket-def* **by** *blast+*
hence $?k <$ *bucket-end* α T $?b$
using *l-bucket-end-le-bucket-end less-le-trans* **by** *blast*

from ‹$?k <$ *l-bucket-end* - - -› ‹*s-bucket-start* - - - $\leq ?k$›
have $b = ?b \Longrightarrow False$
by (*simp add*: *s-bucket-start-eq-l-bucket-end*)
moreover
from *outside-another-bucket*[*OF* - ‹*bucket-start* - - $b \leq ?k$› ‹$?k <$ *bucket-end*
- - b›]
‹*bucket-start* - - $?b \leq ?k$› ‹$?k <$ *bucket-end* - - $?b$›
have $b \neq ?b \Longrightarrow False$
by *blast*
ultimately show *False*
by *blast*
qed
moreover
have *suffix-type* T $(SA ! ?k) = S\text{-}type \Longrightarrow False$
proof −
assume *suffix-type* T $(SA ! ?k) = S\text{-}type$
with *s-seen-invD(3)[OF assms(5)* ‹$?k <$ *length SA*› ‹$i \leq ?k$›]
have *in-s-current-bucket* α T B $?b$ $?k$
by *blast*
hence $B ! ?b \leq ?k$ $?k <$ *bucket-end* α T $?b$
using *in-s-current-bucket-def* **by** *blast+*
hence *bucket-start* α T $?b \leq ?k$
using ‹*bucket-start* α T $?b \leq B ! ?b$› **by** *linarith*

from ‹$B ! ?b \leq ?k$› ‹$?k < B ! b$›
have $b = ?b \Longrightarrow False$
by *simp*
moreover
from *outside-another-bucket*[*OF* - ‹*bucket-start* - - $b \leq ?k$› ‹$?k <$ *bucket-end*
- - b›]
‹*bucket-start* α T $?b \leq ?k$› ‹$?k <$ *bucket-end* α T $?b$›
have $b \neq ?b \Longrightarrow False$
by *blast*
ultimately show *False*
by *blast*
qed

ultimately show *False*
using *SL-types.exhaust* **by** *blast*
qed
ultimately show *False*
by *linarith*
qed

lemma *s-bucket-eq-list-slice*:
assumes *s-distinct-inv* α *T B SA*
and *s-locations-inv* α *T B SA*
and *length SA* = *length T*
and $b \leq \alpha$ (*Max* (*set T*))
and *B* ! *b* = *s-bucket-start* α *T b*
shows *set* (*list-slice SA* (*s-bucket-start* α *T b*) (*bucket-end* α *T b*)) = *s-bucket* α *T b*
(**is** *set ?xs* = *s-bucket* α *T b*)
using *card-subset-eq*[
OF finite-s-bucket s-locations-inv-subset-s-bucket[*OF assms*(*2*,*4*), *simplified assms*(*5*)]]
distinct-card[*OF s-distinct-invD*[*OF assms*(*1*,*4*), *simplified assms*(*5*)]]
bucket-end-eq-s-start-pl-size[*of* α *T b*]
s-bucket-size-def[*of* α *T b*]
by (*metis assms*(*3*) *bucket-end-le-length diff-add-inverse length-list-slice min-def*)

lemma *bucket-eq-list-slice*:
assumes *s-distinct-inv* α *T B SA*
and *s-bucket-ptr-inv* α *T B*
and *s-locations-inv* α *T B SA*
and *s-unchanged-inv* α *T B SA0 SA*
and *length SA0* = *length T*
and *length SA* = *length T*
and *l-types-init* α *T SA0*
and $b \leq \alpha$ (*Max* (*set T*))
and *B* ! *b* = *s-bucket-start* α *T b*
shows *set* (*list-slice SA* (*bucket-start* α *T b*) (*bucket-end* α *T b*)) = *bucket* α *T b*
(**is** *set ?xs* = *bucket* α *T b*)
proof −
let *?ys* = *list-slice SA* (*bucket-start* α *T b*) (*l-bucket-end* α *T b*)
and *?zs* = *list-slice SA* (*s-bucket-start* α *T b*) (*bucket-end* α *T b*)

have *?xs* = *?ys* @ *?zs*
by (*metis list-slice-append bucket-start-le-s-bucket-start l-bucket-end-le-bucket-end s-bucket-start-eq-l-bucket-end*)
hence *set ?xs* = *set ?ys* ∪ *set ?zs*
by *simp*
with *l-types-initD*(*1*)[*OF l-types-init-maintained*[*OF assms*(*2*,*4*−*7*)] *assms*(*8*)]
s-bucket-eq-list-slice[*OF assms*(*1*,*3*,*6*,*8*,*9*)]
have *set ?xs* = *l-bucket* α *T b* ∪ *s-bucket* α *T b*
by *simp*

  then show *?thesis*
    using *l-un-s-bucket* **by** *blast*
qed

lemma *s-index-lower-bound*:
  assumes *s-bucket-ptr-inv* α *T B*
  and *s-seen-inv* α *T B SA n*
  and $i <$ *length SA*
  and $n \leq i$
shows *bucket-start* α *T* (α (*T* ! (*SA* ! *i*))) $\leq i$
    (**is** *bucket-start* α *T ?b* $\leq i$)
proof −

  have *?b* $\leq \alpha$ (*Max* (*set T*))
    by (*meson SL-types.exhaust assms*(*2*−) *in-l-bucket-def in-s-current-bucketD*(*1*) *s-seen-invD*(*2*,*3*))

  have *suffix-type T* (*SA* ! *i*) = *S-type* $\vee$ *suffix-type T* (*SA* ! *i*) = *L-type*
    using *SL-types.exhaust* **by** *blast*
  moreover
  have *suffix-type T* (*SA* ! *i*) = *S-type* $\Longrightarrow$ *?thesis*
  proof −
    assume *suffix-type T* (*SA* ! *i*) = *S-type*
    with *s-seen-invD*(*3*)[*OF assms*(*2*−)] *s-bucket-ptr-lower-bound*[*OF assms*(*1*)]
    show *?thesis*
    by (*meson* ‹*?b* $\leq \alpha$ (*Max* (*set T*))› *bucket-start-le-s-bucket-start dual-order.trans*
        in-s-current-bucketD(*2*))
  qed
  moreover
  have *suffix-type T* (*SA* ! *i*) = *L-type* $\Longrightarrow$ *?thesis*
  proof −
    assume *suffix-type T* (*SA* ! *i*) = *L-type*
    with *s-seen-invD*(*2*)[*OF assms*(*2*−)]
    show *?thesis*
      using *in-l-bucket-def* **by** *blast*
  qed
  ultimately show *?thesis*
    by *blast*
qed

lemma *s-index-upper-bound*:
  assumes *s-bucket-ptr-inv* α *T B*
  and *s-seen-inv* α *T B SA n*
  and $i <$ *length SA*
  and $n \leq i$
shows $i <$ *bucket-end* α *T* (α (*T* ! (*SA* ! *i*)))
    (**is** $i <$ *bucket-end* α *T ?b*)
proof −

have *?b* ≤ *α* (*Max* (*set T*))
by (*meson SL-types.exhaust assms*(*2*−) *in-l-bucket-def in-s-current-bucketD*(*1*) *s-seen-invD*(*2*,*3*))

have *suffix-type T* (*SA* ! *i*) = *S-type* ∨ *suffix-type T* (*SA* ! *i*) = *L-type*
using *SL-types.exhaust* **by** *blast*
moreover
have *suffix-type T* (*SA* ! *i*) = *S-type* ⟹ *?thesis*
proof −
assume *suffix-type T* (*SA* ! *i*) = *S-type*
with *s-seen-invD*(*3*)[*OF assms*(*2*−)] *s-bucket-ptr-upper-bound*[*OF assms*(*1*)]
show *?thesis*
by (*simp add*: *in-s-current-bucket-def*)
qed
moreover
have *suffix-type T* (*SA* ! *i*) = *L-type* ⟹ *?thesis*
proof −
assume *suffix-type T* (*SA* ! *i*) = *L-type*
with *s-seen-invD*(*2*)[*OF assms*(*2*−)]
show *?thesis*
using *in-l-bucket-def l-bucket-end-le-bucket-end less-le-trans* **by** *blast*
qed
ultimately show *?thesis*
by *blast*
qed

83.3 Establishment and Maintenance Steps

83.3.1 Distinctness

lemma *s-distinct-inv-established*:
assumes *s-bucket-init α T B*
and *valid-list T*
and *strict-mono α*
and *α bot* = *0*
shows *s-distinct-inv α T B SA*
unfolding *s-distinct-inv-def*
proof (*intro allI impI*)
fix *b*
let *?goal* = *distinct* (*list-slice SA* (*B* ! *b*) (*bucket-end α T b*))
assume *b* ≤ *α* (*Max* (*set T*))

have *b* > *0* ⟹ *?goal*
proof −
assume *b* > *0*
with *s-bucket-initD*(*1*)[*OF assms*(*1*) ‹*b* ≤ -›]
have *B* ! *b* = *bucket-end α T b*
by *blast*
then show *?goal*
using *list-slice-n-n*

    by (*metis distinct.simps*(*1*))
  qed
  moreover
  have $b = 0 \Longrightarrow$ *?goal*
  proof −
    assume $b = 0$
    with *s-bucket-initD*(*2*)[*OF assms*(*1*) ‹$b \leq$ -›]
    have B ! $b = 0$ **.**
    moreover
    from ‹$b = 0$› *assms*(*2*−*4*)
    have *bucket-end* α T b = *Suc 0*
      by (*simp add*: *valid-list-bucket-end-0*)
    ultimately show *?thesis*
      by (*simp add*: *distinct-conv-nth*)
  qed
  ultimately show *?goal*
    by *blast*
qed

lemma *s-distinct-inv-maintained-step*:
  assumes *s-distinct-inv* α T B SA
  and *s-bucket-ptr-inv* α T B
  and *s-locations-inv* α T B SA
  and *s-unchanged-inv* α T B $SA0$ SA
  and *s-seen-inv* α T B SA i
  and *s-pred-inv* α T B SA i
  and *strict-mono* α
  and α (*Max* (*set* T)) < *length* B
  and *length* $SA0$ = *length* T
  and *length* SA = *length* T
  and *l-types-init* α T $SA0$
  and *valid-list* T
  and α *bot* = *0*
  and i = *Suc* n
  and *Suc* n < *length* SA
  and SA ! *Suc* n = *Suc* j
  and *suffix-type* T j = *S-type*
  and $b = \alpha$ (T ! j)
  and $k = B$! b − *Suc 0*
shows *s-distinct-inv* α T ($B[b := k]$) ($SA[k := j]$)
  unfolding *s-distinct-inv-def*
proof (*intro allI impI*)
  fix b'

  let *?goal* = *distinct* (*list-slice* ($SA[k := j]$) ($B[b := k]$! b') (*bucket-end* α T b'))

  assume $b' \leq \alpha$ (*Max* (*set* T))

  from *s-next-item-not-seen*[*OF assms*(*1*−*7*,*9*−*18*)]

have $j \notin$ *set* (*list-slice SA* (B ! b) (*bucket-end* α T b))**.**

from *s-bucket-ptr-strict-lower-bound*[*OF assms*(*1−7,9−18*)]
have *s-bucket-start* α T $b < B$! b **.**
hence *s-bucket-start* α T $b \le k$ $k < B$! b
  using *assms*(*19*) **by** *linarith+*
hence *bucket-start* α T $b \le k$
  using *bucket-start-le-s-bucket-start le-trans* **by** *blast*

from *assms*(*17*)
have $j <$ *length* T
  by (*simp add*: *suffix-type-s-bound*)
hence $b \le \alpha$ (*Max* (*set* T))
  by (*simp add*: *assms*(*7,18*) *strict-mono-less-eq*)
with *s-bucket-ptr-upper-bound*[*OF assms*(*2*)] ‹$k < B$! b›
have $k <$ *bucket-end* α T b
  using *less-le-trans* **by** *auto*
hence $k <$ *length SA*
  using *assms*(*10*) *bucket-end-le-length less-le-trans* **by** *fastforce*

have $b = b' \Longrightarrow$ *?goal*
proof −
  assume $b = b'$
  hence $B[b := k]$! $b' = k$
    using ‹$b \le \alpha$ (*Max* (*set* T))› *assms*(*8*) **by** *auto*
  from *s-distinct-invD*[*OF assms*(*1*) ‹$b \le$ -›]
  have *distinct* (*list-slice SA* (B ! b) (*bucket-end* α T b)) **.**
  moreover
  from ‹$B[b := k]$! $b' = k$› ‹$b = b'$› ‹$k < B$! b› ‹$k <$ *bucket-end* α T b› ‹$k <$ *length SA*› *assms*(*19*)
  have *list-slice* ($SA[k := j]$) ($B[b := k]$! b') (*bucket-end* α T b')
      $= j$ # *list-slice SA* (B ! b) (*bucket-end* α T b)
   by (*metis Suc-pred diff-is-0-eq′ dual-order.order-iff-strict gr0I length-list-update list-slice-Suc list-slice-update-unchanged-1 nth-list-update-eq*)
  ultimately show *?thesis*
    using ‹$j \notin$ *set* (*list-slice SA* (B ! b) (*bucket-end* α T b))›
    by *simp*
qed
moreover
have $b \ne b' \Longrightarrow$ *?goal*
proof −
  assume $b \ne b'$
  hence $B[b := k]$! $b' = B$! b'
    by *simp*

  from *outside-another-bucket*[*OF* ‹$b \ne b'$› ‹*bucket-start* - - - $\le k$› ‹$k <$ *bucket-end* - - -›]
  have $k <$ *bucket-start* α T $b' \lor$ *bucket-end* α T $b' \le k$
    using *leI* **by** *auto*

moreover
have $k < \textit{bucket-start}\ \alpha\ T\ b' \Longrightarrow$
$\textit{list-slice}\ (SA[k := j])\ (B[b := k]\ !\ b')\ (\textit{bucket-end}\ \alpha\ T\ b')$
$= \textit{list-slice}\ SA\ (B\ !\ b')\ (\textit{bucket-end}\ \alpha\ T\ b')$
proof –
assume $k < \textit{bucket-start}\ \alpha\ T\ b'$
hence $k < B\ !\ b'$
by (*meson* ‹$b' \leq \alpha\ (\textit{Max}\ (\textit{set}\ T))$› *assms*(*2*) *bucket-start-le-s-bucket-start*
less-le-trans
s-bucket-ptr-inv-def)
with *list-slice-update-unchanged-1* ‹$B[b := k]\ !\ b' = B\ !\ b'$›
show *?thesis*
by *simp*
qed
moreover
from ‹$B[b := k]\ !\ b' = B\ !\ b'$›
have $\textit{bucket-end}\ \alpha\ T\ b' \leq k \Longrightarrow$
$\textit{list-slice}\ (SA[k := j])\ (B[b := k]\ !\ b')\ (\textit{bucket-end}\ \alpha\ T\ b')$
$= \textit{list-slice}\ SA\ (B\ !\ b')\ (\textit{bucket-end}\ \alpha\ T\ b')$
by (*simp add*: *list-slice-update-unchanged-2*)
moreover
from *s-distinct-invD*[*OF assms*(*1*) ‹$b' \leq$ -›]
have $\textit{distinct}\ (\textit{list-slice}\ SA\ (B\ !\ b')\ (\textit{bucket-end}\ \alpha\ T\ b'))$ **.**
ultimately show *?goal*
by *auto*
qed
ultimately
show *?goal*
by *blast*
qed

corollary *s-distinct-inv-maintained-perm-step*:
assumes $\textit{s-perm-inv}\ \alpha\ T\ B\ SA0\ SA\ i$
and $i = \textit{Suc}\ n$
and $\textit{Suc}\ n < \textit{length}\ SA$
and $SA\ !\ \textit{Suc}\ n = \textit{Suc}\ j$
and $\textit{suffix-type}\ T\ j = \textit{S-type}$
and $b = \alpha\ (T\ !\ j)$
and $k = B\ !\ b - \textit{Suc}\ 0$
shows $\textit{s-distinct-inv}\ \alpha\ T\ (B[b := k])\ (SA[k := j])$
using *s-distinct-inv-maintained-step*[*OF s-perm-inv-elims*(*1*−*6*,*8*−*14*)[*OF assms*(*1*)]
assms(*2*−)]
by *blast*

83.3.2 Bucket Pointer

lemma *s-bucket-ptr-inv-established*:
assumes $\textit{s-bucket-init}\ \alpha\ T\ B$
and $\textit{valid-list}\ T$

and *strict-mono* α
and α *bot* = *0*
shows *s-bucket-ptr-inv* α *T B*
unfolding *s-bucket-ptr-inv-def*
proof(*intro allI impI*)
fix *b*
let *?goal* = *s-bucket-start* α *T b* $\leq$ *B* ! *b* $\wedge$ *B* ! *b* $\leq$ *bucket-end* α *T b* $\wedge$ (*b* = *0* $\longrightarrow$ *B* ! *b* = *0*)
assume *b* $\leq$ α (*Max* (*set T*))

have *b* > *0* $\Longrightarrow$ *?goal*
proof −
assume *b* > *0*
with *s-bucket-initD*(*1*)[*OF assms*(*1*) ‹*b* $\leq$ -›]
have *B* ! *b* = *bucket-end* α *T b* **.**
then show *?thesis*
by (*metis* ‹*0* < *b*› *l-bucket-end-le-bucket-end less-numeral-extra*(*3*) *order-refl s-bucket-start-eq-l-bucket-end*)
qed
moreover
have *b* = *0* $\Longrightarrow$ *?goal*
proof −
assume *b* = *0*
with *s-bucket-initD*(*2*)[*OF assms*(*1*) ‹*b* $\leq$ -›]
have *B* ! *b* = *0* **.**
with ‹*b* = *0*›
valid-list-bucket-end-0[*OF assms*(*2*−)]
valid-list-s-bucket-start-0[*OF assms*(*2*−)]
show *?thesis*
by *auto*
qed
ultimately show *?goal*
by *auto*
qed

lemma *s-bucket-ptr-inv-maintained-step*:
assumes *s-distinct-inv* α *T B SA*
and *s-bucket-ptr-inv* α *T B*
and *s-locations-inv* α *T B SA*
and *s-unchanged-inv* α *T B SA0 SA*
and *s-seen-inv* α *T B SA i*
and *s-pred-inv* α *T B SA i*
and *strict-mono* α
and α (*Max* (*set T*)) < *length B*
and *length SA0* = *length T*
and *length SA* = *length T*
and *l-types-init* α *T SA0*
and *valid-list T*
and α *bot* = *0*

and $i = Suc\ n$
and $Suc\ n < length\ SA$
and $SA\ !\ Suc\ n = Suc\ j$
and $suffix\text{-}type\ T\ j = S\text{-}type$
and $b = \alpha\ (T\ !\ j)$
and $k = B\ !\ b - Suc\ 0$
shows $s\text{-}bucket\text{-}ptr\text{-}inv\ \alpha\ T\ (B[b := k])$
unfolding $s\text{-}bucket\text{-}ptr\text{-}inv\text{-}def$
proof$(intro\ allI\ impI)$
fix b'
assume $b' \leq \alpha\ (Max\ (set\ T))$

let $?goal = s\text{-}bucket\text{-}start\ \alpha\ T\ b' \leq B[b := k]\ !\ b' \wedge\ B[b := k]\ !\ b' \leq bucket\text{-}end\ \alpha\ T\ b' \wedge$
$(b' = 0 \longrightarrow B[b := k]\ !\ b' = 0)$

from $valid\text{-}list\text{-}length\text{-}ex[OF\ assms(12)]$
obtain m **where**
$length\ T = Suc\ m$
by $blast$
moreover
have $j < length\ T$
by $(simp\ add\text{:}\ assms(17)\ suffix\text{-}type\text{-}s\text{-}bound)$
moreover
from $s\text{-}seen\text{-}invD(1)[OF\ assms(5{,}15)]\ assms(14{,}16)$
have $Suc\ j < length\ T$
by $simp$
ultimately have $j < m$
by $linarith$
hence $b \neq 0$
by $(metis\ \langle length\ T = Suc\ m\rangle\ assms(7{,}12{,}13{,}18)\ diff\text{-}Suc\text{-}1\ strict\text{-}mono\text{-}eq\ valid\text{-}list\text{-}def)$

have $b \neq b' \Longrightarrow ?goal$
proof $-$
assume $b \neq b'$
hence $B[b := k]\ !\ b' = B\ !\ b'$
by $simp$
with $s\text{-}bucket\text{-}ptr\text{-}lower\text{-}bound[OF\ assms(2)\ \langle b' \leq \alpha\ (Max\ (set\ T))\rangle]$
$s\text{-}bucket\text{-}ptr\text{-}upper\text{-}bound[OF\ assms(2)\ \langle b' \leq \alpha\ (Max\ (set\ T))\rangle]$
$s\text{-}bucket\text{-}ptr\text{-}0[OF\ assms(2)]$
show $?thesis$
by $auto$
qed
moreover
have $b = b' \Longrightarrow ?goal$
proof $-$
assume $b = b'$
from $s\text{-}bucket\text{-}ptr\text{-}strict\text{-}lower\text{-}bound[OF\ assms(1{-}7{,}9{-}13{,}14{-}18)]$

have *s-bucket-start* α *T b* $<$ *B* ! *b*.
hence *s-bucket-start* α *T b* $\leq$ *k*
using *assms*(*19*) **by** *linarith*
moreover
from ‹*b* = *b'*› ‹*b'* $\leq$ α (*Max* (*set T*))› *assms*(*2*,*19*)
have *k* $\leq$ *bucket-end* α *T b*
using *le-diff-conv s-bucket-ptr-inv-def trans-le-add1* **by** *blast*
moreover
have *B*[*b* := *k*] ! *b'* = *k*
using ‹*b* = *b'*› ‹*b'* $\leq$ α (*Max* (*set T*))› *assms*(*8*) **by** *auto*
ultimately show *?thesis*
using ‹*b* = *b'*› ‹*b* $\neq$ *0*› **by** *auto*
qed
ultimately show *?goal*
by *linarith*
qed

corollary *s-bucket-ptr-inv-maintained-perm-step*:
assumes *s-perm-inv* α *T B SA0 SA i*
and *i* = *Suc n*
and *Suc n* $<$ *length SA*
and *SA* ! *Suc n* = *Suc j*
and *suffix-type T j* = *S-type*
and *b* = α (*T* ! *j*)
and *k* = *B* ! *b* − *Suc 0*
shows *s-bucket-ptr-inv* α *T* (*B*[*b* := *k*])
using *s-bucket-ptr-inv-maintained-step*[*OF s-perm-inv-elims*(*1*−*6*,*8*−*14*)[*OF assms*(*1*)] *assms*(*2*−)]
by *blast*

83.3.3 Locations

lemma *s-locations-inv-established*:
assumes *s-bucket-init* α *T B*
and *s-type-init T SA*
and *valid-list T*
and *strict-mono* α
and α *bot* = *0*
shows *s-locations-inv* α *T B SA*
unfolding *s-locations-inv-def*
proof(*safe*)
fix *b i*
assume *b* $\leq$ α (*Max* (*set T*)) *B* ! *b* $\leq$ *i i* $<$ *bucket-end* α *T b*
hence *b* $>$ *0* $\Longrightarrow$ *SA* ! *i* $\in$ *s-bucket* α *T b*
by (*metis assms*(*1*) *not-le s-bucket-init-def*)
moreover
have *b* = *0* $\Longrightarrow$ *SA* ! *i* $\in$ *s-bucket* α *T b*
proof −
assume *b* = *0*

have *0* ≤ *i*
  by *blast*
moreover
have *bucket-end* α *T 0* = *1*
  using *assms*(*3*−*5*) *valid-list-bucket-end-0* **by** *blast*
with ‹*i* < *bucket-end* α *T b*› ‹*b* = *0*›
have *i* < *1*
  by *simp*
ultimately have *i* = *0*
  by *blast*
moreover
from *s-type-init-def*[*of T SA*] *assms*(*2*)
obtain *n* **where**
  length T = *Suc n*
  SA ! *0* = *n*
  by *blast*
with *suffix-type-last*[*of T n*]
have *suffix-type T n* = *S-type*
  by *blast*
moreover
have *T* ! *n* = *bot*
  by (*metis* ‹*length T* = *Suc n*› *assms*(*3*) *diff-Suc-1 last-conv-nth less-numeral-extra*(*3*) *list.size*(*3*) *valid-list-def*)
hence α (*T* ! *n*) = *0*
  by (*simp add*: *assms*(*5*))
ultimately show *?thesis*
  by (*simp add*: ‹*SA* ! *0* = *n*›‹*b* = *0*› ‹*length T* = *Suc n*› *bucket-def s-bucket-def*)
qed
ultimately show *SA* ! *i* ∈ *s-bucket* α *T b*
  by *linarith*
qed

lemma *s-locations-inv-maintained-step*:
assumes *s-distinct-inv* α *T B SA*
and *s-bucket-ptr-inv* α *T B*
and *s-locations-inv* α *T B SA*
and *s-unchanged-inv* α *T B SA0 SA*
and *s-seen-inv* α *T B SA i*
and *s-pred-inv* α *T B SA i*
and *strict-mono* α
and α (*Max* (*set T*)) < *length B*
and *length SA0* = *length T*
and *length SA* = *length T*
and *l-types-init* α *T SA0*
and *valid-list T*
and α *bot* = *0*
and *i* = *Suc n*
and *Suc n* < *length SA*
and *SA* ! *Suc n* = *Suc j*

and *suffix-type* T j = *S-type*
and $b = \alpha\ (T\ !\ j)$
and $k = B\ !\ b - Suc\ 0$
shows *s-locations-inv* α T $(B[b := k])$ $(SA[k := j])$
unfolding *s-locations-inv-def*
proof(*safe*)
fix b' i'
assume $b' \leq \alpha\ (Max\ (set\ T))$ $B[b := k]\ !\ b' \leq i'$ $i' <$ *bucket-end* α T b'

from *s-bucket-ptr-strict-lower-bound*[*OF assms*(*1*−*7*,*9*−*18*)]
have *s-bucket-start* α T $b < B\ !\ b$.
hence *s-bucket-start* α T $b \leq k$
using *assms*(*19*) **by** *linarith*

from ‹*s-bucket-start* α T $b < B\ !\ b$›
have $k < B\ !\ b$
using *assms*(*19*) **by** *linarith*

have $j <$ *length* T
by (*simp add*: *assms*(*17*) *suffix-type-s-bound*)
hence $b \leq \alpha\ (Max\ (set\ T))$
by (*simp add*: *assms*(*18*) *assms*(*7*) *strict-mono-less-eq*)
with *s-bucket-ptr-upper-bound*[*OF assms*(*2*)]
have $B\ !\ b \leq$ *bucket-end* α T b
by *blast*

have $b \neq b' \Longrightarrow SA[k := j]\ !\ i' \in$ *s-bucket* α T b'
proof −
assume $b \neq b'$
hence $B[b := k]\ !\ b' = B\ !\ b'$
by *simp*
with ‹$B[b := k]\ !\ b' \leq i'$›
have $B\ !\ b' \leq i'$
by *simp*
with *s-locations-invD*[*OF assms*(*3*) ‹$b' \leq \alpha\ (Max\ (set\ T))$› - ‹$i' <$ *bucket-end* α T b'›]
have $SA\ !\ i' \in$ *s-bucket* α T b'
by *linarith*
moreover
from *s-bucket-ptr-lower-bound*[*OF assms*(*2*) ‹$b' \leq \alpha\ (Max\ (set\ T))$›] ‹$B\ !\ b' \leq i'$›
have *bucket-start* α T $b' \leq i'$
by (*meson bucket-start-le-s-bucket-start dual-order.trans*)
with *outside-another-bucket*[*OF* ‹$b \neq b'$›[*symmetric*] - ‹$i' <$ -›]
‹$B\ !\ b \leq$ *bucket-end* α T b› ‹$k < B\ !\ b$› ‹*s-bucket-start* α T $b \leq k$›
have $k \neq i'$
using *bucket-start-le-s-bucket-start le-trans less-le-trans* **by** *blast*
hence $SA[k := j]\ !\ i' = SA\ !\ i'$
by *simp*

ultimately show *?thesis*
  by *simp*
qed
moreover
have $b = b' \Longrightarrow SA[k := j] \ ! \ i' \in$ *s-bucket* α T b'
proof −
  assume $b = b'$
  hence $k \leq i'$
    using ‹$B[b := k] \ ! \ b' \leq i'$› ‹$b' \leq \alpha$ (*Max* (*set* T))› *assms*(*8*) **by** *auto*
  hence $k = i' \lor k < i'$
    by *linarith*
  moreover
  have $k = i' \Longrightarrow$ *?thesis*
  proof −
    assume $k = i'$
    hence $SA[k := j] \ ! \ i' = j$
    by (*metis* ‹$i' <$ *bucket-end* α T b'› *assms*(*10*) *bucket-end-le-length dual-order.strict-trans1*
        nth-list-update)
    with *assms*(*17*,*18*) ‹$b = b'$› ‹$j <$ *length* T›
    show *?thesis*
      by (*simp add*: *bucket-def s-bucket-def*)
  qed
  moreover
  have $k < i' \Longrightarrow$ *?thesis*
  proof −
    assume $k < i'$
    hence $B \ ! \ b \leq i'$
      using *assms*(*19*) **by** *linarith*
    with *s-locations-invD*[*OF assms*(*3*) ‹$b' \leq$ -› - ‹$i' <$ *bucket-end* - - -›] ‹$b = b'$›
    have $SA \ ! \ i' \in$ *s-bucket* α T b'
      by *blast*
    moreover
    have $SA[k := j] \ ! \ i' = SA \ ! \ i'$
      using ‹$k < i'$› **by** *auto*
    ultimately show *?thesis*
      by *simp*
  qed
  ultimately show *?thesis*
    by *linarith*
qed
ultimately show $SA[k := j] \ ! \ i' \in$ *s-bucket* α T b'
  by *blast*
qed

corollary *s-locations-inv-maintained-perm-step*:
  assumes *s-perm-inv* α T B *SA0* SA i
  and $i = Suc \ n$
  and $Suc \ n <$ *length* SA
  and $SA \ ! \ Suc \ n = Suc \ j$

and *suffix-type* T j = *S-type*
and $b = \alpha\ (T\ !\ j)$
and $k = B\ !\ b - \mathit{Suc}\ 0$
shows *s-locations-inv* α T $(B[b := k])$ $(SA[k := j])$
using *s-locations-inv-maintained-step*[*OF s-perm-inv-elims*(*1*−*6*,*8*−*14*)[*OF assms*(*1*)] *assms*(*2*−)]
by *blast*

83.3.4 Unchanged

lemma *s-unchanged-inv-established*:
shows *s-unchanged-inv* α T B SA SA
by (*simp add*: *s-unchanged-inv-def*)

lemma *s-unchanged-inv-maintained-step*:
assumes *s-distinct-inv* α T B SA
and *s-bucket-ptr-inv* α T B
and *s-locations-inv* α T B SA
and *s-unchanged-inv* α T B $SA0$ SA
and *s-seen-inv* α T B SA i
and *s-pred-inv* α T B SA i
and *strict-mono* α
and $\alpha\ (\mathit{Max}\ (\mathit{set}\ T)) < \mathit{length}\ B$
and $\mathit{length}\ SA0 = \mathit{length}\ T$
and $\mathit{length}\ SA = \mathit{length}\ T$
and *l-types-init* α T $SA0$
and *valid-list* T
and $\alpha\ \mathit{bot} = 0$
and $i = \mathit{Suc}\ n$
and $\mathit{Suc}\ n < \mathit{length}\ SA$
and $SA\ !\ \mathit{Suc}\ n = \mathit{Suc}\ j$
and *suffix-type* T j = *S-type*
and $b = \alpha\ (T\ !\ j)$
and $k = B\ !\ b - \mathit{Suc}\ 0$
shows *s-unchanged-inv* α T $(B[b := k])$ $SA0$ $(SA[k := j])$
unfolding *s-unchanged-inv-def*
proof(*safe*)
fix b' i'
assume $b' \leq \alpha\ (\mathit{Max}\ (\mathit{set}\ T))$ *bucket-start* α T $b' \leq i'$ $i' < B[b := k]\ !\ b'$

from *s-bucket-ptr-strict-lower-bound*[*OF assms*(*1*−*7*,*9*−*18*)]
have *s-bucket-start* α T $b < B\ !\ b$.
hence *s-bucket-start* α T $b \leq k$
using *assms*(*19*) **by** *linarith*
hence *bucket-start* α T $b \leq k$
using *bucket-start-le-s-bucket-start le-trans* **by** *blast*

from ‹*s-bucket-start* α T $b < B\ !\ b$›
have $k < B\ !\ b$

using *assms*(*19*) **by** *linarith*

have $j < \mathit{length}\ T$
by (*simp add*: *assms*(*17*) *suffix-type-s-bound*)
hence $b \leq \alpha\ (\mathit{Max}\ (\mathit{set}\ T))$
by (*simp add*: *assms*(*7*,*18*) *strict-mono-less-eq*)
with *s-bucket-ptr-upper-bound*[*OF assms*(*2*)]
have $B\ !\ b \leq \mathit{bucket\text{-}end}\ \alpha\ T\ b$
by *blast*
with ‹$k < B\ !\ b$›
have $k < \mathit{bucket\text{-}end}\ \alpha\ T\ b$
by *linarith*

have $b = b' \Longrightarrow SA[k := j]\ !\ i' = SA0\ !\ i'$
proof −
assume $b = b'$
hence $B[b := k]\ !\ b' = k$
using ‹$b' \leq \alpha\ (\mathit{Max}\ (\mathit{set}\ T))$› *assms*(*8*) **by** *auto*
with ‹$i' < B[b := k]\ !\ b'$›
have $i' < k$
by *linarith*
hence $SA[k := j]\ !\ i' = SA\ !\ i'$
by *simp*
moreover
from ‹$i' < k$› ‹$k < B\ !\ b$› ‹$b = b'$›
have $i' < B\ !\ b'$
by *simp*
with *s-unchanged-invD*[*OF assms*(*4*) ‹$b' \leq$ -› ‹*bucket-start* - - - $\leq i'$›]
have $SA\ !\ i' = SA0\ !\ i'$
by *simp*
ultimately show *?thesis*
by *simp*
qed
moreover
have $b \neq b' \Longrightarrow SA[k := j]\ !\ i' = SA0\ !\ i'$
proof −
assume $b \neq b'$
with ‹$i' < B[b := k]\ !\ b'$›
have $i' < B\ !\ b'$
by *simp*
with *s-unchanged-invD*[*OF assms*(*4*) ‹$b' \leq$ -› ‹*bucket-start* - - $b' \leq$ -›]
have $SA\ !\ i' = SA0\ !\ i'$
by *blast*
moreover
from *s-bucket-ptr-upper-bound*[*OF assms*(*2*) ‹$b' \leq$ -›] ‹$i' < B\ !\ b'$›
have $i' < \mathit{bucket\text{-}end}\ \alpha\ T\ b'$
by *linarith*
with *outside-another-bucket*[*OF* ‹$b \neq b'$› ‹*bucket-start* - - - $\leq k$› ‹$k <$ *bucket-end* - - -›]

‹*bucket-start* - - $b' \leq i'$›
have $k \neq i'$
by *blast*
hence $SA[k := j] \ ! \ i' = SA \ ! \ i'$
by *simp*
ultimately show *?thesis*
by *simp*
qed
ultimately show $SA[k := j] \ ! \ i' = SA0 \ ! \ i'$
by *blast*
qed

corollary *s-unchanged-inv-maintained-perm-step*:
assumes *s-perm-inv* α *T B SA0 SA i*
and $i = Suc\ n$
and $Suc\ n < length\ SA$
and $SA \ ! \ Suc\ n = Suc\ j$
and *suffix-type T j* = *S-type*
and $b = \alpha\ (T \ ! \ j)$
and $k = B \ ! \ b - Suc\ 0$
shows *s-unchanged-inv* α T $(B[b := k])$ *SA0* $(SA[k := j])$
using *s-unchanged-inv-maintained-step*[*OF s-perm-inv-elims*(*1*−*6*,*8*−*14*)[*OF assms*(*1*)] *assms*(*2*−)]
by *blast*

83.3.5 Seen

lemma *s-seen-inv-established*:
assumes *length SA* = *length T*
and $length\ T \leq n$
shows *s-seen-inv* α *T B SA n*
unfolding *s-seen-inv-def*
using *assms* **by** *auto*

lemma *s-seen-inv-maintained-step-c1*:
assumes *s-bucket-ptr-inv* α *T B*
and *s-unchanged-inv* α *T B SA0 SA*
and *s-seen-inv* α *T B SA i*
and *strict-mono* α
and *length SA0* = *length T*
and *length SA* = *length T*
and *l-types-init* α *T SA0*
and *valid-list T*
and $Suc\ 0 < length\ T$
and $i = Suc\ n$
and $length\ SA \leq Suc\ n$
shows *s-seen-inv* α *T B SA n*
unfolding *s-seen-inv-def*
proof (*intro allI impI*)

fix j
assume $j < length\ SA$ $n \leq j$
hence $n < length\ SA$
  by *simp*
with *assms*(*10*,*11*)
have $length\ SA = Suc\ n$
  by *linarith*

let $?b = \alpha\ (T\ !\ (SA\ !\ j))$
let $?g1 = (suffix\text{-}type\ T\ (SA\ !\ j) = S\text{-}type \longrightarrow in\text{-}s\text{-}current\text{-}bucket\ \alpha\ T\ B\ ?b\ j)$
and $?g2 = (suffix\text{-}type\ T\ (SA\ !\ j) = L\text{-}type \longrightarrow in\text{-}l\text{-}bucket\ \alpha\ T\ ?b\ j)$
and $?g3 = SA\ !\ j < length\ T$

from ‹$n \leq j$›
have $n = j \lor Suc\ n \leq j$
  using *dual-order.antisym not-less-eq-eq* **by** *auto*
moreover
have $n = j \Longrightarrow ?g1 \land ?g2 \land ?g3$
proof −
  let $?b\text{-}max = \alpha\ (Max\ (set\ T))$
  assume $n = j$
  hence $j < bucket\text{-}end\ \alpha\ T\ ?b\text{-}max$
    using ‹$j < length\ SA$› *assms*(*4*,*6*) *bucket-end-Max* **by** *fastforce*
  hence $j < l\text{-}bucket\text{-}end\ \alpha\ T\ ?b\text{-}max$
    using *l-bucket-Max*[*OF assms*(*8*,*9*,*4*)]
   by (*simp add*: *bucket-end-def′ bucket-size-def l-bucket-end-def l-bucket-size-def*)
  moreover
  from ‹$n = j$› ‹$j < length\ SA$› ‹$length\ SA = Suc\ n$› *assms*(*4*,*6*,*10*)
  have $bucket\text{-}start\ \alpha\ T\ ?b\text{-}max \leq j$
   by (*metis Suc-leI antisym bucket-end-eq-length bucket-end-le-length gr-implies-not0*
      index-in-bucket-interval-gen length-0-conv)
  moreover
  have $?b\text{-}max \leq \alpha\ (Max\ (set\ T))$
    by *simp*
  ultimately have $SA\ !\ j \in l\text{-}bucket\ \alpha\ T\ ?b\text{-}max$
   using *l-types-init-nth*[*OF assms*(*6*) *l-types-init-maintained*[*OF assms*(*1*,*2*,*5*−*7*)]
]
    by *blast*
  hence $?b\text{-}max = \alpha\ (T\ !\ (SA\ !\ j))$
    by (*simp add*: *bucket-def l-bucket-def*)
  moreover
  from ‹$SA\ !\ j \in l\text{-}bucket\ \alpha\ T\ ?b\text{-}max$›
  have $?g3$
    by (*simp add*: *bucket-def l-bucket-def*)
  moreover
  from ‹$SA\ !\ j \in l\text{-}bucket\ \alpha\ T\ ?b\text{-}max$›
  have $suffix\text{-}type\ T\ (SA\ !\ j) = L\text{-}type$
    by (*simp add*: *bucket-def l-bucket-def*)
  moreover

have *in-l-bucket* α *T* (α (*T* ! (*SA* ! *j*))) *j*
using ‹*bucket-start* - - - $\leq$ *j*› ‹*j* $<$ *l-bucket-end* - - -› *calculation*(*1*) *in-l-bucket-def*
by *fastforce*
hence *?g2*
using *calculation*(*3*) **by** *blast*
moreover
from *calculation*(*3*)
have *?g1*
by *simp*
ultimately show *?thesis*
by *simp*
qed
moreover
have *Suc n* $\leq$ *j* $\Longrightarrow$ *?g1* $\wedge$ *?g2* $\wedge$ *?g3*
proof −
assume *Suc n* $\leq$ *j*
with *s-seen-invD*[*OF assms*(*3*) ‹*j* $<$ *length SA*›] *assms*(*10*)
show *?thesis*
by *blast*
qed
ultimately show *?g1* $\wedge$ *?g2* $\wedge$ *?g3*
by *blast*
qed

corollary *s-seen-inv-maintained-perm-step-c1*:
assumes *s-perm-inv* α *T B SA0 SA i*
and *i* = *Suc n*
and *length SA* $\leq$ *Suc n*
shows *s-seen-inv* α *T B SA n*
using *s-seen-inv-maintained-step-c1*[*OF s-perm-inv-elims*(*2*,*4*,*5*,*8*,*10*−*13*,*15*)[*OF assms*(*1*)] *assms*(*2*−)]
by *blast*

lemma *s-seen-inv-maintained-step-c1-alt*:
assumes *s-bucket-ptr-inv* α *T B*
and *s-unchanged-inv* α *T B SA0 SA*
and *s-seen-inv* α *T B SA i*
and *strict-mono* α
and *length SA0* = *length T*
and *length SA* = *length T*
and *l-types-init* α *T SA0*
and *valid-list T*
and *Suc 0* $<$ *length T*
and *i* = *Suc n*
and *length T* $\leq$ *SA* ! *Suc n*
shows *s-seen-inv* α *T B SA n*
proof (*cases length SA* $\leq$ *Suc n*)
assume *length SA* $\leq$ *Suc n*

then show *?thesis*
using *assms*(*1−10*) *s-seen-inv-maintained-step-c1* **by** *blast*
next
assume ¬ *length SA* ≤ *Suc n*
hence *Suc n* < *length SA*
by *simp*
show *?thesis*
unfolding *s-seen-inv-def*
proof (*intro allI impI*)
fix *j*
assume *j* < *length SA n* ≤ *j*
hence *n* < *length SA*
by *simp*

let *?b* = α (*T* ! (*SA* ! *j*))
let *?g1* = (*suffix-type T* (*SA* ! *j*) = *S-type* ⟶ *in-s-current-bucket* α *T B ?b j*)
and *?g2* = (*suffix-type T* (*SA* ! *j*) = *L-type* ⟶ *in-l-bucket* α *T ?b j*)
and *?g3* = *SA* ! *j* < *length T*

from ‹*n* ≤ *j*›
have *n* = *j* ∨ *Suc n* ≤ *j*
using *dual-order.antisym not-less-eq-eq* **by** *auto*
moreover
have *n* = *j* ⟹ *?g1* ∧ *?g2* ∧ *?g3*
by (*metis Suc-le-mono* ‹*Suc n* < *length SA*› ‹*n* ≤ *j*› *assms*(*3*,*10*,*11*) *linorder-not-le s-seen-invD*(*1*))
moreover
have *Suc n* ≤ *j* ⟹ *?g1* ∧ *?g2* ∧ *?g3*
proof −
assume *Suc n* ≤ *j*
with *s-seen-invD*[*OF assms*(*3*) ‹*j* < *length SA*›] *assms*(*10*)
show *?thesis*
by *blast*
qed
ultimately show *?g1* ∧ *?g2* ∧ *?g3*
by *blast*
qed
qed

corollary *s-seen-inv-maintained-perm-step-c1-alt*:
assumes *s-perm-inv* α *T B SA0 SA i*
and *i* = *Suc n*
and *length T* ≤ *SA* ! *Suc n*
shows *s-seen-inv* α *T B SA n*
using *s-seen-inv-maintained-step-c1-alt*[*OF s-perm-inv-elims*(*2*,*4*,*5*,*8*,*10−13*,*15*)[*OF assms*(*1*)] *assms*(*2−*)]
by *blast*

lemma *s-seen-inv-maintained-step-c2*:

assumes *s-distinct-inv* α *T B SA*
and *s-bucket-ptr-inv* α *T B*
and *s-locations-inv* α *T B SA*
and *s-unchanged-inv* α *T B SA0 SA*
and *s-seen-inv* α *T B SA i*
and *s-pred-inv* α *T B SA i*
and *s-suc-inv* α *T B SA i*
and *strict-mono* α
and α (*Max* (*set T*)) < *length B*
and *length SA0* = *length T*
and *length SA* = *length T*
and *l-types-init* α *T SA0*
and *valid-list T*
and α *bot* = *0*
and *Suc 0* < *length T*
and *i* = *Suc n*
and *Suc n* < *length SA*
and *SA* ! *Suc n* = *0*
shows *s-seen-inv* α *T B SA n*
unfolding *s-seen-inv-def*
proof (*intro allI impI*)
fix *j*
assume *j* < *length SA n* ≤ *j*
hence *n* < *length SA*
by *simp*
hence *n* < *length T*
by (*simp add*: *assms*(*11*))

let *?b* = α (*T* ! (*SA* ! *j*))
let *?g1* = (*suffix-type T* (*SA* ! *j*) = *S-type* ⟶ *in-s-current-bucket* α *T B ?b j*)
and *?g2* = (*suffix-type T* (*SA* ! *j*) = *L-type* ⟶ *in-l-bucket* α *T ?b j*)
and *?g3* = *SA* ! *j* < *length T*

