

Formalized Burrows-Wheeler Transform

Louis Cheung and Christine Rizkallah

January 17, 2025

Abstract

The Burrows-Wheeler transform (BWT) [2] is an invertible lossless transformation that permutes input sequences into alternate sequences of the same length that frequently contain long localized regions that involve clusters consisting of just a few distinct symbols, and sometimes also include long runs of same-symbol repetitions. Moreover, there is a one-to-one correspondence between the BWT and suffix arrays [7]. As a consequence, the BWT is widely used in data compression and as an indexing data structure for pattern search. In this formalization [4], we present the formal verification of both the BWT and its inverse, building on a formalization of suffix arrays [5]. This is the artefact of our CPP paper [3].

Contents

1	Nat Modulo Helper	3
2	Rotated Sublists	3
3	Counting	11
3.1	Count List	11
3.2	Cardinality	12
3.3	Sorting	14
4	Rank Definition	19
5	Rank Properties	19
5.1	List Properties	19
5.2	Counting Properties	19
5.3	Bound Properties	21
5.4	Sorted Properties	23
6	Select Definition	26

7	Select Properties	27
7.1	Length Properties	27
7.2	List Properties	27
7.3	Bound Properties	28
7.4	Nth Properties	28
7.5	Sorted Properties	32
8	Rank and Select Properties	37
8.1	Correctness of Rank and Select	37
8.1.1	Rank Correctness	37
8.1.2	Select Correctness	37
8.2	Rank and Select	38
8.3	Sorted Properties	39
9	Suffix Array Properties	40
9.1	Bijections	40
9.2	Suffix Properties	41
9.3	General Properties	43
9.4	Nth Properties	43
9.5	Valid List Properties	45
10	Counting Properties on Suffix Arrays	46
10.1	Counting Properties	46
10.2	Ordering Properties	51
11	Burrows-Wheeler Transform	53
12	BWT Verification	54
12.1	List Rotations	54
12.2	Ordering	55
12.3	BWT Equivalence	56
13	BWT and Suffix Array Correspondence	56
13.1	BWT Using Suffix Arrays	57
13.2	BWT Rank Properties	65
13.3	Suffix Array and BWT Rank	68
14	Inverse Burrows-Wheeler Transform	71
14.1	Abstract Versions	71
14.2	Concrete Versions	71
15	List Filter	72

16 Verification of the Inverse Burrows-Wheeler Transform	73
16.1 LF-Mapping Simple Properties	73
16.2 LF-Mapping Correctness	75
16.3 Backwards Inverse BWT Simple Properties	76
16.4 Backwards Inverse BWT Correctness	78
16.5 Concretization	84
16.6 Inverse BWT Correctness	86

```

theory Nat-Mod-Helper
  imports Main
begin

```

1 Nat Modulo Helper

```

lemma nat-mod-add-neq-self:
   $\llbracket a < (n :: nat); b < n; b \neq 0 \rrbracket \implies (a + b) \bmod n \neq a$ 
  by (metis add-diff-cancel-left' mod-if mod-mult-div-eq mod-mult-self1-is-0)

```

```

lemma nat-mod-a-pl-b-eq1:
   $\llbracket n + b \leq a; a < (n :: nat) \rrbracket \implies (a + b) \bmod n = b - (n - a)$ 
  using order-le-less-trans by blast

```

```

lemma not-mod-a-pl-b-eq2:
   $\llbracket n - a \leq b; a < n; b < (n :: nat) \rrbracket \implies (a + b) \bmod n = b - (n - a)$ 
  using Nat.diff-diff-right add.commute mod-if by auto

```

```

end
theory Rotated-Substring
  imports Nat-Mod-Helper
begin

```

2 Rotated Sublists

```

definition is-sublist :: 'a list  $\Rightarrow$  'a list  $\Rightarrow$  bool
  where
  is-sublist xs ys =  $(\exists as\ bs. xs = as @ ys @ bs)$ 

```

```

definition is-rot-sublist :: 'a list  $\Rightarrow$  'a list  $\Rightarrow$  bool
  where
  is-rot-sublist xs ys =  $(\exists n. is-sublist (rotate\ n\ xs)\ ys)$ 