from ‹*n* ≤ *j*›
have *n* = *j* ∨ *Suc n* ≤ *j*
by *linarith*
moreover
have *n* = *j* ⟹ *?g1* ∧ *?g2* ∧ *?g3*
proof −
assume *n* = *j*
with *index-in-bucket-interval-gen*[*OF* ‹*n* < *length T*› *assms*(*8*)]
obtain *b* **where**
b ≤ α (*Max* (*set T*))
bucket-start α *T b* ≤ *j*
j < *bucket-end* α *T b*
by *blast*

have *j* < *l-bucket-end* α *T b* ∨ *s-bucket-start* α *T b* ≤ *j*

by (*metis not-le s-bucket-start-eq-l-bucket-end*)
moreover
have $j < l\text{-}bucket\text{-}end\ \alpha\ T\ b \Longrightarrow ?thesis$
proof −
assume $j < l\text{-}bucket\text{-}end\ \alpha\ T\ b$
with *l-types-init-nth*[*OF assms*(*11*) *l-types-init-maintained*[*OF assms*(*2*,*4*,*10*−*12*)]
‹$b \le$ -›
‹*bucket-start* - - - $\le j$›]
have $SA\ !\ j \in l\text{-}bucket\ \alpha\ T\ b$ **.**
hence $suffix\text{-}type\ T\ (SA\ !\ j) = L\text{-}type\ SA\ !\ j < length\ T$
by (*simp add*: *l-bucket-def bucket-def*)+
moreover
have *?g1*
by(*simp add*: *calculation*(*1*) *SL-types.exhaust*)
moreover
from ‹$SA\ !\ j \in l\text{-}bucket\ \alpha\ T\ b$›
have $b = \alpha\ (T\ !\ (SA\ !\ j))$
by (*metis* (*mono-tags*, *lifting*) *bucket-def l-bucket-def mem-Collect-eq*)
with ‹$bucket\text{-}start\ \alpha\ T\ b \le j$› ‹$j < l\text{-}bucket\text{-}end\ \alpha\ T\ b$› ‹$b \le$ -›
have $in\text{-}l\text{-}bucket\ \alpha\ T\ (\alpha\ (T\ !\ (SA\ !\ j)))\ j$
using *in-l-bucket-def* **by** *blast*
ultimately show *?thesis*
by *blast*
qed
moreover
have $s\text{-}bucket\text{-}start\ \alpha\ T\ b \le j \Longrightarrow ?thesis$
proof −
assume $s\text{-}bucket\text{-}start\ \alpha\ T\ b \le j$
hence $s\text{-}bucket\text{-}start\ \alpha\ T\ b < i$
by (*simp add*: ‹$n = j$› *assms*(*16*))

have $B\ !\ b \le i$
proof(*rule ccontr*)
assume $\neg B\ !\ b \le i$
hence $i < B\ !\ b$
by *simp*
with *s-B-val*[*OF assms*(*1*−*6*,*8*,*10*−*13*,*15*) ‹$b \le \alpha$ -›]
have $B\ !\ b = s\text{-}bucket\text{-}start\ \alpha\ T\ b$ **.**
with ‹$s\text{-}bucket\text{-}start\ \alpha\ T\ b < i$› ‹$i < B\ !\ b$›
show *False*
by *linarith*
qed
hence $B\ !\ b < i \lor B\ !\ b = i$
using *dual-order.order-iff-strict* **by** *blast*
moreover
have $B\ !\ b < i \Longrightarrow ?thesis$
proof −
assume $B\ !\ b < i$
hence $B\ !\ b \le j$

by (*simp add*: ‹$n = j$› *assms*(*16*))
with *s-locations-invD*[*OF assms*(*3*) ‹$b \le$ -› - ‹$j <$ *bucket-end* - - -›]
have *SA* ! $j \in$ *s-bucket* α *T b* **.**
hence *SA* ! $j <$ *length T suffix-type T* (*SA* ! *j*) = *S-type*
by (*simp add*: *s-bucket-def bucket-def*)+
moreover
from *calculation*(*2*)
have *?g2*
by *simp*
moreover
from ‹*SA* ! $j \in$ *s-bucket* α *T b*›
have α (*T* ! (*SA* ! *j*)) = *b*
by (*simp add*: *s-bucket-def bucket-def*)
with ‹*B* ! $b \le j$› ‹$j <$ *bucket-end* α *T b*› ‹$b \le$ -›
have *in-s-current-bucket* α *T B* (α (*T* ! (*SA* ! *j*))) *j*
by (*simp add*: *in-s-current-bucket-def*)
ultimately show *?thesis*
by *blast*
qed
moreover
have *B* ! *b* = *i* $\Longrightarrow$ *?thesis*
proof –
assume *B* ! *b* = *i*
hence *s-bucket-start* α *T b* < *B* ! *b*
using ‹*s-bucket-start* α *T b* < *i*› **by** *blast*

have $b \neq 0$
using ‹*B* ! *b* = *i*› *assms*(*2*,*16*) *less-Suc-eq-0-disj s-bucket-ptr-0* **by** *fastforce*

let *?xs* = *list-slice SA* (*B* ! *b*) (*bucket-end* α *T b*)
let *?B* = *set ?xs*
let *?A* = *s-bucket* α *T b* − *?B*

from *s-locations-inv-subset-s-bucket*[*OF assms*(*3*) ‹$b \le$ -›]
have *?B* $\subseteq$ *s-bucket* α *T b* **.**
hence *?A* $\subseteq$ *s-bucket* α *T b*
by *blast*

have *card* (*s-bucket* α *T b*) = *bucket-end* α *T b* − *s-bucket-start* α *T b*
by (*simp add*: *bucket-end-eq-s-start-pl-size s-bucket-size-def*)

from *s-distinct-invD*[*OF assms*(*1*) ‹$b \le$ -›]
have *card ?B* = *bucket-end* α *T b* − *B* ! *b*
by (*metis assms*(*11*) *bucket-end-le-length distinct-card length-list-slice min.absorb-iff1*)
hence *card ?B* < *card* (*s-bucket* α *T b*)
using ‹*card* (*s-bucket* α *T b*) = *bucket-end* α *T b* − *s-bucket-start* α *T b*› ‹$j <$ *bucket-end* α *T b*› ‹*s-bucket-start* α *T b* < *B* ! *b*› ‹*s-bucket-start* α *T b* $\le j$›

by *linarith*
with *card-psubset*[*OF finite-s-bucket* ‹*?B* ⊆ *s-bucket* α *T b*›]
have *?B* ⊂ *s-bucket* α *T b* **.**
hence *?A* ≠ {}
by *blast*
with *subset-s-bucket-successor*[*OF assms*(*13*,*8*,*14*) ‹*b* ≠ -› ‹*?A* ⊆ -›]
obtain *x* **where**
x ∈ *?A*
Suc x ∈ *?B* ∨ (∃ *b*′. *b* < *b*′ ∧ *Suc x* ∈ *bucket* α *T b*′)
by *blast*
hence *suffix-type T x* = *S-type* α (*T* ! *x*) = *b x* < *length T*
by (*simp add*: *s-bucket-def bucket-def*)+

from ‹*x* ∈ *?A*›
have *x* ∉ *?B*
by *blast*

have *Suc x* ∈ *?B* ⟹ *?thesis*
proof −
assume *Suc x* ∈ *?B*
from *nth-mem-list-slice*[*OF* ‹*Suc x* ∈ *?B*›]
obtain *i*′ **where**
i′ < *length SA*
B ! *b* ≤ *i*′
i′ < *bucket-end* α *T b*
SA ! *i*′ = *Suc x*
by *blast*

have *i* ≠ *i*′
proof (*rule ccontr*)
assume ¬ *i* ≠ *i*′
hence *i* = *i*′
by *auto*
with *assms*(*16*,*18*) ‹*SA* ! *i*′ = *Suc x*›
show *False*
by *simp*
qed
with ‹*B* ! *b* = *i*› ‹*B* ! *b* ≤ *i*′›
have *i* < *i*′
by *simp*
with *s-suc-invD*[*OF assms*(*7*) ‹*i*′ < *length SA*› - ‹*SA* ! *i*′ = *Suc x*›
‹*suffix-type T x* = -›,
simplified ‹α (*T* ! *x*) = *b*›]
obtain *k* **where**
in-s-current-bucket α *T B b k*
SA ! *k* = *x*
k < *i*′
by *blast*
hence *x* ∈ *?B*

by (*meson assms*(*11*) *in-s-current-bucket-list-slice*)
with ‹$x \notin$ *?B*›
have *False*
by *blast*
then show *?thesis*
by *blast*
qed
moreover
have $\exists b'.\ b < b' \wedge$ *Suc* $x \in$ *bucket* α *T* $b' \Longrightarrow$ *?thesis*
proof −
assume $\exists b'.\ b < b' \wedge$ *Suc* $x \in$ *bucket* α *T* b'
then obtain b' **where**
$b < b'$
Suc $x \in$ *bucket* α *T* b'
by *blast*
hence *bucket-end* α *T* $b \leq$ *bucket-start* α *T* b'
by (*simp add*: *less-bucket-end-le-start*)
with *s-bucket-ptr-upper-bound*[*OF assms*(*2*) ‹$b \leq$ - ›] ‹*B* ! $b = i$›
have $i \leq$ *bucket-start* α *T* b'
by *linarith*

from ‹*Suc* $x \in$ *bucket* α *T* b'›
have $b' \leq \alpha$ (*Max* (*set T*))
by (*metis* (*mono-tags*, *lifting*) *Max-greD assms*(*8*) *bucket-def mem-Collect-eq strict-mono-less-eq*)
with ‹$i \leq$ *bucket-start* α *T* b'› *s-bucket-ptr-lower-bound*[*OF assms*(*2*), *of* b']
have $i \leq$ *B* ! b'
by (*metis nat-le-iff-add s-bucket-start-def trans-le-add1*)
hence $i =$ *B* ! $b' \vee i <$ *B* ! b'
using *antisym-conv1* **by** *blast*
hence *B* ! $b' =$ *s-bucket-start* α *T* b'
proof
assume $i =$ *B* ! b'
with ‹$i \leq$ *bucket-start* α *T* b'› *s-bucket-ptr-lower-bound*[*OF assms*(*2*) ‹$b' \leq$ -›]
show *?thesis*
by (*simp add*: *s-bucket-start-def*)
next
assume $i <$ *B* ! b'
with *s-B-val*[*OF assms*(*1*−*6*,*8*,*10*−*13*,*15*) ‹$b' \leq$ -›]
show *?thesis* **.**
qed

from ‹*Suc* $x \in$ *bucket* α *T* b'›
have *Suc* $x \in$ *l-bucket* α *T* $b' \vee$ *Suc* $x \in$ *s-bucket* α *T* b'
by (*simp add*: *l-un-s-bucket*)
moreover
have *Suc* $x \in$ *l-bucket* α *T* $b' \Longrightarrow$ *?thesis*

proof −
assume *Suc x* $\in$ *l-bucket* α *T* b'
with *l-types-initD(1)*[*OF l-types-init-maintained*[*OF assms(2,4,10−12)*]
‹$b' \leq$ -›]
have *Suc x* $\in$ *set* (*list-slice SA* (*bucket-start* α *T* b') (*l-bucket-end* α *T*
b'))
by *simp*
with *nth-mem-list-slice*[*of Suc x SA bucket-start* α *T* b' *l-bucket-end* α
T b']
obtain i' **where**
$i' <$ *length SA*
bucket-start α *T* $b' \leq i'$
$i' <$ *l-bucket-end* α *T* b'
SA ! i' = *Suc x*
by *blast*

have $i \neq i'$
proof (*rule ccontr*)
assume $\neg$ $i \neq i'$
hence $i = i'$
by *auto*
with ‹*SA* ! i' = *Suc x*› *assms(16,18)*
show *False*
by *simp*
qed
hence $i < i'$
using ‹*bucket-start* α *T* $b' \leq i'$› ‹$i \leq$ *bucket-start* α *T* b'› **by** *auto*
with *s-suc-invD*[*OF assms(7)* ‹$i' <$ *length* -› - ‹*SA* ! i' = -› ‹*suffix-type*
T x = -›,
simplified ‹α (*T* ! *x*) = *b*›]
obtain *k* **where**
in-s-current-bucket α *T B b k*
SA ! *k* = *x*
$k < i'$
by *blast*
hence $x \in$ *?B*
by (*meson assms(11) in-s-current-bucket-list-slice*)
with ‹$x \notin$ *?B*›
have *False*
by *blast*
then show *?thesis*
by *blast*
qed
moreover
have *Suc x* $\in$ *s-bucket* α *T* $b' \Longrightarrow$ *?thesis*
proof −
assume *Suc x* $\in$ *s-bucket* α *T* b'

let *?ys* = *list-slice SA* (*s-bucket-start* α *T* b') (*bucket-end* α *T* b')

from *distinct-card*[*OF s-distinct-invD*[*OF assms*(*1*) ‹$b' \leq$ -›],
simplified ‹*B* ! b' = *s-bucket-start* - - -›]
have *card* (*set ?ys*) = *card* (*s-bucket* α *T* b')
by (*metis add-diff-cancel-left′ assms*(*11*) *bucket-end-eq-s-start-pl-size bucket-end-le-length length-list-slice min-def s-bucket-size-def*)
with *card-subset-eq*[
OF finite-s-bucket s-locations-inv-subset-s-bucket[*OF assms*(*3*) ‹$b' \leq$
-›],
simplified ‹*B* ! b' = *s-bucket-start* α *T* b'›]
have *set ?ys* = *s-bucket* α *T* b'
by *blast*
with ‹*Suc x* ∈ *s-bucket* α *T* b'›
have *Suc x* ∈ *set ?ys*
by *simp*
with *nth-mem-list-slice*[*of Suc x*]
obtain i' **where**
i' < *length SA*
s-bucket-start α *T* $b' \leq i'$
i' < *bucket-end* α *T* b'
SA ! i' = *Suc x*
by *blast*

from ‹*SA* ! i' = *Suc x*› *assms*(*16*,*18*)
have $i \neq i'$
using *nat.discI* **by** *blast*
hence $i < i'$
using ‹*B* ! b' = *s-bucket-start* α *T* b'› ‹$i \leq$ *B* ! b'› ‹*s-bucket-start* α *T*
$b' \leq i'$›
by *linarith*
with *s-suc-invD*[*OF assms*(*7*) ‹i' < *length* -› - ‹*SA* ! i' = -› ‹*suffix-type*
T x = -›,
simplified ‹α (*T* ! *x*) = *b*›]
obtain *k* **where**
in-s-current-bucket α *T B b k*
SA ! *k* = *x*
$k < i'$
by *blast*
hence *x* ∈ *?B*
by (*meson assms*(*11*) *in-s-current-bucket-list-slice*)
with ‹*x* ∉ *?B*›
have *False*
by *blast*
then show *?thesis*
by *blast*
qed
ultimately show *?thesis*
by *blast*
qed

ultimately show *?thesis*
  using ‹$Suc\ x \in ?B \lor (\exists b'.\ b < b' \land Suc\ x \in bucket\ \alpha\ T\ b')$› **by** *blast*
qed
ultimately show *?thesis*
  by *blast*
qed
ultimately show *?thesis*
  by *blast*
qed
moreover
have $Suc\ n \leq j \Longrightarrow ?g1 \land ?g2 \land ?g3$
proof −
  assume $Suc\ n \leq j$
  with *s-seen-invD*[*OF assms*(*5*) ‹$j < length\ SA$›] *assms*(*16*)
  show *?thesis*
    by *blast*
qed
ultimately show $?g1 \land ?g2 \land ?g3$
  by *blast*
qed

corollary *s-seen-inv-maintained-perm-step-c2*:
  assumes *s-perm-inv* α *T B SA0 SA i*
  and $i = Suc\ n$
  and $Suc\ n < length\ SA$
  and $SA\ !\ Suc\ n = 0$
shows *s-seen-inv* α *T B SA n*
  using *s-seen-inv-maintained-step-c2*[*OF s-perm-inv-elims*[*OF assms*(*1*)] *assms*(*2*−)]
  by *blast*

lemma *s-seen-inv-maintained-step-c3*:
  assumes *s-distinct-inv* α *T B SA*
  and *s-bucket-ptr-inv* α *T B*
  and *s-locations-inv* α *T B SA*
  and *s-unchanged-inv* α *T B SA0 SA*
  and *s-seen-inv* α *T B SA i*
  and *s-pred-inv* α *T B SA i*
  and *s-suc-inv* α *T B SA i*
  and *strict-mono* α
  and $\alpha\ (Max\ (set\ T)) < length\ B$
  and $length\ SA0 = length\ T$
  and $length\ SA = length\ T$
  and *l-types-init* α *T SA0*
  and *valid-list T*
  and $\alpha\ bot = 0$
  and $Suc\ 0 < length\ T$
  and $i = Suc\ n$
  and $Suc\ n < length\ SA$
  and $SA\ !\ Suc\ n = Suc\ j$

and *suffix-type T j = L-type*
shows *s-seen-inv α T B SA n*
unfolding *s-seen-inv-def*
proof (*intro allI impI*)
fix *k*
assume $k < length\ SA$ $n \leq k$
hence $n < length\ SA$
by *simp*
hence $n < length\ T$
by (*simp add*: *assms*(*11*))

let $?b = \alpha\ (T\ !\ (SA\ !\ k))$
let $?g1 = (suffix\text{-}type\ T\ (SA\ !\ k) = S\text{-}type \longrightarrow in\text{-}s\text{-}current\text{-}bucket\ \alpha\ T\ B\ ?b\ k)$
and $?g2 = (suffix\text{-}type\ T\ (SA\ !\ k) = L\text{-}type \longrightarrow in\text{-}l\text{-}bucket\ \alpha\ T\ ?b\ k)$
and $?g3 = SA\ !\ k < length\ T$

from ‹$n \leq k$›
have $n = k \vee Suc\ n \leq k$
by *linarith*
moreover
have $n = k \Longrightarrow ?g1 \wedge ?g2 \wedge ?g3$
proof −
assume $n = k$
with *index-in-bucket-interval-gen*[*OF* ‹$n < length\ T$› *assms*(*8*)]
obtain *b* **where**
$b \leq \alpha\ (Max\ (set\ T))$
$bucket\text{-}start\ \alpha\ T\ b \leq k$
$k < bucket\text{-}end\ \alpha\ T\ b$
by *blast*

have $k < l\text{-}bucket\text{-}end\ \alpha\ T\ b \vee s\text{-}bucket\text{-}start\ \alpha\ T\ b \leq k$
by (*metis not-le s-bucket-start-eq-l-bucket-end*)
moreover
have $k < l\text{-}bucket\text{-}end\ \alpha\ T\ b \Longrightarrow ?thesis$
proof −
assume $k < l\text{-}bucket\text{-}end\ \alpha\ T\ b$
with *l-types-init-nth*[*OF assms*(*11*) *l-types-init-maintained*[*OF assms*(*2*,*4*,*10*−*12*)]
‹$b \leq$ -› ‹*bucket-start* - - - $\leq$ -›]
have $SA\ !\ k \in l\text{-}bucket\ \alpha\ T\ b$ **.**
hence $SA\ !\ k < length\ T$ $suffix\text{-}type\ T\ (SA\ !\ k) = L\text{-}type$
by (*simp add*: *l-bucket-def bucket-def*)+
moreover
from *calculation*(*2*)
have *?g1*
by *simp*
moreover
from ‹$SA\ !\ k \in l\text{-}bucket\ \alpha\ T\ b$›
have $b = (\alpha\ (T\ !\ (SA\ !\ k)))$
by (*simp add*: *l-bucket-def bucket-def*)

with ‹$b \le$ -› ‹*bucket-start* - - - $\le$ -› ‹$k <$ *l-bucket-end* - - -›
have *in-l-bucket* α T (α (T ! (*SA* ! k))) k
using *in-l-bucket-def* **by** *blast*
ultimately show *?thesis*
by *blast*
qed
moreover
have *s-bucket-start* α T $b \le k \Longrightarrow$ *?thesis*
proof −
assume *s-bucket-start* α T $b \le k$
hence *s-bucket-start* α T $b < i$
by (*simp add*: ‹$n = k$› *assms*(*16*))

have B ! $b \le i$
proof(*rule ccontr*)
assume $\neg B$! $b \le i$
hence $i < B$! b
by *simp*
with *s-B-val*[*OF assms*(*1*−*6*,*8*,*10*−*13*,*15*) ‹$b \le \alpha$ -›]
have B ! b = *s-bucket-start* α T b **.**
with ‹*s-bucket-start* α T $b < i$› ‹$i < B$! b›
show *False*
by *linarith*
qed
hence $i = B$! $b \lor B$! $b < i$
by *linarith*
moreover
have B ! $b < i \Longrightarrow$ *?thesis*
proof −
assume B ! $b < i$
hence B ! $b \le k$
by (*simp add*: ‹$n = k$› *assms*(*16*))
with *s-locations-invD*[*OF assms*(*3*) ‹$b \le$ -› - ‹$k <$ *bucket-end* - - -›]
have *SA* ! $k \in$ *s-bucket* α T b **.**
hence *SA* ! $k <$ *length* T *suffix-type* T (*SA* ! k) = *S-type*
by (*simp add*: *s-bucket-def bucket-def*)+
moreover
from *calculation*(*2*)
have *?g2*
by *simp*
moreover
from ‹*SA* ! $k \in$ *s-bucket* α T b›
have α (T ! (*SA* ! k)) = b
by (*simp add*: *s-bucket-def bucket-def*)
with ‹B ! $b \le k$› ‹$k <$ *bucket-end* α T b› ‹$b \le$ -›
have *in-s-current-bucket* α T B (α (T ! (*SA* ! k))) k
by (*simp add*: *in-s-current-bucket-def*)
ultimately show *?thesis*
by *blast*

qed
moreover
have $i = B\ !\ b \Longrightarrow$ *?thesis*
proof −
assume $i = B\ !\ b$
hence $k < B\ !\ b$
using ‹$n = k$› *assms*(*16*) **by** *linarith*

have *s-bucket-start* α *T b* $<$ $B\ !\ b$
using ‹$i = B\ !\ b$› ‹*s-bucket-start* α *T b* $< i$› **by** *blast*

have $b \neq 0$
by (*metis* ‹$k < B\ !\ b$› *assms*(*2*) *not-less-zero s-bucket-ptr-0*)

let *?xs* = *list-slice SA* ($B\ !\ b$) (*bucket-end* α *T b*)
let *?B* = *set ?xs*
let *?A* = *s-bucket* α *T b* − *?B*

from *s-locations-inv-subset-s-bucket*[*OF assms*(*3*) ‹$b \leq$ -›]
have *?B* $\subseteq$ *s-bucket* α *T b* **.**
hence *?A* $\subseteq$ *s-bucket* α *T b*
by *blast*

have *card* (*s-bucket* α *T b*) = *bucket-end* α *T b* − *s-bucket-start* α *T b*
by (*simp add*: *bucket-end-eq-s-start-pl-size s-bucket-size-def*)

from *s-distinct-invD*[*OF assms*(*1*) ‹$b \leq$ -›]
have *card ?B* = *bucket-end* α *T b* − $B\ !\ b$
by (*metis assms*(*11*) *bucket-end-le-length distinct-card length-list-slice min.absorb-iff1*)
hence *card ?B* $<$ *card* (*s-bucket* α *T b*)
using ‹*card* (*s-bucket* α *T b*) = *bucket-end* α *T b* − *s-bucket-start* α *T b*›
‹$k <$ *bucket-end* α *T b*› ‹*s-bucket-start* α *T b* $< B\ !\ b$› ‹*s-bucket-start* α *T b* $\leq k$›
by *linarith*
with *card-psubset*[*OF finite-s-bucket* ‹*?B* $\subseteq$ *s-bucket* α *T b*›]
have *?B* $\subset$ *s-bucket* α *T b* **.**
hence *?A* $\neq$ {}
by *blast*
with *subset-s-bucket-successor*[*OF assms*(*13*,*8*,*14*) ‹$b \neq 0$› ‹*?A* $\subseteq$ *s-bucket* α *T b*›]
obtain x **where**
$x \in$ *?A*
Suc x $\in$ *s-bucket* α *T b* − *?A* $\vee$ ($\exists\, b'.\ b < b' \wedge$ *Suc x* $\in$ *bucket* α *T* b')
by *blast*
hence *Suc x* $\in$ *?B* $\vee$ ($\exists\, b'.\ b < b' \wedge$ *Suc x* $\in$ *bucket* α *T* b')
by *blast*

from ‹$x \in$ *?A*› ‹*?A* $\subseteq$ *s-bucket* α *T b*›

have *suffix-type* $T\ x = S\text{-}type\ \alpha\ (T\ !\ x) = b$
  by (*simp add*: *s-bucket-def bucket-def*)+

have $x \notin ?B$
  using ‹$x \in ?A$› **by** *blast*

from ‹$Suc\ x \in ?B \lor (\exists b'.\ b < b' \land Suc\ x \in bucket\ \alpha\ T\ b')$›
have *False*
proof
  assume $Suc\ x \in ?B$
  from *nth-mem-list-slice*[*OF* ‹$Suc\ x \in ?B$›]
  obtain i' **where**
    $i' < length\ SA$
    $B\ !\ b \le i'$
    $i' < bucket\text{-}end\ \alpha\ T\ b$
    $SA\ !\ i' = Suc\ x$
    by *blast*

  have $i \neq i'$
  proof (*rule ccontr*)
    assume $\neg\ i \neq i'$
    hence $i = i'$
      by *auto*
    hence $j = x$
      using ‹$SA\ !\ i' = Suc\ x$› *assms*(*16*,*18*) **by** *auto*
    with *assms*(*19*) ‹*suffix-type* $T\ x$ = -›
    show *False*
      by *simp*
  qed
  with ‹$B\ !\ b \le i'$› ‹$i = B\ !\ b$›
  have $i < i'$
    using *nat-less-le* **by** *blast*
  with *s-suc-invD*[*OF assms*(*7*) ‹$i' < length\ SA$ › - ‹$SA\ !\ i'$ = -› ‹*suffix-type*
$T\ x$ = -›]
      ‹$\alpha\ (T\ !\ x) = b$›
  obtain k **where**
    in-s-current-bucket $\alpha\ T\ B\ b\ k$
    $SA\ !\ k = x$
    $k < i'$
    by *blast*
  hence $x \in ?B$
    by (*meson assms*(*11*) *in-s-current-bucket-list-slice*)
  with ‹$x \notin ?B$›
  show *False*
    by *blast*
next
  assume $\exists b'.\ b < b' \land Suc\ x \in bucket\ \alpha\ T\ b'$
  then obtain b' **where**
    $b < b'$

$Suc\ x \in bucket\ \alpha\ T\ b'$
by *blast*
hence $b' \leq \alpha\ (Max\ (set\ T))$
by (*metis* (*mono-tags*, *lifting*) *Max-greD assms*(*8*) *bucket-def mem-Collect-eq strict-mono-less-eq*)

have $suffix\text{-}type\ T\ (Suc\ x) = S\text{-}type \vee suffix\text{-}type\ T\ (Suc\ x) = L\text{-}type$
by (*simp add*: *suffix-type-def*)
hence $Suc\ x \in l\text{-}bucket\ \alpha\ T\ b' \vee Suc\ x \in s\text{-}bucket\ \alpha\ T\ b'$
using ‹$Suc\ x \in bucket\ \alpha\ T\ b'$› *l-bucket-def s-bucket-def* **by** *fastforce*
moreover
have $Suc\ x \in l\text{-}bucket\ \alpha\ T\ b' \Longrightarrow False$
proof −
assume $Suc\ x \in l\text{-}bucket\ \alpha\ T\ b'$
with *l-types-initD*(*1*)[*OF l-types-init-maintained*[*OF assms*(*2*,*4*,*10*−*12*)]
‹$b' \leq$ -›]
have $Suc\ x \in set\ (list\text{-}slice\ SA\ (bucket\text{-}start\ \alpha\ T\ b')\ (l\text{-}bucket\text{-}end\ \alpha\ T\ b'))$
by *blast*
with *nth-mem-list-slice*[*of Suc x*]
obtain i' **where**
$i' < length\ SA$
$bucket\text{-}start\ \alpha\ T\ b' \leq i'$
$i' < l\text{-}bucket\text{-}end\ \alpha\ T\ b'$
$SA\ !\ i' = Suc\ x$
by *blast*

have $i \neq i'$
proof (*rule ccontr*)
assume $\neg\ i \neq i'$
hence $i = i'$
by *auto*
hence $j = x$
using ‹$SA\ !\ i' = Suc\ x$› *assms*(*16*,*18*) **by** *auto*
with *assms*(*19*) ‹$suffix\text{-}type\ T\ x =$ -›
show *False*
by *simp*
qed
moreover
from ‹$b < b'$›
have $bucket\text{-}end\ \alpha\ T\ b \leq bucket\text{-}start\ \alpha\ T\ b'$
by (*simp add*: *less-bucket-end-le-start*)
hence $B\ !\ b \leq i'$
using *s-bucket-ptr-upper-bound*[*OF assms*(*2*) ‹$b \leq \alpha\ (Max\ (set\ T))$›]
‹$bucket\text{-}start\ \alpha\ T\ b' \leq i'$›
by *linarith*
ultimately have $i < i'$
using ‹$i = B\ !\ b$› *nat-less-le* **by** *blast*
with *s-suc-invD*[*OF assms*(*7*) ‹$i' < length\ SA$ › - ‹$SA\ !\ i' =$ -› ‹*suffix-type*

```
T x = -›]
          ‹α (T ! x) = b›
      obtain k where
        in-s-current-bucket α T B b k
        SA ! k = x
        k < i′
        by blast
      hence x ∈ ?B
        by (meson assms(11) in-s-current-bucket-list-slice)
      with ‹x ∉ ?B›
      show False
        by blast
    qed
    moreover
    have Suc x ∈ s-bucket α T b′ ⟹ False
    proof −
      assume Suc x ∈ s-bucket α T b′

      have i ≤ bucket-end α T b
        by (simp add: Suc-le-eq ‹k < bucket-end α T b› ‹n = k› assms(16))
      hence i ≤ bucket-start α T b′
        using ‹b < b′› less-bucket-end-le-start order.trans by blast
      hence i ≤ B ! b′
        using s-bucket-ptr-lower-bound[OF assms(2) ‹b′ ≤ -›]
    by (metis l-bucket-end-def le-trans nat-le-iff-add s-bucket-start-eq-l-bucket-end)
      hence i < B ! b′ ∨ i = B ! b′
        using nat-less-le by blast
      hence B ! b′ = s-bucket-start α T b′
      proof
        assume i < B ! b′
        with s-B-val[OF assms(1−6,8,10−13,15) ‹b′ ≤ -›]
        show B ! b′ = s-bucket-start α T b′
          by blast
      next
        assume i = B ! b′
        with s-bucket-ptr-lower-bound[OF assms(2) ‹b′ ≤ -›]
             ‹i ≤ bucket-start α T b′›
        show B ! b′ = s-bucket-start α T b′
          by (simp add: s-bucket-start-def)
      qed

      let ?ys = list-slice SA (s-bucket-start α T b′) (bucket-end α T b′)

      from distinct-card[OF s-distinct-invD[OF assms(1) ‹b′ ≤ -›],
           simplified ‹B ! b′ = s-bucket-start - - -›]
      have card (set ?ys) = card (s-bucket α T b′)
        by (metis add-diff-cancel-left′ assms(11) bucket-end-eq-s-start-pl-size
             bucket-end-le-length length-list-slice min-def s-bucket-size-def)
      with card-subset-eq[
```

```
          OF finite-s-bucket s-locations-inv-subset-s-bucket[OF assms(3) ‹b′ ≤
-›],
           simplified ‹B ! b′ = s-bucket-start α T b′›]
      have set ?ys = s-bucket α T b′
        by blast
      with ‹Suc x ∈ s-bucket α T b′›
      have Suc x ∈ set ?ys
        by simp
      with nth-mem-list-slice[of Suc x]
      obtain i′ where
        i′ < length SA
        s-bucket-start α T b′ ≤ i′
        i′ < bucket-end α T b′
        SA ! i′ = Suc x
        by blast

      have i ≠ i′
      proof (rule ccontr)
        assume ¬ i ≠ i′
        hence i = i′
          by auto
        hence j = x
          using ‹SA ! i′ = Suc x› assms(16,18) by auto
        with assms(19) ‹suffix-type T x = -›
        show False
          by simp
      qed
      moreover
      have i ≤ i′
        using ‹B ! b′ = s-bucket-start α T b′› ‹i ≤ B ! b′› ‹s-bucket-start α T
b′ ≤ i′›
        by linarith
      ultimately have i < i′
        using dual-order.order-iff-strict by blast
     with s-suc-invD[OF assms(7) ‹i′ < length SA › - ‹SA ! i′ = -› ‹suffix-type
T x = -›]
          ‹α (T ! x) = b›
      obtain k where
        in-s-current-bucket α T B b k
        SA ! k = x
        k < i′
        by blast
      hence x ∈ ?B
        by (meson assms(11) in-s-current-bucket-list-slice)
      with ‹x ∉ ?B›
      show False
        by blast
    qed
    ultimately show False
```

by *blast*
qed
then show *?thesis*
by *blast*
qed
ultimately show *?thesis*
by *blast*
qed
ultimately show *?thesis*
by *blast*
qed
moreover
have $Suc\ n \leq k \Longrightarrow ?g1 \wedge ?g2 \wedge ?g3$
proof −
assume $Suc\ n \leq k$
with *s-seen-invD*[*OF assms*(*5*) ‹*k* < *length SA*›] *assms*(*16*)
show *?thesis*
by *blast*
qed
ultimately show $?g1 \wedge ?g2 \wedge ?g3$
by *blast*
qed

corollary *s-seen-inv-maintained-perm-step-c3*:
assumes *s-perm-inv* α *T B SA0 SA i*
and $i = Suc\ n$
and $Suc\ n < length\ SA$
and $SA\ !\ Suc\ n = Suc\ j$
and *suffix-type T j* = *L-type*
shows *s-seen-inv* α *T B SA n*
using *s-seen-inv-maintained-step-c3*[*OF s-perm-inv-elims*[*OF assms*(*1*)] *assms*(*2*−)]
by *blast*

lemma *s-seen-inv-maintained-step-c4*:
assumes *s-distinct-inv* α *T B SA*
and *s-bucket-ptr-inv* α *T B*
and *s-locations-inv* α *T B SA*
and *s-unchanged-inv* α *T B SA0 SA*
and *s-seen-inv* α *T B SA i*
and *s-pred-inv* α *T B SA i*
and *s-suc-inv* α *T B SA i*
and *strict-mono* α
and $\alpha\ (Max\ (set\ T)) < length\ B$
and $length\ SA0 = length\ T$
and $length\ SA = length\ T$
and *l-types-init* α *T SA0*
and *valid-list T*
and $\alpha\ bot = 0$
and $Suc\ 0 < length\ T$

and $i = Suc\ n$
and $Suc\ n < length\ SA$
and $SA\ !\ Suc\ n = Suc\ j$
and $suffix\text{-}type\ T\ j = S\text{-}type$
and $b = \alpha\ (T\ !\ j)$
and $k = B\ !\ b - Suc\ 0$
shows $s\text{-}seen\text{-}inv\ \alpha\ T\ (B[b := k])\ (SA[k := j])\ n$
unfolding *s-seen-inv-def*
proof(*intro allI impI*)
fix i'
assume $i' < length\ (SA[k := j])\ n \leq i'$

let $?g1 = (suffix\text{-}type\ T\ (SA[k := j]\ !\ i') = S\text{-}type \longrightarrow$
$in\text{-}s\text{-}current\text{-}bucket\ \alpha\ T\ (B[b := k])\ (\alpha\ (T\ !\ (SA[k := j]\ !\ i')))\ i')$ **and**
$?g2 = (suffix\text{-}type\ T\ (SA[k := j]\ !\ i') = L\text{-}type \longrightarrow$
$in\text{-}l\text{-}bucket\ \alpha\ T\ (\alpha\ (T\ !\ (SA[k := j]\ !\ i')))\ i')$ **and**
$?g3 = SA[k := j]\ !\ i' < length\ T$

from *s-bucket-ptr-strict-lower-bound*[*OF assms*(*1*−*6*,*8*,*10*−*14*,*16*−*20*)]
have $s\text{-}bucket\text{-}start\ \alpha\ T\ b < B\ !\ b$**.**
hence $s\text{-}bucket\text{-}start\ \alpha\ T\ b \leq k$
using *assms*(*21*) **by** *linarith*
hence $bucket\text{-}start\ \alpha\ T\ b \leq k$
using *bucket-start-le-s-bucket-start le-trans* **by** *blast*

from ‹$s\text{-}bucket\text{-}start\ \alpha\ T\ b < B\ !\ b$›
have $k < B\ !\ b$
using *assms*(*21*) **by** *linarith*

have $j < length\ T$
by (*simp add*: *assms*(*19*) *suffix-type-s-bound*)
hence $b \leq \alpha\ (Max\ (set\ T))$
by (*simp add*: *assms*(*8*,*20*) *strict-mono-less-eq*)
with *s-bucket-ptr-upper-bound*[*OF assms*(*2*)]
have $B\ !\ b \leq bucket\text{-}end\ \alpha\ T\ b$
by *blast*
with ‹$k < B\ !\ b$›
have $k < bucket\text{-}end\ \alpha\ T\ b$
by *linarith*

have $B\ !\ b \leq i$
proof(*rule ccontr*)
assume $\neg B\ !\ b \leq i$
hence $i < B\ !\ b$
by *simp*
with *s-B-val*[*OF assms*(*1*−*6*,*8*,*10*−*13*,*15*) ‹$b \leq \alpha$ -›]
have $B\ !\ b = s\text{-}bucket\text{-}start\ \alpha\ T\ b$ **.**
with ‹$s\text{-}bucket\text{-}start\ \alpha\ T\ b < B\ !\ b$›
show *False*

    by *linarith*
qed
hence $k < i$
  using ‹$k < B ! b$› *less-le-trans* **by** *blast*

have $k = i' \Longrightarrow n = i'$
  using ‹$k < i$› ‹$n \leq i'$› *assms*(*16*) *le-less-Suc-eq* **by** *blast*

have $k \leq i'$
  using ‹$k < i$› ‹$n \leq i'$› *assms*(*16*) **by** *linarith*

have $i' <$ *length* T
  using ‹$i' <$ *length* (*SA*[$k := j$])› *assms*(*11*) **by** *auto*
with *index-in-bucket-interval-gen*[*OF* - *assms*(*8*), *of* i' *T*]
obtain b' **where**
  $b' \leq \alpha$ (*Max* (*set* *T*))
  bucket-start α *T* $b' \leq i'$
  $i' <$ *bucket-end* α *T* b'
  by *blast*
hence $n <$ *bucket-end* α *T* b'
  using ‹$n \leq i'$› *dual-order.strict-trans2* **by** *blast*
hence $i \leq$ *bucket-end* α *T* b'
  using *assms*(*16*) **by** *linarith*

have $b \leq b'$
proof (*rule ccontr*)
  assume $\neg b \leq b'$
  hence $b' < b$
    by *linarith*
  hence *bucket-end* α *T* $b' \leq$ *bucket-start* α *T* b
    by (*simp add*: *less-bucket-end-le-start*)
  with ‹$i \leq$ *bucket-end* α *T* b'› ‹*bucket-start* α *T* $b \leq k$› ‹$k < B ! b$›
  have $i < B ! b$
    by *linarith*
  with ‹$B ! b \leq i$›
  show *False*
    by *linarith*
qed

have *in-s-current-bucket* α *T* (*B*[$b := k$]) *b* *k*
  unfolding *in-s-current-bucket-def*
  using ‹$b \leq \alpha$ (*Max* (*set* *T*))› ‹$k <$ *bucket-end* α *T* b› *assms*(*9*) **by** *auto*

have $b < b' \Longrightarrow$ *?g1* $\wedge$ *?g2* $\wedge$ *?g3*
proof −
  assume $b < b'$
  hence *bucket-end* α *T* $b \leq$ *bucket-start* α *T* b'
    by (*simp add*: *less-bucket-end-le-start*)
  with ‹$k <$ *bucket-end* - - b› ‹*bucket-start* - - $b' \leq i'$›

have $k < i'$
by *linarith*
hence $SA[k := j] ! i' = SA ! i'$
by *simp*

from ‹$b < b'$›
have $B[b := k] ! b' = B ! b'$
by *simp*

have $i' <$ *l-bucket-end* α T b' $\vee$ $B ! b' \leq i'$ $\vee$ (*s-bucket-start* α T $b' \leq i' \wedge i' < B ! b'$)
by (*metis not-le s-bucket-start-eq-l-bucket-end*)
moreover
have $B ! b' \leq i' \Longrightarrow$ *?thesis*
proof –
assume $B ! b' \leq i'$
with *s-locations-invD*[*OF assms*(*3*) ‹$b' \leq$ -› - ‹$i' <$ *bucket-end* - - -›]
have $SA ! i' \in$ *s-bucket* α T b' **.**
hence *suffix-type* T $(SA ! i')$ = *S-type* α $(T ! (SA ! i')) = b'$ $SA ! i' <$ *length* T
by (*simp add*: *s-bucket-def bucket-def*)+
moreover
from ‹$B[b := k] ! b' = B ! b'$› ‹$b' \leq \alpha$ -› ‹$B ! b' \leq i'$› ‹$i' <$ *bucket-end* α T b'›
have *in-s-current-bucket* α T $(B[b := k])$ b' i'
by (*simp add*: *in-s-current-bucket-def*)
ultimately show *?thesis*
by (*simp add*: ‹$SA[k := j] ! i' = SA ! i'$›)
qed
moreover
have $i' <$ *l-bucket-end* α T b' $\Longrightarrow$ *?thesis*
proof –
assume $i' <$ *l-bucket-end* α T b'
hence *in-l-bucket* α T b' i'
by (*simp add*: ‹*bucket-start* α T $b' \leq i'$› ‹$b' \leq \alpha$ -› *in-l-bucket-def*)
moreover
from *l-types-init-nth*[*OF assms*(*11*) *l-types-init-maintained*[*OF assms*(*2*,*4*,*10*–*12*)]
‹$b' \leq \alpha$ -› ‹*bucket-start* - - - $\leq i'$› ‹$i' <$ *l-bucket-end* - - -›]
have $SA ! i' \in$ *l-bucket* α T b' **.**
hence $SA ! i' <$ *length* T α $(T ! (SA ! i')) = b'$ *suffix-type* T $(SA ! i')$ = *L-type*
by (*simp add*: *l-bucket-def bucket-def*)+
ultimately show *?thesis*
using ‹$SA[k := j] ! i' = SA ! i'$›
by *simp*
qed
moreover
have $[\![$*s-bucket-start* α T $b' \leq i'$; $i' < B ! b']\!] \Longrightarrow$ *?thesis*
proof –

assume *s-bucket-start* α T $b' \leq i'$ $i' < B$! b'
have B ! $b' = i$
proof (*rule ccontr*)
  assume B ! $b' \neq i$
  hence $i < B$! $b' \vee B$! $b' < i$
    by *linarith*
  moreover
  have B ! $b' < i \Longrightarrow$ *False*
    using ‹$i' < B$! b'› ‹$n \leq i'$› *assms*(*16*) **by** *linarith*
  moreover
  have $i < B$! $b' \Longrightarrow$ *False*
  proof –
    assume $i < B$! b'
    with *s-B-val*[*OF assms*(*1–6*,*8*,*10–13*,*15*) ‹$b' \leq \alpha$ -›]
    have B ! b' = *s-bucket-start* α T b' **.**
    with ‹*s-bucket-start* α T $b' \leq i'$› ‹$i' < B$! b'›
    show *False*
      by *linarith*
  qed
  ultimately show *False*
    by *linarith*
qed

have *s-bucket-start* α T $b' < B$! b'
  using ‹$i' < B$! b'› ‹*s-bucket-start* α T $b' \leq i'$› **by** *linarith*
hence *s-bucket-start* α T b' < *bucket-end* α T b'
  using ‹B ! $b' = i$› ‹$i \leq$ *bucket-end* α T b'› *order.strict-trans2* **by** *blast*
hence *s-bucket* α T $b' \neq \{\}$
by (*metis add.commute bucket-end-eq-s-start-pl-size distinct-card distinct-conv-nth empty-set less-irrefl-nat less-nat-zero-code list.size*(*3*) *plus-nat.add-0 s-bucket-size-def*)

have *bucket-end* α T $b' \leq$ *length SA*
  by (*simp add*: *assms*(*11*) *bucket-end-le-length*)

let *?xs* = *list-slice SA* (B ! b') (*bucket-end* α T b')

have *set ?xs* $\subset$ *s-bucket* α T b'
proof
  from *s-locations-inv-subset-s-bucket*[*OF assms*(*3*) ‹$b' \leq$ -›]
  show *set ?xs* $\subseteq$ *s-bucket* α T b' **.**
next
  from ‹*s-bucket-start* α T $b' < B$! b'› ‹*s-bucket-start* α T b' < *bucket-end* α T b'›
  have *bucket-end* α T $b' - B$! b' < *bucket-end* α T b' – *s-bucket-start* α T b'
    using *diff-less-mono2* **by** *blast*
  hence *length ?xs* < *s-bucket-size* α T b'
    by (*metis* ‹*bucket-end* α T $b' \leq$ *length SA*› *add-diff-cancel-left'*

bucket-end-eq-s-start-pl-size length-list-slice min-def)
hence *card* (*set ?xs*) $\neq$ *card* (*s-bucket* α *T* b')
by (*metis card-length not-le s-bucket-size-def*)
then show *set ?xs* $\neq$ *s-bucket* α *T* b'
by *auto*
qed

have *P0*: $\forall$ *i0* < *length T*. α (*T* ! *i0*) = b' $\longrightarrow$ *T* ! *i0* $\neq$ *bot*
using ‹*b* < b'› *assms*(*8*,*20*) *strict-mono-less* **by** *fastforce*
hence *P1*: $\forall$ *i0* < *length T*. α (*T* ! *i0*) = b' $\longrightarrow$ *Suc i0* < *length T*
by (*metis Suc-leI assms*(*13*) *diff-Suc-1 last-conv-nth le-imp-less-or-eq length-greater-0-conv*
valid-list-def)

let *?S* = *s-bucket* α *T* b' − *set ?xs*

from ‹*set ?xs* $\subset$ *s-bucket* α *T* b'›
have *?S* $\neq$ {}
by *blast*
have *?S* $\subseteq$ *s-bucket* α *T* b'
by *blast*
hence *P2*: $\forall$ *x* $\in$ *?S*. α (*T* ! *x*) = b' $\wedge$ *suffix-type T x* = *S-type* $\wedge$ *x* < *length T*
by (*simp add*: *bucket-def s-bucket-def*)

have *P3*: $\forall$ *x* $\in$ *?S*. *Suc x* < *length T* $\wedge$ α (*T* ! *Suc x*) $\geq$ b'
proof
fix *x*
assume *x* $\in$ *?S*
with *P2*
have α (*T* ! *x*) = b' *suffix-type T x* = *S-type x* < *length T*
by *blast+*
with *P1*
have *Suc x* < *length T*
by *blast*
moreover
from ‹*suffix-type T x* = *S-type*› ‹*x* < *length T*›
have *T* ! *x* $\leq$ *T* ! *Suc x*
using *calculation nth-gr-imp-l-type* **by** *fastforce*
hence α (*T* ! *Suc x*) $\geq$ b'
using ‹α (*T* ! *x*) = b'› *assms*(*8*) *strict-mono-leD* **by** *blast*
ultimately show *Suc x* < *length T* $\wedge$ α (*T* ! *Suc x*) $\geq$ b'
by *blast*
qed

have *finite ?S*
by (*simp add*: *finite-s-bucket*)

have $\exists$ *x* $\in$ *?S*. α (*T* ! *Suc x*) > b' $\vee$ *Suc x* $\in$ *set ?xs*

proof (*rule ccontr*)
  assume $\neg$ ($\exists x \in$ *?S*. $b' < \alpha$ (T ! *Suc x*) $\vee$ *Suc x* $\in$ *set ?xs*)
  hence $\forall x \in$ *?S*. α (T ! *Suc x*) $\leq b' \wedge$ *Suc x* $\notin$ *set ?xs*
    using *not-le-imp-less* **by** *blast*
  with *P3*
  have *P4*: $\forall x \in$ *?S*. α (T ! *Suc x*) $= b' \wedge$ *Suc x* $\notin$ *set ?xs*
    using *dual-order.antisym* **by** *blast*
  hence *P5*: $\forall x \in$ *?S*. *suffix-type T* (*Suc x*) = *S-type*
    by (*metis P2 P3 assms*(*8*) *strict-mono-eq suffix-type-neq*)
  hence *P6*: $\forall x \in$ *?S*. *Suc x* $\in$ *?S*
    by (*metis* (*mono-tags*, *lifting*) *Diff-iff P3 P4 bucket-def mem-Collect-eq s-bucket-def*)
  with ‹*?S* $\neq$ {}› ‹*finite ?S*›
  show *False*
    using *Suc-le-lessD infinite-growing* **by** *blast*
qed
then obtain x **where**
  $x \in$ *?S*
  α (T ! *Suc x*) $> b' \vee$ *Suc x* $\in$ *set ?xs*
  by *blast*
with *P3*
have *Suc x* < *length T*
  by *blast*

from ‹$x \in$ *?S*›
have *suffix-type T x* = *S-type* α (T ! x) $= b'$ x < *length T*
  using *P2* **by** *blast+*

have *P4*: $\forall$ *b0* $\leq \alpha$ (*Max* (*set T*)). $b' <$ *b0* $\longrightarrow$ *B* ! *b0* = *s-bucket-start* α *T b0*
proof(*safe*)
  fix *b0*
  assume *b0* $\leq \alpha$ (*Max* (*set T*)) $b' <$ *b0*
  hence *bucket-end* α *T* $b' \leq$ *bucket-start* α *T b0*
    by (*simp add*: *less-bucket-end-le-start*)
  with *s-bucket-ptr-upper-bound*[*OF assms*(*2*) ‹$b' \leq$ - ›]
      s-bucket-ptr-lower-bound[*OF assms*(*2*) ‹*b0* $\leq$ - ›]
  have *B* ! $b' \leq$ *B* ! *b0*
    by (*meson bucket-start-le-s-bucket-start le-trans*)
  hence *B* ! b' = *B* ! *b0* $\vee$ *B* ! b' < *B* ! *b0*
    by *linarith*
  moreover
  have *B* ! b' = *B* ! *b0* $\Longrightarrow$ *B* ! *b0* = *s-bucket-start* α *T b0*
    by (*metis* ‹*B* ! b' = *i*› ‹*bucket-end* α *T* $b' \leq$ *bucket-start* α *T b0*› *le-trans* ‹$i \leq$ *bucket-end* α *T* b'› ‹*s-bucket-start* α *T b0* $\leq$ *B* ! *b0*› *dual-order.antisym bucket-start-le-s-bucket-start*)
  moreover
  have *B* ! b' < *B* ! *b0* $\Longrightarrow$ *i* < *B* ! *b0*
    by (*simp add*: ‹*B* ! b' = *i*›)

with *s-B-val*[*OF assms*(*1−6*,*8*,*10−13*,*15*) ‹*b0* ≤ -›]
have *B* ! *b′* < *B* ! *b0* ⟹ *B* ! *b0* = *s-bucket-start α T b0*
by *blast*
ultimately show *B* ! *b0* = *s-bucket-start α T b0*
by *blast*
qed

from ‹*α* (*T* ! *Suc x*) > *b′* ∨ *Suc x* ∈ *set ?xs*›
show *?thesis*
proof
let *?b* = *α* (*T* ! *Suc x*)
let *?ys* = *list-slice SA* (*bucket-start α T ?b*) (*l-bucket-end α T ?b*)
and *?zs* = *list-slice SA* (*s-bucket-start α T ?b*) (*bucket-end α T ?b*)

assume *b′* < *?b*
with *P4* ‹*Suc x* < *length T*›
have *B* ! *?b* = *s-bucket-start α T ?b*
by (*simp add*: *assms*(*8*) *strict-mono-less-eq*)

from ‹*Suc x* < *length T*›
have *?b* ≤ *α* (*Max* (*set T*))
by (*simp add*: *assms*(*8*) *strict-mono-leD*)

have *bucket-end α T b′* ≤ *bucket-start α T ?b*
using ‹*b′* < *α* (*T* ! *Suc x*)› *less-bucket-end-le-start* **by** *blast*
hence *i* ≤ *bucket-start α T ?b*
using ‹*i* ≤ *bucket-end α T b′*› *order.trans* **by** *blast*

have *set ?zs* = *s-bucket α T ?b*
proof (*rule card-subset-eq*[*OF finite-s-bucket*])
show *set ?zs* ⊆ *s-bucket α T ?b*
by (*metis Max-greD* ‹*B* ! *?b* = *s-bucket-start α T ?b*› ‹*Suc x* < *length*
T› *assms*(*3*,*8*)
s-locations-inv-subset-s-bucket strict-mono-leD)
next
from *distinct-card*[*OF s-distinct-invD*[*OF assms*(*1*) ‹*?b* ≤ -›]]
‹*B* ! *?b* = *s-bucket-start α T ?b*›
have *card* (*set ?zs*) = *length ?zs*
by *simp*
moreover
have *length ?zs* = *bucket-end α T ?b* − *s-bucket-start α T ?b*
by (*metis assms*(*11*) *bucket-end-le-length length-list-slice min-def*)
moreover
have *s-bucket-size α T ?b* = *bucket-end α T ?b* − *s-bucket-start α T ?b*
by (*simp add*: *bucket-end-eq-s-start-pl-size*)
hence *card* (*s-bucket α T ?b*) = *bucket-end α T ?b* − *s-bucket-start α T*
?b
by (*simp add*: *s-bucket-size-def*)
ultimately show *card* (*set ?zs*) = *card* (*s-bucket α T ?b*)

by *simp*
qed

have *suffix-type T (Suc x) = L-type* $\Longrightarrow$ *?thesis*
proof −
assume *suffix-type T (Suc x) = L-type*
with *l-types-initD(1)[OF l-types-init-maintained[OF assms(2,4,10−12)]*
‹*?b* ≤ -›]
have *Suc x* ∈ *set ?ys*
by (*simp add*: ‹*Suc x* < *length T*› *bucket-def l-bucket-def*)

from *nth-mem-list-slice[OF* ‹*Suc x* ∈ *set ?ys*›]
obtain *i0* **where**
i0 < *length SA*
bucket-start α *T ?b* ≤ *i0*
i0 < *l-bucket-end* α *T ?b*
SA ! *i0* = *Suc x*
by *blast*
hence *i* ≤ *i0*
using ‹*i* ≤ *bucket-start* α *T ?b*› *dual-order.trans* **by** *blast*
hence *i* = *i0* ∨ *i* < *i0*
by *linarith*
then show *?thesis*
proof
assume *i* = *i0*
hence *x* = *j*
using ‹*SA* ! *i0* = *Suc x*› *assms(16,18)* **by** *auto*
then show *?thesis*
using ‹α (*T* ! *x*) = *b*′› ‹*b* < *b*′› *assms(20)* **by** *blast*
next
assume *i* < *i0*
with *s-suc-invD[OF assms(7)* ‹*i0* < *length* -› - ‹*SA* ! *i0* = -› ‹*suffix-type*
T x = *S-type*›]
‹α (*T* ! *x*) = *b*′›
obtain *i1* **where**
in-s-current-bucket α *T B b*′ *i1*
SA ! *i1* = *x*
i1 < *i0*
by *auto*
with *in-s-current-bucket-list-slice[OF assms(11)]*
have *x* ∈ *set ?xs*
by *blast*
then show *?thesis*
using ‹*x* ∈ *?S*› **by** *blast*
qed
qed
moreover
have *suffix-type T (Suc x) = S-type* $\Longrightarrow$ *?thesis*
proof −

assume *suffix-type T (Suc x) = S-type*
with ‹*set ?zs = s-bucket α T ?b*› ‹*Suc x < length T*›
have *Suc x ∈ set ?zs*
by (*simp add: s-bucket-def bucket-def*)

from *nth-mem-list-slice*[*OF* ‹*Suc x ∈ set ?zs*›]
obtain *i0* **where**
i0 < length SA
s-bucket-start α T ?b ≤ i0
i0 < bucket-end α T ?b
SA ! i0 = Suc x
by *blast*
hence *i ≤ i0*
by (*meson* ‹*i ≤ bucket-start α T ?b*› *bucket-start-le-s-bucket-start dual-order.trans*)
hence *i = i0 ∨ i < i0*
by *linarith*
then show *?thesis*
proof
assume *i = i0*
hence *x = j*
using ‹*SA ! i0 = Suc x*› *assms*(*16*,*18*) **by** *auto*
then show *?thesis*
using ‹*α (T ! x) = b′*› ‹*b < b′*› *assms*(*20*) **by** *blast*
next
assume *i < i0*
with *s-suc-invD*[*OF assms*(*7*) ‹*i0 < length -*› - ‹*SA ! i0 = -*› ‹*suffix-type T x = S-type*›]
‹*α (T ! x) = b′*›
obtain *i1* **where**
in-s-current-bucket α T B b′ i1
SA ! i1 = x
i1 < i0
by *auto*
with *in-s-current-bucket-list-slice*[*OF assms*(*11*)]
have *x ∈ set ?xs*
by *blast*
then show *?thesis*
using ‹*x ∈ ?S*› **by** *blast*
qed
qed
ultimately show *?thesis*
using *SL-types.exhaust* **by** *blast*
next
assume *Suc x ∈ set ?xs*

from *nth-mem-list-slice*[*OF* ‹*Suc x ∈ set ?xs*›]
obtain *i0* **where**
i0 < length SA

```
        B ! b' ≤ i0
        i0 < bucket-end α T b'
        SA ! i0 = Suc x
        by blast
      with ‹B ! b' = i›
      have i ≤ i0
        by blast
      hence i = i0 ∨ i < i0
        by linarith
      then show ?thesis
      proof
        assume i = i0
        hence x = j
          using ‹SA ! i0 = Suc x› assms(16,18) by auto
        then show ?thesis
          using ‹α (T ! x) = b'› ‹b < b'›  assms(20) by blast
      next
        assume i < i0
        with s-suc-invD[OF assms(7) ‹i0 < length -› - ‹SA ! i0 = -› ‹suffix-type
T x = -›]
            ‹α (T ! x) = b'›
        obtain i1 where
          in-s-current-bucket α T B b' i1
          SA ! i1 = x
          i1 < i0
          by blast
        with in-s-current-bucket-list-slice[OF assms(11)]
        have x ∈ set ?xs
          by blast
        then show ?thesis
          using ‹x ∈ ?S› by blast
      qed
    qed
  qed
  ultimately show ?thesis
    by linarith
qed
moreover
have b = b' ⟹ ?g1 ∧ ?g2 ∧ ?g3
proof -
  assume b = b'
  have k = i' ⟹ ?thesis
  proof -
    assume k = i'
    hence SA[k := j] ! i' = j
      using ‹i' < length (SA[k := j])› by auto
    with ‹suffix-type T j = S-type› ‹j < length T› ‹in-s-current-bucket α T (B[b
:= k]) b k›
          assms(20) ‹k = i'›
```

show *?thesis*
by *simp*
qed
moreover
have $k < i' \Longrightarrow$ *?thesis*
proof −
assume $k < i'$
hence $B ! b \leq i'$
using *assms*(*21*) **by** *linarith*
with *s-locations-invD*[*OF assms*(*3*) ‹$b' \leq \alpha$ -› - ‹$i' <$ *bucket-end* - - -›] ‹$b = b'$›
have $SA ! i' \in$ *s-bucket* α T b'
by *blast*
hence *suffix-type* T $(SA ! i')$ = *S-type* α $(T ! (SA ! i')) = b'$
by (*simp add*: *s-bucket-def bucket-def*)+
moreover
have $SA[k := j] ! i' = SA ! i'$
using ‹$k < i'$› **by** *simp*
moreover
have *in-s-current-bucket* α T $(B[b := k])$ b' i'
by (*metis* (*no-types*, *lifting*) ‹$b = b'$› ‹$b' \leq \alpha$ (*Max* (*set* T))› ‹$i' <$ *bucket-end* α T b'›
‹$k \leq i'$› *assms*(*9*) *dual-order.strict-trans2 in-s-current-bucket-def nth-list-update-eq*)
ultimately show *?thesis*
by (*simp add*: *suffix-type-s-bound*)
qed
ultimately show *?thesis*
using ‹$k \leq i'$› *dual-order.order-iff-strict* **by** *blast*
qed
ultimately show *?g1* $\wedge$ *?g2* $\wedge$ *?g3*
using ‹$b \leq b'$› *dual-order.order-iff-strict* **by** *blast*
qed

corollary *s-seen-inv-maintained-perm-step-c4*:
assumes *s-perm-inv* α T B *SA0* SA i
and i = *Suc* n
and *Suc* $n <$ *length* SA
and SA ! *Suc* n = *Suc* j
and *suffix-type* T j = *S-type*
and $b = \alpha$ $(T ! j)$
and $k = B ! b -$ *Suc 0*
shows *s-seen-inv* α T $(B[b := k])$ $(SA[k := j])$ n
using *s-seen-inv-maintained-step-c4*[*OF s-perm-inv-elims*[*OF assms*(*1*)] *assms*(*2*−)]
by *blast*

lemmas *s-seen-inv-maintained-perm-step* =
s-seen-inv-maintained-perm-step-c1
s-seen-inv-maintained-perm-step-c2

s-seen-inv-maintained-perm-step-c3
s-seen-inv-maintained-perm-step-c4

83.3.6 Predecessor

lemma *s-pred-inv-established*:
assumes *s-bucket-init* α *T B*
shows *s-pred-inv* α *T B SA n*
unfolding *s-pred-inv-def*
proof (*safe*)
fix *b i*
assume *A*: *in-s-current-bucket* α *T B b i* $0 < b$

let *?goal* = $\exists j$<*length SA*. *SA* ! *j* = *Suc* (*SA* ! *i*) $\wedge$ $i < j \wedge n < j$

have $b = 0 \vee 0 < b$
by *blast*
moreover
from *A*(*2*)
have $b = 0 \Longrightarrow$ *?goal*
by *blast*
moreover
have $0 < b \Longrightarrow$ *?goal*
proof −
assume $0 < b$
with *s-bucket-initD*(*1*)[*OF assms*(*1*) *in-s-current-bucketD*(*1*)[*OF A*(*1*)]]
have *B* ! *b* = *bucket-end* α *T b* **.**
with *in-s-current-bucketD*(*2*,*3*)[*OF A*(*1*)]
show *?goal*
by *linarith*
qed
ultimately show *?goal*
by *blast*
qed

lemma *s-pred-inv-maintained-step-alt*:
assumes *s-pred-inv* α *T B SA i*
and $i = Suc\ n$
shows *s-pred-inv* α *T B SA n*
unfolding *s-pred-inv-def*
proof (*intro allI impI*; *elim conjE*)
fix *b i'*
assume *in-s-current-bucket* α *T B b i'* $b \neq 0$
with *s-pred-invD*[*OF assms*(*1*), *of b i'*] *assms*(*2*)
show $\exists j$<*length SA*. *SA* ! *j* = *Suc* (*SA* ! *i'*) $\wedge$ $i' < j \wedge n < j$
using *Suc-lessD* **by** *blast*
qed

corollary *s-pred-inv-maintained-perm-step-alt*:

assumes *s-perm-inv* α *T B SA0 SA i*
and $i = Suc\ n$
shows *s-pred-inv* α *T B SA n*
using *s-pred-inv-maintained-step-alt*[*OF s-perm-inv-elims*(*6*), *OF assms*]
by *blast*

lemma *s-pred-inv-maintained-step*:
assumes *s-distinct-inv* α *T B SA*
and *s-bucket-ptr-inv* α *T B*
and *s-locations-inv* α *T B SA*
and *s-unchanged-inv* α *T B SA0 SA*
and *s-seen-inv* α *T B SA i*
and *s-pred-inv* α *T B SA i*
and *s-suc-inv* α *T B SA i*
and *strict-mono* α
and $\alpha\ (Max\ (set\ T)) < length\ B$
and $length\ SA0 = length\ T$
and $length\ SA = length\ T$
and *l-types-init* α *T SA0*
and *valid-list T*
and $\alpha\ bot = 0$
and $Suc\ 0 < length\ T$
and $i = Suc\ n$
and $Suc\ n < length\ SA$
and $SA\ !\ Suc\ n = Suc\ j$
and *suffix-type T j = S-type*
and $b = \alpha\ (T\ !\ j)$
and $k = B\ !\ b - Suc\ 0$
shows *s-pred-inv* α $T\ (B[b := k])\ (SA[k := j])\ n$
unfolding *s-pred-inv-def*
proof(*safe*)
fix $b'\ i'$
assume *in-s-current-bucket* $\alpha\ T\ (B[b := k])\ b'\ i'\ 0 < b'$
hence $b' \neq 0$
by *linarith*

let $?goal = \exists j'{<}length\ (SA[k := j]).\ SA[k := j]\ !\ j' = Suc\ (SA[k := j]\ !\ i') \wedge i'$
$< j' \wedge n < j'$

from *s-bucket-ptr-strict-lower-bound*[*OF assms*(*1*−*6*,*8*,*10*−*14*,*16*−*20*)]
have *s-bucket-start* $\alpha\ T\ b < B\ !\ b$.
hence *s-bucket-start* $\alpha\ T\ b \leq k$
using *assms*(*21*) **by** *linarith*
hence *bucket-start* $\alpha\ T\ b \leq k$
using *bucket-start-le-s-bucket-start le-trans* **by** *blast*

from ‹*s-bucket-start* $\alpha\ T\ b < B\ !\ b$›
have $k < B\ !\ b$
using *assms*(*21*) **by** *linarith*

have j < *length* T
by (*simp add*: *assms*(*19*) *suffix-type-s-bound*)
hence $b \leq \alpha$ (*Max* (*set* T))
by (*simp add*: *assms*(*8*,*20*) *strict-mono-less-eq*)
with *s-bucket-ptr-upper-bound*[*OF assms*(*2*)]
have B ! $b \leq$ *bucket-end* α T b
by *blast*
with ‹k < B ! b›
have k < *bucket-end* α T b
by *linarith*

have B ! $b \leq i$
proof(*rule ccontr*)
assume ¬B ! $b \leq i$
hence i < B ! b
by *simp*
with *s-B-val*[*OF assms*(*1*−*6*,*8*,*10*−*13*,*15*) ‹$b \leq \alpha$ (*Max* (*set* T))›] ‹*s-bucket-start* α T b < B ! b›
show *False*
by *simp*
qed
with ‹k < B ! b›
have k < i
by *linarith*

have $b \neq b' \Longrightarrow$ *?goal*
proof −
assume $b \neq b'$
hence $B[b := k]$! $b' = B$! b'
by *simp*
with ‹*in-s-current-bucket* α T ($B[b := k]$) b' i'›
have *in-s-current-bucket* α T B b' i'
by (*simp add*: *in-s-current-bucket-def*)
with *s-pred-invD*[*OF assms*(*6*) - ‹$b' \neq 0$›]
obtain j' **where**
j' < *length SA*
SA ! j' = *Suc* (*SA* ! i')
$i' < j'$
$i < j'$
by *blast*
moreover
from ‹*in-s-current-bucket* α T B b' i'›
have B ! $b' \leq i'$ i' < *bucket-end* α T b'
by (*simp-all add*: *in-s-current-bucket-def*)
with *s-bucket-ptr-lower-bound*[*OF assms*(*2*)]
in-s-current-bucketD(*1*)[*OF* ‹*in-s-current-bucket* - - B - -›]
have *bucket-start* α T $b' \leq i'$
by (*meson bucket-start-le-s-bucket-start le-trans*)

with *outside-another-bucket*[*OF* ‹$b \neq b'$› ‹*bucket-start* - - - $\leq k$› ‹$k <$ *bucket-end* - - -›]
‹$i' <$ *bucket-end* α T b'›
have $k \neq i'$
by *blast*
hence $SA[k := j]$! $i' = SA$! i'
by *simp*
moreover
from ‹$i < j'$› *assms*(*16*)
have $n < j'$
using *Suc-lessD* **by** *blast*
moreover
have $SA[k := j]$! $j' = SA$! j'
using ‹$k < i$› *calculation*(*4*) **by** *auto*
ultimately show *?thesis*
by *auto*
qed
moreover
have $b = b' \Longrightarrow$ *?goal*
proof −
assume $b = b'$
hence $B[b := k]$! $b' = k$
using ‹$b \leq \alpha$ (*Max* (*set* T))› *assms*(*9*) **by** *auto*

have $k = i' \Longrightarrow$ *?goal*
proof −
assume $k = i'$
hence $SA[k := j]$! $i' = j$
using ‹$k < i$› *assms*(*16*,*17*) **by** *auto*
moreover
have $SA[k := j]$! $i = SA$! i
using ‹$k < i$› **by** *auto*
ultimately show *?goal*
using *assms*(*16*−*18*) ‹$k = i'$› ‹$k < i$›
by *auto*
qed
moreover
have $k \neq i' \Longrightarrow$ *?goal*
proof −
assume $k \neq i'$
with ‹*in-s-current-bucket* α T ($B[b := k]$) b' i'› ‹$B[b := k]$! $b' = k$›
have $k < i'$
by (*simp add*: *in-s-current-bucket-def*)
hence B ! $b' \leq i'$
using *assms*(*21*) ‹$b = b'$› ‹$k < B$! b› **by** *simp*
hence *in-s-current-bucket* α T B b' i'
using ‹*in-s-current-bucket* α T ($B[b := k]$) b' i'› *in-s-current-bucket-def* **by** *blast*
with *s-pred-invD*[*OF assms*(*6*) - ‹$b' \neq 0$›]

obtain j' **where**
$j' < length\ SA$
$SA\ !\ j' = Suc\ (SA\ !\ i')$
$i' < j'$
$i < j'$
by *blast*
moreover
have $SA[k := j]\ !\ i' = SA\ !\ i'$
using ‹$k \neq i'$› **by** *simp*
moreover
have $SA[k := j]\ !\ j' = SA\ !\ j'$
using ‹$k < i'$› ‹$i' < j'$›
by *auto*
ultimately show *?goal*
using *assms*(*16*) **by** *auto*
qed
ultimately show *?goal*
by *blast*
qed
ultimately show *?goal*
by *blast*
qed

corollary *s-pred-inv-maintained-perm-step*:
assumes $s\text{-}perm\text{-}inv\ \alpha\ T\ B\ SA0\ SA\ i$
and $i = Suc\ n$
and $Suc\ n < length\ SA$
and $SA\ !\ Suc\ n = Suc\ j$
and $suffix\text{-}type\ T\ j = S\text{-}type$
and $b = \alpha\ (T\ !\ j)$
and $k = B\ !\ b - Suc\ 0$
shows $s\text{-}pred\text{-}inv\ \alpha\ T\ (B[b := k])\ (SA[k := j])\ n$
using *s-pred-inv-maintained-step*[*OF s-perm-inv-elims*[*OF assms*(*1*)] *assms*(*2*−)]
by *blast*

83.3.7 Successor

lemma *s-suc-inv-established*:
assumes $length\ SA = length\ T$
and $length\ T \leq n$
shows $s\text{-}suc\text{-}inv\ \alpha\ T\ B\ SA\ n$
unfolding *s-suc-inv-def*
using *assms*(*1*) *assms*(*2*) **by** *linarith*

lemma *s-suc-inv-maintained-step-c1*:
assumes $length\ SA \leq Suc\ n$
shows $s\text{-}suc\text{-}inv\ \alpha\ T\ B\ SA\ n$
unfolding *s-suc-inv-def*
proof (*intro allI impI*; *elim conjE*)

fix i' j
assume $i' <$ *length SA* $n < i'$ *SA* ! $i' =$ *Suc j suffix-type T j* $=$ *S-type*
with *assms*
have *False*
using *less-trans-Suc not-less* **by** *blast*
then show $\exists k.$ *in-s-current-bucket* α *T B* (α (*T* ! *j*)) *k* $\wedge$ *SA* ! *k* $=$ *j* $\wedge$ *k* $< i'$
by *blast*
qed

corollary *s-suc-inv-maintained-perm-step-c1*:
assumes *s-perm-inv* α *T B SA0 SA i*
and $i =$ *Suc n*
and *length SA* $\leq$ *Suc n*
shows *s-suc-inv* α *T B SA n*
by (*simp add*: *assms*(*3*) *s-suc-inv-maintained-step-c1*)

lemma *s-suc-inv-maintained-step-c1-alt*:
assumes *s-suc-inv* α *T B SA i*
and *s-bucket-ptr-inv* α *T B*
and *s-locations-inv* α *T B SA*
and *strict-mono* α
and α (*Max* (*set T*)) $<$ *length B*
and *valid-list T*
and α *bot* $=$ *0*
and $i =$ *Suc n*
and *length T* $\leq$ *SA* ! *Suc n*
shows *s-suc-inv* α *T B SA n*
proof (*cases length SA* $\leq$ *Suc n*)
case *True*
then show *?thesis*
by (*simp add*: *s-suc-inv-maintained-step-c1*)
next
case *False*
hence *Suc n* $<$ *length SA*
by *simp*
show *?thesis*
unfolding *s-suc-inv-def*
proof (*intro allI impI*; *elim conjE*)
fix i' j
let *?goal* $= \exists k.$ *in-s-current-bucket* α *T B* (α (*T* ! *j*)) *k* $\wedge$ *SA* ! *k* $=$ *j* $\wedge$ *k* $< i'$
assume $i' <$ *length SA* $n < i'$ *SA* ! $i' =$ *Suc j suffix-type T j* $=$ *S-type*
hence $i' =$ *Suc n* $\vee$ *Suc n* $< i'$
using *Suc-lessI* **by** *blast*
moreover
from *s-suc-invD*[*OF assms*(*1*) ‹$i' <$ *length SA*› - ‹*SA* ! $i' =$ *Suc j*› ‹*suffix-type T j* $=$ *S-type*›]
have *Suc n* $< i' \Longrightarrow$ *?goal*
using ‹$i =$ *Suc n*› **by** *blast*
moreover

have $i' = Suc\ n \Longrightarrow ?goal$
proof −
assume $i' = Suc\ n$
have $j < length\ T \vee length\ T \leq j$
using *linorder-not-le* **by** *blast*
moreover
have $length\ T \leq j \Longrightarrow ?goal$
by (*meson* ‹*suffix-type* $T\ j$ = *S-type*› *linorder-not-le suffix-type-s-bound*)
moreover
have $j < length\ T \Longrightarrow ?goal$
proof −
assume $j < length\ T$
hence $length\ T = Suc\ j$
using ‹$SA\ !\ i' = Suc\ j$› ‹$i' = Suc\ n$› ‹$length\ T \leq SA\ !\ Suc\ n$› **by** *force*
hence $T\ !\ j = bot$
by (*metis* ‹*valid-list* T› *diff-Suc-1 last-conv-nth length-greater-0-conv valid-list-def*)
hence $\alpha\ (T\ !\ j) = 0$
using ‹$\alpha\ bot = 0$› **by** *presburger*
hence *in-s-current-bucket* $\alpha\ T\ B\ (\alpha\ (T\ !\ j))\ 0$
unfolding *in-s-current-bucket-def*
using *One-nat-def assms*(*2*,*4*,*6*,*7*) *lessI s-bucket-ptr-0 valid-list-bucket-end-0*
by *fastforce*
moreover
{
have $0 < bucket\text{-}end\ \alpha\ T\ 0$
using ‹$\alpha\ (T\ !\ j) = 0$› *calculation in-s-current-bucket-def* **by** *fastforce*
with *s-bucket-ptr-0*[*OF assms*(*2*), *of 0*, *simplified*]
s-locations-invD[*OF assms*(*3*), *of 0 0*, *simplified*]
have $SA\ !\ 0 \in s\text{-}bucket\ \alpha\ T\ 0$
by *simp*
moreover
have $s\text{-}bucket\ \alpha\ T\ 0 = \{j\}$
by (*simp add*: ‹$length\ T = Suc\ j$› *assms*(*4*) *assms*(*6*) *assms*(*7*) *s-bucket-0*)
ultimately have $SA\ !\ 0 = j$
by *blast*
}
ultimately show *?goal*
using ‹$i' = Suc\ n$› **by** *blast*
qed
ultimately show *?goal*
by *blast*
qed
ultimately show *?goal*
by *blast*
qed
qed

corollary *s-suc-inv-maintained-perm-step-c1-alt*:

assumes *s-perm-inv* α *T B SA0 SA i*
and $i = Suc\ n$
and *length* $T \leq SA\ !\ Suc\ n$
shows *s-suc-inv* α *T B SA n*
using *assms s-perm-inv-def s-suc-inv-maintained-step-c1-alt* **by** *blast*

lemma *s-suc-inv-maintained-step-c2*:
assumes *s-suc-inv* α *T B SA i*
and $i = Suc\ n$
and $Suc\ n < length\ SA$
and $SA\ !\ Suc\ n = 0$
shows *s-suc-inv* α *T B SA n*
unfolding *s-suc-inv-def*
proof (*intro allI impI*; *elim conjE*)
fix i' j
assume $i' < length\ SA$ $n < i'$ $SA\ !\ i' = Suc\ j$ *suffix-type* $T\ j = S\text{-}type$

let $?goal = \exists\, k.\ in\text{-}s\text{-}current\text{-}bucket\ \alpha\ T\ B\ (\alpha\ (T\ !\ j))\ k \wedge SA\ !\ k = j \wedge k < i'$

from ‹$n < i'$› ‹$i = Suc\ n$›
have $i = i' \vee i < i'$
by *linarith*
moreover
from *assms*(*2*,*4*) ‹$SA\ !\ i' = Suc\ j$›
have $i = i' \Longrightarrow ?goal$
by *simp*
moreover
from *s-suc-invD*[*OF assms*(*1*) ‹$i' <$ -› - ‹$SA\ !\ i' =$ -› ‹*suffix-type* $T\ j =$ -›] *assms*(*2*)
have $i < i' \Longrightarrow ?goal$
by *blast*
ultimately show *?goal*
by *blast*
qed

corollary *s-suc-inv-maintained-perm-step-c2*:
assumes *s-perm-inv* α *T B SA0 SA i*
and $i = Suc\ n$
and $Suc\ n < length\ SA$
and $SA\ !\ Suc\ n = 0$
shows *s-suc-inv* α *T B SA n*
using *assms s-perm-inv-elims*(*7*) *s-suc-inv-maintained-step-c2* **by** *blast*

lemma *s-suc-inv-maintained-step-c3*:
assumes *s-suc-inv* α *T B SA i*
and $i = Suc\ n$
and $Suc\ n < length\ SA$
and $SA\ !\ Suc\ n = Suc\ j$
and *suffix-type* $T\ j = L\text{-}type$

shows *s-suc-inv* α *T B SA n*
  unfolding *s-suc-inv-def*
proof (*intro allI impI*; *elim conjE*)
  fix i' j'
  assume $i' <$ *length SA* $n < i'$ *SA* ! $i' =$ *Suc* j' *suffix-type* $T\, j' =$ *S-type*

  let *?goal* $= \exists\, k.$ *in-s-current-bucket* α *T B* (α (*T* ! j')) $k \wedge$ *SA* ! $k = j' \wedge k < i'$

  from ‹$n < i'$› *assms*(*2*)
  have $i = i' \vee i < i'$
    using *Suc-lessI* **by** *blast*
  moreover
  from *assms*(*2*,*4*,*5*) ‹*SA* ! $i' =$ -› ‹*suffix-type* $T\, j' =$ -›
  have $i = i' \Longrightarrow$ *?goal*
    by *simp*
  moreover
  from *s-suc-invD*[*OF assms*(*1*) ‹$i' <$ -› - ‹*SA* ! $i' =$ -› ‹*suffix-type* $T\, j' =$ -›]
  have $i < i' \Longrightarrow$ *?goal*
    by *blast*
  ultimately show *?goal*
    by *blast*
qed

corollary *s-suc-inv-maintained-perm-step-c3*:
  assumes *s-perm-inv* α *T B SA0 SA i*
  and $i =$ *Suc n*
  and *Suc* $n <$ *length SA*
  and *SA* ! *Suc* $n =$ *Suc j*
  and *suffix-type* $T\, j =$ *L-type*
shows *s-suc-inv* α *T B SA n*
  using *assms s-perm-inv-elims*(*7*) *s-suc-inv-maintained-step-c3* **by** *blast*

lemma *s-suc-inv-maintained-step-c4*:
  assumes *s-distinct-inv* α *T B SA*
  and *s-bucket-ptr-inv* α *T B*
  and *s-locations-inv* α *T B SA*
  and *s-unchanged-inv* α *T B SA0 SA*
  and *s-seen-inv* α *T B SA i*
  and *s-pred-inv* α *T B SA i*
  and *s-suc-inv* α *T B SA i*
  and *strict-mono* α
  and α (*Max* (*set T*)) $<$ *length B*
  and *length SA0* $=$ *length T*
  and *length SA* $=$ *length T*
  and *l-types-init* α *T SA0*
  and *valid-list T*
  and α *bot* $= 0$
  and *Suc* $0 <$ *length T*
  and $i =$ *Suc n*

and *Suc n < length SA*
and *SA ! Suc n = Suc j*
and *suffix-type T j = S-type*
and *b = α (T ! j)*
and *k = B ! b − Suc 0*
shows *s-suc-inv α T (B[b := k]) (SA[k := j]) n*
unfolding *s-suc-inv-def*
proof(*safe*)
fix *i′ j′*
assume *i′ < length (SA[k := j]) n < i′ SA[k := j] ! i′ = Suc j′ suffix-type T j′ = S-type*
hence *i′ < length SA*
by *simp*

let *?b = α (T ! j′)*
and *?B = B[b := k]*
and *?SA = SA[k := j]*

let *?goal = ∃ k′. in-s-current-bucket α T ?B ?b k′ ∧ ?SA ! k′ = j′ ∧ k′ < i′*

from ‹*suffix-type T j′ = -*›
have *j′ < length T*
by (*simp add*: *suffix-type-s-bound*)
hence *?b ≤ α (Max (set T))*
using ‹*strict-mono -*›
by (*simp add*: *strict-mono-less-eq*)

from *s-bucket-ptr-strict-lower-bound*[*OF assms*(*1−6*,*8*,*10−14*,*16−20*)]
have *s-bucket-start α T b < B ! b*.
hence *s-bucket-start α T b ≤ k*
using *assms*(*21*) **by** *linarith*
hence *bucket-start α T b ≤ k*
using *bucket-start-le-s-bucket-start le-trans* **by** *blast*

from ‹*s-bucket-start α T b < B ! b*›
have *k < B ! b*
using *assms*(*21*) **by** *linarith*

have *j < length T*
by (*simp add*: *assms*(*19*) *suffix-type-s-bound*)
hence *b ≤ α (Max (set T))*
by (*simp add*: *assms*(*8*,*20*) *strict-mono-less-eq*)
with *s-bucket-ptr-upper-bound*[*OF assms*(*2*)]
have *B ! b ≤ bucket-end α T b*
by *blast*
with ‹*k < B ! b*›
have *k < bucket-end α T b*
by *linarith*

have $B\ !\ b \le i$
proof(*rule ccontr*)
  assume $\neg B\ !\ b \le i$
  hence $i < B\ !\ b$
    by *simp*
  with *s-B-val*[*OF assms*(*1−6*,*8*,*10−13*,*15*) ‹$b \le \alpha$ (*Max* (*set T*))›] ‹*s-bucket-start* α *T* $b < B\ !\ b$›
  show *False*
    by *simp*
qed
with ‹$k < B\ !\ b$›
have $k < i$
  by *linarith*
hence $k \le n$
  by (*simp add*: *assms*(*16*))
with ‹$n < i'$›
have $k < i'$
  using *dual-order.strict-trans2* **by** *blast*
hence $SA[k := j]\ !\ i' = SA\ !\ i'$
  by *simp*
with ‹$SA[k := j]\ !\ i' = Suc\ j'$›
have $SA\ !\ i' = Suc\ j'$
  by *simp*

have $i \le i'$
  by (*simp add*: *Suc-leI* ‹$n < i'$› *assms*(*16*))
hence $i = i' \lor i < i'$
  by (*simp add*: *nat-less-le*)
moreover
have $i = i' \Longrightarrow$ *?goal*
proof −
  assume $i = i'$
  hence $j = j'$
    using ‹$SA\ !\ i' = Suc\ j'$› *assms*(*16*,*18*) **by** *auto*
  hence $SA[k := j]\ !\ k = j'$
    using ‹$k \le n$› *assms*(*17*) **by** *auto*
  moreover
  have *?b* = *b*
    using ‹$j = j'$› *assms*(*20*) **by** *blast*
  hence *in-s-current-bucket* α *T ?B ?b k* = *in-s-current-bucket* α *T ?B b k*
    by *simp*
  moreover
  from ‹α ($T\ !\ j'$) $\le \alpha$ (*Max* (*set T*))›
      ‹*?b* = *b*›[*symmetric*]
  have *in-s-current-bucket* α *T ?B b k*
    unfolding *in-s-current-bucket-def*
    using ‹$k <$ *bucket-end* α *T b*› *assms*(*9*) **by** *auto*
  ultimately show *?goal*
    using ‹$k < i'$› **by** *blast*

qed
moreover
have $i < i' \Longrightarrow$ *?goal*
proof –
assume $i < i'$
with *s-suc-invD*[*OF assms*(*7*) ‹$i' <$ *length SA*› - ‹*SA* ! $i' =$ *Suc* j'› ‹*suffix-type T* $j' =$ -›]
obtain k' **where**
in-s-current-bucket α *T B ?b* k'
SA ! $k' = j'$
$k' < i'$
by *blast*
moreover
from *in-s-current-bucketD*[*OF* ‹*in-s-current-bucket* α *T B ?b* k'›]
have *in-s-current-bucket* α *T ?B ?b* k'
unfolding *in-s-current-bucket-def*
proof (*safe*)
show *?B* ! *?b* $\leq k'$
by (*metis* ‹*B* ! *?b* $\leq k'$› ‹$k <$ *B* ! *b*› *dual-order.trans list-update-beyond nat-le-linear*
not-less nth-list-update-eq nth-list-update-neq)
qed
moreover
from *in-s-current-bucketD*(*2*)[*OF* ‹*in-s-current-bucket* α *T B ?b* k'›]
have *B* ! *?b* $\leq k'$ **.**
hence *s-bucket-start* α *T ?b* $\leq k'$
by (*meson* ‹*?b* $\leq \alpha$ (*Max* (*set T*))› *assms*(*2*) *le-less-trans not-le s-bucket-ptr-inv-def*)
hence *bucket-start* α *T ?b* $\leq k'$
using *bucket-start-le-s-bucket-start dual-order.trans* **by** *blast*

have $b =$ *?b* $\vee$ $b \neq$ *?b*
by *blast*
hence $k \neq k'$
proof
assume $b =$ *?b*
with ‹*B* ! *?b* $\leq k'$› ‹$k <$ *B* ! *b*›
show *?thesis*
by *simp*
next
assume $b \neq$ *?b*
with *outside-another-bucket*[*OF* - ‹*bucket-start* - - - $\leq k$› ‹$k <$ *bucket-end* - - -›]
‹*bucket-start* α *T ?b* $\leq k'$› *in-s-current-bucketD*(*3*)[*OF* ‹*in-s-current-bucket* α *T B ?b* k'›]
show *?thesis*
by *blast*
qed
hence *SA*[$k := j$] ! $k' =$ *SA* ! k'
by *simp*

ultimately show *?goal*
by *blast*
qed
ultimately show *?goal*
by *blast*
qed

corollary *s-suc-inv-maintained-perm-step-c4*:
assumes *s-perm-inv* α *T B SA0 SA i*
and $i = Suc\ n$
and $Suc\ n < length\ SA$
and $SA\ !\ Suc\ n = Suc\ j$
and *suffix-type T j = S-type*
and $b = \alpha\ (T\ !\ j)$
and $k = B\ !\ b - Suc\ 0$
shows *s-suc-inv* α $T\ (B[b := k])\ (SA[k := j])\ n$
using *s-suc-inv-maintained-step-c4*[*OF s-perm-inv-elims*[*OF assms*(*1*)] *assms*(*2*−)]
by *blast*

lemmas *s-suc-inv-maintained-perm-step* =
s-suc-inv-maintained-step-c1
s-suc-inv-maintained-perm-step-c2
s-suc-inv-maintained-perm-step-c3
s-suc-inv-maintained-perm-step-c4

83.3.8 Combined Permutation Invariant

lemma *s-perm-inv-established*:
assumes *s-bucket-init* α *T B*
and *s-type-init T SA*
and *strict-mono* α
and $\alpha\ (Max\ (set\ T)) < length\ B$
and $length\ SA = length\ T$
and *l-types-init* α *T SA*
and *valid-list T*
and $\alpha\ bot = 0$
and $Suc\ 0 < length\ T$
and $length\ T \leq n$
shows *s-perm-inv* α *T B SA SA n*
unfolding *s-perm-inv-def*
by (*simp add*: *assms s-distinct-inv-established s-bucket-ptr-inv-established s-locations-inv-established s-unchanged-inv-established s-seen-inv-established s-pred-inv-established s-suc-inv-established*)

lemma *s-perm-inv-maintained-step-c1*:
assumes *s-perm-inv* α *T B SA0 SA i*
and $i = Suc\ n$
and $length\ SA \leq Suc\ n$
shows *s-perm-inv* α *T B SA0 SA n*

```
  unfolding s-perm-inv-def
  by (clarsimp simp: s-perm-inv-elims[OF assms(1)]
                     s-seen-inv-maintained-perm-step-c1[OF assms]
                     s-pred-inv-maintained-perm-step-alt[OF assms(1,2)]
                     s-suc-inv-maintained-step-c1[OF assms(3)])

lemma s-perm-inv-maintained-step-c1-alt:
  assumes s-perm-inv α T B SA0 SA i
  and     i = Suc n
  and     length T ≤ SA ! Suc n
shows s-perm-inv α T B SA0 SA n
proof (cases length T ≤ Suc n)
  case True
  then show ?thesis
   by (metis assms(1) assms(2) s-perm-inv-elims(11) s-perm-inv-maintained-step-c1)
next
  case False
  hence Suc n < length T
    by simp
  then show ?thesis
    unfolding s-perm-inv-def
    by (metis assms s-perm-inv-def s-pred-inv-maintained-step-alt
          s-seen-inv-maintained-perm-step-c1-alt s-suc-inv-maintained-perm-step-c1-alt)
qed

lemma s-perm-inv-maintained-step-c2:
  assumes s-perm-inv α T B SA0 SA i
  and     i = Suc n
  and     Suc n < length SA
  and     SA ! Suc n = 0
shows s-perm-inv α T B SA0 SA n
  unfolding s-perm-inv-def
  by (clarsimp simp: s-perm-inv-elims[OF assms(1)]
                     s-seen-inv-maintained-perm-step-c2[OF assms]
                     s-pred-inv-maintained-perm-step-alt[OF assms(1,2)]
                     s-suc-inv-maintained-perm-step-c2[OF assms])

lemma s-perm-inv-maintained-step-c3:
  assumes s-perm-inv α T B SA0 SA i
  and     i = Suc n
  and     Suc n < length SA
  and     SA ! Suc n = Suc j
  and     suffix-type T j = L-type
shows s-perm-inv α T B SA0 SA n
  unfolding s-perm-inv-def
  by (clarsimp simp: s-perm-inv-elims[OF assms(1)]
                     s-seen-inv-maintained-perm-step-c3[OF assms]
                     s-pred-inv-maintained-perm-step-alt[OF assms(1,2)]
                     s-suc-inv-maintained-perm-step-c3[OF assms])
```

lemma *s-perm-inv-maintained-step-c4*:
  assumes *s-perm-inv* α *T B SA0 SA i*
  and $i = Suc\ n$
  and $Suc\ n < length\ SA$
  and $SA\ !\ Suc\ n = Suc\ j$
  and $suffix\text{-}type\ T\ j = S\text{-}type$
  and $b = \alpha\ (T\ !\ j)$
  and $k = B\ !\ b - Suc\ 0$
shows *s-perm-inv* α $T\ (B[b := k])\ SA0\ (SA[k := j])\ n$
  unfolding *s-perm-inv-def*
  by (*clarsimp simp*: *s-perm-inv-elims*[*OF assms*(*1*)]
      s-distinct-inv-maintained-perm-step[*OF assms*]
      s-bucket-ptr-inv-maintained-perm-step[*OF assms*]
      s-locations-inv-maintained-perm-step[*OF assms*]
      s-unchanged-inv-maintained-perm-step[*OF assms*]
      s-seen-inv-maintained-perm-step-c4[*OF assms*]
      s-pred-inv-maintained-perm-step[*OF assms*]
      s-suc-inv-maintained-perm-step-c4[*OF assms*])

theorem *abs-induce-s-perm-step*:
  assumes *s-perm-inv* α *T B SA0 SA i*
  and *abs-induce-s-step* $(B, SA, i)\ (\alpha, T) = (B', SA', i')$
shows *s-perm-inv* $\alpha\ T\ B'\ SA0\ SA'\ i'$
proof (*cases i*)
  case *0*
  then show *?thesis*
    using *assms* **by** *force*
next
  case (*Suc n*)
  assume $i = Suc\ n$
  have $T \neq []$
    using *s-perm-inv-elims*(*15*)[*OF assms*(*1*)] **by** *fastforce*
  show *?thesis*
  proof (*cases* $Suc\ n < length\ SA \land SA\ !\ Suc\ n < length\ T$)
    assume $Suc\ n < length\ SA \land SA\ !\ Suc\ n < length\ T$
    hence $Suc\ n < length\ SA$ $SA\ !\ Suc\ n < length\ T$
      by *blast+*
    show *?thesis*
    proof (*cases* $SA\ !\ Suc\ n$)
      case *0*
      then show *?thesis*
        using *s-perm-inv-maintained-step-c2*[*OF assms*(*1*) ‹$i = Suc\ n$› ‹$Suc\ n <$
length SA› *0*] *assms*
       by (*clarsimp simp*: ‹$i = Suc\ n$› ‹$Suc\ n < length\ SA$› *0* ‹$T \neq []$›)
    next
      case (*Suc j*)
      assume $SA\ !\ Suc\ n = Suc\ j$
      hence $Suc\ j < length\ T$

using ‹SA ! $Suc\ n < length\ T$› **by** *auto*
show *?thesis*
proof (*cases suffix-type T j*)
case *S-type*
then show *?thesis*
using *assms* ‹$i = Suc\ n$› ‹$Suc\ n < length\ SA$› ‹SA ! $Suc\ n = Suc\ j$›
s-perm-inv-maintained-step-c4[*OF assms*(*1*), *of n j* α (T ! j) B ! α (T ! j) $-$ *Suc 0*]
by (*clarsimp simp*: *Let-def* ‹$Suc\ j < length\ T$›)
next
case *L-type*
then show *?thesis*
using *assms* ‹$i = Suc\ n$› ‹$Suc\ n < length\ SA$› ‹SA ! $Suc\ n = Suc\ j$›
s-perm-inv-maintained-step-c3[*OF assms*(*1*)]
by (*clarsimp simp*: *Let-def* ‹$Suc\ j < length\ T$›)
qed
qed
next
assume $\neg$($Suc\ n < length\ SA \wedge SA$! $Suc\ n < length\ T$)
hence $\neg$ $Suc\ n < length\ SA \vee \neg$ SA ! $Suc\ n < length\ T$
by *blast*
then show *?thesis*
proof
assume $\neg$ $Suc\ n < length\ SA$
then show *?thesis*
using *assms* ‹$i = Suc\ n$› *s-perm-inv-maintained-step-c1*[*OF assms*(*1*)] **by** *force*
next
assume $\neg$ SA ! $Suc\ n < length\ T$
hence $length\ T \leq SA$! $Suc\ n$
by *simp*
then show *?thesis*
using *assms* ‹$i = Suc\ n$› *s-perm-inv-maintained-step-c1-alt*[*OF assms*(*1*)] **by** *simp*
qed
qed
qed

corollary *abs-induce-s-perm-step-alt*:
$\bigwedge a$. *s-perm-inv-alt* α T *SA0* $a \Longrightarrow$ *s-perm-inv-alt* α T *SA0* (*abs-induce-s-step* a (α, T))
by (*metis abs-induce-s-perm-step s-perm-inv-alt.elims*(*2*) *s-perm-inv-alt.elims*(*3*))

theorem *abs-induce-s-perm-alt-maintained*:
assumes *s-perm-inv-alt* α T *SA0* (B, SA, *length* T)
shows *s-perm-inv-alt* α T *SA0* (*abs-induce-s-base* α T B SA)
unfolding *abs-induce-s-base-def*
using *repeat-maintain-inv*[*of s-perm-inv-alt* α T *SA0 abs-induce-s-step* (α, T), *OF - assms*(*1*)]

abs-induce-s-perm-step-alt
by *blast*

corollary *abs-induce-s-perm-maintained*:
assumes *abs-induce-s-base* α *T B SA* = (*B′*, *SA′*, *n*)
and *s-perm-inv* α *T B SA0 SA* (*length T*)
shows *s-perm-inv* α *T B′ SA0 SA′ n*
using *assms abs-induce-s-perm-alt-maintained* **by** *force*

lemma *s-perm-inv-0-B-val*:
assumes *s-perm-inv* α *T B SA SA′ 0*
and $b \le \alpha$ (*Max* (*set T*))
shows *B* ! *b* = *s-bucket-start* α *T b*
proof −
from *s-bucket-ptr-lower-bound*[*OF s-perm-inv-elims*(*2*)[*OF assms*(*1*)] *assms*(*2*)]
have *s-bucket-start* α *T b* ≤ *B* ! *b* **.**

have *s-bucket-start* α *T b* ≥ *0*
by *blast*
hence *s-bucket-start* α *T b* = *0* ∨ *0* < *s-bucket-start* α *T b*
by *blast*
with ‹*s-bucket-start* α *T b* ≤ *B* ! *b*›
have *B* ! *b* = *s-bucket-start* α *T b* ∨ *0* < *B* ! *b*
by *linarith*
then show *?thesis*
proof
assume *B* ! *b* = *s-bucket-start* α *T b*
then show *?thesis* **.**
next
assume *0* < *B* ! *b*
with *s-B-val*[*OF s-perm-inv-elims*(*1*−*6*,*8*,*10*−*13*,*15*)[*OF assms*(*1*)] *assms*(*2*)]
show *?thesis* **.**
qed
qed

lemma *s-perm-inv-0-list-slice-bucket*:
assumes *s-perm-inv* α *T B SA SA′ 0*
and $b \le \alpha$ (*Max* (*set T*))
shows *set* (*list-slice SA′* (*bucket-start* α *T b*) (*bucket-end* α *T b*)) = *bucket* α *T b*
by (*meson assms bucket-eq-list-slice s-perm-inv-0-B-val s-perm-inv-elims*(*1*−*4*,*10*−*12*))

lemma *s-perm-inv-0-distinct-list-slice*:
assumes *s-perm-inv* α *T B SA SA′ 0*
and $b \le \alpha$ (*Max* (*set T*))
shows *distinct* (*list-slice SA′* (*bucket-start* α *T b*) (*bucket-end* α *T b*))
(**is** *distinct ?xs*)
proof −
let *?ys* = *list-slice SA′* (*bucket-start* α *T b*) (*l-bucket-end* α *T b*)

and *?zs* = *list-slice SA′* (*s-bucket-start* α *T b*) (*bucket-end* α *T b*)

have *?xs* = *?ys* @ *?zs*
by (*metis list-slice-append bucket-start-le-s-bucket-start l-bucket-end-le-bucket-end s-bucket-start-eq-l-bucket-end*)

from *l-types-initD*(*1*)[*OF l-types-init-maintained*[*OF s-perm-inv-elims*(*2*,*4*,*10*−*12*)[*OF assms*(*1*)]]
assms(*2*)]
have *set ?ys* = *l-bucket* α *T b* **.**
moreover
from *s-bucket-eq-list-slice*[*OF s-perm-inv-elims*(*1*,*3*,*11*)[*OF assms*(*1*)] *assms*(*2*)
s-perm-inv-0-B-val[*OF assms*]]
have *set ?zs* = *s-bucket* α *T b* **.**
ultimately have *set ?ys* ∩ *set ?zs* = {}
using *disjoint-l-s-bucket* **by** *blast*
with *s-distinct-invD*[*OF s-perm-inv-elims*(*1*), *OF assms*, *simplified s-perm-inv-0-B-val*[*OF assms*]]
l-types-initD(*2*)[*OF l-types-init-maintained*[*OF s-perm-inv-elims*(*2*,*4*,*10*−*12*)[*OF assms*(*1*)]]
assms(*2*)]
‹*?xs* = *?ys* @ *?zs*›
show *?thesis*
by *auto*
qed

lemma *abs-induce-s-base-distinct*:
assumes *abs-induce-s-base* α *T B SA* = (*B′*, *SA′*, *n*)
and *s-perm-inv* α *T B′ SA SA′ n*
shows *distinct SA′*
proof(*intro distinct-conv-nth*[*THEN iffD2*] *allI impI*)
fix *i j*
assume *i* < *length SA′ j* < *length SA′ i* ≠ *j*
hence *i* < *length T j* < *length T*
using *assms*(*2*) *s-perm-inv-elims*(*11*) **by** *fastforce*+

from *abs-induce-s-base-index*[*of* α *T B SA*] *assms*(*1*)
have *n* = *0*
by *simp*

from *index-in-bucket-interval-gen*[*OF* ‹*i* < *length T*› *s-perm-inv-elims*(*8*)[*OF assms*(*2*)]]
obtain *b0* **where**
b0 ≤ α (*Max* (*set T*))
bucket-start α *T b0* ≤ *i*
i < *bucket-end* α *T b0*
by *blast*

have *bucket-end* α *T b0* ≤ *length SA′*

using *assms*(*2*) *bucket-end-le-length s-perm-inv-elims*(*11*) **by** *fastforce*

let *?xs* = *list-slice SA′* (*bucket-start* α *T b0*) (*bucket-end* α *T b0*)

from *index-in-bucket-interval-gen*[*OF* ‹*j* < *length T*› *s-perm-inv-elims*(*8*)[*OF assms*(*2*)]]
obtain *b1* **where**
b1 ≤ α (*Max* (*set T*))
bucket-start α *T b1* ≤ *j*
j < *bucket-end* α *T b1*
by *blast*

have *bucket-end* α *T b1* ≤ *length SA′*
using *assms*(*2*) *bucket-end-le-length s-perm-inv-elims*(*11*) **by** *fastforce*

have *b0* ≠ *b1* ⟹ *SA′* ! *i* ≠ *SA′* ! *j*
proof −
assume *b0* ≠ *b1*
hence *bucket* α *T b0* ∩ *bucket* α *T b1* = {}
by (*metis* (*mono-tags*, *lifting*) *Int-emptyI bucket-def mem-Collect-eq*)
moreover
from *s-perm-inv-0-list-slice-bucket*[*OF assms*(*2*)[*simplified* ‹*n* = *0*›] ‹*b0* ≤ *-*›]
list-slice-nth-mem[*OF* ‹*bucket-start* α *T b0* ≤ *i*› ‹*i* < *bucket-end* α *T b0*›
‹*bucket-end* α *T b0* ≤ *-*›]
have *SA′* ! *i* ∈ *bucket* α *T b0*
by *blast*
moreover
from *s-perm-inv-0-list-slice-bucket*[*OF assms*(*2*)[*simplified* ‹*n* = *0*›] ‹*b1* ≤ *-*›]
list-slice-nth-mem[*OF* ‹*bucket-start* α *T b1* ≤ *j*› ‹*j* < *bucket-end* α *T b1*›
‹*bucket-end* α *T b1* ≤ *-*›]
have *SA′* ! *j* ∈ *bucket* α *T b1*
by *blast*
ultimately show *?thesis*
by *auto*
qed
moreover
have *b0* = *b1* ⟹ *SA′* ! *i* ≠ *SA′* ! *j*
proof −
assume *b0* = *b1*
with ‹*bucket-start* α *T b1* ≤ *j*› ‹*j* < *bucket-end* α *T b1*›
have *bucket-start* α *T b0* ≤ *j j* < *bucket-end* α *T b0*
by *simp-all*
with *list-slice-nth-eq-iff-index-eq*[
OF s-perm-inv-0-distinct-list-slice[*OF assms*(*2*)[*simplified*‹*n* = *0*›] ‹*b0* ≤ *-*›]
‹*bucket-end - - b0* ≤ *-*› ‹*bucket-start* α *T b0* ≤ *i*› ‹*i* < *bucket-end* α *T b0*›, *of j*]
‹*i* ≠ *j*›
show *?thesis*

    by *blast*
  qed
  ultimately show SA' ! $i \neq SA'$! j
    by *blast*
qed

lemma *abs-induce-s-base-subset-upt*:
  assumes *abs-induce-s-base* α T B SA = (B', SA', n)
  and *s-perm-inv* α T B' SA SA' n
shows *set* $SA' \subseteq \{0..<length\ T\}$
proof
  fix x
  assume $x \in set\ SA'$
  from *in-set-conv-nth*[*THEN iffD1*, *OF* ‹$x \in set\ SA'$›]
  obtain i **where**
    $i < length\ SA'$
    SA' ! $i = x$
    by *blast*
  hence $i < length\ T$
    using *assms*(*2*) *s-perm-inv-elims*(*11*) **by** *fastforce*
  with *index-in-bucket-interval-gen*[*OF - s-perm-inv-elims*(*8*)[*OF assms*(*2*)]]
  obtain b **where**
    $b \leq \alpha$ (*Max* (*set* T))
    bucket-start α T $b \leq i$
    $i <$ *bucket-end* α T b
    by *blast*

  from *abs-induce-s-base-index*[*of* α T B SA] *assms*(*1*)
  have $n = 0$
    by *simp*

  have *bucket-end* α T $b \leq$ *length* SA'
    using *assms*(*2*) *bucket-end-le-length* *s-perm-inv-elims*(*11*) **by** *fastforce*
  with *s-perm-inv-0-list-slice-bucket*[*OF assms*(*2*)[*simplified* ‹$n = 0$›] ‹$b \leq$ -›]
      ‹SA' ! $i = x$› ‹*bucket-start* α T $b \leq i$› ‹$i <$ *bucket-end* α T b›
  have $x \in$ *bucket* α T b
    using *list-slice-nth-mem* **by** *blast*
  hence $x < length\ T$
    using *bucket-def* **by** *blast*
  then show $x \in \{0..< length\ T\}$
    by *simp*
qed

corollary *abs-induce-s-base-eq-upt*:
  assumes *abs-induce-s-base* α T B SA = (B', SA', n)
  and *s-perm-inv* α T B' SA SA' n
shows *set* $SA' = \{0..<length\ T\}$
  by (*rule card-subset-eq*[*OF finite-atLeastLessThan abs-induce-s-base-subset-upt*[*OF assms*]];

clarsimp simp: *distinct-card*[*OF abs-induce-s-base-distinct*[*OF assms*]]
s-perm-inv-elims(*11*)[*OF assms*(*2*)]])

theorem *abs-induce-s-base-perm*:
assumes *abs-induce-s-base* α *T B SA* = (*B'*, *SA'*, *n*)
and *s-perm-inv* α *T B' SA SA' n*
shows *SA'* <~~> [*0*..< *length T*]
by (*rule perm-distinct-set-of-upt-iff*[*THEN iffD2*];
clarsimp simp: *abs-induce-s-base-distinct*[*OF assms*] *abs-induce-s-base-eq-upt*[*OF assms*])

83.3.9 Sorted

lemma *s-sorted-established*:
assumes *s-bucket-init* α *T B*
and *strict-mono* α
and *valid-list T*
and α *bot* = *0*
and *b* ≤ α (*Max* (*set T*))
shows *sorted-wrt R* (*list-slice SA* (*B* ! *b*) (*bucket-end* α *T b*))
(**is** *sorted-wrt R ?xs*)
proof −
have *b* = *0* ∨ *0* < *b*
by *blast*
moreover
have *0* < *b* ⟹ *?thesis*
proof −
assume *0* < *b*
hence *B* ! *b* = *bucket-end* α *T b*
by (*simp add*: ‹*b* ≤ α (*Max* (*set T*))› *assms*(*1*) *s-bucket-initD*)
then show *?thesis*
by *simp*
qed
moreover
have *b* = *0* ⟹ *?thesis*
proof −
assume *b* = *0*
hence *bucket-end* α *T b* = *Suc 0*
by (*simp add*: *assms*(*2*−*4*) *valid-list-bucket-end-0*)
moreover
from ‹*b* = *0*›
have *B* ! *b* = *0*
using *assms*(*1*) *s-bucket-initD*(*2*) **by** *auto*
ultimately show *?thesis*
by (*simp add*: *sorted-wrt01*)
qed
ultimately show *?thesis*
by *blast*
qed

lemma *s-sorted-inv-established*:
  assumes *s-bucket-init* α *T B*
  and *strict-mono* α
  and *valid-list T*
  and α *bot* = *0*
shows *s-sorted-inv* α *T B SA*
  unfolding *s-sorted-inv-def*
  using *assms ordlistns.sorted-map s-sorted-established* **by** *blast*

lemma *s-prefix-sorted-inv-established*:
  assumes *s-bucket-init* α *T B*
  and *strict-mono* α
  and *valid-list T*
  and α *bot* = *0*
shows *s-prefix-sorted-inv* α *T B SA*
  unfolding *s-prefix-sorted-inv-def*
  using *assms ordlistns.sorted-map s-sorted-established* **by** *blast*

lemma *s-sorted-maintained-unchanged-step*:
  assumes *s-perm-inv* α *T B SA0 SA i*
  and $i = Suc\ n$
  and $Suc\ n < length\ SA$
  and $SA\ !\ Suc\ n = Suc\ j$
  and $suffix\text{-}type\ T\ j = S\text{-}type$
  and $b = \alpha\ (T\ !\ j)$
  and $k = B\ !\ b - Suc\ 0$
  and $b' \leq \alpha\ (Max\ (set\ T))$
  and $sorted\text{-}wrt\ R\ (list\text{-}slice\ SA\ (B\ !\ b')\ (bucket\text{-}end\ \alpha\ T\ b'))$
  and $b \neq b'$
shows $sorted\text{-}wrt\ R\ (list\text{-}slice\ (SA[k := j])\ ((B[b := k])\ !\ b')\ (bucket\text{-}end\ \alpha\ T\ b'))$
proof –
  let $?xs = list\text{-}slice\ (SA[k := j])\ (B[b := k]\ !\ b')\ (bucket\text{-}end\ \alpha\ T\ b')$

  have $bucket\text{-}end\ \alpha\ T\ b \leq length\ T$
    using *bucket-end-le-length* **by** *blast*
  moreover
  have $B\ !\ b \leq bucket\text{-}end\ \alpha\ T\ b$
    using *assms*(*1*,*5*,*6*) *s-bucket-ptr-upper-bound s-perm-inv-elims*(*2*,*8*) *strict-mono-less-eq suffix-type-s-bound* **by** *fastforce*
  ultimately have $k < length\ T$
    using *assms*(*1*,*7*) *s-perm-inv-elims*(*15*) **by** *fastforce*
  hence $k < length\ SA$
    by (*metis assms*(*1*) *s-perm-inv-def*)

  from *s-bucket-ptr-strict-lower-bound*[*OF s-perm-inv-elims*(*1*–*6*,*8*,*10*–*14*)][*OF assms*(*1*)] *assms*(*2*–*6*)]
  have $s\text{-}bucket\text{-}start\ \alpha\ T\ b < B\ !\ b$ **.**
  hence $k < B\ !\ b$

using *assms*(*7*) *diff-less gr-implies-not-zero* **by** *blast*

have *s-bucket-start* α *T b* $\leq$ *k*
using *assms s-bucket-ptr-strict-lower-bound s-perm-inv-def* **by** *fastforce*
hence *bucket-start* α *T b* $\leq$ *k*
using *bucket-start-le-s-bucket-start le-trans* **by** *blast*

from ‹$b \neq b'$›
have $B[b := k]$! $b' = B$! b'
by *simp*

have $k < B$! $b' \vee$ *bucket-end* α *T* $b' \leq k$
proof –
from ‹$b \neq b'$›
have $b < b' \vee b' < b$
using *nat-neq-iff* **by** *blast*
moreover
have $b < b' \Longrightarrow k < B$! b'
proof –
assume $b < b'$
hence *bucket-end* α *T b* $\leq$ *bucket-start* α *T* b'
by (*simp add*: *less-bucket-end-le-start*)
hence $k <$ *bucket-start* α *T* b'
using ‹*B* ! *b* $\leq$ *bucket-end* α *T b*› ‹$k < B$! *b*› **by** *linarith*
with *s-bucket-ptr-lower-bound*[*OF s-perm-inv-elims*(*2*)[*OF assms*(*1*)] ‹$b' \leq$ -›]
show *?thesis*
by (*meson bucket-start-le-s-bucket-start order.strict-trans2*)
qed
moreover
have $b' < b \Longrightarrow$ *bucket-end* α *T* $b' \leq k$
proof –
assume $b' < b$
hence *bucket-end* α *T* $b' \leq$ *bucket-start* α *T b*
by (*simp add*: *less-bucket-end-le-start*)
then show *?thesis*
using ‹*bucket-start* α *T b* $\leq$ *k*› **by** *linarith*
qed
ultimately show *?thesis*
by *blast*
qed
with *list-slice-update-unchanged-1*
list-slice-update-unchanged-2
have *?xs* = *list-slice SA* (*B* ! b') (*bucket-end* α *T* b')
using ‹$B[b := k]$! $b' = B$! b'› **by** *auto*
then show *?thesis*
using *assms*(*9*) **by** *auto*
qed

lemma *s-sorted-inv-maintained-step*:

assumes *s-perm-inv* α *T B SA0 SA i*
and *s-sorted-pre* α *T SA0*
and *s-sorted-inv* α *T B SA*
and $i = Suc\ n$
and $Suc\ n < length\ SA$
and $SA\ !\ Suc\ n = Suc\ j$
and *suffix-type* $T\ j = S\text{-}type$
and $b = \alpha\ (T\ !\ j)$
and $k = B\ !\ b - Suc\ 0$
shows *s-sorted-inv* α $T\ (B[b := k])\ (SA[k := j])$
unfolding *s-sorted-inv-def*
proof (*safe*)
fix b'
assume $b' \leq \alpha\ (Max\ (set\ T))$
let *?xs* = *list-slice* $(SA[k := j])\ (B[b := k]\ !\ b')$ (*bucket-end* α $T\ b'$)

have *bucket-end* α $T\ b \leq length\ T$
using *bucket-end-le-length* **by** *blast*

moreover
have $B\ !\ b \leq$ *bucket-end* α $T\ b$
using *assms*(*1*,*7*,*8*) *s-bucket-ptr-upper-bound suffix-type-s-bound*
s-perm-inv-elims(*2*,*8*) *strict-mono-less-eq*
by *fastforce*

ultimately have $k < length\ T$
using *assms*(*1*,*9*) *s-perm-inv-elims*(*15*) **by** *fastforce*
hence $k < length\ SA$
by (*metis assms*(*1*) *s-perm-inv-def*)

from *s-bucket-ptr-strict-lower-bound*
[*OF s-perm-inv-elims*(*1*−*6*,*8*,*10*−*14*)
[*OF assms*(*1*)] *assms*(*4*−*8*)]
have *s-bucket-start* α $T\ b < B\ !\ b$ **.**
hence $k < B\ !\ b$
using *assms*(*9*) *diff-less gr-implies-not-zero* **by** *blast*

have *s-bucket-start* α $T\ b \leq k$
using *assms s-bucket-ptr-strict-lower-bound s-perm-inv-def*
by *fastforce*
hence *bucket-start* α $T\ b \leq k$
using *bucket-start-le-s-bucket-start le-trans*
by *blast*
hence $b \leq \alpha\ (Max\ (set\ T))$
by (*metis* ‹$k < length\ SA$› *assms*(*1*) *bucket-end-Max dual-order.trans*
less-bucket-end-le-start s-perm-inv-elims(*8*,*11*) *leD leI*)

have $b = b' \lor b \neq b'$
by *blast*

moreover
have $b = b' \Longrightarrow$ *ordlistns.sorted* (*map* (*suffix T*) *?xs*)
proof −
  assume $b = b'$
  hence $B[b := k] \mathbin{!} b' = k$
    by (*meson* ‹$b' \leq \alpha$ (*Max* (*set T*))› *assms*(*1*) *le-less-trans nth-list-update-eq*
        s-perm-inv-elims(*9*))

  have $SA[k := j] \mathbin{!} k = j$
    by (*simp add*: ‹$k <$ *length SA*›)

  from *list-slice-update-unchanged-1*
      ‹$k < B \mathbin{!} b$›
      ‹$SA[k := j] \mathbin{!} k = j$›
      ‹$B[b := k] \mathbin{!} b' = k$›
      ‹$B \mathbin{!} b \leq$ *bucket-end* α *T b*›
      ‹$b = b'$› ‹$k <$ *length SA*›
  have *?xs* = *j* # *list-slice SA* (*B* ! *b*) (*bucket-end* α *T b*)
    by (*metis Suc-pred assms*(*9*) *length-list-update not-le*
          less-nat-zero-code list-slice-Suc less-le-trans)
  moreover
  have *ordlistns.sorted*
        (*map* (*suffix T*) (*j* # *list-slice SA* (*B* ! *b*) (*bucket-end* α *T b*)))
  proof −
    let *?ys* = *list-slice SA* (*B* ! *b*) (*bucket-end* α *T b*)

    have *A*: *map* (*suffix T*) (*j* # *?ys*) = (*suffix T j*) # *map* (*suffix T*) *?ys*
      by *simp*

    from *s-sorted-invD*[*OF assms*(*3*) ‹$b \leq$ -›]
    have *B*: *ordlistns.sorted* (*map* (*suffix T*) *?ys*) **.**

    have *?ys* = [] ∨ *?ys* ≠ []
      by *blast*
    hence *map* (*suffix T*) *?ys* = [] ∨ *map* (*suffix T*) *?ys* ≠ []
      by *simp*
    moreover
    have *map* (*suffix T*) *?ys* = [] $\Longrightarrow$ *?thesis*
      using *ordlistns.sorted-cons-nil* **by** *fastforce*
    moreover
    have *map* (*suffix T*) *?ys* ≠ [] $\Longrightarrow$
          ordlistns.sorted ((*suffix T j*) # *map* (*suffix T*) *?ys*)
    proof (*rule ordlistns.sorted-consI*[*OF* - *B*])
      assume *map* (*suffix T*) (*list-slice SA* (*B* ! *b*) (*bucket-end* α *T b*)) ≠ []
      then
      show *map* (*suffix T*) (*list-slice SA* (*B* ! *b*) (*bucket-end* α *T b*)) ≠ []
        by *simp*
    next
      assume *map* (*suffix T*) *?ys* ≠ []

hence *map* (*suffix T*) *?ys* ! *0* = *suffix T* (*?ys* ! *0*)
  by (*metis length-greater-0-conv list.simps*(*8*) *nth-map*)
moreover
have *list-less-eq-ns* (*suffix T j*) (*suffix T* (*?ys* ! *0*))
proof −
  have *?ys* ! *0* ∈ *s-bucket* α *T b*
    by (*metis assms*(*1*) *length-greater-0-conv s-perm-inv-elims*(*3*)
          length-map nth-mem s-locations-inv-in-list-slice
          ‹*b* = *b*′›
          ‹*b*′ ≤ α (*Max* (*set T*))›
          ‹*map* (*suffix T*) *?ys* ≠ []›)
  hence α (*T* ! (*?ys* ! *0*)) = *b suffix-type T* (*?ys* ! *0*) = *S-type*
    by (*simp add*: *s-bucket-def bucket-def*)+
  hence *T* ! *j* = *T* ! (*?ys* ! *0*)
    using *assms*(*1*,*8*) *s-perm-inv-elims*(*8*) *strict-mono-eq* **by** *fastforce*

  have *?ys* ! *0* = *SA* ! (*B* ! *b*)
    using ‹*map* (*suffix T*) *?ys* ≠ []› *nth-list-slice* **by** *fastforce*

  have *b* ≠ *0*
    by (*metis* ‹*s-bucket-start* α *T b* < *B* ! *b*› *assms*(*1*)
          gr-implies-not-zero s-bucket-ptr-0 s-perm-inv-elims(*2*))

  have *in-s-current-bucket* α *T B b* (*B* ! *b*)
    using ‹*b* = *b*′› ‹*b*′ ≤ α (*Max* (*set T*))› ‹*map* (*suffix T*) *?ys* ≠ []›
    by (*metis* ‹*B* ! *b* ≤ *bucket-end* α *T b*› *in-s-current-bucket-def*
          le-eq-less-or-eq list.map-disc-iff list-slice-n-n)
    with *s-pred-invD*
        [*OF s-perm-inv-elims*(*6*)[*OF assms*(*1*)] - ‹*b* ≠ *0*›,
         of B ! *b*]
  obtain *i*′ **where** *i*′*-assms*:
    i′ < *length SA*
    SA ! *i*′ = *Suc* (*SA* ! (*B* ! *b*))
    B ! *b* < *i*′
    i < *i*′
    by *blast*

  let *?b0* = α (*T* ! (*Suc j*))
  and *?b1* = α (*T* ! (*Suc* (*SA* ! (*B* ! *b*))))

  have *i-less*: *i* < *length SA*
    by (*simp add*: *assms*(*4*−*5*))

  have *?b0* ≤ α (*Max* (*set T*))
    by (*metis Max-greD Suc-leI assms*(*1*,*4*−*6*) *strict-mono-leD*
          s-perm-inv-elims(*5*,*8*) *s-seen-invD*(*1*) *lessI*)

  have *?b1* ≤ α (*Max* (*set T*))
    by (*metis Max-greD i*′*-assms*(*1*,*2*,*4*) *assms*(*1*)

less-imp-le-nat s-perm-inv-elims(*5* ,*8*)
s-seen-invD(*1*) *strict-mono-leD*)

have *S0*: *suffix T j* = *T* ! *j* # *suffix T* (*Suc j*)
using *assms*(*7*) *suffix-cons-Suc suffix-type-s-bound*
by *blast*

have *S1*: *suffix T* (*?ys* ! *0*) =
T ! (*?ys* ! *0*) # *suffix T* (*Suc* (*SA* ! (*B* ! *b*)))
using ‹*?ys* ! *0* = *SA* ! (*B* ! *b*)›
‹*suffix-type T* (*?ys* ! *0*) = *S-type*› *suffix-cons-Suc*
suffix-type-s-bound **by** *auto*

have *?b0* ≤ *?b1*
proof(*rule ccontr*)
assume ¬*?b0* ≤ *?b1*
hence *?b1* < *?b0*
by *simp*
hence *bucket-end* α *T ?b1* ≤ *bucket-start* α *T ?b0*
by (*simp add*: *less-bucket-end-le-start*)
with *s-index-upper-bound*[*OF s-perm-inv-elims*(*2* ,*5*)[*OF assms*(*1*)]
i′*-assms*(*1*)]
s-index-lower-bound[*OF s-perm-inv-elims*(*2* ,*5*)[*OF assms*(*1*)] *i-less*,
simplified]
order.*strict-implies-order*[*OF i*′*-assms*(*4*)]
show *False*
using *i*′*-assms*(*2*) *assms*(*4* ,*6*) **by** *auto*
qed
hence *?b0* = *?b1* ∨ *?b0* < *?b1*
by *linarith*
moreover
have *?b0* < *?b1* =⇒
list-less-eq-ns
(*suffix T* (*Suc j*))
(*suffix T* (*Suc* (*SA* ! (*B* ! *b*))))
proof −
assume *?b0* < *?b1*
hence *T* ! (*Suc j*) < *T* ! (*Suc* (*SA* ! (*B* ! *b*)))
using *assms*(*1*) *s-perm-inv-elims*(*8*) *strict-mono-less* **by** *blast*
then show *?thesis*
by (*metis i*′*-assms*(*1* ,*2* ,*4*) *assms*(*1* ,*4* −*6*) *s-perm-inv-def*
leD s-seen-invD(*1*) *list-less-eq-ns-linear*
suffix-cons-Suc list-less-eq-ns-cons)
qed
moreover
have *?b0* = *?b1* =⇒
list-less-eq-ns
(*suffix T* (*Suc j*))
(*suffix T* (*Suc* (*SA* ! (*B* ! *b*))))

proof −
  assume *?b0* = *?b1*
  with *s-index-upper-bound*
        [*OF s-perm-inv-elims*(*2*,*5*)[*OF assms*(*1*)]
      ‹$i' <$ -›] i'*-assms*(*4*)
  have $i' <$ *bucket-end* α *T ?b0*
    by (*simp add*: ‹*SA* ! i' = *Suc* (*SA* ! (*B* ! *b*))›)

  have *suffix-type T* (*SA* ! *i*) = *S-type* ∨
      suffix-type T (*SA* ! *i*) = *L-type*
    using *SL-types.exhaust* **by** *blast*
  moreover
  have *suffix-type T* (*SA* ! *i*) = *S-type* ⟹ *?thesis*
  proof −
    assume *suffix-type T* (*SA* ! *i*) = *S-type*
    with *s-seen-invD*(*3*)
        [*OF s-perm-inv-elims*(*5*)[*OF assms*(*1*)] *i-less*,
         simplified]
    have *in-s-current-bucket* α *T B ?b0 i*
      by (*simp add*: *assms*(*4*) *assms*(*6*))
    hence *B* ! *?b0* ≤ *i*
      using *in-s-current-bucket-def* **by** *blast*
    hence ∃ *m*. *B* ! *?b0* + *m* = *i*
      using *less-eqE* **by** *blast*
    then obtain *m0* **where** *m0-assm*:
      B ! *?b0* + *m0* = *i*
      by *blast*

    from ‹*B* ! *?b0* ≤ *i*› i'*-assms*(*4*)
    have ∃ *m*. *B* ! *?b0* + *m* = i'
      by *presburger*
    then obtain *m1* **where** *m1-assm*:
      B ! *?b0* + *m1* = i'
      by *blast*
    hence *B* ! *?b0* + *m0* ≤ *B* ! *?b0* + *m1*
      by (*simp add*: *m0-assm* i'*-assms*(*4*) *dual-order.order-iff-strict*)
    hence *m0* ≤ *m1*
      using *add-le-imp-le-left* **by** *blast*

    have (*list-slice SA* (*B* ! *?b0*) (*bucket-end* α *T ?b0*)) ! *m0* =
        Suc j
      using *m0-assm i-less*
          ‹*in-s-current-bucket* α *T B ?b0 i*› *assms*(*4*,*6*)
          in-s-current-bucketD(*3*)
      by (*metis* ‹*B* ! α (*T* ! *Suc j*) ≤ *i*› *diff-add-inverse list-slice-nth*)
    moreover
    have (*list-slice SA* (*B* ! *?b0*) (*bucket-end* α *T ?b0*)) ! *m1* =
        Suc (*SA* ! (*B* ! *b*))
      using *m1-assm* i'*-assms*

‹$i' <$ *bucket-end* α *T ?b0*›
by (*metis diff-add-inverse le-add1 list-slice-nth*)
moreover
have *length* (*list-slice SA* (*B* ! *?b0*) (*bucket-end* α *T ?b0*)) =
(*bucket-end* α *T ?b0*) − *B* ! *?b0*
by (*metis assms*(*1*) *bucket-end-le-length length-list-slice min-def*
s-perm-inv-def)
with ‹*B* ! *?b0* + *m1* = i'›
‹$i' <$ *bucket-end* α *T ?b0*›
have *m1* < *length* (*list-slice SA* (*B* ! *?b0*) (*bucket-end* α *T ?b0*))
by *linarith*
ultimately
show *?thesis*
using *ordlistns.sorted-nth-mono*
[*OF s-sorted-invD*[*OF assms*(*3*) ‹*?b0* ≤ α (*Max* -)›]
‹*m0* ≤ *m1*›]
‹*m0* ≤ *m1*›
by *auto*
qed
moreover
have *suffix-type T* (*SA* ! *i*) = *L-type* ⟹ *?thesis*
proof −
assume *suffix-type T* (*SA* ! *i*) = *L-type*
with *s-seen-invD*(*2*)
[*OF s-perm-inv-elims*(*5*)[*OF assms*(*1*)] *i-less*,
simplified]
have *in-l-bucket* α *T ?b0 i*
by (*simp add*: *assms*(*4*) *assms*(*6*))
hence *bucket-start* α *T ?b0* ≤ *i*
by (*simp add*: *in-l-bucket-def*)
hence ∃ *m*. *bucket-start* α *T ?b0* + *m* = *i*
using *less-eqE* **by** *blast*
then obtain *m0* **where** *start-plus-m0-eq*:
bucket-start α *T ?b0* + *m0* = *i*
by *blast*

have *suffix-type T* (*SA* ! i') = *L-type* ∨
suffix-type T (*SA* ! i') = *S-type*
using *SL-types.exhaust* **by** *blast*
moreover
have *suffix-type T* (*SA* ! i') = *S-type* ⟹ *?thesis*
proof −
assume *suffix-type T* (*SA* ! i') = *S-type*

have *SA* ! *i* < *length T*
by (*meson* ‹*i* < *length SA*› *assms*(*1*) *order-refl s-perm-inv-elims*(*5*)
s-seen-invD(*1*))

have *SA* ! i' < *length T*

by (*simp add*: ‹*suffix-type T* (*SA* ! *i*′) = *S-type*› *suffix-type-s-bound*)

from ‹*?b0* = *?b1*›
have *T* ! (*Suc j*) = *T* ! *Suc* (*SA* ! (*B* ! *b*))
  using *assms*(*1*) *s-perm-inv-def strict-mono-eq* **by** *blast*
hence *hd* (*suffix T* (*SA* ! *i*′)) = *hd* (*suffix T* (*SA* ! *i*))
  by (*metis assms*(*4*,*6*) *list.sel*(*1*) *suffix-cons-Suc*
      ‹*SA* ! *i* < *length T*› ‹*SA* ! *i*′ < *length T*›
      ‹*SA* ! *i*′ = *Suc* (*SA* ! (*B* ! *b*))›)
with *l-less-than-s-type*
    [*OF s-perm-inv-elims*(*13*)[*OF assms*(*1*)]
      ‹*SA* ! *i*′ < *length T*›
      ‹*SA* ! *i* < *length T*› -
      ‹*suffix-type T* (*SA* ! *i*′) = -›
      ‹*suffix-type T* (*SA* ! *i*) = -›]
have *list-less-ns* (*suffix T* (*SA* ! *i*)) (*suffix T* (*SA* ! *i*′)).
then show *?thesis*
  by (*simp add*: ‹*SA* ! *i*′ = *Suc* (*SA* ! (*B* ! *b*))› *assms*(*4*,*6*))
qed
moreover
have *suffix-type T* (*SA* ! *i*′) = *L-type* ⟹ *?thesis*
proof −
assume *suffix-type T* (*SA* ! *i*′) = *L-type*
with *s-seen-invD*(*2*)
    [*OF s-perm-inv-elims*(*5*)[*OF assms*(*1*)]
      ‹*i*′ < *length SA*›,
    simplified
      ‹*?b0* = *?b1*›[*symmetric*]
      ‹*SA* ! *i*′ = *Suc* (*SA* ! (*B* ! *b*))›]
      ‹*SA* ! *i*′ = *Suc* (*SA* ! (*B* ! *b*))›
have *in-l-bucket* α *T ?b0 i*′
  by (*simp add*: ‹*i* < *i*′› *dual-order.order-iff-strict*)
hence *i*′*-le-end*: *i*′ < *l-bucket-end* α *T ?b0*
  by (*simp add*: *in-l-bucket-def*)
hence ∃ *m*. *bucket-start* α *T ?b0* + *m* = *i*′
  by (*metis* ‹*in-l-bucket* α *T ?b0 i*′› *in-l-bucket-def less-eqE*)
then obtain *m1* **where** *start-plus-m1-eq*:
  bucket-start α *T ?b0* + *m1* = *i*′
  by *blast*

let *?zs* =
  list-slice SA
    (*bucket-start* α *T ?b0*)
    (*l-bucket-end* α *T ?b0*)

have *?zs* ! *m0* = *Suc j*
  by (*metis* ‹*bucket-start* α *T* (α (*T* ! *Suc j*)) ≤ *i*›
      assms(*4*,*6*) *i*′*-assms*(*4*) *i*′*-le-end*
      diff-add-inverse dual-order.order-iff-strict

i-less list-slice-nth order.strict-trans1
start-plus-m0-eq)
moreover
have *?zs* ! *m1* = *Suc* (*SA* ! (*B* ! *b*))
using *list-slice-nth dual-order.order-iff-strict i′-assms*(*4*)
le-trans [*OF* ‹*bucket-start* α *T* (α (*T* ! *Suc j*)) $\leq$ *i*›]
‹*SA* ! *i′* = *Suc* (*SA* ! (*B* ! *b*))›
‹*bucket-start* α *T ?b0* + *m1* = *i′*›
‹*i′* < *l-bucket-end* α *T ?b0*›
‹*i′* < *length SA*›
by (*metis diff-add-inverse*)
moreover
have *m0* $\leq$ *m1*
using *start-plus-m0-eq start-plus-m1-eq i′-assms*(*4*)
by *linarith*
moreover
have *length ?zs* = *l-bucket-end* α *T ?b0* − *bucket-start* α *T ?b0*
by (*metis* ‹*?b0* $\leq$ α (*Max* (*set T*))› *assms*(*1*)
add-diff-cancel-left′ distinct-card
l-bucket-end-def l-bucket-size-def
l-types-init-def s-perm-inv-def
l-types-init-maintained)
hence *m1* < *length ?zs*
using *start-plus-m1-eq i′-le-end* **by** *linarith*
moreover
from *s-sorted-pre-maintained*
[*OF s-perm-inv-elims*(*2*,*4*,*10*,*11*)[*OF assms*(*1*)]
assms(*2*)]
have *s-sorted-pre* α *T SA* **.**
ultimately show *?thesis*
using *ordlistns.sorted-nth-mono*
[*OF s-sorted-preD*
[*of* α *T SA*,
OF - ‹*?b0* $\leq$ α (*Max* -)›],
of m0 m1]
by (*simp add*: ‹*m1* < *length ?zs*› *le-less-trans*
‹*s-sorted-pre* α *T SA*›)
qed
ultimately show *?thesis*
by *blast*
qed
ultimately show *?thesis*
by *blast*
qed
ultimately
have *list-less-eq-ns*
(*suffix T* (*Suc j*))
(*suffix T* (*Suc* (*SA* ! (*B* ! *b*))))
by *blast*

with *S0 S1* ‹T ! j = T ! (*?ys* ! *0*)›
show *?thesis*
by (*simp add*: *list-less-eq-ns-cons*)
qed
ultimately
show *list-less-eq-ns* (*suffix T j*) (*map* (*suffix T*) *?ys* ! *0*)
by *simp*
qed
ultimately show *?thesis*
using *A* **by** *fastforce*
qed
ultimately show *?thesis*
by *simp*
qed
moreover
have $b \neq b' \Longrightarrow$ *ordlistns.sorted* (*map* (*suffix T*) *?xs*)
proof −
assume $b \neq b'$
with *s-sorted-maintained-unchanged-step*[*OF assms*(*1*,*4*−) ‹$b' \leq$ -›]
s-sorted-invD[*OF assms*(*3*) ‹$b' \leq$ -›]
show *?thesis*
using *ordlistns.sorted-map* **by** *blast*
qed
ultimately show *ordlistns.sorted* (*map* (*suffix T*) *?xs*)
by *blast*
qed

lemma *s-prefix-sorted-inv-maintained-step*:
assumes *s-perm-inv* α *T B SA0 SA i*
and *s-prefix-sorted-pre* α *T SA0*
and *s-prefix-sorted-inv* α *T B SA*
and $i = Suc\ n$
and $Suc\ n < length\ SA$
and $SA\ !\ Suc\ n = Suc\ j$
and *suffix-type T j* = *S-type*
and $b = \alpha\ (T\ !\ j)$
and $k = B\ !\ b - Suc\ 0$
shows *s-prefix-sorted-inv* α *T* ($B[b := k]$) ($SA[k := j]$)
unfolding *s-prefix-sorted-inv-def*
proof (*safe*)
fix b'
assume $b' \leq \alpha\ (Max\ (set\ T))$
let *?xs* = *list-slice* ($SA[k := j]$) ($B[b := k]\ !\ b'$) (*bucket-end* α *T* b')

have *bucket-end* α *T b* $\leq$ *length T*
using *bucket-end-le-length* **by** *blast*
moreover
have $B\ !\ b \leq$ *bucket-end* α *T b*
using *assms*(*1*,*7*,*8*) *s-bucket-ptr-upper-bound s-perm-inv-elims*(*2*,*8*) *strict-mono-less-eq*

suffix-type-s-bound **by** *fastforce*

ultimately have $k < length\ T$
using *assms*(*1*,*9*) *s-perm-inv-elims*(*15*) **by** *fastforce*
hence $k < length\ SA$
by (*metis assms*(*1*) *s-perm-inv-def*)

from *s-bucket-ptr-strict-lower-bound*
[*OF s-perm-inv-elims*(*1*−*6*,*8*,*10*−*14*)[*OF assms*(*1*)] *assms*(*4*−*8*)]
have *s-bucket-start* α $T\ b < B\ !\ b$ **.**
hence $k < B\ !\ b$
using *assms*(*9*) *diff-less gr-implies-not-zero* **by** *blast*

have *s-bucket-start* α $T\ b \leq k$
using *assms s-bucket-ptr-strict-lower-bound s-perm-inv-def* **by** *fastforce*
hence *bucket-start* α $T\ b \leq k$
using *bucket-start-le-s-bucket-start le-trans* **by** *blast*
hence $b \leq \alpha$ (*Max* (*set* T))
by (*metis* ‹$k < length\ SA$› *assms*(*1*) *bucket-end-Max dual-order.trans*
leD leI *s-perm-inv-elims*(*8*,*11*) *less-bucket-end-le-start*)

have $b = b' \lor b \neq b'$
by *blast*
moreover
have $b = b' \Longrightarrow$ *ordlistns.sorted* (*map* (*lms-slice* T) *?xs*)
proof −
assume $b = b'$
hence $B[b := k]\ !\ b' = k$
by (*meson* ‹$b' \leq \alpha$ (*Max* (*set* T))› *assms*(*1*) *le-less-trans*
nth-list-update-eq s-perm-inv-elims(*9*))

have $SA[k := j]\ !\ k = j$
by (*simp add*: ‹$k < length\ SA$›)

from *list-slice-update-unchanged-1*[*OF* ‹$k < B\ !\ b$›]
‹$k < B\ !\ b$› ‹$SA[k := j]\ !\ k = j$› ‹$B[b := k]\ !\ b' = k$›
‹$B\ !\ b \leq$ *bucket-end* α $T\ b$›
‹$b = b'$› ‹$k < length\ SA$›
have *?xs* $= j$ # *list-slice SA* ($B\ !\ b$) (*bucket-end* α $T\ b$)
by (*metis Suc-pred assms*(*9*) *length-list-update not-le*
less-nat-zero-code list-slice-Suc less-le-trans)
moreover
have *ordlistns.sorted*
(*map* (*lms-slice* T)
(j # *list-slice SA* ($B\ !\ b$)
(*bucket-end* α $T\ b$)))
proof −
let *?ys* = *list-slice SA* ($B\ !\ b$) (*bucket-end* α $T\ b$)

have A: *map* (*lms-slice* T) (j # *?ys*) = (*lms-slice* T j) # *map* (*lms-slice* T) *?ys*
  by *simp*

from *s-prefix-sorted-invD*[*OF assms*(*3*) ‹$b \leq$ -›]
have B: *ordlistns.sorted* (*map* (*lms-slice* T) *?ys*) **.**

have *?ys* = [] ∨ *?ys* ≠ []
  by *blast*
hence *map* (*lms-slice* T) *?ys* = [] ∨ *map* (*lms-slice* T) *?ys* ≠ []
  by *simp*
moreover
have *map* (*lms-slice* T) *?ys* = [] ⟹ *?thesis*
  using *ordlistns.sorted-cons-nil* **by** *fastforce*
moreover
have *map* (*lms-slice* T) *?ys* ≠ [] ⟹
    ordlistns.sorted ((*lms-slice* T j) # *map* (*lms-slice* T) *?ys*)
proof (*rule ordlistns.sorted-consI*[*OF* - B])
  assume *map* (*lms-slice* T) *?ys* ≠ []
  hence *map* (*lms-slice* T) *?ys* ! *0* = *lms-slice* T (*?ys* ! *0*)
    by (*metis length-greater-0-conv list.simps*(*8*) *nth-map*)
  moreover
  have *list-less-eq-ns* (*lms-slice* T j) (*lms-slice* T (*?ys* ! *0*))
  proof −
    have *?ys* ! *0* ∈ *s-bucket* α T b
      by (*metis* ‹$b = b'$› ‹$b' \leq \alpha$ (*Max* (*set* T))› ‹*map* (*lms-slice* T) *?ys* ≠ []› *assms*(*1*)
          length-greater-0-conv length-map nth-mem s-locations-inv-in-list-slice
            s-perm-inv-elims(*3*))
    hence α (T ! (*?ys* ! *0*)) = b *suffix-type* T (*?ys* ! *0*) = *S-type*
      by (*simp add*: *s-bucket-def bucket-def*)+
    hence T ! j = T ! (*?ys* ! *0*)
      using *assms*(*1*,*8*) *s-perm-inv-elims*(*8*) *strict-mono-eq* **by** *fastforce*

    have *?ys* ! *0* = *SA* ! (B ! b)
      using ‹*map* (*lms-slice* T) *?ys* ≠ []› *nth-list-slice* **by** *fastforce*

    have $b \neq 0$
      by (*metis* ‹*s-bucket-start* α T b < B ! b› *assms*(*1*)
          gr-implies-not-zero s-bucket-ptr-0 s-perm-inv-elims(*2*))

    have *in-s-current-bucket* α T B b (B ! b)
      using ‹$b = b'$› ‹$b' \leq \alpha$ (*Max* (*set* T))›
          ‹*map* (*lms-slice* T) *?ys* ≠ []›
          in-s-current-bucket-def
      by (*metis* ‹B ! b ≤ *bucket-end* α T b›
            dual-order.order-iff-strict list.map-disc-iff
            list-slice-n-n)
      with *s-pred-invD*

[*OF s-perm-inv-elims*(*6*)[*OF assms*(*1*)] - ‹*b* ≠ *0*›,
of B ! *b*]

obtain *i'* **where**
i' < *length SA*
SA ! *i'* = *Suc* (*SA* ! (*B* ! *b*))
B ! *b* < *i'*
i < *i'*
by *blast*

let *?b0* = α (*T* ! (*Suc j*))
and *?b1* = α (*T* ! (*Suc* (*SA* ! (*B* ! *b*))))

have *i* < *length SA*
by (*simp add*: *assms*(*4*−*5*))

have *?b0* ≤ α (*Max* (*set T*))
by (*metis Max-greD Suc-leI assms*(*1*,*4*−*6*) *lessI s-perm-inv-elims*(*5*,*8*)
s-seen-invD(*1*)
strict-mono-leD)

have *?b1* ≤ α (*Max* (*set T*))
by (*metis Max-greD* ‹*SA* ! *i'* = *Suc* (*SA* ! (*B* ! *b*))› ‹*i* < *i'*› ‹*i'* < *length*
SA› *assms*(*1*)
less-imp-le-nat s-perm-inv-elims(*5*) *s-perm-inv-elims*(*8*) *s-seen-invD*(*1*)
strict-mono-leD)

have *S0*: *lms-slice T j* = *T* ! *j* # *lms-slice T* (*Suc j*)
using *assms*(*7*) *lms-slice-cons suffix-type-s-bound* **by** *blast*

have *S1*:
lms-slice T (*?ys* ! *0*) = *T* ! (*?ys* ! *0*) # *lms-slice T* (*Suc* (*SA* ! (*B* ! *b*)))
using ‹*?ys* ! *0* = *SA* ! (*B* ! *b*)› ‹*suffix-type T* (*?ys* ! *0*) = *S-type*›
lms-slice-cons
suffix-type-s-bound **by** *auto*

have *?b0* ≤ *?b1*
proof(*rule ccontr*)
assume ¬*?b0* ≤ *?b1*
hence *?b1* < *?b0*
by *simp*
hence *bucket-end* α *T ?b1* ≤ *bucket-start* α *T ?b0*
by (*simp add*: *less-bucket-end-le-start*)
with *s-index-upper-bound*[*OF s-perm-inv-elims*(*2*,*5*)[*OF assms*(*1*)] ‹*i'* <
length SA›]
s-index-lower-bound[*OF s-perm-inv-elims*(*2*,*5*)[*OF assms*(*1*)] ‹*i* <
length SA›,
simplified]
order.strict-implies-order[*OF* ‹*i* < *i'*›]
show *False*

using ‹SA ! i' = Suc (SA ! (B ! b))› $assms(4,6)$ **by** $auto$
qed
hence $?b0 = ?b1 \lor ?b0 < ?b1$
by $linarith$
moreover
have
$?b0 < ?b1 \Longrightarrow$
list-less-eq-ns (*lms-slice* T (Suc j)) (*lms-slice* T (Suc (SA ! (B ! b))))
proof −
assume $?b0 < ?b1$
hence T ! (Suc j) < T ! (Suc (SA ! (B ! b)))
using *assms*(1) *s-perm-inv-elims*(8) *strict-mono-less* **by** *blast*
moreover
have Suc j < *length* T
using ‹i < *length* SA› *assms*(1,4,6) *s-perm-inv-elims*(5) *s-seen-invD*(1)
by *fastforce*
hence ∃ *as*. *lms-slice* T (Suc j) = T ! (Suc j) # *as*
by (*metis dual-order.strict-trans abs-find-next-lms-lower-bound-1 lessI*
list-slice-Suc
lms-slice-def)
then obtain *as* **where**
lms-slice T (Suc j) = T ! (Suc j) # *as*
by *blast*
moreover
have Suc (SA ! (B ! b)) < *length* T
using ‹SA ! i' = Suc (SA ! (B ! b))› ‹$i < i'$› ‹i' < *length* SA› *assms*(1)
s-perm-inv-elims(5) *s-seen-invD*(1) **by** *fastforce*
hence ∃ *bs*. *lms-slice* T (Suc (SA ! (B ! b))) = T ! (Suc (SA ! (B ! b)))
bs
by (*metis abs-find-next-lms-lower-bound-1 less-Suc-eq*
list-slice-Suc lms-slice-def)
then obtain *bs* **where**
lms-slice T (Suc (SA ! (B ! b))) = T ! (Suc (SA ! (B ! b))) # *bs*
by *blast*
ultimately show *?thesis*
using *list-less-eq-ns-cons* **by** *fastforce*
qed
moreover
have
$?b0 = ?b1 \Longrightarrow$
list-less-eq-ns (*lms-slice* T (Suc j)) (*lms-slice* T (Suc (SA ! (B ! b))))
proof −
assume $?b0 = ?b1$
with *s-index-upper-bound*[*OF s-perm-inv-elims*(2,5)[*OF assms*(1)] ‹i' <
-›] ‹$i < i'$›
have i' < *bucket-end* α T $?b0$
by (*simp add*: ‹SA ! i' = Suc (SA ! (B ! b))›)

have *suffix-type* T (SA ! i) = *S-type* ∨ *suffix-type* T (SA ! i) = *L-type*

using *SL-types.exhaust* **by** *blast*
moreover
have *suffix-type T (SA ! i) = S-type* $\Longrightarrow$ *?thesis*
proof −
assume *suffix-type T (SA ! i) = S-type*
with *s-seen-invD(3)[OF s-perm-inv-elims(5)[OF assms(1)] ‹i < length SA›, simplified]*
have *in-s-current-bucket* α *T B ?b0 i*
by *(simp add: assms(4) assms(6))*
hence *B ! ?b0* $\leq$ *i*
using *in-s-current-bucket-def* **by** *blast*
hence $\exists$ *m. B ! ?b0 + m = i*
using *less-eqE* **by** *blast*
then obtain *m0* **where**
B ! ?b0 + m0 = i
by *blast*

from *‹B ! ?b0* $\leq$ *i› ‹i < i′›*
have $\exists$ *m. B ! ?b0 + m = i′*
by *presburger*
then obtain *m1* **where**
B ! ?b0 + m1 = i′
by *blast*
hence *B ! ?b0 + m0* $\leq$ *B ! ?b0 + m1*
by *(simp add: ‹B !* α *(T ! Suc j) + m0 = i›*
‹i < i′› dual-order.order-iff-strict)
hence *m0* $\leq$ *m1*
using *add-le-imp-le-left* **by** *blast*

have *(list-slice SA (B ! ?b0) (bucket-end* α *T ?b0)) ! m0 = Suc j*
using *‹B !* α *(T ! Suc j) + m0 = i› ‹i < length SA›*
‹in-s-current-bucket α *T B ?b0 i› assms(4,6)*
in-s-current-bucketD(3)
by *(metis ‹B !* α *(T ! Suc j)* $\leq$ *i› diff-add-inverse list-slice-nth)*
moreover
have *(list-slice SA (B ! ?b0) (bucket-end* α *T ?b0)) ! m1 = Suc (SA ! (B ! b))*
using *‹B ! ?b0 + m1 = i′› ‹SA ! i′ = Suc (SA ! (B ! b))› ‹i′ < bucket-end* α *T ?b0›*
‹i′ < length SA›
by *(metis diff-add-inverse le-add1 list-slice-nth)*
moreover
have *length (list-slice SA (B ! ?b0) (bucket-end* α *T ?b0))*
= (bucket-end α *T ?b0) − B ! ?b0*
by *(metis assms(1) bucket-end-le-length length-list-slice min-def s-perm-inv-def)*
with *‹B ! ?b0 + m1 = i′› ‹i′ < bucket-end* α *T ?b0›*
have *m1 < length (list-slice SA (B ! ?b0) (bucket-end* α *T ?b0))*
by *linarith*

ultimately
show *?thesis*
using *ordlistns.sorted-nth-mono*[*OF*
s-prefix-sorted-invD[*OF assms*(*3*) ‹*?b0* $\leq$ α (*Max* -)›] ‹*m0* $\leq$ *m1*›]
‹*m0* $\leq$ *m1*› **by** *auto*
qed
moreover
have *suffix-type T* (*SA* ! *i*) = *L-type* $\Longrightarrow$ *?thesis*
proof −
assume *suffix-type T* (*SA* ! *i*) = *L-type*
with *s-seen-invD*(*2*)[*OF s-perm-inv-elims*(*5*)[*OF assms*(*1*)] ‹*i* < *length SA*›, *simplified*]
have *in-l-bucket* α *T ?b0 i*
by (*simp add*: *assms*(*4*) *assms*(*6*))
hence *bucket-start* α *T ?b0* $\leq$ *i*
by (*simp add*: *in-l-bucket-def*)
hence $\exists$ *m*. *bucket-start* α *T ?b0* + *m* = *i*
using *less-eqE* **by** *blast*
then obtain *m0* **where**
bucket-start α *T ?b0* + *m0* = *i*
by *blast*

have *suffix-type T* (*SA* ! *i'*) = *L-type* $\vee$ *suffix-type T* (*SA* ! *i'*) = *S-type*
using *SL-types.exhaust* **by** *blast*
moreover
have *suffix-type T* (*SA* ! *i'*) = *S-type* $\Longrightarrow$ *?thesis*
proof −
assume *suffix-type T* (*SA* ! *i'*) = *S-type*

have *SA* ! *i* < *length T*
by (*meson* ‹*i* < *length SA*› *assms*(*1*) *order-refl s-perm-inv-elims*(*5*) *s-seen-invD*(*1*))

have *SA* ! *i'* < *length T*
by (*simp add*: ‹*suffix-type T* (*SA* ! *i'*) = *S-type*› *suffix-type-s-bound*)

from ‹*?b0* = *?b1*›
have *T* ! (*Suc j*) = *T* ! *Suc* (*SA* ! (*B* ! *b*))
using *assms*(*1*) *s-perm-inv-def strict-mono-eq* **by** *blast*
hence *hd* (*suffix T* (*SA* ! *i'*)) = *hd* (*suffix T* (*SA* ! *i*))
by (*metis* ‹*SA* ! *i* < *length T*› ‹*SA* ! *i'* < *length T*› ‹*SA* ! *i'* = *Suc* (*SA* ! (*B* ! *b*))›
assms(*4*) *assms*(*6*) *list.sel*(*1*) *suffix-cons-Suc*)
hence *list-less-ns* (*lms-slice T* (*SA* ! *i*)) (*lms-slice T* (*SA* ! *i'*))
using ‹*SA* ! *i* < *length T*› ‹*SA* ! *i'* < *length T*› ‹*SA* ! *i'* = *Suc* (*SA* ! (*B* ! *b*))›
‹*T* ! *Suc j* = *T* ! *Suc* (*SA* ! (*B* ! *b*))› ‹*suffix-type T* (*SA* ! *i'*) = *S-type*›

‹*suffix-type T* (*SA* ! *i*) = *L-type*› *assms*(*1*,*4*,*6*) *s-perm-inv-elims*(*13*)
lms-slice-l-less-than-s-type **by** *fastforce*
then show *?thesis*
by (*simp add*: ‹*SA* ! *i'* = *Suc* (*SA* ! (*B* ! *b*))› *assms*(*4*,*6*))
qed
moreover
have *suffix-type T* (*SA* ! *i'*) = *L-type* ⟹ *?thesis*
proof −
assume *suffix-type T* (*SA* ! *i'*) = *L-type*
with *s-seen-invD*(*2*)[*OF s-perm-inv-elims*(*5*)[*OF assms*(*1*)] ‹*i'* < *length SA*›,
simplified ‹*?b0* = *?b1*›[*symmetric*] ‹*SA* ! *i'* = *Suc* (*SA* ! (*B* ! *b*))›]
‹*SA* ! *i'* = *Suc* (*SA* ! (*B* ! *b*))›
have *in-l-bucket* α *T ?b0 i'*
by (*simp add*: ‹*i* < *i'*› *dual-order.order-iff-strict*)
hence *i'* < *l-bucket-end* α *T ?b0*
by (*simp add*: *in-l-bucket-def*)
hence ∃ *m*. *bucket-start* α *T ?b0* + *m* = *i'*
by (*metis* ‹*in-l-bucket* α *T ?b0 i'*› *in-l-bucket-def less-eqE*)
then obtain *m1* **where**
bucket-start α *T ?b0* + *m1* = *i'*
by *blast*

let *?zs* = *list-slice SA* (*bucket-start* α *T ?b0*) (*l-bucket-end* α *T ?b0*)

have *?zs* ! *m0* = *Suc j*
using ‹*bucket-start* α *T ?b0* + *m0* = *i*› ‹*i* < *i'*› ‹*i* < *length SA*›
‹*i'* < *l-bucket-end* α *T ?b0*› *assms*(*4*,*6*)
by (*metis* ‹*bucket-start* α *T* (α (*T* ! *Suc j*)) ≤ *i*›
diff-add-inverse dual-order.order-iff-strict
list-slice-nth order.strict-trans1)
moreover
have *?zs* ! *m1* = *Suc* (*SA* ! (*B* ! *b*))
using ‹*SA* ! *i'* = *Suc* (*SA* ! (*B* ! *b*))› ‹*bucket-start* α *T ?b0* + *m1* = *i'*›
‹*i'* < *l-bucket-end* α *T ?b0*› ‹*i'* < *length SA*›
by (*metis* ‹*in-l-bucket* α *T* (α (*T* ! *Suc j*)) *i'*›
diff-add-inverse in-l-bucket-def list-slice-nth)
moreover
have *m0* ≤ *m1*
using ‹*bucket-start* α *T ?b0* + *m0* = *i*› ‹*bucket-start* α *T ?b0* + *m1* = *i'*› ‹*i* < *i'*›
by *linarith*
moreover
have *length ?zs* = *l-bucket-end* α *T ?b0* − *bucket-start* α *T ?b0*
by (*metis* ‹*?b0* ≤ α (*Max* (*set T*))› *add-diff-cancel-left'* *assms*(*1*) *distinct-card*
l-bucket-end-def l-bucket-size-def l-types-init-def l-types-init-maintained
s-perm-inv-def)

hence *m1* < *length ?zs*
using ‹*bucket-start* α *T ?b0* + *m1* = *i*′› ‹*i*′ < *l-bucket-end* α *T ?b0*› **by** *linarith*
moreover
from *s-prefix-sorted-pre-maintained*[*OF s-perm-inv-elims*(*2*,*4*,*10*,*11*)[*OF assms*(*1*)]
assms(*2*)]
have *s-prefix-sorted-pre* α *T SA* **.**
ultimately show *?thesis*
using *ordlistns.sorted-nth-mono*[*OF s-prefix-sorted-preD*[*of* α *T SA*,
OF - ‹*?b0* ≤ α (*Max* -)›], *of m0 m1*]
by (*simp add*: ‹*m1* < *length ?zs*› ‹*s-prefix-sorted-pre* α *T SA*› *le-less-trans*)
qed
ultimately show *?thesis*
by *blast*
qed
ultimately show *?thesis*
by *blast*
qed
ultimately have
list-less-eq-ns (*lms-slice T* (*Suc j*)) (*lms-slice T* (*Suc* (*SA* ! (*B* ! *b*))))
by *blast*
with *S0 S1* ‹*T* ! *j* = *T* ! (*?ys* ! *0*)›
show *?thesis*
by (*simp add*: *list-less-eq-ns-cons*)
qed
ultimately show *list-less-eq-ns* (*lms-slice T j*) (*map* (*lms-slice T*) *?ys* ! *0*)
by *simp*
qed
ultimately show *?thesis*
using *A* **by** *fastforce*
qed
ultimately show *?thesis*
by *simp*
qed
moreover
have *b* ≠ *b*′ ⟹ *ordlistns.sorted* (*map* (*lms-slice T*) *?xs*)
proof −
assume *b* ≠ *b*′
with *s-sorted-maintained-unchanged-step*[*OF assms*(*1*,*4*−) ‹*b*′ ≤ -›]
s-prefix-sorted-invD[*OF assms*(*3*) ‹*b*′ ≤ -›]
show *?thesis*
using *ordlistns.sorted-map* **by** *blast*
qed
ultimately show *ordlistns.sorted* (*map* (*lms-slice T*) *?xs*)
by *blast*
qed

theorem *abs-induce-s-sorted-step*:
  assumes *s-perm-inv* α *T B SA0 SA i*
  and *s-sorted-pre* α *T SA0*
  and *s-sorted-inv* α *T B SA*
  and *abs-induce-s-step* (B, SA, i) $(\alpha, T) = (B', SA', i')$
shows *s-sorted-inv* α *T* B' SA'
proof (*rule abs-induce-s-step.elims*[*OF assms*(*4*)];
    clarsimp simp: *assms*(*3*,*4*) *Let-def not-less*
      split: *if-splits SL-types.splits nat.splits*[**where** *?nat* = *i*] *nat.splits*)
  assume $B = B'$ $SA' = SA$
  with *assms*(*3*)
  show *s-sorted-inv* α *T* B' *SA*
    by *simp*
next
  assume $B = B'$ $SA' = SA$
  with *assms*(*3*)
  show *s-sorted-inv* α *T* B' *SA*
    by *simp*
next
  assume $B = B'$ $SA' = SA$
  with *assms*(*3*)
  show *s-sorted-inv* α *T* B' *SA*
    by *simp*
next
  assume $B = B'$ $SA' = SA$
  with *assms*(*3*)
  show *s-sorted-inv* α *T* B' *SA*
    by *simp*
next
  fix *j*

  let *?b* = α (*T* ! *j*)
  let *?k* = *B* ! *?b* − *Suc 0*

  assume *A*: *i* = *Suc* i' *Suc* i' < *length SA SA* ! *Suc* i' = *Suc j suffix-type T j* = *S-type*
      B' = *B*[*?b* := *?k*] SA' = *SA*[*?k* := *j*]

  from *s-sorted-inv-maintained-step*[*OF assms*(*1*−*3*) *A*(*1*−*4*), *of ?b ?k*, *simplified*]
  show *s-sorted-inv* α *T* (*B*[*?b* := *?k*]) (*SA*[*?k* := *j*]) **.**
qed

corollary *abs-induce-s-sorted-step-alt*:
$\bigwedge$*a*. *s-sorted-inv-alt* α *T SA0 a* $\Longrightarrow$ *s-sorted-inv-alt* α *T SA0* (*abs-induce-s-step a* (α, *T*))
proof −
  fix *a*
  assume *s-sorted-inv-alt* α *T SA0 a*

have $\exists$ *B SA i*. *a* = (*B*, *SA*, *i*)
  by (*meson prod-cases3*)
then obtain *B SA i* **where**
  a = (*B*, *SA*, *i*)
  by *blast*
with ‹*s-sorted-inv-alt* α *T SA0 a*›
have *P*: *s-perm-inv* α *T B SA0 SA i s-sorted-pre* α *T SA0 s-sorted-inv* α *T B SA*
  by *simp-all*

from *abs-induce-s-step-ex*[*of* (*B*, *SA*, *i*) (α, *T*)]
obtain *B′ SA′ i′* **where**
  Q: *abs-induce-s-step* (*B*, *SA*, *i*) (α, *T*) = (*B′*, *SA′*, *i′*)
  by *blast*

from *abs-induce-s-sorted-step*[*OF P Q*] *abs-induce-s-perm-step*[*OF P*(*1*) *Q*] ‹*s-sorted-pre* - - -›
show *s-sorted-inv-alt* α *T SA0* (*abs-induce-s-step a* (α, *T*))
  using *Q* ‹*a* = (*B*, *SA*, *i*)› **by** *auto*
qed

theorem *abs-induce-s-sorted-alt-maintained*:
assumes *s-sorted-inv-alt* α *T SA0* (*B*, *SA*, *length T*)
shows *s-sorted-inv-alt* α *T SA0* (*abs-induce-s-base* α *T B SA*)
unfolding *abs-induce-s-base-def*
using *repeat-maintain-inv*
  [*of s-sorted-inv-alt* α *T SA0 abs-induce-s-step* (α, *T*), *OF - assms*(*1*)]
  abs-induce-s-sorted-step-alt
by *blast*

corollary *abs-induce-s-sorted-maintained*:
assumes *abs-induce-s-base* α *T B SA* = (*B′*, *SA′*, *n*)
and *s-perm-inv* α *T B SA0 SA* (*length T*)
and *s-sorted-pre* α *T SA0*
and *s-sorted-inv* α *T B SA*
shows *s-sorted-inv* α *T B′ SA′*
using *assms abs-induce-s-sorted-alt-maintained* **by** *force*

theorem *abs-induce-s-prefix-sorted-step*:
assumes *s-perm-inv* α *T B SA0 SA i*
and *s-prefix-sorted-pre* α *T SA0*
and *s-prefix-sorted-inv* α *T B SA*
and *abs-induce-s-step* (*B*, *SA*, *i*) (α, *T*) = (*B′*, *SA′*, *i′*)
shows *s-prefix-sorted-inv* α *T B′ SA′*
proof (*rule abs-induce-s-step.elims*[*OF assms*(*4*)];
  clarsimp simp: *assms*(*3*,*4*) *Let-def not-less*
    split: *if-splits SL-types.splits nat.splits*[**where** *?nat* = *i*] *nat.splits*)
assume *B* = *B′*
with *assms*(*3*)

show *s-prefix-sorted-inv* α T B' SA
  by *simp*
next
  assume $B = B'$
  with *assms*(*3*)
  show *s-prefix-sorted-inv* α T B' SA
    by *simp*
next
  assume $B = B'$
  with *assms*(*3*)
  show *s-prefix-sorted-inv* α T B' SA
    by *simp*
next
  assume $B = B'$
  with *assms*(*3*)
  show *s-prefix-sorted-inv* α T B' SA
    by *simp*
next
  fix j

  let $?b = \alpha\ (T\ !\ j)$
  let $?k = B\ !\ ?b - Suc\ 0$

  assume A: $i = Suc\ i'$ $Suc\ i' < length\ SA$ $SA\ !\ Suc\ i' = Suc\ j$ *suffix-type* $T\ j$ = *S-type*
    $B' = B[?b := ?k]$ $SA' = SA[?k := j]$

  from *s-prefix-sorted-inv-maintained-step*[*OF assms*(*1−3*) *A*(*1−4*), *of ?b ?k*, *simplified*]
  show *s-prefix-sorted-inv* α T $(B[?b := ?k])$ $(SA[?k := j])$ **.**
qed

corollary *abs-induce-s-prefix-sorted-step-alt*:
$\bigwedge a$. *s-prefix-sorted-inv-alt* α T $SA0$ $a \Longrightarrow$
    s-prefix-sorted-inv-alt α T $SA0$ (*abs-induce-s-step* a $(\alpha,\ T)$)
proof −
  fix a
  assume *s-prefix-sorted-inv-alt* α T $SA0$ a

  have $\exists\, B\ SA\ i.\ a = (B,\ SA,\ i)$
    by (*meson prod-cases3*)
  then obtain B SA i **where**
    $a = (B,\ SA,\ i)$
    by *blast*
  with ‹*s-prefix-sorted-inv-alt* α T $SA0$ a›
  have P: *s-perm-inv* α T B $SA0$ SA i *s-prefix-sorted-pre* α T $SA0$ *s-prefix-sorted-inv* α T B SA
    by *simp-all*

from *abs-induce-s-step-ex*[*of* (*B*, *SA*, *i*) (α, *T*)]
obtain B' SA' i' **where**
Q: *abs-induce-s-step* (*B*, *SA*, *i*) (α, *T*) = (B', SA', i')
by *blast*

from *abs-induce-s-prefix-sorted-step*[*OF P Q*] *abs-induce-s-perm-step*[*OF P*(*1*) *Q*]
‹*s-prefix-sorted-pre - - -*›
show *s-prefix-sorted-inv-alt* α *T SA0* (*abs-induce-s-step a* (α, *T*))
using *Q* ‹*a* = (*B*, *SA*, *i*)› **by** *auto*
qed

theorem *abs-induce-s-prefix-sorted-alt-maintained*:
assumes *s-prefix-sorted-inv-alt* α *T SA0* (*B*, *SA*, *length T*)
shows *s-prefix-sorted-inv-alt* α *T SA0* (*abs-induce-s-base* α *T B SA*)
unfolding *abs-induce-s-base-def*
using *repeat-maintain-inv*[*of s-prefix-sorted-inv-alt* α *T SA0 abs-induce-s-step* (α,
T),
OF - assms(*1*)]
abs-induce-s-prefix-sorted-step-alt
by *blast*

corollary *abs-induce-s-prefix-sorted-maintained*:
assumes *abs-induce-s-base* α *T B SA* = (B', SA', *n*)
and *s-perm-inv* α *T B SA0 SA* (*length T*)
and *s-prefix-sorted-pre* α *T SA0*
and *s-prefix-sorted-inv* α *T B SA*
shows *s-prefix-sorted-inv* α *T* B' SA'
using *assms abs-induce-s-prefix-sorted-alt-maintained* **by** *force*

theorem *s-sorted-bucket*:
assumes *s-perm-inv* α *T B SA0 SA 0*
and *s-sorted-pre* α *T SA0*
and *s-sorted-inv* α *T B SA*
and $b \le \alpha$ (*Max* (*set T*))
shows *ordlistns.sorted* (*map* (*suffix T*) (*list-slice SA* (*bucket-start* α *T b*) (*bucket-end*
α *T b*)))
(**is** *ordlistns.sorted* (*map* (*suffix T*) *?xs*))
proof −
let *?ys* = *list-slice SA* (*bucket-start* α *T b*) (*l-bucket-end* α *T b*)
and *?zs* = *list-slice SA* (*s-bucket-start* α *T b*) (*bucket-end* α *T b*)

from *s-perm-inv-0-B-val*[*OF assms*(*1*,*4*)]
have *B* ! *b* = *s-bucket-start* α *T b* **.**

from *s-sorted-pre-maintained*[*OF s-perm-inv-elims*(*2*,*4*,*10*,*11*)[*OF assms*(*1*)] *assms*(*2*)]
have *s-sorted-pre* α *T SA* **.**

have *?xs* = *?ys* @ *?zs*
by (*metis bucket-start-le-s-bucket-start l-bucket-end-le-bucket-end list-slice-append*

s-bucket-start-eq-l-bucket-end)
hence *map* (*suffix T*) *?xs* = *map* (*suffix T*) *?ys* @ *map* (*suffix T*) *?zs*
by *simp*
moreover
from *s-sorted-preD*[*OF* ‹*s-sorted-pre - - SA*› ‹*b* ≤ -›]
have *ordlistns.sorted* (*map* (*suffix T*) *?ys*) **.**
moreover
from *s-sorted-invD*[*OF assms*(*3*,*4*), *simplified* ‹- = *s-bucket-start* α - -›]
have *ordlistns.sorted* (*map* (*suffix T*) *?zs*) **.**
moreover
have ∀ *y* ∈ *set* (*map* (*suffix T*) *?ys*). ∀ *z* ∈ *set* (*map* (*suffix T*) *?zs*). *list-less-eq-ns y z*
proof(*safe*)
fix *y z*
assume *y* ∈ *set* (*map* (*suffix T*) *?ys*) *z* ∈ *set* (*map* (*suffix T*) *?zs*)
with *in-set-mapD*[*of y suffix T ?ys*]
in-set-mapD[*of z suffix T ?zs*]
obtain *i j* **where**
y = *suffix T i i* ∈ *set ?ys*
z = *suffix T j j* ∈ *set ?zs*
by *blast*

from *l-types-initD*(*1*)[*OF l-types-init-maintained*[*OF s-perm-inv-elims*(*2*,*4*,*10*−*12*)[*OF assms*(*1*)]]
‹*b* ≤ -›]
‹*i* ∈ *set ?ys*›
have *i* ∈ *l-bucket* α *T b*
by *blast*
hence *suffix-type T i* = *L-type* α (*T* ! *i*) = *b i* < *length T*
by (*simp add*: *l-bucket-def bucket-def*)+
moreover
from *s-bucket-eq-list-slice*[*OF s-perm-inv-elims*(*1*,*3*,*11*)[*OF assms*(*1*)] ‹*b* ≤ -›
‹*B* ! *b* = *s-bucket-start* α - -›]
‹*j* ∈ *set ?zs*›
have *j* ∈ *s-bucket* α *T b*
by *blast*
hence *suffix-type T j* = *S-type* α (*T* ! *j*) = *b j* < *length T*
by (*simp add*: *s-bucket-def bucket-def*)+
moreover
have *T* ! *i* = *T* ! *j*
using *calculation*(*2*,*5*) *s-perm-inv-elims*(*8*)[*OF assms*(*1*)] *strict-mono-eq* **by** *fastforce*
ultimately have *list-less-eq-ns* (*suffix T i*) (*suffix T j*)
using *l-less-than-s-type-suffix*[*of j T i*] **by** *simp*
then show *list-less-eq-ns y z*
using ‹*y* = *suffix T i*› ‹*z* = *suffix T j*› **by** *blast*
qed
ultimately show *?thesis*
by (*simp add*: *ordlistns.sorted-append*)

qed

theorem *abs-induce-s-base-sorted*:
  assumes *abs-induce-s-base* α *T B SA* = (*B*′, *SA*′, *n*)
  and *s-perm-inv* α *T B SA0 SA* (*length T*)
  and *s-sorted-pre* α *T SA0*
  and *s-sorted-inv* α *T B SA*
shows *ordlistns.sorted* (*map* (*suffix T*) *SA*′)
proof (*intro sorted-wrt-mapI*)
  fix *i j*
  assume *i* < *j j* < *length SA*′
  hence *i* < *length T j* < *length T*
    by (*metis* (*no-types*, *lifting*) *abs-induce-s-perm-maintained*
        assms(*1*,*2*) *order.strict-trans s-perm-inv-elims*(*11*))+

  let *?goal* = *list-less-eq-ns* (*suffix T* (*SA*′ ! *i*)) (*suffix T* (*SA*′ ! *j*))

  from *index-in-bucket-interval-gen*[*OF* ‹*i* < *length* -› *s-perm-inv-elims*(*8*)[*OF assms*(*2*)]]
  obtain *b0* **where**
    b0 ≤ α (*Max* (*set T*))
    bucket-start α *T b0* ≤ *i*
    i < *bucket-end* α *T b0*
    by *blast*

  from *index-in-bucket-interval-gen*[*OF* ‹*j* < *length T*› *s-perm-inv-elims*(*8*)[*OF assms*(*2*)]]
  obtain *b1* **where**
    b1 ≤ α (*Max* (*set T*))
    bucket-start α *T b1* ≤ *j*
    j < *bucket-end* α *T b1*
    by *blast*

  from *abs-induce-s-perm-maintained*[*OF assms*(*1*,*2*)]
  have *s-perm-inv* α *T B*′ *SA0 SA*′ *n* **.**
  moreover
  have *n* = *0*
    by (*metis Pair-inject assms*(*1*) *abs-induce-s-base-index*)
  ultimately have *s-perm-inv* α *T B*′ *SA0 SA*′ *0*
    by *simp*

  let *?xs* = *list-slice SA*′ (*bucket-start* α *T b0*) (*bucket-end* α *T b0*)
  and *?ys* = *list-slice SA*′ (*bucket-start* α *T b1*) (*bucket-end* α *T b1*)

  have *b0* ≤ *b1*
  proof (*rule ccontr*)
    assume ¬ *b0* ≤ *b1*
    hence *b1* < *b0*
      by (*simp add*: *not-le*)
    hence *bucket-end* α *T b1* ≤ *bucket-start* α *T b0*

by (*simp add*: *less-bucket-end-le-start*)
with ‹$j < bucket\text{-}end\ \alpha\ T\ b1$› ‹$bucket\text{-}start\ \alpha\ T\ b0 \le i$› ‹$i < j$›
show *False*
by *linarith*
qed
hence $b0 = b1 \lor b0 < b1$
by (*simp add*: *nat-less-le*)
moreover
have $b0 < b1 \Longrightarrow ?goal$
proof −
assume $b0 < b1$
moreover
from *s-perm-inv-0-list-slice-bucket*[*OF* ‹*s-perm-inv - - - - - 0*› ‹$b0 \le \alpha$ -›]
have *set ?xs* = *bucket* α *T b0* **.**
hence SA' ! $i \in$ *bucket* α *T b0*
by (*metis* ‹$bucket\text{-}start\ \alpha\ T\ b0 \le i$› ‹$i < bucket\text{-}end\ \alpha\ T\ b0$› ‹*s-perm-inv* α *T* B' *SA0* SA' *0*›
bucket-end-le-length list-slice-nth-mem s-perm-inv-elims(*11*))
hence α (T ! (SA' ! i)) = *b0* SA' ! $i < length\ T$
by (*simp add*: *bucket-def*)+
moreover
from *s-perm-inv-0-list-slice-bucket*[*OF* ‹*s-perm-inv - - - - - 0*› ‹$b1 \le \alpha$ -›]
have *set ?ys* = *bucket* α *T b1* **.**
hence SA' ! $j \in$ *bucket* α *T b1*
by (*metis* ‹$bucket\text{-}start\ \alpha\ T\ b1 \le j$› ‹$j < bucket\text{-}end\ \alpha\ T\ b1$› ‹*s-perm-inv* α *T* B' *SA0* SA' *0*›
bucket-end-le-length list-slice-nth-mem s-perm-inv-elims(*11*))
hence α (T ! (SA' ! j)) = *b1* SA' ! $j < length\ T$
by (*simp add*: *bucket-def*)+
ultimately have T ! (SA' ! i) < T ! (SA' ! j)
using *assms*(*2*) *s-perm-inv-elims*(*8*) *strict-mono-less* **by** *blast*
then show *?thesis*
by (*metis* ‹SA' ! $i < length\ T$› ‹SA' ! $j < length\ T$› *less-imp-le list-less-eq-ns-cons neq-iff*
suffix-cons-Suc)
qed
moreover
have $b0 = b1 \Longrightarrow ?goal$
proof −
assume $b0 = b1$
with ‹$j < bucket\text{-}end\ \alpha\ T\ b1$›
have $j < bucket\text{-}end\ \alpha\ T\ b0$
by *simp*

have $\exists i'.\ i = bucket\text{-}start\ \alpha\ T\ b0 + i'$
using ‹$bucket\text{-}start\ \alpha\ T\ b0 \le i$› *nat-le-iff-add* **by** *blast*
then obtain i' **where**
$i = bucket\text{-}start\ \alpha\ T\ b0 + i'$
by *blast*

have $\exists j'.\ j = bucket\text{-}start\ \alpha\ T\ b0 + j'$
by (*simp add*: ‹$b0 = b1$› ‹$bucket\text{-}start\ \alpha\ T\ b1 \leq j$› *le-Suc-ex*)
then obtain j' **where**
$j = bucket\text{-}start\ \alpha\ T\ b0 + j'$
by *blast*
with ‹$i < j$› ‹$i = bucket\text{-}start\ \alpha\ T\ b0 + i'$›
have $i' \leq j'$
by *linarith*

have $j' < length\ ?xs$
by (*metis* ‹$b0 = b1$› ‹$j < bucket\text{-}end\ \alpha\ T\ b1$› ‹$j = bucket\text{-}start\ \alpha\ T\ b0 + j'$›
‹*s-perm-inv* α T B' *SA0* SA' n› *add.commute bucket-end-le-length*
length-list-slice
less-diff-conv min-def s-perm-inv-elims(*11*))
with *ordlistns.sorted-nth-mono*[*OF s-sorted-bucket*[*OF* ‹*s-perm-inv* - - - - - *0*›
assms(*3*)
abs-induce-s-sorted-maintained[*OF assms*] ‹$b0 \leq \alpha$ -›] ‹$i' \leq j'$›]
have *list-less-eq-ns* (*suffix* T (*?xs* ! i')) (*suffix* T (*?xs* ! j'))
using ‹$i' \leq j'$› *nth-map* **by** *auto*
moreover
have SA' ! i = *?xs* ! i'
using ‹$i < bucket\text{-}end\ \alpha\ T\ b0$› ‹$i < length\ T$› ‹$i = bucket\text{-}start\ \alpha\ T\ b0 + i'$›
‹*s-perm-inv* α T B' *SA0* SA' n› *s-perm-inv-elims*(*11*)
by (*metis* ‹$bucket\text{-}start\ \alpha\ T\ b0 \leq i$› *diff-add-inverse list-slice-nth*)
moreover
have SA' ! j = *?xs* ! j'
using ‹$j < bucket\text{-}end\ \alpha\ T\ b0$› ‹$j < length\ SA'$›
‹$j = bucket\text{-}start\ \alpha\ T\ b0 + j'$›
by (*simp add*: *nth-list-slice*)
ultimately show *?goal*
by *simp*
qed
ultimately show *?goal*
by *blast*
qed

theorem *s-prefix-sorted-bucket*:
assumes *s-perm-inv* α T B *SA0* *SA* *0*
and *s-prefix-sorted-pre* α T *SA0*
and *s-prefix-sorted-inv* α T B *SA*
and $b \leq \alpha\ (Max\ (set\ T))$
shows *ordlistns.sorted* (*map* (*lms-slice* T) (*list-slice SA* (*bucket-start* α T b) (*bucket-end*
α T b)))
(**is** *ordlistns.sorted* (*map* (*lms-slice* T) *?xs*))
proof −
let *?ys* = *list-slice SA* (*bucket-start* α T b) (*l-bucket-end* α T b)
and *?zs* = *list-slice SA* (*s-bucket-start* α T b) (*bucket-end* α T b)

from *s-perm-inv-0-B-val*[*OF assms*(*1*,*4*)]
have *B* ! *b* = *s-bucket-start* α *T b* **.**

from *s-prefix-sorted-pre-maintained*[*OF s-perm-inv-elims*(*2*,*4*,*10*,*11*)[*OF assms*(*1*)]
assms(*2*)]
have *s-prefix-sorted-pre* α *T SA* **.**

have *?xs* = *?ys* @ *?zs*
by (*metis bucket-start-le-s-bucket-start l-bucket-end-le-bucket-end list-slice-append*
s-bucket-start-eq-l-bucket-end)
hence *map* (*lms-slice T*) *?xs* = *map* (*lms-slice T*) *?ys* @ *map* (*lms-slice T*) *?zs*
by *simp*
moreover
from *s-prefix-sorted-preD*[*OF* ‹*s-prefix-sorted-pre - - SA*› ‹*b* ≤ -›]
have *ordlistns.sorted* (*map* (*lms-slice T*) *?ys*) **.**
moreover
from *s-prefix-sorted-invD*[*OF assms*(*3*,*4*), *simplified* ‹- = *s-bucket-start* α - -›]
have *ordlistns.sorted* (*map* (*lms-slice T*) *?zs*) **.**
moreover
have ∀ *y* ∈ *set* (*map* (*lms-slice T*) *?ys*). ∀ *z* ∈ *set* (*map* (*lms-slice T*) *?zs*).
list-less-eq-ns y z
proof *safe*
fix *y z*
assume *y* ∈ *set* (*map* (*lms-slice T*) *?ys*)
z ∈ *set* (*map* (*lms-slice T*) *?zs*)
with *in-set-mapD*[*of y lms-slice T ?ys*]
in-set-mapD[*of z lms-slice T ?zs*]
obtain *i j* **where**
y = *lms-slice T i*
i ∈ *set ?ys*
z = *lms-slice T j*
j ∈ *set ?zs*
by *blast*

from *l-types-initD*(*1*)[*OF l-types-init-maintained*[*OF s-perm-inv-elims*(*2*,*4*,*10*−*12*)[*OF*
assms(*1*)]]
‹*b* ≤ -›]
‹*i* ∈ *set ?ys*›
have *i* ∈ *l-bucket* α *T b*
by *blast*
hence *suffix-type T i* = *L-type* α (*T* ! *i*) = *b i* < *length T*
by (*simp add*: *l-bucket-def bucket-def*)+
moreover
from *s-bucket-eq-list-slice*[*OF s-perm-inv-elims*(*1*,*3*,*11*)[*OF assms*(*1*)] ‹*b* ≤ -›
‹*B* ! *b* = *s-bucket-start* α - -›]
‹*j* ∈ *set ?zs*›
have *j* ∈ *s-bucket* α *T b*
by *blast*
hence *suffix-type T j* = *S-type* α (*T* ! *j*) = *b j* < *length T*

by (*simp add*: *s-bucket-def bucket-def*)+
moreover
have $T ! i = T ! j$
using *assms*(*1*) *calculation*(*2*,*5*) *s-perm-inv-elims*(*8*) *strict-mono-eq* **by** *fast-force*
ultimately have *list-less-eq-ns* (*lms-slice* T i) (*lms-slice* T j)
using *lms-slice-l-less-than-s-type*[*of* i T j] **by** *simp*
then show *list-less-eq-ns* y z
by (*simp add*: ‹y = *lms-slice* T i› ‹z = *lms-slice* T j›)
qed
ultimately show *?thesis*
by (*simp add*: *ordlistns.sorted-append*)
qed

theorem *abs-induce-s-base-prefix-sorted*:
assumes *abs-induce-s-base* α T B SA = (B', SA', n)
and *s-perm-inv* α T B *SA0* SA (*length* T)
and *s-prefix-sorted-pre* α T *SA0*
and *s-prefix-sorted-inv* α T B SA
shows *ordlistns.sorted* (*map* (*lms-slice* T) SA')
proof (*intro sorted-wrt-mapI*)
fix i j
assume $i < j$ $j <$ *length* SA'
hence $i <$ *length* T $j <$ *length* T
by (*metis* (*no-types*, *lifting*) *abs-induce-s-perm-maintained*
assms(*1*,*2*) *order.strict-trans s-perm-inv-elims*(*11*))+

let *?goal* = *list-less-eq-ns* (*lms-slice* T (SA' ! i)) (*lms-slice* T (SA' ! j))

from *index-in-bucket-interval-gen*[*OF* ‹$i <$ *length* -› *s-perm-inv-elims*(*8*)[*OF assms*(*2*)]]
obtain *b0* **where**
b0 $\leq \alpha$ (*Max* (*set* T))
bucket-start α T *b0* $\leq i$
$i <$ *bucket-end* α T *b0*
by *blast*

from *index-in-bucket-interval-gen*[*OF* ‹$j <$ *length* T› *s-perm-inv-elims*(*8*)[*OF assms*(*2*)]]
obtain *b1* **where**
b1 $\leq \alpha$ (*Max* (*set* T))
bucket-start α T *b1* $\leq j$
$j <$ *bucket-end* α T *b1*
by *blast*

from *abs-induce-s-perm-maintained*[*OF assms*(*1*,*2*)]
have *s-perm-inv* α T B' *SA0* SA' n **.**
moreover
have $n = 0$
by (*metis Pair-inject assms*(*1*) *abs-induce-s-base-index*)

ultimately have *s-perm-inv* α *T B' SA0 SA' 0*
  by *simp*

let *?xs* = *list-slice SA'* (*bucket-start* α *T b0*) (*bucket-end* α *T b0*)
and *?ys* = *list-slice SA'* (*bucket-start* α *T b1*) (*bucket-end* α *T b1*)

have *b0* $\leq$ *b1*
proof (*rule ccontr*)
  assume $\neg$*b0* $\leq$ *b1*
  hence *b1* < *b0*
    by (*simp add*: *not-le*)
  hence *bucket-end* α *T b1* $\leq$ *bucket-start* α *T b0*
    by (*simp add*: *less-bucket-end-le-start*)
  with ‹*j* < *bucket-end* α *T b1*› ‹*bucket-start* α *T b0* $\leq$ *i*› ‹*i* < *j*›
  show *False*
    by *linarith*
qed
hence *b0* = *b1* $\vee$ *b0* < *b1*
  by (*simp add*: *nat-less-le*)
moreover
have *b0* < *b1* $\Longrightarrow$ *?goal*
proof −
  assume *b0* < *b1*
  moreover
  from *s-perm-inv-0-list-slice-bucket*[*OF* ‹*s-perm-inv - - - - - 0*› ‹*b0* $\leq$ α *-*›]
  have *set ?xs* = *bucket* α *T b0* **.**
  hence *SA'* ! *i* $\in$ *bucket* α *T b0*
    by (*metis* ‹*bucket-start* α *T b0* $\leq$ *i*› ‹*i* < *bucket-end* α *T b0*› ‹*s-perm-inv* α
T B' SA0 SA' 0›
        bucket-end-le-length list-slice-nth-mem s-perm-inv-elims(*11*))
  hence α (*T* ! (*SA'* ! *i*)) = *b0 SA'* ! *i* < *length T*
    by (*simp add*: *bucket-def*)+
  moreover
  from *s-perm-inv-0-list-slice-bucket*[*OF* ‹*s-perm-inv - - - - - 0*› ‹*b1* $\leq$ α *-*›]
  have *set ?ys* = *bucket* α *T b1* **.**
  hence *SA'* ! *j* $\in$ *bucket* α *T b1*
    by (*metis* ‹*bucket-start* α *T b1* $\leq$ *j*› ‹*j* < *bucket-end* α *T b1*› ‹*s-perm-inv* α
T B' SA0 SA' 0›
        bucket-end-le-length list-slice-nth-mem s-perm-inv-elims(*11*))
  hence α (*T* ! (*SA'* ! *j*)) = *b1 SA'* ! *j* < *length T*
    by (*simp add*: *bucket-def*)+
  ultimately have *T* ! (*SA'* ! *i*) < *T* ! (*SA'* ! *j*)
    using *assms*(*2*) *s-perm-inv-elims*(*8*) *strict-mono-less* **by** *blast*
  then show *?thesis*
    by (*metis* ‹*SA'* ! *i* < *length T*› ‹*SA'* ! *j* < *length T*› *abs-find-next-lms-lower-bound-1*

      less-SucI less-imp-le list-less-eq-ns-cons list-slice-Suc neq-iff lms-slice-def)
qed
moreover

have $b0 = b1 \Longrightarrow$ *?goal*
proof −
  assume $b0 = b1$
  with ‹$j <$ *bucket-end* α T $b1$›
  have $j <$ *bucket-end* α T $b0$
    by *simp*

  have $\exists\, i'.\ i =$ *bucket-start* α T $b0 + i'$
    using ‹*bucket-start* α T $b0 \leq i$› *nat-le-iff-add* **by** *blast*
  then obtain i' **where**
    $i =$ *bucket-start* α T $b0 + i'$
    by *blast*

  have $\exists\, j'.\ j =$ *bucket-start* α T $b0 + j'$
    by (*simp add*: ‹$b0 = b1$› ‹*bucket-start* α T $b1 \leq j$› *le-Suc-ex*)
  then obtain j' **where**
    $j =$ *bucket-start* α T $b0 + j'$
    by *blast*
  with ‹$i < j$› ‹$i =$ *bucket-start* α T $b0 + i'$›
  have $i' \leq j'$
    by *linarith*

  have $j' <$ *length ?xs*
    by (*metis* ‹$b0 = b1$›
        ‹$j <$ *bucket-end* α T $b1$›
        ‹$j =$ *bucket-start* α T $b0 + j'$›
        ‹*s-perm-inv* α T B' *SA0* SA' n›
        add.commute bucket-end-le-length length-list-slice
        less-diff-conv min-def s-perm-inv-elims(*11*))
  with *ordlistns.sorted-nth-mono*
      [*OF s-prefix-sorted-bucket*
          [*OF* ‹*s-perm-inv* - - - - - *0*› *assms*(*3*)
            abs-induce-s-prefix-sorted-maintained
              [*OF assms*] ‹$b0 \leq \alpha$ -›] ‹$i' \leq j'$›]
  have *list-less-eq-ns* (*lms-slice* T (*?xs* ! i')) (*lms-slice* T (*?xs* ! j'))
    using ‹$i' \leq j'$› *nth-map* **by** *auto*
  moreover
  have SA' ! i = *?xs* ! i'
    using ‹$i <$ *bucket-end* α T $b0$›
        ‹$i <$ *length* T›
        ‹$i =$ *bucket-start* α T $b0 + i'$›
        ‹*s-perm-inv* α T B' *SA0* SA' n›
        ‹$j' <$ *length* (*list-slice* SA'
              (*bucket-start* α T $b0$)
              (*bucket-end* α T $b0$))›
    by (*metis* ‹$i' \leq j'$› *nth-list-slice order.strict-trans1*)
  moreover
  have SA' ! j = *?xs* ! j'
    using ‹$j <$ *bucket-end* α T $b0$› ‹$j <$ *length* SA'›

‹j = *bucket-start* α *T b0* + j'›
by (*simp add*: *nth-list-slice*)
ultimately show *?goal*
by *simp*
qed
ultimately show *?goal*
by *blast*
qed

84 Induce S Correctness Theorems

theorem *abs-induce-s-perm*:
assumes *s-perm-pre* α *T B SA* (*length T*)
shows *abs-induce-s* α *T B SA* <~~> [*0*..< *length T*]
proof −
from *s-perm-inv-established assms*[*simplified s-perm-pre-def*]
have *s-perm-inv* α *T B SA SA* (*length T*)
by *blast*
moreover
from *abs-induce-s-base-index*[*of* α *T B SA*]
obtain B' SA' **where**
abs-induce-s-base α *T B SA* = (B', SA', *0*)
by *blast*
ultimately have *s-perm-inv* α *T* B' *SA* SA' *0*
using *abs-induce-s-perm-maintained*[*of* α *T B SA* B' SA' *0 SA*]
by *blast*
hence SA' <~~> [*0*..< *length T*]
using *abs-induce-s-base-perm*[*OF* ‹*abs-induce-s-base* α *T B SA* = (B', SA', *0*)›]
by *blast*
with ‹*abs-induce-s-base* α *T B SA* = (B', SA', *0*)›
show *?thesis*
by (*simp add*: *abs-induce-s-def*)
qed

— Used in SAIS algorithm as part of inducing the suffix ordering based on LMS
theorem *abs-induce-s-sorted*:
assumes *s-perm-pre* α *T B SA* (*length T*)
and *s-sorted-pre* α *T SA*
shows *ordlistns.sorted* (*map* (*suffix T*) (*abs-induce-s* α *T B SA*))
proof −
from *s-perm-inv-established assms*(*1*)[*simplified s-perm-pre-def*]
have *s-perm-inv* α *T B SA SA* (*length T*)
by *blast*
moreover
from *abs-induce-s-base-index*[*of* α *T B SA*]
obtain B' SA' **where**
abs-induce-s-base α *T B SA* = (B', SA', *0*)
by *blast*

moreover
from *s-sorted-inv-established assms*(*1*)[*simplified s-perm-pre-def*]
have *s-sorted-inv* α *T B SA*
by *blast*
ultimately have *ordlistns.sorted* (*map* (*suffix T*) *SA′*)
using *abs-induce-s-base-sorted*[*OF - - assms*(*2*), *of B SA B′ SA′ 0*]
by *blast*
then show *?thesis*
by (*simp add*: ‹*abs-induce-s-base* α *T B SA* = (*B′*, *SA′*, *0*)› *abs-induce-s-def*)
qed

— Used in SAIS algorithm as part of inducing the prefix ordering based on LMS
theorem *abs-induce-s-prefix-sorted*:
assumes *s-perm-pre* α *T B SA* (*length T*)
and *s-prefix-sorted-pre* α *T SA*
shows *ordlistns.sorted* (*map* (*lms-slice T*) (*abs-induce-s* α *T B SA*))
proof −

from *s-perm-inv-established assms*(*1*)[*simplified s-perm-pre-def*]
have *s-perm-inv* α *T B SA SA* (*length T*)
by *blast*
moreover
from *abs-induce-s-base-index*[*of* α *T B SA*]
obtain *B′ SA′* **where**
abs-induce-s-base α *T B SA* = (*B′*, *SA′*, *0*)
by *blast*
moreover
from *s-prefix-sorted-inv-established assms*(*1*)[*simplified s-perm-pre-def*]
have *s-prefix-sorted-inv* α *T B SA*
by *blast*
ultimately have *ordlistns.sorted* (*map* (*lms-slice T*) *SA′*)
using *abs-induce-s-base-prefix-sorted*[*OF - - assms*(*2*), *of B SA B′ SA′ 0*]
by *blast*
then show *?thesis*
by (*simp add*: ‹*abs-induce-s-base* α *T B SA* = (*B′*, *SA′*, *0*)› *abs-induce-s-def*)
qed

end
theory *Abs-Induce-Verification*
imports
Abs-Induce-L-Verification
Abs-Induce-S-Verification
Abs-Bucket-Insert-Verification
begin

85 Bucket Initialisation Properties

lemma *l-bucket-init-map-bucket-start*:
l-bucket-init α *T* (*map* (*bucket-start* α *T*) [*0*..<*Suc* (α (*Max* (*set T*)))])

unfolding *l-bucket-init-def*
by (*metis add.left-neutral diff-zero le-imp-less-Suc length-map length-upt lessI nth-map-upt*)

lemma *lms-bucket-init-map-bucket-end*:
lms-bucket-init α *T* (*map* (*bucket-end* α *T*) [*0*..<*Suc* (α (*Max* (*set T*)))])
unfolding *lms-bucket-init-def*
by (*metis add.left-neutral diff-zero le-imp-less-Suc length-map length-upt lessI nth-map-upt*)

lemma *s-bucket-init-map-bucket-end*:
s-bucket-init α *T* ((*map* (*bucket-end* α *T*) [*0*..<*Suc* (α (*Max* (*set T*)))])[*0* := *0*])
unfolding *s-bucket-init-def*
by (*metis* (*no-types*, *lifting*) *length-greater-0-conv length-list-update list.size*(*3*)
nth-list-update lms-bucket-init-def lms-bucket-init-map-bucket-end)

abbreviation *bucket-starts* α *T* $\equiv$ *map* (*bucket-start* α *T*) [*0*..<*Suc* (α (*Max* (*set T*)))]

abbreviation *bucket-ends* α *T* $\equiv$ *map* (*bucket-end* α *T*) [*0*..<*Suc* (α (*Max* (*set T*)))]

86 Bucket Insert Precondition

lemma *lms-pre-established*:
assumes *set LMS* = {*i*. *abs-is-lms T i*}
and *distinct LMS*
and *strict-mono* α
shows *lms-pre* α *T* (*bucket-ends* α *T*) (*replicate* (*length T*) (*length T*)) (*rev LMS*)
(**is** *lms-pre* α *T ?B ?SA* (*rev LMS*))
proof −
from *lms-bucket-init-map-bucket-end*[*of* α *T*]
have *lms-bucket-init* α *T ?B* **.**
then show *lms-pre* α *T ?B ?SA0* (*rev LMS*)
by (*clarsimp simp*: *lms-pre-def* ‹*lms-bucket-init* α *T ?B*› *assms*
simp del: *upt-Suc*)
qed

87 Induce L Precondition

lemma *l-perm-pre-established*:
assumes *valid-list T*
and *strict-mono* α
and *lms-pre* α *T B SA* (*rev LMS*)
shows *l-perm-pre* α *T* (*bucket-starts* α *T*) (*abs-bucket-insert* α *T B SA* (*rev LMS*))
(**is** *l-perm-pre* α *T ?B ?SA*)
unfolding *l-perm-pre-def*
proof *safe*

show *lms-init* α *T ?SA*
by (*metis assms*(*3*) *abs-bucket-insert-vals lms-init-def lms-vals-postD*)
next
show *l-init* α *T ?SA*
unfolding *l-init-def*
proof (*intro allI impI*; *elim conjE*)
fix *b i*
assume $b \leq \alpha$ (*Max* (*set T*)) $i <$ *length ?SA bucket-start* α *T* $b \leq i$
$i <$ *l-bucket-end* α *T b*
hence $i <$ *lms-bucket-start* α *T b*
using *l-bucket-end-le-lms-bucket-start less-le-trans* **by** *blast*
with *lms-unknowns-postD*[*OF abs-bucket-insert-unknowns*[*OF assms*(*3*)] ‹$b \leq$ -›
‹*bucket-start* - - - $\leq$ -›]
show *?SA* ! $i =$ *length T* **.**
qed
next
show *s-init* α *T ?SA*
unfolding *s-init-def*
proof (*intro allI impI*; *elim conjE*)
fix *b i*
assume $b \leq \alpha$ (*Max* (*set T*)) $i <$ *length ?SA l-bucket-end* α *T* $b \leq i$
$i <$ *lms-bucket-start* α *T b*
hence *bucket-start* α *T* $b \leq i$
by (*simp add*: *l-bucket-end-def*)
with *lms-unknowns-postD*[*OF abs-bucket-insert-unknowns*[*OF assms*(*3*)] ‹$b \leq$ -› -
‹$i <$ *lms-bucket-start* - - -›]
show *?SA* ! $i =$ *length T* **.**
qed
next
from *l-bucket-init-map-bucket-start*[*of* α *T*]
show *l-bucket-init* α *T ?B* **.**
next
show *length ?SA* = *length T*
by (*metis assms*(*3*) *abs-bucket-insert-length lms-pre-elims*(*2*))
qed (*force simp*: *valid-list-not-nil*[*OF assms*(*1*)] *assms*(*2*))+

88 Induce S Precondition

lemma *s-perm-pre-established*:
assumes *valid-list T*
and *strict-mono* α
and α *bot* = *0*
and *Suc 0* < *length T*
and *lms-pre* α *T B0 SA0* (*rev LMS*)
and *SA1* = *abs-bucket-insert* α *T B0 SA0* (*rev LMS*)
and *l-perm-pre* α *T B1 SA1*
shows *s-perm-pre* α *T* ((*bucket-ends* α *T*)[*0* := *0*]) (*abs-induce-l* α *T B1 SA1*)

(*length T*)
    (**is** *s-perm-pre* α *T ?B ?SA ?n*)
  unfolding *s-perm-pre-def*
proof (*intro conjI*)
  from *s-bucket-init-map-bucket-end*[*of* α *T*]
  show *s-bucket-init* α *T ?B* **.**
next
  from *valid-list-length-ex*[*OF assms*(*1*)]
  obtain *n* **where**
    length T = *Suc n*
    by *blast*
  hence $\exists$ *m*. *length T* = *Suc* (*Suc m*)
    using *assms*(*4*) *old.nat.exhaust* **by** *auto*
  then obtain *m* **where**
    length T = *Suc* (*Suc m*)
    by *blast*

  from *abs-bucket-insert-bot-first*[*OF assms*(*5*,*1*) ‹*length T* = *Suc* (*Suc m*)› *assms*(*3*)]
  have *SA1* ! *0* = *n*
    using ‹*length T* = *Suc* (*Suc m*)› ‹*length T* = *Suc n*› *assms*(*6*) **by** *auto*

  have *0* $\leq$ α (*Max* (*set T*))
    by *simp*
  moreover
  have *s-bucket-start* α *T 0* $\leq$ *0*
    by (*simp add*: *assms*(*1*−*3*) *valid-list-s-bucket-start-0*)
  moreover
  have *0* < *bucket-end* α *T 0*
    by (*simp add*: *assms*(*1*−*3*) *valid-list-bucket-end-0*)
  ultimately have *?SA* ! *0* = *SA1* ! *0*
    using *abs-induce-l-unchanged*[*OF* ‹*l-perm-pre - - - -*›, *of 0 0*]
    by *blast*
  with ‹*SA1* ! *0* = *n*›
  have *?SA* ! *0* = *n*
    by *simp*
  with ‹*length T* = *Suc n*›
  show *s-type-init T ?SA*
    using *s-type-init-def* **by** *blast*
next
  show *length ?SA* = *length T*
    by (*metis abs-induce-l-length assms*(*7*) *l-perm-pre-elims*(*7*))
next
  from *abs-induce-l-distinct-l-bucket*[*OF assms*(*7*)]
      abs-induce-l-list-slice-l-bucket[*OF assms*(*7*)]
  show *l-types-init* α *T ?SA*
    by (*simp add*: *l-types-init-def*)
qed(*force simp*: *assms*)+

89 Permutation

lemma *abs-sa-induce-permutation*:
assumes *set LMS* = {*i. abs-is-lms T i*}
and *distinct LMS*
and *valid-list T*
and *strict-mono α*
and *α bot* = *0*
and *Suc 0* < *length T*
shows *abs-sa-induce α T LMS* <~~> [*0*..< *length T*]
proof −
let *?B0* = *map* (*bucket-end α T*) [*0*..<*Suc* (*α* (*Max* (*set T*)))] **and**
?B1 = *map* (*bucket-start α T*) [*0*..<*Suc* (*α* (*Max* (*set T*)))] **and**
?SA0 = *replicate* (*length T*) (*length T*)

let *?B2* = *?B0*[*0* := *0*]
let *?SA1* = *abs-bucket-insert α T ?B0 ?SA0* (*rev LMS*)
let *?SA2* = *abs-induce-l α T ?B1 ?SA1*
let *?SA3* = *abs-induce-s α T* (*?B0*[*0* := *0*]) *?SA2*

from *lms-pre-established*[*OF assms*(*1*,*2*,*4*)]
have *lms-pre α T ?B0 ?SA0* (*rev LMS*) **.**

have *l-perm-pre α T ?B1 ?SA1*
using ‹*lms-pre α T ?B0 ?SA0* (*rev LMS*)› *assms*(*3*,*4*) *l-perm-pre-established*
by *blast*

have *s-perm-pre α T ?B2 ?SA2* (*length T*)
using ‹*l-perm-pre α T ?B1 ?SA1*› ‹*lms-pre α T ?B0 ?SA0* (*rev LMS*)›
assms(*3*−*6*)
s-perm-pre-established **by** *blast*
with *abs-induce-s-perm*[*of α T ?B0*[*0* := *0*] *?SA2*]
have *?SA3* <~~> [*0*..< *length T*]
by *blast*
then show *?thesis*
by (*metis abs-sa-induce-def*)
qed

90 Sorting

lemma *abs-sa-induce-suffix-sorted*:
assumes *set LMS* = {*i. abs-is-lms T i*}
and *distinct LMS*
and *valid-list T*
and *strict-mono α*
and *α bot* = *0*
and *Suc 0* < *length T*
and *ordlistns.sorted* (*map* (*suffix T*) *LMS*)
shows *ordlistns.sorted* (*map* (*suffix T*) (*abs-sa-induce α T LMS*))

proof −
  let *?B0* = *map* (*bucket-end* α *T*) [*0*..<*Suc* (α (*Max* (*set T*)))] **and**
      ?B1 = *map* (*bucket-start* α *T*) [*0*..<*Suc* (α (*Max* (*set T*)))] **and**
      ?SA0 = *replicate* (*length T*) (*length T*)

  let *?B2* = *?B0*[*0* := *0*]
  let *?SA1* = *abs-bucket-insert* α *T ?B0 ?SA0* (*rev LMS*)
  let *?SA2* = *abs-induce-l* α *T ?B1 ?SA1*
  let *?SA3* = *abs-induce-s* α *T* (*?B0*[*0* := *0*]) *?SA2*

  from *lms-pre-established*[*OF assms*(*1*,*2*,*4*)]
  have *lms-pre* α *T ?B0 ?SA0* (*rev LMS*) **.**

  have *P0*:
    ∀ *b* ≤ α (*Max* (*set T*)).
      ordlistns.sorted (*map* (*suffix T*)
        (*list-slice ?SA1* (*lms-bucket-start* α *T b*) (*bucket-end* α *T b*)))
  proof(*intro allI impI*)
    fix *b*
    assume *b* ≤ α (*Max* (*set T*))
    from *lms-suffix-sorted-bucket*[*OF* ‹*lms-pre* - - - - -› - ‹*b* ≤ -›] *assms*(*7*)
    show *ordlistns.sorted* (*map* (*suffix T*)
          (*list-slice ?SA1* (*lms-bucket-start* α *T b*) (*bucket-end* α *T b*)))
      by *simp*
  qed

  have *l-perm-pre* α *T ?B1 ?SA1*
    using ‹*lms-pre* α *T ?B0 ?SA0* (*rev LMS*)› *assms*(*3*,*4*) *l-perm-pre-established*
by *blast*
  moreover
  have *l-suffix-sorted-pre* α *T ?SA1*
    using *P0 l-suffix-sorted-pre-def* **by** *blast*
  ultimately have *P1*:
    ∀ *b* ≤ α (*Max* (*set T*)).
     ordlistns.sorted (*map* (*suffix T*) (*list-slice ?SA2* (*bucket-start* α *T b*) (*l-bucket-end*
α *T b*)))
    using *abs-induce-l-suffix-sorted-l-bucket* **by** *blast*

  have *s-perm-pre* α *T ?B2 ?SA2* (*length T*)
      using ‹*l-perm-pre* α *T ?B1 ?SA1*› ‹*lms-pre* α *T ?B0 ?SA0* (*rev LMS*)›
assms(*3*−*6*)
          s-perm-pre-established **by** *blast*
  moreover
  have *s-sorted-pre* α *T ?SA2*
    using *P1 s-sorted-pre-def* **by** *blast*
  ultimately show *?thesis*
    by (*metis abs-induce-s-sorted abs-sa-induce-def*)
qed

— Used in SAIS algorithm to induce the prefix ordering based on LMS

```
theorem abs-sa-induce-prefix-sorted:
  assumes set LMS = {i. abs-is-lms T i}
  and     distinct LMS
  and     valid-list T
  and     strict-mono α
  and     α bot = 0
  and     Suc 0 < length T
shows ordlistns.sorted (map (lms-slice T) (abs-sa-induce α T LMS))
proof −
  let ?B0 = map (bucket-end α T) [0..<Suc (α (Max (set T)))] and
      ?B1 = map (bucket-start α T) [0..<Suc (α (Max (set T)))] and
      ?SA0 = replicate (length T) (length T)

  let ?B2 = ?B0[0 := 0]
  let ?SA1 = abs-bucket-insert α T ?B0 ?SA0 (rev LMS)
  let ?SA2 = abs-induce-l α T ?B1 ?SA1
  let ?SA3 = abs-induce-s α T (?B0[0 := 0]) ?SA2

  from lms-pre-established[OF assms(1,2,4)]
  have lms-pre α T ?B0 ?SA0 (rev LMS) .

  have P0:
    ∀ b ≤ α (Max (set T)).
      ordlistns.sorted (map (lms-prefix T)
        (list-slice ?SA1 (lms-bucket-start α T b) (bucket-end α T b)))
  proof(intro allI impI)
    fix b
    assume b ≤ α (Max (set T))
    from lms-prefix-sorted-bucket[OF ‹lms-pre - - - - -› ‹b ≤ -›]
    show ordlistns.sorted (map (lms-prefix T)
            (list-slice ?SA1 (lms-bucket-start α T b) (bucket-end α T b)))
      by simp
  qed

  have l-perm-pre α T ?B1 ?SA1
    using ‹lms-pre α T ?B0 ?SA0 (rev LMS)› assms(3,4) l-perm-pre-established
by blast
  moreover
  have l-prefix-sorted-pre α T ?SA1
    using P0 l-prefix-sorted-pre-def by blast
  ultimately have P1:
    ∀ b ≤ α (Max (set T)).
      ordlistns.sorted (map (lms-prefix T)
        (list-slice ?SA2 (bucket-start α T b) (l-bucket-end α T b)))
    using abs-induce-l-prefix-sorted-l-bucket by blast

  have P2:
```

$\forall b \leq \alpha$ (*Max* (*set* T)).
ordlistns.sorted (*map* (*lms-slice* T)
(*list-slice ?SA2* (*bucket-start* α T b) (*l-bucket-end* α T b)))
proof (*intro allI impI sorted-wrt-mapI*)
fix b i j

let *?xs* = *list-slice ?SA2* (*bucket-start* α T b) (*l-bucket-end* α T b) **and**
?R1 = (λx y. *list-less-eq-ns* (*lms-prefix* T x) (*lms-prefix* T y)) **and**
?R2 = (λx y. *list-less-eq-ns* (*lms-slice* T x) (*lms-slice* T y))

assume $b \leq \alpha$ (*Max* (*set* T)) $i < j$ $j <$ *length ?xs*
with *P1*
have *ordlistns.sorted* (*map* (*lms-prefix* T) *?xs*)
by *blast*
with *sorted-wrt-mapD*[*OF* - ‹$i < j$› ‹$j <$ *length ?xs*›]
have *list-less-eq-ns* (*lms-prefix* T (*?xs* ! i)) (*lms-prefix* T (*?xs* ! j))
by *blast*
moreover
from *abs-induce-l-list-slice-l-bucket*[*OF* ‹*l-perm-pre* - - - -› ‹$b \leq$ -›]
have *?xs* ! $i \in$ *l-bucket* α T b
using *Suc-lessD* ‹$i < j$› ‹$j <$ *length ?xs*› *less-trans-Suc nth-mem* **by** *blast*
hence *suffix-type* T (*?xs* ! i) = *L-type*
using *l-bucket-def bucket-def* **by** *blast*
hence *lms-prefix* T (*?xs* ! i) = *lms-slice* T (*?xs* ! i)
by (*metis SL-types.distinct*(*1*) *abs-is-lms-def not-lms-imp-next-eq-lms-prefix*)
moreover
from *abs-induce-l-list-slice-l-bucket*[*OF* ‹*l-perm-pre* - - - -› ‹$b \leq$ -›]
have *?xs* ! $j \in$ *l-bucket* α T b
using ‹$j <$ *length ?xs*› *less-trans-Suc nth-mem* **by** *blast*
hence *suffix-type* T (*?xs* ! j) = *L-type*
using *l-bucket-def bucket-def* **by** *blast*
hence *lms-prefix* T (*?xs* ! j) = *lms-slice* T (*?xs* ! j)
by (*metis SL-types.distinct*(*1*) *abs-is-lms-def not-lms-imp-next-eq-lms-prefix*)
ultimately show *list-less-eq-ns* (*lms-slice* T (*?xs* ! i)) (*lms-slice* T (*?xs* ! j))
by *order*
qed

have *s-perm-pre* α T *?B2 ?SA2* (*length* T)
using ‹*l-perm-pre* α T *?B1 ?SA1*› ‹*lms-pre* α T *?B0 ?SA0* (*rev LMS*)›
assms(*3*−*6*)
s-perm-pre-established **by** *blast*
moreover
have *s-prefix-sorted-pre* α T *?SA2*
using *P2 s-prefix-sorted-pre-def* **by** *blast*
ultimately show *?thesis*
by (*metis abs-induce-s-prefix-sorted abs-sa-induce-def*)
qed

end
theory *Abs-Extract-LMS-Verification*
  imports *../abs−def/Abs-SAIS Abs-Induce-Verification*
begin

91 Extract LMS types Proofs

lemma *abs-extract-lms-correct*:
  xs <~~> [*0*..<*length T*] $\Longrightarrow$
   distinct (*abs-extract-lms T xs*) $\wedge$ *set* (*abs-extract-lms T xs*) = {*i*. *abs-is-lms T i*}
  by (*metis comp-apply distinct-filter distinct-upt filter-set get-lms-correct*
      get-lms-set-n-gre-length order.refl perm-distinct-iff perm-set-eq set-rev)

lemma *set-abs-extract-lms-eq-all-lms*:
  set (*abs-extract-lms T* [*0*..<*length T*]) = {*i*. *abs-is-lms T i*}
  using *abs-extract-lms-correct* **by** *blast*

lemma *distinct-abs-extract-lms*:
  distinct (*abs-extract-lms T* [*0*..<*length T*])
  using *abs-extract-lms-correct* **by** *blast*

lemma *filter-abs-sa-induce-eq-all-lms*:
  ⟦*set LMS* = {*i*. *abs-is-lms T i*}; *distinct LMS*; *valid-list T*; *strict-mono* α; α *bot* = *0*;
    Suc 0 < *length T*⟧ $\Longrightarrow$
   set (*abs-extract-lms T* (*abs-sa-induce* α *T LMS*)) = {*i*. *abs-is-lms T i*}
  using *abs-extract-lms-correct abs-sa-induce-permutation* **by** *blast*

lemma *distinct-filter-abs-sa-induce*:
  ⟦*set LMS* = {*i*. *abs-is-lms T i*}; *distinct LMS*; *valid-list T*; *strict-mono* α; α *bot* = *0*;
    Suc 0 < *length T*⟧ $\Longrightarrow$
   distinct (*abs-extract-lms T* (*abs-sa-induce* α *T LMS*))
  using *abs-extract-lms-correct abs-sa-induce-permutation* **by** *blast*

end
theory *Abs-Order-LMS-Verification*
  imports *../abs−def/Abs-SAIS*
begin

92 Order LMS-types Proofs

lemma *abs-order-lms-eq-map-nth*:
order-lms LMS xs = *map* (*nth LMS*) *xs*
by (*induct xs*; *simp*)

theorem *distinct-abs-order-lms*:
⟦*xs* <~~> [*0*..<*length LMS*]; *distinct LMS*⟧ ⟹
distinct (*order-lms LMS xs*)
apply (*subst abs-order-lms-eq-map-nth*)
apply (*erule distinct-map-nth-perm*[*of* - *length LMS*]; *simp*)
done

theorem *abs-order-lms-eq-all-lms*:
⟦*xs* <~~> [*0*..<*length LMS*]; *set LMS* = *S*⟧ ⟹
set (*order-lms LMS xs*) = *S*
apply (*subst abs-order-lms-eq-map-nth*)
apply (*frule set-map-nth-perm-eq*)
apply *simp*
done

end
theory *Abs-Rename-LMS-Verification*
imports ../*abs*−*def*/*Abs-SAIS*

begin

93 Rename Mapping Proofs

lemma *abs-rename-mapping′-length*:
length (*abs-rename-mapping′ T LMS names i*) = *length names*
by (*induct rule*: *abs-rename-mapping′.induct*[*of* - *T LMS names i*]; *simp*)

lemma *abs-rename-mapping-length*:
length (*abs-rename-mapping T LMS*) = *length T*
by (*clarsimp simp*: *abs-rename-mapping-def abs-rename-mapping′-length*)

lemma *rename-mapping′-unchanged*:
⟦*x* ∉ *set LMS*; *x* < *length names*⟧ ⟹
(*abs-rename-mapping′ T LMS names i*) ! *x* = *names* ! *x*
by (*induct rule*: *abs-rename-mapping′.induct*[*of* - *T LMS names i*]; *simp*)

lemma *rename-mapping′-lms*:
assumes *distinct LMS*
and *ordlistns.sorted* (*map* (*lms-slice T*) *LMS*)

and $i \in set\ LMS$
and $i < length\ names$
shows $(abs\text{-}rename\text{-}mapping'\ T\ LMS\ names\ j)\ !\ i =$
$j + (ordlistns.elem\text{-}rank\ ((lms\text{-}slice\ T)\ `\ set\ LMS)\ (lms\text{-}slice\ T\ i))$
using *assms*
proof (*induct arbitrary*: *i rule*: *abs-rename-mapping'.induct*[*of - T LMS names j*])
case (*1 uu names uv*)
then show *?case*
by *force*
next
case (*2 uw x names j*)
note $A = this$
hence $x = i$
by *force*
hence $lms\text{-}slice\ uw\ `\ set\ [x] = \{lms\text{-}slice\ uw\ i\}$
using $A(1)$ **by** *auto*
hence $ordlistns.elem\text{-}rank\ (lms\text{-}slice\ uw\ `\ set\ [x])\ (lms\text{-}slice\ uw\ i) = 0$
by (*simp add*: *ordlistns.elem-rank-def elm-rank-def*)
then show *?case*
by (*simp add*: ‹$x = i$› $A(4)$)
next
case (*3 T a b xs names j*)
note $IH = this$

have *A1*: $distinct\ (b\ \#\ xs)$
using $IH(3)$ **by** *fastforce*

have *A2*: $ordlistns.sorted\ (map\ (lms\text{-}slice\ T)\ (b\ \#\ xs))$
using $IH(4)$ **by** *fastforce*

have *A3*: $i < length\ (names[a := j])$
by (*simp add*: $IH(6)$)

have *A4*: $a \notin set\ (b\ \#\ xs)$
using $IH(3)$ **by** *auto*

have *A5*: $ordlistns.elem\text{-}rank\ (lms\text{-}slice\ T\ `\ set\ (a\ \#\ b\ \#\ xs))\ (lms\text{-}slice\ T\ a) =$
0
unfolding *ordlistns.elem-rank-def elm-rank-def* **using** $IH(4)$ **by** *auto*

note $IH1 = IH(1)[OF\ \text{-}\ A1\ A2\ \text{-}\ A3]$
note $IH2 = IH(2)[OF\ \text{-}\ A1\ A2\ \text{-}\ A3]$

have *P*: $i \in set\ (b\ \#\ xs) \lor i = a$
using $IH(5)$ **by** *force*

have $lms\text{-}slice\ T\ a = lms\text{-}slice\ T\ b\ \lor$
$lms\text{-}slice\ T\ a \neq lms\text{-}slice\ T\ b$
by *blast*

then show *?case*
proof
assume *B*: *lms-slice T a* = *lms-slice T b*
hence *C*: *abs-rename-mapping′ T* (*a* # *b* # *xs*) *names j* =
abs-rename-mapping′ T (*b* # *xs*) (*names*[*a* := *j*]) *j*
by *simp*

from *IH1*[*OF B*] *C*
have $i \in$ *set* (*b* # *xs*) $\Longrightarrow$ *?thesis*
by (*simp add*: *B list.set-map*)
moreover
from *rename-mapping′-unchanged*[*OF A4*, *of names*[*a* := *j*] *T j*, *simplified C*[*symmetric*]] *IH*(*6*) *A5*
have $i = a \Longrightarrow$ *?thesis*
by *simp*
moreover
note *P*
ultimately show *?thesis*
by *blast*
next
assume *B*: *lms-slice T a* $\neq$ *lms-slice T b*
hence *C*: *abs-rename-mapping′ T* (*a* # *b* # *xs*) *names j* =
abs-rename-mapping′ T (*b* # *xs*) (*names*[*a* := *j*]) (*Suc j*)
by *simp*

have *D*: *lms-slice T a* $\notin$ *lms-slice T* ‘ *set* (*b* # *xs*)
proof
assume *lms-slice T a* $\in$ *lms-slice T* ‘ *set* (*b* # *xs*)
moreover
from *IH*(*4*) *ordlistns.sorted-simps*(*2*)[*of lms-slice T a map* (*lms-slice T*) (*b* # *xs*)]
have $\forall$ *y* $\in$ *set* (*map* (*lms-slice T*) (*b* # *xs*)). *list-less-eq-ns* (*lms-slice T a*) *y*
by *auto*
ultimately show *False*
using *A2 B* **by** *auto*
qed

from *rename-mapping′-unchanged*[*OF A4*, *of names*[*a* := *j*] *T Suc j*, *simplified C*[*symmetric*]]
have $i = a \Longrightarrow$ *?thesis*
using *A5 IH*(*6*) **by** *auto*
moreover
have $i \in$ *set* (*b* # *xs*) $\Longrightarrow$ *?thesis*
proof −
assume $i \in$ *set* (*b* # *xs*)
with *IH2*[*OF B*, *simplified C*[*symmetric*]] *C*
have *abs-rename-mapping′ T* (*a* # *b* # *xs*) *names j* ! *i* =
j + *Suc* (*ordlistns.elem-rank* (*lms-slice T* ‘ *set* (*b* # *xs*)) (*lms-slice T i*))
by *linarith*

moreover
have *lms-slice T ‘ set* (*a* # *b* # *xs*) =
insert (*lms-slice T a*) (*lms-slice T ‘ set* (*b* # *xs*))
by *simp*
moreover
have *ordlistns.elem-rank* (*insert* (*lms-slice T a*) (*lms-slice T ‘ set* (*b* # *xs*)))
(*lms-slice T i*) =
Suc (*ordlistns.elem-rank* (*lms-slice T ‘ set* (*b* # *xs*)) (*lms-slice T i*))
proof (*intro ordlistns.elem-rank-insert-min*)
from *D*
show *lms-slice T a* ∉ *lms-slice T ‘ set* (*b* # *xs*) **.**
next
show *finite* (*lms-slice T ‘ set* (*b* # *xs*))
by *blast*
next
show ∀ *y*∈*lms-slice T ‘ set* (*b* # *xs*). *list-less-ns* (*lms-slice T a*) *y*
using *D IH*(*4*) **by** *fastforce*
next
show *lms-slice T i* ∈ *lms-slice T ‘ set* (*b* # *xs*)
using ‹*i* ∈ *set* (*b* # *xs*)› **by** *blast*
qed
ultimately show *?thesis*
by *presburger*
qed
moreover
note *P*
ultimately show *?thesis*
by *blast*
qed
qed

lemma *abs-rename-mapping-lms*:
assumes *distinct LMS*
and *ordlistns.sorted* (*map* (*lms-slice T*) *LMS*)
and *i* ∈ *set LMS*
and *i* < *length T*
shows (*abs-rename-mapping T LMS*) ! *i* =
ordlistns.elem-rank ((*lms-slice T*) ‘ *set LMS*) (*lms-slice T i*)
unfolding *abs-rename-mapping-def*
using *rename-mapping′-lms*[**where** *j* = *0*, *simplified*, *OF assms*(*1*−*3*)] *assms*(*4*)
abs-rename-mapping-length[*of T LMS*]
by *auto*

lemma *abs-rename-mapping-lms-all*:
assumes *distinct LMS*
and *ordlistns.sorted* (*map* (*lms-slice T*) *LMS*)
and ∀ *x* ∈ *set LMS*. *x* < *length T*
shows ∀ *x* ∈ *set LMS*. (!) (*abs-rename-mapping T LMS*) *x* =

ordlistns.elem-rank (*lms-slice* *T* ‘ *set* *LMS*) (*lms-slice* *T* *x*)
using *assms*(*1*) *assms*(*2*) *assms*(*3*) *abs-rename-mapping-lms* **by** *blast*

lemma *map-abs-rename-mapping*:
assumes *distinct LMS*
and *ordlistns.sorted* (*map* (*lms-slice* *T*) *LMS*)
and $\forall x \in$ *set LMS*. $x <$ *length T*
and *set xs* $\subseteq$ *set LMS*
shows *map* ((!) (*abs-rename-mapping T LMS*)) *xs* =
map (*ordlistns.elem-rank* (*lms-slice* *T* ‘ *set* *LMS*)) (*map* (*lms-slice* *T*) *xs*)
using *assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *abs-rename-mapping-lms-all* **by** *fastforce*

94 Rename String Proofs

lemma *rename-list-length*:
length (*rename-string xs names*) = *length xs*
by (*induct xs*; *simp*)

theorem *rename-list-correct*:
rename-string T names = *map* (λx. *names* ! *x*) *T*
by (*induct T*; *simp*)

corollary *rename-list-nth*:
$i <$ *length T* $\Longrightarrow$ (*rename-string T names*) ! *i* = *names* ! (*T* ! *i*)
by (*simp add*: *rename-list-correct*)

end
theory *Abs-SAIS-Verification-With-Valid-Precondition*
imports
Abs-Induce-Verification
Abs-Rename-LMS-Verification
Abs-Extract-LMS-Verification
Abs-Order-LMS-Verification
begin

95 SAIS General Helpers

termination *abs-sais*
by (*relation measure* (λxs. *length xs*))
(*simp* (*no-asm-simp*)
del: *List.list.size*(*4*)
add: *rename-list-length length-filter-lms*)+

lemma *abs-sais-reduced-string*:

assumes *LMS1* = *lms0-seq T*
and *distinct LMS2*
and *set LMS2* = {*i. abs-is-lms T i*}
and *ordlistns.sorted* (*map* (*lms-slice T*) *LMS2*)
and *names* = *abs-rename-mapping T LMS2*
and *T'* = *rename-string LMS1 names*
shows *T'* = *lms-map T* (*lms0-suffix T*)
proof −
let *?T'* = *lms0-map T*

have *set-LMS1*: *set LMS1* = {*i. abs-is-lms T i*}
using *assms*(*1*) *lms0-seq-has-all-lms* **by** *blast*

have *distinct LMS1*
by (*simp add*: *assms*(*1*) *lms-seq-distinct*)

have *T'* = *map* (λ*x. names* ! *x*) *LMS1*
using *rename-list-correct assms*(*6*) **by** *auto*

have ∀ *x* ∈ *set LMS2*. *x* < *length T*
using *assms*(*3*) *abs-is-lms-imp-less-length* **by** *blast*
with *map-abs-rename-mapping*[*OF assms*(*2*,*4*), *simplified assms*(*3*),
of LMS1, *simplified* ‹*LMS1* = *lms0-seq T*›]
‹*T'* = *map* (λ*x. names* ! *x*) *LMS1*› ‹*set LMS2* = {*i. abs-is-lms T i*}›
have *T'* = *map* (*ordlistns.elem-rank* (*lms-substrs T*)) (*map* (*lms-slice T*) (*lms0-seq T*))
using *assms*(*1*) *assms*(*5*) *set-LMS1* **by** *blast*
moreover
have *map* (*ordlistns.elem-rank* (*lms-substrs T*)) (*map* (*lms-slice T*) (*lms0-seq T*)) = *?T'*
unfolding *lms-map-def*
by (*metis* (*no-types*, *opaque-lifting*) *comp-apply lms-substr-seq-def lms-subtrs-seq-id-suffix*)
ultimately show *T'* = *?T'*
by (*simp only*:)
qed

96 SAIS cases simplifications

lemma *abs-sais-distinct-simp*:
assumes *T* = *a* # *b* # *xs*
and *LMS0* = *abs-extract-lms T* [*0*..<*length T*]
and *SA* = *abs-sa-induce id T LMS0*
and *LMS* = *abs-extract-lms T SA*
and *names* = *abs-rename-mapping T LMS*
and *T'* = *rename-string LMS0 names*
and *distinct T'*
shows *abs-sais T* = *abs-sa-induce id T LMS*
proof −
let *?P* = λ*ys. abs-sais* (*a* # *b* # *xs*) = *ys*

from *subst*[*OF abs-sais.simps*(*3*), *of ?P a b xs*,
simplified Let-def assms(*1*−*6*)[*symmetric*] *assms*(*7*),
simplified]
show *?thesis*
by *simp*
qed

lemma *abs-sais-not-distinct-simp*:
assumes *T* = *a* # *b* # *xs*
and *LMS0* = *abs-extract-lms T* [*0*..<*length T*]
and *SA* = *abs-sa-induce id T LMS0*
and *LMS* = *abs-extract-lms T SA*
and *names* = *abs-rename-mapping T LMS*
and *T′* = *rename-string LMS0 names*
and *LMS1* = *order-lms LMS0* (*abs-sais T′*)
and ¬ *distinct T′*
shows *abs-sais T* = *abs-sa-induce id T LMS1*
proof −
let *?P* = *λys. abs-sais* (*a* # *b* # *xs*) = *ys*
from *subst*[*OF abs-sais.simps*(*3*), *of ?P a b xs*,
simplified Let-def assms(*1*−*7*)[*symmetric*] *assms*(*8*),
simplified]
show *?thesis*
by *simp*
qed

97 SAIS returns a permutation

theorem *abs-sais-permutation*:
valid-list T ⟹ *abs-sais T* <~~> [*0*..<*length T*]
proof(*induct rule*: *abs-sais.induct*[*of - T*])
case *1*
then show *?case* **by** *simp*
next
case (*2 x*)
then show *?case* **by** *simp*
next
case (*3 a b xs*)
note *IH* = *this*
let *?T* = *a* # *b* # *xs*
have *T*: *?T* = *a* # *b* # *xs*
by (*simp only*:)
let *?LMS1* = *abs-extract-lms ?T* [*0*..<*length ?T*]
have *LMS1*: *?LMS1* = *abs-extract-lms ?T* [*0*..<*length ?T*]
by (*simp only*:)
let *?SA1* = *abs-sa-induce id ?T ?LMS1*
have *SA1*: *?SA1* = *abs-sa-induce id ?T ?LMS1*
by (*simp only*:)
let *?LMS2* = *abs-extract-lms ?T ?SA1*

have *LMS2*: *?LMS2* = *abs-extract-lms ?T ?SA1*
  by (*simp only*:)
let *?names* = *abs-rename-mapping ?T ?LMS2*
have *names*: *?names* = *abs-rename-mapping ?T ?LMS2*
  by (*simp only*:)
let *?T′* = *rename-string ?LMS1 ?names*
have *T′*: *?T′* = *rename-string ?LMS1 ?names*
  by (*simp only*:)
let *?LMS3* = *order-lms ?LMS1* (*abs-sais ?T′*)
have *LMS3*: *?LMS3* = *order-lms ?LMS1* (*abs-sais ?T′*)
  by (*simp only*:)

from *IH*(*1*)[*OF T LMS1 SA1 LMS2 names T′*]
have *IH′*: ⟦¬*distinct ?T′*; *valid-list ?T′*⟧ ⟹ *abs-sais ?T′* <~~> [*0*..<*length ?T′*]
  by *assumption*

have *distinct ?LMS1*
  using *distinct-abs-extract-lms*
  by *fastforce*

have *set ?LMS1* = {*i*. *abs-is-lms ?T i*}
  using *set-abs-extract-lms-eq-all-lms*
  by (*metis comp-apply*)

have *id*: *strict-mono* (*id* :: *nat* ⇒ *nat*) (*id* :: *nat* ⇒ *nat*) *bot* = *0*
  by (*simp add*: *strict-mono-def bot-nat-def*)+

have *len*: *Suc 0* < *length ?T*
  by *simp*

from *distinct-filter-abs-sa-induce*
    [*OF* ‹*set ?LMS1* = {*i*. *abs-is-lms ?T i*}› ‹*distinct ?LMS1*› *IH*(*2*) *id len*]
have *distinct-LMS2*: *distinct ?LMS2*
  by (*metis comp-apply*)

from *filter-abs-sa-induce-eq-all-lms*
    [*OF* ‹*set ?LMS1* = {*i*. *abs-is-lms ?T i*}› ‹*distinct ?LMS1*› *IH*(*2*) *id len*]
have *set-LMS2*: *set ?LMS2* = {*i*. *abs-is-lms ?T i*}
  by *blast*

from *abs-sa-induce-permutation*
    [*OF* ‹*set ?LMS2* = {*i*. *abs-is-lms ?T i*}› ‹*distinct ?LMS2*› *IH*(*2*) *id len*]
have *abs-sa-induce id ?T ?LMS2* <~~> [*0*..<*length ?T*]
  by *assumption*

from *rename-list-length*[*of ?LMS1 ?names*]
have *length ?T′* = *length ?LMS1*
  by *assumption*

have *sorted-LMS2*: *ordlistns.sorted* (*map* (*lms-slice ?T*) *?LMS2*)
by (*metis 3.prems* ‹*distinct ?LMS1*› ‹*set ?LMS1* = {*i*. *abs-is-lms ?T i*}› *comp-apply len id*
ordlistns.sorted-filter abs-sa-induce-prefix-sorted)

have *?LMS1* = *lms0-seq ?T*
by (*metis comp-apply lms-seq-0-zeroth-lms lms-seq-def*)
with *abs-sais-reduced-string*[*OF - distinct-LMS2 set-LMS2 sorted-LMS2 names T'*]
have *?T'* = *lms0-map ?T*
by *blast*

have *abs-is-lms ?T* (*lms0 ?T*)
by (*metis 3.prems abs-find-next-lms-less-length-abs-is-lms length-Cons*
abs-is-lms-last len no-lms-between-i-and-next not-less-eq)

from *valid-list-lms-map*[*OF IH*(*2*) ‹*abs-is-lms ?T* (*lms0 ?T*)›]
have *valid-list* (*lms0-map ?T*) **.**
hence *valid-list ?T'*
by (*simp only*: ‹*?T'* = (*lms0-map ?T*)›)

from ‹*length ?T'* = *length ?LMS1*› ‹*distinct ?LMS1*› ‹*set ?LMS1* = {*i*. *abs-is-lms ?T i*}›
have *R1*: *abs-sais ?T'* <~~> [*0..*<*length ?T'*] ⟹ *distinct ?LMS3*
by (*metis distinct-abs-order-lms*)

from ‹*length ?T'* = *length ?LMS1*› ‹*distinct ?LMS1*› ‹*set ?LMS1* = {*i*. *abs-is-lms ?T i*}›
have *R2*: *abs-sais ?T'* <~~> [*0..*<*length ?T'*] ⟹ *set ?LMS3* = {*i*. *abs-is-lms ?T i*}
by (*metis* (*no-types*, *lifting*) *abs-order-lms-eq-all-lms*)

from *abs-sa-induce-permutation*[*OF R2 R1 IH*(*2*) *id len*]
have *R3*: *abs-sais ?T'* <~~> [*0..*<*length ?T'*] ⟹
abs-sa-induce id ?T ?LMS3 <~~> [*0..*<*length ?T*]
by *blast*

from *IH'*[*OF -* ‹*valid-list ?T'*›]
have ¬*distinct ?T'* ⟹ *abs-sais ?T'* <~~> [*0..*<*length ?T'*]
by *assumption*
with *R3*
have ¬*distinct ?T'* ⟹ *abs-sa-induce id ?T ?LMS3* <~~> [*0..*<*length ?T*]
by *blast*

have *distinct ?T'* ∨ ¬*distinct ?T'*
by *blast*
then show *?case*
proof
assume *A*: *distinct ?T'*

from *abs-sais-distinct-simp*[*OF T LMS1 SA1 LMS2 names T′ A*]
‹*abs-sa-induce id ?T ?LMS2* <~~> [*0..<length ?T*]›
show *?thesis*
by *metis*
next
assume *A*: ¬*distinct ?T′*
from *abs-sais-not-distinct-simp*[*OF T LMS1 SA1 LMS2 names T′ LMS3 A*]
‹¬*distinct ?T′* ⟹ *abs-sa-induce id ?T ?LMS3* <~~> [*0..<length ?T*]›[*OF A*]
show *?thesis*
by *metis*
qed
qed

98 SAIS Sorted Helpers

lemma *abs-sais-subset-idx*:
assumes *valid-list T*
shows *set* (*abs-sais T*) ⊆ {*0..<length T*}
using *assms perm-distinct-set-of-upt-iff abs-sais-permutation* **by** *auto*

99 SAIS sorts suffixes

theorem *abs-sais-sorted-alt*:
valid-list T ⟹
ordlistns.strict-sorted (*map* (*suffix T*) (*abs-sais T*))
proof(*induct rule*: *abs-sais.induct*[*of - T*])
case *1*
then show *?case* **by** *simp*
next
case (*2 x*)
then show *?case* **by** *simp*
next
case (*3 a b xs*)
note *IH* = *this*
let *?T* = *a* # *b* # *xs*
have *T*: *?T* = *a* # *b* # *xs*
by (*simp only*:)
let *?LMS1* = *abs-extract-lms ?T* [*0..<length ?T*]
have *LMS1*: *?LMS1* = *abs-extract-lms ?T* [*0..<length ?T*]
by (*simp only*:)
let *?SA1* = *abs-sa-induce id ?T ?LMS1*
have *SA1*: *?SA1* = *abs-sa-induce id ?T ?LMS1*
by (*simp only*:)
let *?LMS2* = *abs-extract-lms ?T ?SA1*
have *LMS2*: *?LMS2* = *abs-extract-lms ?T ?SA1*
by (*simp only*:)
let *?names* = *abs-rename-mapping ?T ?LMS2*

have *names*: *?names = abs-rename-mapping ?T ?LMS2*
  by (*simp only*:)
let *?T′ = rename-string ?LMS1 ?names*
have *T′*: *?T′ = rename-string ?LMS1 ?names*
  by (*simp only*:)
let *?LMS3 = order-lms ?LMS1* (*abs-sais ?T′*)
have *LMS3*: *?LMS3 = order-lms ?LMS1* (*abs-sais ?T′*)
  by (*simp only*:)

from *IH*(*1*)[*OF T LMS1 SA1 LMS2 names T′*]
have *IH′*: ⟦¬*distinct ?T′*; *valid-list ?T′*⟧ ⟹
    ordlistns.strict-sorted (*map* (*suffix ?T′*) (*abs-sais ?T′*))
  by *blast*

from *set-abs-extract-lms-eq-all-lms*[*of ?T*]
have *set-LMS1*: *set ?LMS1* = {*i. abs-is-lms ?T i*}
  by *simp*

from *distinct-abs-extract-lms*[*of ?T*]
have *distinct-LMS1*: *distinct ?LMS1*
  by *simp*

have *id*: *strict-mono* (*id* :: *nat* ⇒ *nat*) (*id* :: *nat* ⇒ *nat*) *bot* = *0*
  by (*simp add*: *strict-mono-def bot-nat-def*)+

have *len*: *Suc 0* < *length ?T*
  by *simp*

from *distinct-filter-abs-sa-induce*[*OF* ‹*set ?LMS1* = {*i. abs-is-lms ?T i*}› ‹*distinct ?LMS1*› *IH*(*2*) *id len*]
have *distinct-LMS2*: *distinct ?LMS2*
  by *blast*

from *filter-abs-sa-induce-eq-all-lms*[*OF* ‹*set ?LMS1* = {*i. abs-is-lms ?T i*}› ‹*distinct ?LMS1*› *IH*(*2*) *id len*]
have *set-LMS2*: *set ?LMS2* = {*i. abs-is-lms ?T i*}
  by *blast*

from *distinct-set-imp-perm*[*OF* ‹*distinct ?LMS1*› ‹*distinct ?LMS2*›]
    ‹*set ?LMS1* = {*i. abs-is-lms ?T i*}›
    ‹*set ?LMS2* = {*i. abs-is-lms ?T i*}›
have *?LMS1* <~~> *?LMS2*
  by *blast*

have *sorted-LMS2*: *ordlistns.sorted* (*map* (*lms-slice ?T*) *?LMS2*)
  by (*metis 3.prems* ‹*distinct ?LMS1*› ‹*set ?LMS1* = {*i. abs-is-lms ?T i*}› *comp-apply len id*
    ordlistns.sorted-filter abs-sa-induce-prefix-sorted)

```
  have ?LMS1 = lms0-seq ?T
    by (metis comp-apply lms-seq-0-zeroth-lms lms-seq-def)

  with abs-sais-reduced-string[OF - distinct-LMS2 set-LMS2 sorted-LMS2 names
T']
  have ?T' = lms0-map ?T
    by blast

  have abs-is-lms ?T (lms0 ?T)
    by (metis 3.prems abs-find-next-lms-less-length-abs-is-lms abs-is-lms-last
              len length-Cons no-lms-between-i-and-next not-less-eq)

  from valid-list-lms-map[OF IH(2) ‹abs-is-lms ?T (lms0 ?T)›]
  have valid-list (lms0-map ?T) .
  hence valid-list ?T'
    by (simp only: ‹?T' = (lms0-map ?T)›)

  have distinct ?T' ∨ ¬distinct ?T'
    by blast
  hence ordlistns.sorted (map (suffix ?T) (abs-sais ?T))
  proof
    assume A: distinct ?T'
    hence distinct (lms0-map ?T)
      by (simp only: ‹?T' = (lms0-map ?T)›)
    with sorted-distinct-lms-substr-perm[OF sorted-LMS2]
    have ordlistns.sorted (map (suffix ?T) ?LMS2)
      by (metis ‹?LMS1 = lms0-seq ?T› ‹?LMS1 <~~> ?LMS2›)
    with abs-sa-induce-suffix-sorted[OF set-LMS2 distinct-LMS2 IH(2) id len]
    have ordlistns.sorted (map (suffix ?T) (abs-sa-induce id ?T ?LMS2))
      using ordlistns.strict-sorted-imp-sorted by blast
    with abs-sais-distinct-simp[OF T LMS1 SA1 LMS2 names T' A]
    show ?thesis
      by presburger
  next
    assume A: ¬distinct ?T'
    with IH'[OF - ‹valid-list ?T'›]
    have ordlistns.strict-sorted (map (suffix ?T') (abs-sais ?T'))
      by blast
   hence C1: ordlistns.strict-sorted (map (suffix (lms0-map ?T)) (abs-sais (lms0-map
?T)))
      by (simp only: ‹?T' = (lms0-map ?T)›)

    from abs-order-lms-eq-map-nth[of ?LMS1 abs-sais ?T']
         ‹?LMS1 = lms0-seq ?T› ‹?T' = lms0-map ?T›
    have C2: order-lms ?LMS1 (abs-sais ?T') =
             map (nth (lms0-seq ?T)) (abs-sais (lms0-map ?T))
      by (simp only:)

    note perm = abs-sais-permutation[OF ‹valid-list ?T'›,
```

simplified rename-list-length]

note *set-LMS3* = *abs-order-lms-eq-all-lms*[*OF perm set-LMS1*]
note *distinct-LMS3* = *distinct-abs-order-lms*[*OF perm distinct-LMS1*]

from *abs-sais-permutation*[*OF* ‹*valid-list* (*lms0-map ?T*)›]
have ∀ *y* ∈ *set* (*abs-sais* (*lms0-map ?T*)). *y* < *card* {*i*. *abs-is-lms ?T i*}
by (*metis atLeastLessThan-iff card-lms-suffixes length-reduced-seq perm-set-eq set-upt*)
with *sorted-reduced-seq-imp-lms*[*OF C1*] *C2*
have *ordlistns.strict-sorted* (*map* (*suffix ?T*) *?LMS3*)
by *presburger*
with *abs-sa-induce-suffix-sorted*[*OF set-LMS3 distinct-LMS3 IH*(*2*) *id len*]
have *ordlistns.sorted* (*map* (*suffix ?T*) (*abs-sa-induce id ?T ?LMS3*))
using *ordlistns.strict-sorted-imp-sorted* **by** *blast*
with *abs-sais-not-distinct-simp*[*OF T LMS1 SA1 LMS2 names T′ LMS3 A*]
show *?thesis*
by *presburger*
qed
moreover
from *perm-distinct-set-of-upt-iff* [*THEN iffD1*, *OF abs-sais-permutation*[*OF IH*(*2*)]]
have *distinct* (*map* (*suffix ?T*) (*abs-sais ?T*))
by (*metis atLeastLessThan-iff distinct-suffixes*)
ultimately show *?case*
using *ordlistns.strict-sorted-iff* **by** *blast*
qed

theorem *abs-sais-sorted*:
valid-list T ⟹
strict-sorted (*map* (*suffix T*) (*abs-sais T*))
using *abs-sais-sorted-alt abs-sais-subset-idx valid-list-ordlist-ordlistns-strict-sorted-eq*
by *blast*

100 Verification of a SAIS construction algorithm

interpretation *abs-sais*: *Suffix-Array-Restricted abs-sais*
using *Suffix-Array-Restricted.intro abs-sais-permutation abs-sais-sorted* **by** *blast*

end
theory *Abs-SAIS-Verification*
imports *Abs-SAIS-Verification-With-Valid-Precondition*
begin

101 Final Theorem: Verification of a generalised SAIS construction algorithm

The @term *abs-sais* implementation produces an output that is equivalent to that of a suffix array construction algorithm for lists of any type that can be linearly ordered. This lifts the restriction that the algorithm only operates on natural numbers terminated by a bottom element.

interpretation *abs-sais-gen*: *Suffix-Array-General sa-nat-wrapper map-to-nat abs-sais*
 by (*simp add*: *Suffix-Array-Restricted-imp-General abs-sais.Suffix-Array-Restricted-axioms*)

theorem *abs-sais-gen-is-Suffix-Array-General*:
 Suffix-Array-General sa $\longleftrightarrow$ *sa* = *sa-nat-wrapper map-to-nat abs-sais*
 using *Suffix-Array-General-determinism abs-sais-gen.Suffix-Array-General-axioms*
by *auto*

end
theory *Bucket-Insert*
 imports
  ../../util/Repeat
begin

102 Bucket Insert

fun *bucket-insert-step* ::
 $nat\ list \times nat\ list \times nat \Rightarrow$
  $(('a :: \{linorder,\ order\text{-}bot\}) \Rightarrow nat) \times 'a\ list \times nat\ list \Rightarrow$
  $nat\ list \times nat\ list \times nat$
 where
bucket-insert-step $(B,\ SA,\ i)\ (\alpha,\ T,\ LMS) =$
 $(let\ b = \alpha\ (T\ !\ (LMS\ !\ i));$
   $k = B\ !\ b - Suc\ 0$
  $in\ (B[b := k],\ SA[k := LMS\ !\ i],\ Suc\ i))$

definition *bucket-insert-base* ::
 $(('a :: \{linorder,\ order\text{-}bot\}) \Rightarrow nat) \Rightarrow 'a\ list \Rightarrow nat\ list \Rightarrow nat\ list \Rightarrow nat\ list$
$\Rightarrow$
  $nat\ list \times nat\ list \times nat$
 where
bucket-insert-base α *T B SA LMS* = *repeat* (*length LMS*) *bucket-insert-step* (*B*, *SA*, *0*) (α, *T*, *LMS*)

definition *bucket-insert* ::
 $(('a :: \{linorder,\ order\text{-}bot\}) \Rightarrow nat) \Rightarrow 'a\ list \Rightarrow nat\ list \Rightarrow nat\ list \Rightarrow nat\ list$
$\Rightarrow$
  nat list
 where
bucket-insert α *T B SA LMS* =
 (*let* $(B',\ SA',\ i)$ = *bucket-insert-base* α *T B SA LMS*

in SA′)

end
theory *Get-Types*
imports
../prop/List-Type
../prop/LMS-List-Slice-Util
../../util/Repeat
begin

103 Suffix Types

fun
get-suffix-types-step-r0 ::
SL-types list × *nat* ⇒ *′a* :: {*linorder*, *order-bot*} *list* ⇒ *SL-types list* × *nat*
where
get-suffix-types-step-r0 (*xs*, *i*) *ys* =
(*case i of*
0 ⇒ (*xs*, *0*)
| *Suc j* ⇒
(*if Suc j* < *length xs* ∧ *Suc j* < *length ys then*
(*if ys* ! *j* < *ys* ! *Suc j then*
(*xs*[*j* := *S-type*], *j*)
else if ys ! *j* > *ys* ! *Suc j then*
(*xs*[*j* := *L-type*], *j*)
else
(*xs*[*j* := *xs* ! *Suc j*], *j*))
else
(*xs*, *j*)))

definition *get-suffix-types-base*
where
get-suffix-types-base xs ≡
repeat (*length xs* − *Suc 0*) *get-suffix-types-step-r0*
(*replicate* (*length xs*) *S-type*, *length xs* − *Suc 0*) *xs*

definition *get-suffix-types*
where
get-suffix-types xs ≡ *fst* (*get-suffix-types-base xs*)

104 LMS types

fun *is-lms-ref*
where
is-lms-ref ST 0 = *False* |
is-lms-ref ST (*Suc i*) =
(*if Suc i* < *length ST then ST* ! *i* = *L-type* ∧ *ST* ! (*Suc i*) = *S-type else False*)

105 Extracting LMS types

abbreviation *extract-lms ST xs* ≡ *filter* (*λi. is-lms-ref ST i*) *xs*

106 LMS Substrings

definition *find-next-lms* :: *SL-types list* ⇒ *nat* ⇒ *nat*
where
find-next-lms ST i =
(*case find* (*λj. is-lms-ref ST j*) [*Suc i..<length ST*] *of*
Some j ⇒ *j*
| - ⇒ *length ST*)

definition
lms-slice-ref ::
(*'a* :: {*linorder*, *order-bot*}) *list* ⇒ *SL-types list* ⇒ *nat* ⇒ *'a list*
where
lms-slice-ref T ST i =
list-slice T i (*Suc* (*find-next-lms ST i*))

107 Rename Mapping

fun *rename-mapping'* ::
(*'a* :: {*linorder*, *order-bot*}) *list* ⇒ *SL-types list* ⇒
nat list ⇒ *nat list* ⇒ *nat* ⇒ *nat list*
where
rename-mapping' - - [] *names* - = *names* |
rename-mapping' - - [*x*] *names i* = *names*[*x* := *i*] |
rename-mapping' T ST (*a* # *b* # *xs*) *names i* =
(*if lms-slice-ref T ST a* = *lms-slice-ref T ST b*
then
rename-mapping' T ST (*b* # *xs*) (*names*[*a* := *i*]) *i*
else
rename-mapping' T ST (*b* # *xs*) (*names*[*a* := *i*]) (*Suc i*))

definition
rename-mapping ::
(*'a* :: {*linorder*, *order-bot*}) *list* ⇒ *SL-types list* ⇒ *nat list* ⇒ *nat list*
where
rename-mapping T ST LMS =
rename-mapping' T ST LMS (*replicate* (*length T*) (*length T*)) *0*

end
theory *Induce-L*
imports
../../util/Repeat
../prop/Buckets
begin

108 Induce L Refinement

```
fun induce-l-step-r0 ::
  nat list × nat list × nat ⇒
   (('a :: {linorder, order-bot}) ⇒ nat) × 'a list ⇒
   nat list × nat list × nat
  where
induce-l-step-r0 (B, SA, i) (α, T) =
  (if SA ! i < length T
   then
     (case SA ! i of
        Suc j ⇒
          (case suffix-type T j of
             L-type ⇒
               (let k = α (T ! j);
                    l = B ! k
                in (B[k := Suc l], SA[l := j], Suc i))
             | - ⇒ (B, SA, Suc i))
        | - ⇒ (B, SA, Suc i))
   else (B, SA, Suc i))

fun induce-l-step ::
  nat list × nat list × nat ⇒
   (('a :: {linorder, order-bot}) ⇒ nat) × 'a list × SL-types list⇒
   nat list × nat list × nat
  where
induce-l-step (B, SA, i) (α, T, ST) =
  (if SA ! i < length T
   then
     (case SA ! i of
        Suc j ⇒
          (case ST ! j of
             L-type ⇒
               (let k = α (T ! j);
                    l = B ! k
                in (B[k := Suc (B ! k)], SA[l := j], Suc i))
             | - ⇒ (B, SA, Suc i))
        | - ⇒ (B, SA, Suc i))
   else (B, SA, Suc i))

definition induce-l-base ::
  (('a :: {linorder, order-bot}) ⇒ nat) ⇒
   'a list ⇒
   SL-types list ⇒
   nat list ⇒
   nat list ⇒
   nat list × nat list × nat
  where
induce-l-base α T ST B SA = repeat (length T) induce-l-step (B, SA, 0) (α, T,
```

ST)

definition *induce-l* ::
  ((*'a* :: {*linorder*, *order-bot*}) ⇒ *nat*) ⇒
   'a list ⇒
   SL-types list ⇒
   nat list ⇒
   nat list ⇒
   nat list
  where
induce-l α *T ST B SA* = (*let* (*B'*, *SA'*, *i*) = *induce-l-base* α *T ST B SA in SA'*)

end
theory *Induce-S*
  imports ../*abs−proof*/*Abs-Induce-S-Verification*
begin

109 Induce S Refinement

fun *induce-s-step-r0* ::
  nat list × *nat list* × *nat* ⇒
   ((*'a* :: {*linorder*, *order-bot*}) ⇒ *nat*) × *'a list* ⇒
   nat list × *nat list* × *nat*
  where
induce-s-step-r0 (*B*, *SA*, *i*) (α, *T*) =
  (*case i of*
    Suc n ⇒
      (*if Suc n* < *length SA* ∧ *SA* ! *Suc n* < *length T then*
        (*case SA* ! *Suc n of*
          Suc j ⇒
            (*case suffix-type T j of*
              S-type ⇒
                (*let b* = α (*T* ! *j*);
                      k = *B* ! *b* − *Suc 0*
                 in (*B*[*b* := *k*], *SA*[*k* := *j*], *n*)
)
              | - ⇒ (*B*, *SA*, *n*)
)
          | - ⇒ (*B*, *SA*, *n*)
)
        else
          (*B*, *SA*, *n*)
)
    | -   ⇒ (*B*, *SA*, *0*)
)

fun *induce-s-step-r1* ::
  nat list × *nat list* × *nat* ⇒
   ((*'a* :: {*linorder*, *order-bot*}) ⇒ *nat*) × *'a list* × *SL-types list* ⇒

```
  nat list × nat list × nat
 where
induce-s-step-r1 (B, SA, i) (α, T, ST) =
 (case i of
   Suc n ⇒
    (if Suc n < length SA ∧ SA ! Suc n < length T then
      (case SA ! Suc n of
        Suc j ⇒
         (case ST ! j of
           S-type ⇒
            (let b = α (T ! j);
                 k = B ! b − Suc 0
             in (B[b := k], SA[k := j], n)
            )
           | - ⇒ (B, SA, n)
         )
        | - ⇒ (B, SA, n)
      )
      else
        (B, SA, n)
    )
   | -   ⇒ (B, SA, 0)
 )

fun induce-s-step-r2 ::
 nat list × nat list × nat ⇒
  (('a :: {linorder, order-bot}) ⇒ nat) × 'a list × SL-types list ⇒
  nat list × nat list × nat
 where
induce-s-step-r2 (B, SA, i) (α, T, ST) =
 (case i of
   Suc n ⇒
    (if Suc n < length SA  then
      (case SA ! Suc n of
        Suc j ⇒
         (case ST ! j of
           S-type ⇒
            (let b = α (T ! j);
                 k = B ! b − Suc 0
             in (B[b := k], SA[k := j], n)
            )
           | - ⇒ (B, SA, n)
         )
        | - ⇒ (B, SA, n)
      )
      else
        (B, SA, n)
    )
   | -   ⇒ (B, SA, 0)
```

```
  )

fun induce-s-step ::
  nat list × nat list × nat ⇒
   (('a :: {linorder, order-bot}) ⇒ nat) × 'a list × SL-types list ⇒
   nat list × nat list × nat
  where
induce-s-step (B, SA, i) (α, T, ST) =
  (case i of
    Suc n ⇒
    (case SA ! Suc n of
        Suc j ⇒
          (case ST ! j of
            S-type ⇒
              (let b = α (T ! j);
                   k = B ! b − Suc 0
               in (B[b := k], SA[k := j], n)
              )
            | - ⇒ (B, SA, n)
          )
        | - ⇒ (B, SA, n)
      )
    | -   ⇒ (B, SA, 0)
  )

definition induce-s-base ::
  (('a :: {linorder, order-bot}) ⇒ nat) ⇒
   'a list ⇒
   SL-types list ⇒
   nat list ⇒
   nat list ⇒
   nat list × nat list × nat
  where
induce-s-base α T ST B SA = repeat (length T − Suc 0) induce-s-step (B, SA,
length T − Suc 0) (α, T, ST)

definition induce-s ::
  (('a :: {linorder, order-bot}) ⇒ nat) ⇒
   'a list ⇒
   SL-types list ⇒
   nat list ⇒
   nat list ⇒
   nat list
  where
induce-s α T ST B SA = (let (B', SA', i) = induce-s-base α T ST B SA in SA')

end
theory Induce
  imports Induce-S Induce-L Bucket-Insert
```

begin

110 Induce

definition *sa-induce-r0* ::
  $(('a :: \{linorder, order\text{-}bot\}) \Rightarrow nat) \Rightarrow$
   $'a\ list \Rightarrow$
   $nat\ list \Rightarrow$
   $nat\ list$
  where
sa-induce-r0 α *T LMS* =
  (*let*
    $B0 = map\ (bucket\text{-}end\ \alpha\ T)\ [0..<Suc\ (\alpha\ (Max\ (set\ T)))]$;
    $B1 = map\ (bucket\text{-}start\ \alpha\ T)\ [0..<Suc\ (\alpha\ (Max\ (set\ T)))]$;

    — Initialise SA
    $SA = replicate\ (length\ T)\ (length\ T)$;

    — Get the suffix types
    $ST = abs\text{-}get\text{-}suffix\text{-}types\ T$;

    — Insert the LMS types into the suffix array
    $SA = abs\text{-}bucket\text{-}insert\ \alpha\ T\ B0\ SA\ (rev\ LMS)$;

    — Insert the L types into the suffix array
    $SA = induce\text{-}l\ \alpha\ T\ ST\ B1\ SA$

   — Insert the S types into the suffix array
   in $induce\text{-}s\ \alpha\ T\ ST\ (B0[0 := 0])\ SA)$

definition *sa-induce-r1* ::
  $(('a :: \{linorder, order\text{-}bot\}) \Rightarrow nat) \Rightarrow$
   $'a\ list \Rightarrow$
   $SL\text{-}types\ list \Rightarrow$
   $nat\ list \Rightarrow$
   $nat\ list$
  where
sa-induce-r1 α *T ST LMS* =
  (*let*
    $B0 = map\ (bucket\text{-}end\ \alpha\ T)\ [0..<Suc\ (\alpha\ (Max\ (set\ T)))]$;
    $B1 = map\ (bucket\text{-}start\ \alpha\ T)\ [0..<Suc\ (\alpha\ (Max\ (set\ T)))]$;

    — Initialise SA
    $SA = replicate\ (length\ T)\ (length\ T)$;

    — Insert the LMS types into the suffix array
    $SA = abs\text{-}bucket\text{-}insert\ \alpha\ T\ B0\ SA\ (rev\ LMS)$;

    — Insert the L types into the suffix array

SA = *induce-l* α *T ST B1 SA*

— Insert the S types into the suffix array
in induce-s α *T ST* (*B0*[*0* := *0*]) *SA*)

definition *sa-induce-r2* ::
(('*a* :: {*linorder*, *order-bot*}) $\Rightarrow$ *nat*) $\Rightarrow$
'*a list* $\Rightarrow$
SL-types list $\Rightarrow$
nat list $\Rightarrow$
nat list
where
sa-induce-r2 α *T ST LMS* =
(*let*
B0 = *map* (*bucket-end* α *T*) [*0*..<*Suc* (α (*Max* (*set T*)))];
B1 = *map* (*bucket-start* α *T*) [*0*..<*Suc* (α (*Max* (*set T*)))];

— Initialise SA
SA = *replicate* (*length T*) (*length T*);

— Insert the LMS types into the suffix array
SA = *bucket-insert* α *T B0 SA* (*rev LMS*);

— Insert the L types into the suffix array
SA = *induce-l* α *T ST B1 SA*

— Insert the S types into the suffix array
in induce-s α *T ST* (*B0*[*0* := *0*]) *SA*)

abbreviation *sa-induce* $\equiv$ *sa-induce-r2*

end
theory *SAIS*
imports *Induce Get-Types*
begin

111 SAIS

function *sais-r0* ::
nat list $\Rightarrow$
nat list
where
sais-r0 [] = [] |
sais-r0 [*x*] = [*0*] |
sais-r0 (*a* # *b* # *xs*) =
(*let*
T = *a* # *b* # *xs*;

— Compute the suffix types

$ST = abs\text{-}get\text{-}suffix\text{-}types\ T$;

— Extract the LMS types
$LMS0 = extract\text{-}lms\ ST\ [0..<length\ T]$;

— Induce the prefix ordering based on LMS
$SA = sa\text{-}induce\ id\ T\ ST\ LMS0$;

— Extract the LMS types
$LMS1 = extract\text{-}lms\ ST\ SA$;

— Create a new alphabet
$names = rename\text{-}mapping\ T\ ST\ LMS1$;

— Make a reduced string (2 lines)
$T' = rename\text{-}string\ LMS0\ names$;

— Obtain the correct ordering of LMS-types
$LMS2 = (if\ distinct\ T'\ then\ LMS1\ else\ order\text{-}lms\ LMS0\ (sais\text{-}r0\ T'))$

— Induce the suffix ordering based of LMS
in sa-induce id T ST LMS2)
by *pat-completeness blast+*

function *sais-r1* ::
nat list ⇒
nat list
where
sais-r1 [] = [] |
sais-r1 [*x*] = [*0*] |
sais-r1 (*a* # *b* # *xs*) =
(*let*
$T = a\ \#\ b\ \#\ xs$;

— Compute the suffix types
$ST = get\text{-}suffix\text{-}types\ T$;

— Extract the LMS types
$LMS0 = extract\text{-}lms\ ST\ [0..<length\ T]$;

— Induce the prefix ordering based on LMS
$SA = sa\text{-}induce\ id\ T\ ST\ LMS0$;

— Extract the LMS types
$LMS1 = extract\text{-}lms\ ST\ SA$;

— Create a new alphabet
$names = rename\text{-}mapping\ T\ ST\ LMS1$;

— Make a reduced string
T′ = rename-string LMS0 names;

— Obtain the correct ordering of LMS-types
LMS2 = (*if distinct T′ then LMS1 else order-lms LMS0* (*sais-r1 T′*))

— Induce the suffix ordering based of LMS
in sa-induce id T ST LMS2)
by *pat-completeness blast+*

abbreviation *sais* ≡ *sais-r1*

end
theory *Bucket-Insert-Verification*
imports
../abs−proof/Abs-Bucket-Insert-Verification
../def/Bucket-Insert
begin

112 Bucket Insert

lemma *abs-bucket-insert-step-cons*:
assumes *bucket-insert-step* (*B*, *SA*, *Suc i*) (α, *T*, *a* # *xs*) = (*B1*, *SA1*, *j1*)
and *bucket-insert-step* (*B*, *SA*, *i*) (α, *T*, *xs*) = (*B2*, *SA2*, *j2*)
shows *B1* = *B2* ∧ *SA1* = *SA2*
by (*metis assms*(*1*) *assms*(*2*) *bucket-insert-step.simps nth-Cons-Suc prod.sel*(*1*) *prod.sel*(*2*))

lemma *abs-bucket-insert-base-cons′*:
assumes *repeat n bucket-insert-step* (*B*, *SA*, *Suc i*) (α, *T*, *x* # *xs*) = (*B1*, *SA1*, *j1*)
and *repeat n bucket-insert-step* (*B*, *SA*, *i*) (α, *T*, *xs*) = (*B2*, *SA2*, *j2*)
shows *B1* = *B2* ∧ *SA1* = *SA2*
using *assms*
proof (*induct n arbitrary*: *B SA i*)
case *0*
then show *?case*
by (*simp add*: *repeat-0*)
next
case (*Suc n*)
note *IH* = *this*

let *?b* = α (*T* ! (*xs* ! *i*))
let *?k* = *B* ! *?b* − *Suc 0*

have *bucket-insert-step* (*B*, *SA*, *Suc i*) (α, *T*, *x* # *xs*)
= (*B*[*?b* := *?k*], *SA*[*?k* := *xs* ! *i*], *Suc* (*Suc i*))
by (*metis bucket-insert-step.simps nth-Cons-Suc*)
with *IH*(*2*) *repeat-step-forward*[*of n bucket-insert-step* (*B*, *SA*, *Suc i*) (α, *T*, *x*

xs)]
  have *repeat n bucket-insert-step* (*B*[*?b* := *?k*], *SA*[*?k* := *xs* ! *i*], *Suc* (*Suc i*)) (α, *T*, *x* # *xs*)
      = (*B1*, *SA1*, *j1*)
    by *simp*
  moreover
  have *bucket-insert-step* (*B*, *SA*, *i*) (α, *T*, *xs*) = (*B*[*?b* := *?k*], *SA*[*?k* := *xs* ! *i*], *Suc i*)
    by (*metis bucket-insert-step.simps*)
  with *IH*(*3*) *repeat-step-forward*[*of n bucket-insert-step* (*B*, *SA*, *i*) (α, *T*, *xs*)]
  have *repeat n bucket-insert-step* (*B*[*?b* := *?k*], *SA*[*?k* := *xs* ! *i*], *Suc i*) (α, *T*, *xs*)
      = (*B2*, *SA2*, *j2*)
    by *simp*
  ultimately show *?case*
    using *IH*(*1*)[*of B*[*?b* := *?k*] *SA*[*?k* := *xs* ! *i*] *Suc i*]
    by *blast*
qed

lemma *bucket-insert-base-cons*:
  assumes *b* = α (*T* ! *a*)
  and *k* = *B* ! *b* − *Suc 0*
  and *bucket-insert-base* α *T B SA* (*a* # *xs*) = (*B1*, *SA1*, *j1*)
  and *bucket-insert-base* α *T* (*B*[*b* := *k*]) (*SA*[*k* := *a*]) *xs* = (*B2*, *SA2*, *j2*)
shows *B1* = *B2* ∧ *SA1* = *SA2*
proof −
  from *assms*(*1*,*2*)
  have *bucket-insert-step* (*B*, *SA*, *0*) (α, *T*, *a* # *xs*) = (*B*[*b* := *k*], *SA*[*k* := *a*], *Suc 0*)
    by (*metis bucket-insert-step.simps nth-Cons-0*)
  with *assms*(*3*)[*simplified bucket-insert-base-def*, *simplified*]
    repeat-step-forward[*of length xs bucket-insert-step* (*B*, *SA*, *0*) (α, *T*, *a* # *xs*)]
  have *A*: *repeat* (*length xs*) *bucket-insert-step* (*B*[*b* := *k*], *SA*[*k* := *a*], *Suc 0*) (α, *T*, *a* # *xs*)
      = (*B1*, *SA1*, *j1*)
    by *simp*
  with *abs-bucket-insert-base-cons'*[*of length xs B*[*b* := *k*] *SA*[*k* := *a*] *0* α *T a xs B1 SA1 j1 B2 SA2 j2*]
    assms(*4*)[*simplified bucket-insert-base-def*]
  show *?thesis*
    by *simp*
qed

lemma *bucket-insert-cons*:
  assumes *b* = α (*T* ! *a*)
  and *k* = *B* ! *b* − *Suc 0*
shows *bucket-insert* α *T B SA* (*a* # *xs*) = *bucket-insert* α *T* (*B*[*b* := *k*]) (*SA*[*k* := *a*]) *xs*
  by (*clarsimp simp*: *bucket-insert-def Let-def bucket-insert-base-cons*[*of* - α, *OF assms*]

split: *prod.splits*)

lemma *abs-bucket-insert-eq*:
abs-bucket-insert α *T B SA xs* = *bucket-insert* α *T B SA xs*
proof (*induct xs arbitrary*: *B SA*)
case *Nil*
then show *?case*
unfolding *bucket-insert-def bucket-insert-base-def*
by (*simp add*: *repeat-0*)
next
case (*Cons a xs*)
note *IH* = *this*

let *?b* = α (*T* ! *a*)
let *?k* = *B* ! *?b* − *Suc 0*

have *abs-bucket-insert* α *T B SA* (*a* # *xs*) = *abs-bucket-insert* α *T* (*B*[*?b* := *?k*]) (*SA*[*?k* := *a*]) *xs*
by (*meson abs-bucket-insert.simps*(*2*))
moreover
from *bucket-insert-cons*[*of ?b* α *T a ?k B SA xs*, *simplified*]
have *bucket-insert* α *T B SA* (*a* # *xs*) = *bucket-insert* α *T* (*B*[*?b* := *?k*]) (*SA*[*?k* := *a*]) *xs* **.**
ultimately show *?case*
using *IH*[*of B*[*?b* := *?k*] *SA*[*?k* := *a*]]
by *simp*
qed

end
theory *Induce-L-Verification*
imports
../abs−proof/Abs-Induce-L-Verification
../def/Induce-L
begin

113 Induce L Refinement

lemma *abs-induce-l-step-to-r0*:
i < *length SA* $\Longrightarrow$ *abs-induce-l-step* (*B*, *SA*, *i*) (α, *T*) = *induce-l-step-r0* (*B*, *SA*, *i*) (α, *T*)
by (*clarsimp simp*: *Let-def split*: *prod.splits nat.splits SL-types.splits*)

lemma *induce-l-step-r0-to*:
$\llbracket$*length ST* = *length T*; $\forall$ *k* < *length ST*. *ST* ! *k* = *suffix-type T k*$\rrbracket$ $\Longrightarrow$
induce-l-step-r0 (*B*, *SA*, *i*) (α, *T*) = *induce-l-step* (*B*, *SA*, *i*) (α, *T*, *ST*)
by (*clarsimp simp*: *Let-def split*: *prod.splits nat.splits SL-types.splits*)

lemma *abs-induce-l-step-to*:
assumes *i* < *length SA*

and *length ST* = *length T*
and ∀ *k* < *length ST*. *ST* ! *k* = *suffix-type T k*
shows *abs-induce-l-step* (*B*, *SA*, *i*) (α, *T*) = *induce-l-step* (*B*, *SA*, *i*) (α, *T*, *ST*)
by (*metis assms induce-l-step-r0-to abs-induce-l-step-to-r0*)

lemma *repeat-abs-induce-l-step-to*:
assumes *n* ≤ *length SA*
and *length ST* = *length T*
and ∀ *k* < *length ST*. *ST* ! *k* = *suffix-type T k*
shows *repeat n abs-induce-l-step* (*B*, *SA*, *0*) (α, *T*) = *repeat n induce-l-step* (*B*, *SA*, *0*) (α, *T*, *ST*)
using *assms*(*1*)
proof (*induct n*)
case *0*
then show *?case*
by (*simp add*: *repeat-0*)
next
case (*Suc n*)
note *IH* = *this*

from *repeat-step*[*of n abs-induce-l-step* (*B*, *SA*, *0*) (α, *T*)]
have *A*: *repeat* (*Suc n*) *abs-induce-l-step* (*B*, *SA*, *0*) (α, *T*) =
abs-induce-l-step (*repeat n abs-induce-l-step* (*B*, *SA*, *0*) (α, *T*)) (α, *T*)
by *assumption*

from *repeat-step*[*of n induce-l-step* (*B*, *SA*, *0*) (α, *T*, *ST*)]
have *B*: *repeat* (*Suc n*) *induce-l-step* (*B*, *SA*, *0*) (α, *T*, *ST*) =
induce-l-step (*repeat n induce-l-step* (*B*, *SA*, *0*) (α, *T*, *ST*)) (α, *T*, *ST*)
by *assumption*

from *repeat-abs-induce-l-step-index*[*of n B SA 0 α T*]
obtain *B′ SA′* **where**
C: *repeat n abs-induce-l-step* (*B*, *SA*, *0*) (α, *T*) = (*B′*, *SA′*, *n*)
by *auto*
with *IH*
have *D*: *repeat n induce-l-step* (*B*, *SA*, *0*) (α, *T*, *ST*) = (*B′*, *SA′*, *n*)
by *simp*

from *IH*(*2*)
have *n* < *length SA*
by *simp*
with *repeat-abs-induce-l-step-lengths*[*OF C*]
have *n* < *length SA′*
by *simp*

from *abs-induce-l-step-to*[*OF* ‹*n* < *length SA′*› *assms*(*2*−), *of B′*]
A B C D
show *?case*
by *simp*

qed

lemma *abs-induce-l-base-to*:
  assumes *length SA = length T*
  and *length ST = length T*
  and $\forall i <$ *length ST*. *ST* ! *i = suffix-type T i*
shows *abs-induce-l-base* α *T B SA = induce-l-base* α *T ST B SA*
  unfolding *induce-l-base-def abs-induce-l-base-def*
  by (*simp add*: *assms*(*1*, *2*,*3*) *repeat-abs-induce-l-step-to*)

lemma *abs-induce-l-eq*:
  assumes *length SA = length T*
  and *length ST = length T*
  and $\forall i <$ *length ST*. *ST* ! *i = suffix-type T i*
shows *abs-induce-l* α *T B SA = induce-l* α *T ST B SA*
  by (*metis assms abs-induce-l-base-to abs-induce-l-def induce-l-def*)

end
theory *Induce-S-Verification*
  imports
    ../abs−proof/Abs-Induce-S-Verification
    ../def/Induce-S
begin

114 Induce S Refinement

lemma *abs-induce-s-step-to-r0*:
  shows *induce-s-step-r0* (*B*, *SA*, *i*) (α, *T*) = *abs-induce-s-step* (*B*, *SA*, *i*) (α, *T*)
proof (*cases i*)
  case *0*
  then show *?thesis*
    by *simp*
next
  case (*Suc n*)
  assume *i = Suc n*
  then show *?thesis*
  proof (*cases Suc n < length SA*)
    assume *Suc n < length SA*
    show *?thesis*
    proof (*cases SA* ! *Suc n < length T*)
      assume *SA* ! *Suc n < length T*
      show *?thesis*
      proof (*cases SA* ! *Suc n*)
        case *0*
        then show *?thesis*
          by (*clarsimp simp*: ‹*i* = -› ‹*Suc n < length* -› ‹*SA* ! - < -›)
      next
        case (*Suc j*)
        assume *SA* ! *Suc n = Suc j*

hence *Suc j* < *length T*
using ‹*SA* ! *Suc n* < *length T*› **by** *auto*
then show *?thesis*
by (*clarsimp simp*: ‹*i* = -› ‹*Suc n* < *length* -› ‹*SA* ! - < -›)
qed
next
assume ¬ *SA* ! *Suc n* < *length T*
then show *?thesis*
by *simp*
qed
next
assume ¬ *Suc n* < *length SA*
show *?thesis*
by (*clarsimp simp*: ‹*i* = -› ‹¬ -›)
qed
qed

lemma *induce-s-step-r0-to-r1*:
assumes *length ST* = *length T*
and ∀ *k* < *length ST*. *ST* ! *k* = *suffix-type T k*
shows *induce-s-step-r1* (*B*, *SA*, *i*) (α, *T*, *ST*) = *induce-s-step-r0* (*B*, *SA*, *i*) (α, *T*)
proof (*cases i*)
case *0*
then show *?thesis*
by *auto*
next
case (*Suc n*)
assume *i* = *Suc n*
then show *?thesis*
proof (*cases Suc n* < *length SA* ∧ *SA* ! *Suc n* < *length T*)
assume *Suc n* < *length SA* ∧ *SA* ! *Suc n* < *length T*
hence *Suc n* < *length SA SA* ! *Suc n* < *length T*
by *blast*+
then show *?thesis*
proof (*cases SA* ! *Suc n*)
case *0*
then show *?thesis*
by (*clarsimp simp*: ‹*i* = -› ‹*Suc n* < *length* -› ‹*SA* ! - < -›)
next
case (*Suc j*)
assume *SA* ! *Suc n* = *Suc j*
hence *ST* ! *j* = *suffix-type T j*
using ‹*SA* ! *Suc n* < *length T*› *assms*(*1*,*2*) **by** *force*
then show *?thesis*
by (*clarsimp simp*: ‹*i* = -› ‹*Suc n* < *length* -› ‹*SA* ! - < -› ‹*SA* ! - = -›)
qed
next
assume ¬ (*Suc n* < *length SA* ∧ *SA* ! *Suc n* < *length T*)

show *?thesis*
  by (*simp add*: ‹¬ -›‹i = *Suc n*›)
qed
qed

lemma *abs-induce-s-step-to-r1*:
assumes *length ST* = *length T*
and $\forall k <$ *length ST*. *ST* ! k = *suffix-type T k*
shows *induce-s-step-r1* (B, *SA*, i) (α, T, *ST*) = *abs-induce-s-step* (B, *SA*, i) (α, T)
by (*metis assms induce-s-step-r0-to-r1 abs-induce-s-step-to-r0*)

lemma *induce-s-step-r1-to-r2*:
assumes *s-perm-inv* α *T B SA0 SA i*
shows *induce-s-step-r2* (B, *SA*, i) (α, T, *ST*) = *induce-s-step-r1* (B, *SA*, i) (α, T, *ST*)
proof (*cases i*)
case *0*
then show *?thesis*
by *simp*
next
case (*Suc n*)
then show *?thesis*
proof (*cases Suc n* < *length SA*)
assume *Suc n* < *length SA*
moreover
have *SA* ! *Suc n* < *length T*
by (*metis Suc assms calculation dual-order.refl s-perm-inv-elims*(*5*) *s-seen-invD*(*1*))
ultimately show *?thesis*
proof (*cases SA* ! *Suc n*)
case *0*
then show *?thesis*
using ‹i = *Suc n*› ‹*Suc n* < *length SA*› ‹*SA* ! *Suc n* < *length T*›
by *simp*
next
case (*Suc j*)
assume *SA* ! *Suc n* = *Suc j*
then show *?thesis*
proof (*cases ST* ! j)
assume *ST* ! j = *S-type*
then show *?thesis*
using ‹i = *Suc n*› ‹*Suc n* < *length SA*› ‹*SA* ! *Suc n* < *length T*› ‹*SA* ! *Suc n* = *Suc j*›
by (*clarsimp simp*: *Let-def*)
next
assume *ST* ! j = *L-type*
then show *?thesis*
using ‹i = *Suc n*› ‹*Suc n* < *length SA*› ‹*SA* ! *Suc n* < *length T*› ‹*SA* ! *Suc n* = *Suc j*›

by (*clarsimp simp*: *Let-def*)
qed
qed
next
assume $i = Suc\ n$ $\neg Suc\ n < length\ SA$
then show *?thesis*
by *simp*
qed
qed

lemma *abs-induce-s-step-to-r2*:
assumes *s-perm-inv* α *T B SA0 SA i*
and *length ST* = *length T*
and $\forall k <$ *length ST*. *ST* ! k = *suffix-type T k*
shows *induce-s-step-r2* (*B*, *SA*, *i*) (α, *T*, *ST*) = *abs-induce-s-step* (*B*, *SA*, *i*) (α, *T*)
by (*metis assms induce-s-step-r1-to-r2 induce-s-step-r0-to-r1 abs-induce-s-step-to-r0*)

lemma *induce-s-step-r2-to*:
$i <$ *length SA* $\Longrightarrow$ *induce-s-step* (*B*, *SA*, *i*) (α, *T*, *ST*) = *induce-s-step-r2* (*B*, *SA*, *i*) (α, *T*, *ST*)
by (*clarsimp simp*: *Let-def split*: *nat.splits*)

lemma *abs-induce-s-step-to*:
assumes *s-perm-inv* α *T B SA0 SA i*
and *length ST* = *length T*
and $\forall k <$ *length ST*. *ST* ! k = *suffix-type T k*
and $i <$ *length SA*
shows *induce-s-step* (*B*, *SA*, *i*) (α, *T*, *ST*) = *abs-induce-s-step* (*B*, *SA*, *i*) (α, *T*)
by (*metis abs-induce-s-step-to-r2 assms induce-s-step-r2-to*)

lemma *abs-induce-s-base-to'*:
assumes *s-perm-inv* α *T B SA0 SA n*
and *length ST* = *length T*
and $\forall k <$ *length ST*. *ST* ! k = *suffix-type T k*
and $n <$ *length SA*
shows *repeat m induce-s-step* (*B*, *SA*, *n*) (α, *T*, *ST*) = *repeat m abs-induce-s-step* (*B*, *SA*, *n*) (α, *T*)
using *assms*(*1*,*4*)
proof (*induct m arbitrary*: *B SA n*)
case *0*
then show *?case*
by (*simp add*: *repeat-0*)
next
case (*Suc m*)
note *IH* = *this* **and**
R0 = *repeat-step*[*of m abs-induce-s-step* (*B*, *SA*, *n*) (α, *T*)] **and**
R1 = *repeat-step*[*of m induce-s-step* (*B*, *SA*, *n*) (α, *T*, *ST*)]

from *repeat-abs-induce-s-step-index*[*of* m B SA n α T]
obtain B' SA' **where** S:
repeat m *abs-induce-s-step* (B, SA, n) $(\alpha, T) = (B', SA', n - m)$
length $SA' =$ *length* SA
length $B' =$ *length* B
by *blast*

have $n - m <$ *length* SA
using *Suc.prems*(*2*) **by** *auto*
hence $n - m <$ *length* SA'
using S(*2*) **by** *fastforce*

from *IH*(*1*)[*OF IH*(*2*,*3*)] *R1* S
have *repeat* (*Suc* m) *induce-s-step* (B, SA, n) (α, T, ST)
= *induce-s-step* $(B', SA', n - m)$ (α, T, ST)
by *simp*
moreover
from *IH*(*1*)[*OF IH*(*2*)] *R0* S
have *repeat* (*Suc* m) *abs-induce-s-step* (B, SA, n) (α, T)
= *abs-induce-s-step* $(B', SA', n - m)$ (α, T)
by *simp*
moreover
let *?P* = $\lambda(B, SA, i)$. *s-perm-inv* α T B *SA0* SA i
have *s-perm-inv* α T B' *SA0* SA' $(n - m)$
by (*rule repeat-maintain-inv*[*of ?P abs-induce-s-step* (α, T) (B, SA, n) m,
simplified S, *simplified*, *OF* - *IH*(*2*)];
clarsimp simp del: *abs-induce-s-step.simps*;
erule (*1*) *abs-induce-s-perm-step*)
with *abs-induce-s-step-to*[*OF* - *assms*(*2*,*3*) ‹$n - m <$ *length* SA'›, *of* α B' *SA0*]
have *induce-s-step* $(B', SA', n - m)$ (α, T, ST) = *abs-induce-s-step* $(B', SA', n$
$- m)$ (α, T)
by *blast*
ultimately show *?case*
by *simp*
qed

lemma *repeat-abs-induce-step-gre-length*:
assumes *length* SA = *length* T
shows
length $T \leq$ *Suc* $n \Longrightarrow$
repeat (*Suc* m) *abs-induce-s-step* $(B, SA,$ *Suc* $n)$ (α, T)
= *repeat* m *abs-induce-s-step* (B, SA, n) (α, T)
proof (*induct* m *arbitrary*: n)
case *0*
then show *?case*
by (*simp add*: *repeat-0 repeat-step Let-def assms*)
next
case (*Suc* m)
note *IH* = *this*

from *repeat-step*[*of Suc m abs-induce-s-step* (*B*, *SA*, *Suc n*) (α, *T*)]
IH(*1*)[*OF IH*(*2*)]
have *repeat* (*Suc* (*Suc m*)) *abs-induce-s-step* (*B*, *SA*, *Suc n*) (α, *T*)
= *abs-induce-s-step* (*repeat m abs-induce-s-step* (*B*, *SA*, *n*) (α, *T*)) (α, *T*)
by *presburger*
with *repeat-step*[*of m abs-induce-s-step* (*B*, *SA*, *n*) (α, *T*)]
show *?case*
by *presburger*
qed

lemma *abs-induce-s-base-to*:
assumes *s-perm-pre* α *T B SA* (*length T*)
and *length ST* = *length T*
and $\forall$ *k* < *length ST*. *ST* ! *k* = *suffix-type T k*
shows *induce-s-base* α *T ST B SA* = *abs-induce-s-base* α *T B SA*
proof −
note *A* = *assms*(*1*)[*simplified s-perm-pre-def*]

from *assms*(*1*)[*simplified s-perm-pre-def*]
have *s-perm-inv* α *T B SA SA* (*length T*)
by (*simp add*: *s-perm-inv-established*)
with *abs-induce-s-base-to'*[*OF - assms*(*2*−)]
have *repeat* (*length T* − *Suc 0*) *induce-s-step* (*B*, *SA*, *length T* − *Suc 0*) (α, *T*, *ST*)
= *repeat* (*length T* − *Suc 0*) *abs-induce-s-step* (*B*, *SA*, *length T* − *Suc 0*) (α, *T*)
by (*metis Suc-lessD Suc-pred A diff-Suc-less s-perm-inv-maintained-step-c1*)
moreover
have *repeat* (*length T*) *abs-induce-s-step* (*B*, *SA*, *length T*) (α, *T*)
= *repeat* (*length T* − *Suc 0*) *abs-induce-s-step* (*B*, *SA*, *length T* − *Suc 0*) (α, *T*)
by (*metis Suc-lessD Suc-pred A repeat-abs-induce-step-gre-length*)
ultimately show *?thesis*
by (*simp add*: *abs-induce-s-base-def induce-s-base-def*)
qed

lemma *abs-induce-s-eq*:
assumes *s-perm-pre* α *T B SA* (*length T*)
and *length ST* = *length T*
and $\forall$ *k* < *length ST*. *ST* ! *k* = *suffix-type T k*
shows *abs-induce-s* α *T B SA* = *induce-s* α *T ST B SA*
by (*simp add*: *assms abs-induce-s-base-to abs-induce-s-def induce-s-def*)

end
theory *Induce-Verification*
imports
../abs−proof/Abs-Induce-Verification
../def/Induce

Induce-S-Verification Induce-L-Verification Bucket-Insert-Verification
begin

115 Induce

lemma *sa-induce-to-r0*:
assumes *set LMS* = {*i. abs-is-lms T i*}
and *distinct LMS*
and *valid-list T*
and *strict-mono* α
and α *bot* = *0*
and *Suc 0* < *length T*
shows *abs-sa-induce* α *T LMS* = *sa-induce-r0* α *T LMS*
proof −

let *?ST* = *abs-get-suffix-types T*

note *A* = *length-abs-get-suffix-types*[*of T*]

from *get-suffix-types-correct*[*of T*] *A*
have *B*: ∀ *i* < *length ?ST*. *?ST* ! *i* = *suffix-type T i*
by *simp*

let *?B0* = *map* (*bucket-end* α *T*) [*0*..<*Suc* (α (*Max* (*set T*)))] **and**
?B1 = *map* (*bucket-start* α *T*) [*0*..<*Suc* (α (*Max* (*set T*)))] **and**
?SA0 = *replicate* (*length T*) (*length T*)

let *?B2* = *?B0*[*0* := *0*]
let *?SA1* = *abs-bucket-insert* α *T ?B0 ?SA0* (*rev LMS*)
let *?SA2* = *abs-induce-l* α *T ?B1 ?SA1*
let *?SA3* = *abs-induce-s* α *T* (*?B0*[*0* := *0*]) *?SA2*

let *?SA2'* = *induce-l* α *T ?ST ?B1 ?SA1*
let *?SA3'* = *induce-s* α *T ?ST* (*?B0*[*0* := *0*]) *?SA2*

from *lms-pre-established*[*OF assms*(*1*,*2*,*4*)]
have *lms-pre* α *T ?B0 ?SA0* (*rev LMS*) **.**

have *l-perm-pre* α *T ?B1 ?SA1*
using ‹*lms-pre* α *T ?B0 ?SA0* (*rev LMS*)›
assms(*3*,*4*) *l-perm-pre-established* **by** *blast*
with *A B*
have *?SA2* = *?SA2'*
using *abs-induce-l-eq l-perm-pre-elims*(*7*) **by** *blast*

have *s-perm-pre* α *T ?B2 ?SA2* (*length T*)
using ‹*l-perm-pre* α *T ?B1 ?SA1*› ‹*lms-pre* α *T ?B0 ?SA0* (*rev LMS*)›
assms(*3*−*6*)
s-perm-pre-established **by** *blast*

```
  with A B
  have ?SA3 = ?SA3'
    using abs-induce-s-eq by blast
  then show ?thesis
    by (metis ‹?SA2 = ?SA2'› abs-sa-induce-def sa-induce-r0-def)
qed

definition sa-induce-r1 ::
  (('a :: {linorder, order-bot}) ⇒ nat) ⇒
   'a list ⇒
   SL-types list ⇒
   nat list ⇒
   nat list
  where
sa-induce-r1 α T ST LMS =
  (let
      B0 = map (bucket-end α T) [0..<Suc (α (Max (set T)))];
      B1 = map (bucket-start α T) [0..<Suc (α (Max (set T)))];

      — Initialise SA
      SA = replicate (length T) (length T);

      — Insert the LMS types into the suffix array
      SA = abs-bucket-insert α T B0 SA (rev LMS);

      — Insert the L types into the suffix array
      SA = induce-l α T ST B1 SA

   — Insert the S types into the suffix array
   in induce-s α T ST (B0[0 := 0]) SA)

lemma sa-induce-r0-to-r1:
  assumes length ST = length T
  and      ∀ i < length ST. ST ! i = suffix-type T i
shows sa-induce-r0 α T LMS = sa-induce-r1 α T ST LMS
proof –
  let ?ST = abs-get-suffix-types T

  note A = length-abs-get-suffix-types[of T]

  from get-suffix-types-correct[of T] A
  have B: ∀ i < length ?ST. ?ST ! i = suffix-type T i
    by simp
  with A
  have ?ST = ST
    by (simp add: assms nth-equalityI)
  then show ?thesis
    by (simp add: sa-induce-r0-def sa-induce-r1-def)
qed
```

lemma *sa-induce-to-r1*:
assumes *set LMS* = {*i*. *abs-is-lms T i*}
and *distinct LMS*
and *valid-list T*
and *strict-mono* α
and α *bot* = *0*
and *Suc 0* < *length T*
and *length ST* = *length T*
and $\forall$ *i* < *length ST*. *ST* ! *i* = *suffix-type T i*
shows *abs-sa-induce* α *T LMS* = *sa-induce-r1* α *T ST LMS*
by (*simp add*: *assms sa-induce-r0-to-r1 sa-induce-to-r0*)

lemma *sa-induce-r1-to-r2*:
sa-induce-r1 α *T ST LMS* = *sa-induce-r2* α *T ST LMS*
by (*simp add*: *abs-bucket-insert-eq sa-induce-r1-def sa-induce-r2-def*)

lemma *abs-sa-induce-to-r2*:
assumes *set LMS* = {*i*. *abs-is-lms T i*}
and *distinct LMS*
and *valid-list T*
and *strict-mono* α
and α *bot* = *0*
and *Suc 0* < *length T*
and *length ST* = *length T*
and $\forall$ *i* < *length ST*. *ST* ! *i* = *suffix-type T i*
shows *abs-sa-induce* α *T LMS* = *sa-induce-r2* α *T ST LMS*
by (*metis assms sa-induce-r1-to-r2 sa-induce-to-r1*)

end
theory *Get-Types-Verification*
imports
../abs−def/Abs-SAIS
../def/Get-Types
begin

116 Suffix Types

lemma *get-suffix-types-step-r0-ret*:
$\exists$ *xs′ i′*. *get-suffix-types-step-r0* (*xs*, *i*) *ys* = (*xs′*, *i′*) $\wedge$
length xs′ = *length xs* $\wedge$ (*i* = *0* $\longrightarrow$ *i′* = *0*) $\wedge$ ($\exists$ *j*. *i* = *Suc j* $\longrightarrow$ *i′* = *j*)
by (*cases i*; *simp*)

lemma *get-suffix-types-step-r0-0*:
get-suffix-types-step-r0 (*xs*, *0*) *ys* = (*xs*, *0*)
by *simp*

lemma *get-suffix-types-step-r0-Suc*:
⟦*Suc i* < *length xs*; *length xs* = *length ys*; $\forall$ *k* < *length xs*. *i* < *k* $\longrightarrow$ *xs* ! *k* =

suffix-type ys k]] $\Longrightarrow$
  get-suffix-types-step-r0 (*xs*, *Suc i*) *ys* = (*xs*[*i* := *suffix-type ys i*], *i*)
 apply *clarsimp*
 apply (*intro conjI impI arg-cong*[**where** *f* = λx. *xs*[*i* := *x*]])
  apply (*simp add*: *nth-gr-imp-l-type*)
 apply (*simp add*: *nth-less-imp-s-type*)
 by (*metis suffix-type-neq*)

fun *get-suffix-types-inv*
 where
get-suffix-types-inv ys (*xs*, *i*) =
 (*length xs* = *length ys* $\wedge$ *i* < *length xs* $\wedge$ ($\forall$ *k* < *length xs*. *i* $\leq$ *k* $\longrightarrow$ *xs* ! *k* = *suffix-type ys k*))

lemma *get-suffix-types-inv-maintained*:
 assumes *get-suffix-types-inv ys* (*xs*, *i*)
shows *get-suffix-types-inv ys* (*get-suffix-types-step-r0* (*xs*, *i*) *ys*)
proof (*cases i*)
 case *0*
 hence *get-suffix-types-step-r0* (*xs*, *i*) *ys* = (*xs*, *0*)
  using *get-suffix-types-step-r0-0* **by** *simp*
 moreover
 have *get-suffix-types-inv ys* (*xs*, *0*)
  using *0 assms* **by** *auto*
 ultimately show *?thesis*
  by *presburger*
next
 case (*Suc n*)
 hence *get-suffix-types-step-r0* (*xs*, *i*) *ys* = (*xs*[*n* := *suffix-type ys n*], *n*)
  by (*metis Suc-leI assms get-suffix-types-inv.simps get-suffix-types-step-r0-Suc*)
 moreover
 have *get-suffix-types-inv ys* (*xs*[*n* := *suffix-type ys n*], *n*)
  using *Suc assms le-eq-less-or-eq* **by** *fastforce*
 ultimately show *?thesis*
  by *simp*
qed

lemma *get-suffix-types-inv-established*:
 xs $\neq$ [] $\Longrightarrow$ *get-suffix-types-inv xs* (*replicate* (*length xs*) *S-type*, *length xs* − *Suc 0*)
 by (*simp add*: *suffix-type-last*)

lemma *get-suffix-types-base-prod'*:
 $\exists$ *xs'*. *repeat n get-suffix-types-step-r0* (*xs*, *m*) *ys* = (*xs'*, *m* − *n*)
proof (*induct n arbitrary*: *xs m*)
 case *0*
 then show *?case*
  by (*simp add*: *repeat-0*)
next

case (*Suc n*)
note *IH* = *this*

from *repeat-step*[*of n get-suffix-types-step-r0* (*xs*, *m*) *ys*]
have *repeat* (*Suc n*) *get-suffix-types-step-r0* (*xs*, *m*) *ys*
= *get-suffix-types-step-r0* (*repeat n get-suffix-types-step-r0* (*xs*, *m*) *ys*) *ys* **.**
moreover
from *IH*[*of xs m*]
obtain *xs′* **where**
repeat n get-suffix-types-step-r0 (*xs*, *m*) *ys* = (*xs′*, *m* − *n*)
by *blast*
moreover
have ∃ *xs″*. *get-suffix-types-step-r0* (*xs′*, *m* − *n*) *ys* = (*xs″*, *m* − *Suc n*)
proof (*cases m*−*n*)
case *0*
hence *get-suffix-types-step-r0* (*xs′*, *m* − *n*) *ys* = (*xs′*, *0*)
by *auto*
moreover
have *m* − *Suc n* = *0*
using *0* **by** *auto*
ultimately show *?thesis*
by *simp*
next
case (*Suc k*)
have (*m* − *n* < *length xs′* ∧ *m* − *n* < *length ys*) ∨ ¬(*m* − *n* < *length xs′* ∧ *m* − *n* < *length ys*)
by *blast*
moreover
have *m* − *n* < *length xs′* ∧ *m* − *n* < *length ys* ⟹ *?thesis*
by (*clarsimp simp add*: *Suc diff-Suc*)
moreover
have ¬(*m* − *n* < *length xs′* ∧ *m* − *n* < *length ys*) ⟹ *?thesis*
by (*clarsimp simp add*: *Suc diff-Suc*)
ultimately show *?thesis*
by *blast*
qed
ultimately show *?case*
by *presburger*
qed

lemma *get-suffix-types-inv-holds*:
assumes *xs* ≠ []
shows *get-suffix-types-inv xs* (*get-suffix-types-base xs*)
unfolding *get-suffix-types-base-def*
apply (*rule repeat-maintain-inv*)
apply (*metis get-suffix-types-inv-maintained prod.collapse*)
apply (*rule get-suffix-types-inv-established*[*OF assms*])
done

lemma *get-suffix-types-base-prod*:
$\exists$ *xs'*. *get-suffix-types-base xs* = (*xs'*, *0*)
unfolding *get-suffix-types-base-def*
by (*metis cancel-comm-monoid-add-class.diff-cancel get-suffix-types-base-prod'*)

lemma *get-suffix-types-base-ref*:
get-suffix-types-base xs = (*abs-get-suffix-types xs*, *0*)
proof (*cases xs* $\neq$ [])
assume $\neg$ *xs* $\neq$ []
then show *?thesis*
by (*clarsimp simp*: *get-suffix-types-base-def repeat-0 get-suffix-types-def*)
next
assume *xs* $\neq$ []
with *get-suffix-types-inv-holds*
have *get-suffix-types-inv xs* (*get-suffix-types-base xs*)
by *blast*
moreover
from *get-suffix-types-base-prod*[*of xs*]
obtain *xs'* **where**
get-suffix-types-base xs = (*xs'*, *0*)
by *blast*
ultimately have *get-suffix-types-inv xs* (*xs'*, *0*)
by *auto*
moreover
have *abs-get-suffix-types xs* = *xs'*
unfolding *list-eq-iff-nth-eq*
by (*metis bot-nat-0.extremum calculation get-suffix-types-correct*
get-suffix-types-inv.simps
length-abs-get-suffix-types)
ultimately show *?thesis*
by (*simp add*: ‹*get-suffix-types-base xs* = (*xs'*, *0*)›)
qed

lemma *get-suffix-types-eq*:
get-suffix-types xs = *abs-get-suffix-types xs*
by (*simp add*: *get-suffix-types-base-ref get-suffix-types-def*)

lemmas *length-get-suffix-types* =
length-abs-get-suffix-types[*simplified get-suffix-types-eq*]

117 LMS types

lemma *is-lms-refinement*:
assumes *length ST* = *length T* $\forall$ *i* < *length T*. *ST* ! *i* = *suffix-type T i*
shows *is-lms-ref ST* = *abs-is-lms T*
proof
fix *i*
show *is-lms-ref ST i* = *abs-is-lms T i*
proof (*cases i*)

case *0*
then show *?thesis*
by (*simp add*: *abs-is-lms-0*)
next
case (*Suc n*)
then show *?thesis*
by (*metis Suc-lessD assms abs-is-lms-def abs-is-lms-imp-less-length is-lms-ref.simps*(*2*))
qed
qed

118 Extracting LMS types

lemma *extract-lms-eq*:
⟦*length ST* = *length T*; ∀ *i* < *length T*. *ST* ! *i* = *suffix-type T i*⟧ ⟹
extract-lms ST = *abs-extract-lms T*
by (*clarsimp simp*: *fun-eq-iff is-lms-refinement*)

119 LMS Substrings

lemma *find-next-lms-refinement*:
⟦*length ST* = *length T*; ∀ *i* < *length T*. *ST* ! *i* = *suffix-type T i*⟧ ⟹
find-next-lms ST= *abs-find-next-lms T*
unfolding *find-next-lms-def abs-find-next-lms-def*
apply (*clarsimp simp*: *is-lms-refinement fun-eq-iff*)
by *argo*

lemma *lms-slice-refinement*:
⟦*length ST* = *length T*; ∀ *i* < *length T*. *ST* ! *i* = *suffix-type T i*⟧ ⟹
lms-slice-ref T ST = *lms-slice T*
unfolding *lms-slice-ref-def lms-slice-def*
by (*clarsimp simp*: *find-next-lms-refinement fun-eq-iff*)

120 Rename Mapping

lemma *rename-mapping′-refinement*:
assumes *length ST* = *length T* ∀ *i* < *length T*. *ST* ! *i* = *suffix-type T i*
shows *rename-mapping′ T ST* = *abs-rename-mapping′ T*
proof (*intro fun-eq-iff*[*THEN iffD2*] *allI*)
fix *xs ns i*
show *rename-mapping′ T ST xs ns i* = *abs-rename-mapping′ T xs ns i*
using *assms*
proof (*induct rule*: *rename-mapping′.induct*[*of - T ST xs ns i*])
case (*1 T ST ns i*)
then show *?case*
by *simp*
next
case (*2 T ST x ns i*)
then show *?case*

```
      by simp
  next
    case (3 T ST a b xs ns i)
    then show ?case
      by (simp add: lms-slice-refinement)
  qed
qed

lemma rename-mapping-refinement:
  assumes length ST = length T
  assumes ∀ i < length T. ST ! i = suffix-type T i
  shows rename-mapping T ST =  abs-rename-mapping T
 by (clarsimp simp: fun-eq-iff assms rename-mapping'-refinement abs-rename-mapping-def
                      rename-mapping-def)

end
theory SAIS-Verification
  imports
      Get-Types-Verification
      Induce-Verification
      ../abs−proof/Abs-SAIS-Verification-With-Valid-Precondition
      ../def/SAIS

begin
```

121 SAIS

```
termination sais-r0
  apply (relation measure (λxs. length xs))
   apply simp
  apply (simp (no-asm-simp)
             del: List.list.size(4)
             only: extract-lms-eq[OF length-get-suffix-types get-suffix-types-correct]
                   rename-mapping-refinement[OF length-get-suffix-types
                   get-suffix-types-correct] get-suffix-types-eq)
  apply (simp (no-asm-simp)
           del: List.list.size(4)
           add: rename-list-length length-filter-lms)
  done

lemma abs-sais-r0-distinct-simp:
  assumes T = a # b # xs
  and      ST = abs-get-suffix-types T
  and      LMS0 = extract-lms ST [0..<length T]
  and      SA = sa-induce id T ST LMS0
  and      LMS = extract-lms ST SA
  and      names = rename-mapping T ST LMS
  and      T' = rename-string LMS0 names
  and      distinct T'
```

shows *sais-r0 T* = *sa-induce id T ST LMS*
proof −
let *?P* = *λys. sais-r0* (*a* # *b* # *xs*) = *ys*
from *subst*[*OF sais-r0.simps*(*3*), *of ?P a b xs*, *simplified Let-def*
assms(*1*−*7*)[*symmetric*] *assms*(*8*), *simplified*]
show *?thesis*
by *simp*
qed

lemma *abs-sais-r0-not-distinct-simp*:
assumes *T* = *a* # *b* # *xs*
and *ST* = *abs-get-suffix-types T*
and *LMS0* = *extract-lms ST* [*0*..<*length T*]
and *SA* = *sa-induce id T ST LMS0*
and *LMS* = *extract-lms ST SA*
and *names* = *rename-mapping T ST LMS*
and *T′* = *rename-string LMS0 names*
and *LMS1* = *order-lms LMS0* (*sais-r0 T′*)
and ¬*distinct T′*
shows *sais-r0 T* = *sa-induce id T ST LMS1*
proof −
let *?P* = *λys. sais-r0* (*a* # *b* # *xs*) = *ys*
from *subst*[*OF sais-r0.simps*(*3*), *of ?P a b xs*, *simplified Let-def*
assms(*1*−*8*)[*symmetric*] *assms*(*9*), *simplified*]
show *?thesis*
by *simp*
qed

lemma *abs-sais-to-r0*:
valid-list T ⟹ *abs-sais T* = *sais-r0 T*
proof(*induct rule*: *abs-sais.induct*[*of* - *T*])
case *1*
then show *?case*
by *simp*
next
case (*2 x*)
then show *?case*
by *simp*
next
case (*3 a b xs*)
note *IH* = *this*

let *?T* = *a* # *b* # *xs*
have *T*: *?T* = *a* # *b* # *xs*
by (*simp only*:)

let *?ST* = *abs-get-suffix-types ?T*
have *ST*: *?ST* = *abs-get-suffix-types ?T*
by (*simp only*:)

from *get-suffix-types-correct*[*of ?T*] *length-abs-get-suffix-types*[*of ?T*]
have $\forall$ *i* < *length ?ST*. *?ST* ! *i* = *suffix-type ?T i*
by (*simp add*: *get-suffix-types-eq*)
note *st-thms* = *length-get-suffix-types*[*of ?T*]
‹$\forall$ *i* < *length ?ST*. *?ST* ! *i* = *suffix-type ?T i*›

let *?LMS1* = *abs-extract-lms ?T* [*0..<length ?T*]
let *?LMS1'* = *extract-lms ?ST* [*0..<length ?T*]
have *LMS1*: *?LMS1* = *abs-extract-lms ?T* [*0..<length ?T*]
by (*simp only*:)
have *LMS1'*: *?LMS1'* = *extract-lms ?ST* [*0..<length ?T*]
by (*simp only*:)
have *distinct ?LMS1*
using *distinct-abs-extract-lms*
by *fastforce*
have *set ?LMS1* = {*i. abs-is-lms ?T i*}
using *set-abs-extract-lms-eq-all-lms*
by (*metis comp-apply*)
have *?LMS1* = *?LMS1'*
by (*metis extract-lms-eq get-suffix-types-correct length-get-suffix-types*)
note *lms1-thms* = ‹*set ?LMS1* = {*i. abs-is-lms ?T i*}› ‹*distinct ?LMS1*› ‹*?LMS1* = *?LMS1'*›

have *id*: *strict-mono* (*id* :: *nat* ⇒ *nat*) (*id* :: *nat* ⇒ *nat*) *bot* = *0*
by (*simp add*: *strict-mono-def bot-nat-def*)+

have *len*: *Suc 0* < *length ?T*
by *simp*

let *?SA1* = *abs-sa-induce id ?T ?LMS1*
have *SA1*: *?SA1* = *abs-sa-induce id ?T ?LMS1*
by (*simp only*:)
let *?SA1'* = *sa-induce id ?T ?ST ?LMS1'*
have *SA1'*: *?SA1'* = *sa-induce id ?T ?ST ?LMS1'*
by (*simp only*:)
have *?SA1* = *?SA1'*
by (*metis 3.prems len lms1-thms id st-thms abs-sa-induce-to-r2*)

let *?LMS2* = *abs-extract-lms ?T ?SA1*
let *?LMS2'* = *extract-lms ?ST ?SA1'*
have *LMS2*: *?LMS2* = *abs-extract-lms ?T ?SA1*
by (*simp only*:)
have *LMS2'*: *?LMS2'* = *extract-lms ?ST ?SA1'*
by (*simp only*:)
have *?LMS2* = *?LMS2'*
using ‹*?SA1* = *?SA1'*› *st-thms comp-apply is-lms-refinement*
by (*metis* (*no-types*, *lifting*) *get-suffix-types-eq*)
have *?LMS1* <~~> *?LMS2*
by (*metis 3.prems distinct-filter-abs-sa-induce lms1-thms len id*

distinct-set-imp-perm filter-abs-sa-induce-eq-all-lms)
hence *distinct ?LMS2 set ?LMS2* = {*i. abs-is-lms ?T i*}
using *lms1-thms perm-distinct-iff perm-set-eq* **by** *blast*+
have *ordlistns.sorted* (*map* (*lms-slice ?T*) *?LMS2*)
by (*metis 3.prems* ‹*distinct ?LMS1*› ‹*set ?LMS1* = {*i. abs-is-lms ?T i*}›
comp-apply len id
ordlistns.sorted-filter abs-sa-induce-prefix-sorted)
note *lms2-thms* = ‹*distinct ?LMS2*› ‹*set ?LMS2* = {*i. abs-is-lms ?T i*}›
‹*ordlistns.sorted* (*map* (*lms-slice ?T*) *?LMS2*)›
‹*?LMS2* = *?LMS2′*›

let *?names* = *abs-rename-mapping ?T ?LMS2*
let *?names′* = *rename-mapping ?T ?ST ?LMS2′*
have *names*: *?names* = *abs-rename-mapping ?T ?LMS2*
by (*simp only*:)
have *names′*: *?names′* = *rename-mapping ?T ?ST ?LMS2′*
by (*simp only*:)
have *?names* = *?names′*
by (*metis st-thms lms2-thms*(*4*) *rename-mapping-refinement*)

let *?T′* = *rename-string ?LMS1 ?names*
let *?T″* = *rename-string ?LMS1′ ?names′*
have *T′*: *?T′* = *rename-string ?LMS1 ?names*
by (*simp only*:)
have *T″*: *?T″* = *rename-string ?LMS1′ ?names′*
by (*simp only*:)
have *?T′* = *?T″*
using ‹*?names* = *?names′*› *lms1-thms*(*3*) **by** *argo*

let *?LMS3* = *order-lms ?LMS1* (*abs-sais ?T′*)
let *?LMS3′* = *order-lms ?LMS1′* (*sais-r0 ?T″*)
have *LMS3*: *?LMS3* = *order-lms ?LMS1* (*abs-sais ?T′*)
by (*simp only*:)
have *LMS3′*: *?LMS3′* = *order-lms ?LMS1′* (*sais-r0 ?T″*)
by (*simp only*:)

have *?LMS1* = *lms0-seq ?T*
by (*metis comp-apply lms-seq-0-zeroth-lms lms-seq-def*)
with *abs-sais-reduced-string*[*OF - lms2-thms*(*1*−*3*) *names T′*]
have *?T′* = *lms0-map ?T*
by *blast*

have *abs-is-lms ?T* (*lms0 ?T*)
by (*metis 3.prems abs-find-next-lms-less-length-abs-is-lms abs-is-lms-last*
len length-Cons no-lms-between-i-and-next not-less-eq)

from *valid-list-lms-map*[*OF IH*(*2*) ‹*abs-is-lms ?T* (*lms0 ?T*)›]
have *valid-list* (*lms0-map ?T*) **.**
hence *valid-list ?T′*

by (*simp only*: ‹*?T′* = (*lms0-map ?T*)›)

have *distinct ?T′* ⟹ *?case*
proof −
assume *distinct ?T′*
hence *distinct ?T″*
by (*simp only*: ‹*?T′* = *?T″*›)
note *lms2-thms* = *filter-abs-sa-induce-eq-all-lms*[*OF lms1-thms*(*1*,*2*) *IH*(*2*) *id len*]
distinct-filter-abs-sa-induce[*OF lms1-thms*(*1*,*2*) *IH*(*2*) *id len*]
from *abs-sa-induce-to-r2 lms2-thms*(*1*,*2*) *IH*(*2*) *id len st-thms*
have *abs-sa-induce id ?T ?LMS2* = *sa-induce id ?T ?ST ?LMS2′*
by (*metis* ‹*?LMS2* = *?LMS2′*› *comp-apply*)
with *abs-sais-distinct-simp*[*OF T LMS1 SA1 LMS2 names T′* ‹*distinct ?T′*›]
abs-sais-r0-distinct-simp[*OF T ST LMS1′ SA1′ LMS2′ names′ T″* ‹*distinct ?T″*›]
show *?thesis*
by *presburger*
qed
moreover
have ¬*distinct ?T′* ⟹ *?case*
proof −
assume ¬*distinct ?T′*
hence ¬*distinct ?T″*
using ‹*?T′* = *?T″*› **by** *argo*

from *IH*(*1*)[*OF T LMS1 SA1 LMS2 names T′*, *OF* ‹¬*distinct ?T′*› ‹*valid-list ?T′*›]
have *abs-sais ?T′* = *sais-r0 ?T″*
using ‹*?T′* = *?T″*› **by** *argo*
hence *?LMS3′* = *?LMS3*
using *lms1-thms*(*3*) **by** *argo*

have *abs-sais-perm*: *abs-sais ?T′* <~~> [*0..<length ?LMS1*]
using *abs-sais-permutation*[*OF* ‹*valid-list ?T′*›, *simplified rename-list-length*]
by *blast*

note *lms3-thms* = *abs-order-lms-eq-all-lms*[*OF abs-sais-perm lms1-thms*(*1*)]
distinct-abs-order-lms[*OF abs-sais-perm lms1-thms*(*2*)]
from *abs-sa-induce-to-r2*[*OF lms3-thms* ‹*valid-list ?T*› *id len st-thms*]
have *abs-sa-induce id ?T ?LMS3* = *sa-induce id ?T ?ST ?LMS3′*
using ‹*abs-sais ?T′* = *sais-r0 ?T″*› *lms1-thms*(*3*) **by** *argo*
with *abs-sais-not-distinct-simp*[*OF T LMS1 SA1 LMS2 names T′ LMS3* ‹¬*distinct ?T′*›]
abs-sais-r0-not-distinct-simp[*OF T ST LMS1′ SA1′ LMS2′ names′ T″ LMS3′* ‹¬*distinct ?T″*›]
‹*abs-sais ?T′* = *sais-r0 ?T′*›
show *?thesis*
by *presburger*

qed
ultimately show *?case*
by *blast*
qed

termination *sais-r1*
apply (*relation measure* (λ*xs. length xs*))
apply *simp*
apply (*simp* (*no-asm-simp*)
del: *List.list.size*(*4*)
only: *extract-lms-eq*[*OF length-abs-get-suffix-types get-suffix-types-correct*]
rename-mapping-refinement[*OF length-abs-get-suffix-types get-suffix-types-correct*])
apply (*simp* (*no-asm-simp*)
del: *List.list.size*(*4*)
add: *rename-list-length length-filter-lms*)
apply (*metis get-suffix-types-correct get-suffix-types-eq is-lms-refinement length-filter-lms length-get-suffix-types list.discI*)
done

lemma *abs-sais-r0-to-r1*:
sais-r1 T = *sais-r0 T*
apply (*induct rule*: *sais-r0.induct*[*of - T*])
apply *simp*
apply *simp*
apply (*subst sais-r1.simps*)
apply (*subst get-suffix-types-eq*)
apply (*subst sais-r0.simps*)
apply (*clarsimp simp only*: *Let-def split*: *if-splits*)
by *presburger*

lemma *abs-sais-to-r1*:
valid-list T $\Longrightarrow$ *sais-r1 T* = *abs-sais T*
by (*simp add*: *abs-sais-r0-to-r1 abs-sais-to-r0*)

122 Correctness

interpretation *sais*: *Suffix-Array-Restricted sais*
by (*simp add*: *Suffix-Array-Restricted.intro Suffix-Array-Restricted.sa-r-permutation Suffix-Array-Restricted.sa-r-sorted abs-sais.Suffix-Array-Restricted-axioms abs-sais-to-r1*)

interpretation *abs-sais-ref-gen*: *Suffix-Array-General sa-nat-wrapper map-to-nat sais*
by (*simp add*: *Suffix-Array-Restricted-imp-General sais.Suffix-Array-Restricted-axioms*)

theorem *sais-gen-is-Suffix-Array-General*:
Suffix-Array-General sa $\longleftrightarrow$ *sa* = *sa-nat-wrapper map-to-nat sais*
using *Suffix-Array-General-determinism abs-sais-ref-gen.Suffix-Array-General-axioms*

```
by auto

end
theory Code-Extraction
  imports ../abs−proof/Abs-SAIS-Verification
          ../proof/SAIS-Verification
begin

lemma [code]:
abs-is-lms T i =
  (if i > 0 then
    if suffix-type T i = S-type ∧ suffix-type T (i − 1) = L-type
    then True
    else False
   else False)
  by (metis abs-is-lms-0 One-nat-def Suc-pred bot-nat-0.not-eq-extremum
          i-s-type-imp-Suc-i-not-lms abs-is-lms-def suffix-type-def)

definition
  bucket-upt-code :: ('a :: {linorder,order-bot} ⇒ nat) ⇒ 'a list ⇒ nat ⇒ nat set
where
  bucket-upt-code α T b ≡
    set (filter (λx. α (T ! x) < b) [0..<length T])

lemma [code]:
  bucket-upt α T b = bucket-upt-code α T b
proof(safe)
  fix x
  assume x ∈ bucket-upt α T b
  hence x < length T α (T ! x) < b
    by (simp add: bucket-upt-def)+
  then show x ∈ bucket-upt-code α T b
    by (simp add: bucket-upt-code-def)
next
  fix x
  assume x ∈ bucket-upt-code α T b
  hence x < length T α (T ! x) < b
    by (simp add: bucket-upt-code-def)+
  then show x ∈ bucket-upt α T b
    by (simp add: bucket-upt-def)
qed


export-code abs-sais in Haskell
  module-name SAIS file-prefix abs-sais

export-code sais in Haskell
  module-name SAIS-REF file-prefix sais
```

end
theory *SACA-Equiv*
  imports *sais/abs−proof/Abs-SAIS-Verification*
    simple/Simple-SACA-Verification
    sais/proof/SAIS-Verification
begin

lemma *Suffix-Array-General-imp-suffix-array*:
  Suffix-Array-General sa $\Longrightarrow$
   sa s = *simple-saca s*
 using *Suffix-Array-General-determinism simple-saca.Suffix-Array-General-axioms*
by *blast*

theorem *Suffix-Array-General-equiv-spec*:
  Suffix-Array-General sa $\longleftrightarrow$
   sa = *simple-saca*
 using *Suffix-Array-General-imp-suffix-array simple-saca.Suffix-Array-General-axioms*
by *blast*

corollary *abs-sais-equiv-simple-saca*:
  sa-nat-wrapper map-to-nat abs-sais = *simple-saca*
  using *Suffix-Array-General-equiv-spec abs-sais-gen.Suffix-Array-General-axioms*
by *auto*

corollary *sais-equiv-simple-saca*:
  sa-nat-wrapper map-to-nat sais = *simple-saca*
  using *sais-gen-is-Suffix-Array-General*
    Suffix-Array-General-equiv-spec
  by *auto*

end

References

[1] P. Ko and S. Aluru. Space efficient linear time construction of suffix arrays. *Journal of Discrete Algorithms*, 3(2-4):143–156, 2005.

[2] U. Manber and E. W. Myers. Suffix arrays: A new method for on-line string searches. *SIAM Journal on Computing*, 22(5):935–948, 1993.

[3] G. Nong, S. Zhang, and W. H. Chan. Linear suffix array construction by almost pure induced-sorting. In *Proc. Data Compression Conference*, pages 193–202. IEEE Computing Society, 2009.