```

```

definition inc-one-bounded :: nat  $\Rightarrow$  nat list  $\Rightarrow$  bool
  where
  inc-one-bounded n xs  $\equiv$ 
   $(\forall i. Suc\ i < length\ xs \longrightarrow xs\ !\ Suc\ i = Suc\ (xs\ !\ i) \bmod\ n) \wedge$ 
   $(\forall i < length\ xs. xs\ !\ i < n)$ 

```

```

lemma inc-one-boundedD:

```

$\llbracket \text{inc-one-bounded } n \text{ } xs; \text{Suc } i < \text{length } xs \rrbracket \implies xs ! \text{Suc } i = \text{Suc } (xs ! i) \text{ mod } n$
 $\llbracket \text{inc-one-bounded } n \text{ } xs; i < \text{length } xs \rrbracket \implies xs ! i < n$
using *inc-one-bounded-def* **by** *blast+*

lemma *inc-one-bounded-nth-plus*:

$\llbracket \text{inc-one-bounded } n \text{ } xs; i + k < \text{length } xs \rrbracket \implies xs ! (i + k) = (xs ! i + k) \text{ mod } n$

proof (*induct k*)

case *0*

then show *?case*

by (*simp add: inc-one-boundedD(2)*)

next

case (*Suc k*)

then show *?case*

by (*metis Suc-lessD add-Suc-right inc-one-bounded-def mod-Suc-eq*)

qed

lemma *inc-one-bounded-neq*:

$\llbracket \text{inc-one-bounded } n \text{ } xs; \text{length } xs \leq n; i + k < \text{length } xs; k \neq 0 \rrbracket \implies xs ! (i + k) \neq xs ! i$

using *inc-one-bounded-nth-plus nat-mod-add-neq-self*

by (*simp add: inc-one-boundedD(2) linorder-not-le*)

corollary *inc-one-bounded-neq-nth*:

assumes *inc-one-bounded n xs*

and $\text{length } xs \leq n$

and $i < \text{length } xs$

and $j < \text{length } xs$

and $i \neq j$

shows $xs ! i \neq xs ! j$

proof (*cases i < j*)

assume $i < j$

then show *?thesis*

by (*metis assms(1,2,4) canonically-ordered-monoid-add-class.lessE inc-one-bounded-neq*)

next

assume $\neg i < j$

then show *?thesis*

by (*metis assms(1,2,3,5) canonically-ordered-monoid-add-class.lessE inc-one-bounded-neq le-neq-implies-less linorder-not-le*)

qed

lemma *inc-one-bounded-distinct*:

$\llbracket \text{inc-one-bounded } n \text{ } xs; \text{length } xs \leq n \rrbracket \implies \text{distinct } xs$

using *distinct-conv-nth inc-one-bounded-neq-nth* **by** *blast*

lemma *inc-one-bounded-subset-upt*:

$\llbracket \text{inc-one-bounded } n \text{ } xs; \text{length } xs \leq n \rrbracket \implies \text{set } xs \subseteq \{0..<n\}$

by (*metis atLeastLessThan-iff in-set-conv-nth inc-one-boundedD(2) less-eq-nat.simps(1) subset-code(1)*)

lemma *inc-one-bounded-consD*:

inc-one-bounded n ($x \# xs$) \implies *inc-one-bounded* n xs

unfolding *inc-one-bounded-def*

using *bot-nat-0.not-eq-extremum lessI less-zeroE mod-less-divisor* **by** *fastforce*

lemma *inc-one-bounded-nth*:

$\llbracket \text{inc-one-bounded } n \text{ } xs; i < \text{length } xs \rrbracket \implies xs ! i = ((\lambda x. \text{Suc } x \text{ mod } n) \overset{\sim}{\sim} i) (xs ! 0)$

proof (*induct* i)

case 0

then show *?case*

by *simp*

next

case (*Suc* i)

note $IH = \text{this}$

from IH

have $xs ! i = ((\lambda x. \text{Suc } x \text{ mod } n) \overset{\sim}{\sim} i) (xs ! 0)$

by *simp*

hence $\text{Suc } (xs ! i) \text{ mod } n = ((\lambda x. \text{Suc } x \text{ mod } n) \overset{\sim}{\sim} \text{Suc } i) (xs ! 0)$

by *force*

moreover

from *inc-one-boundedD(1)[OF IH(2,3)]*

have $xs ! \text{Suc } i = \text{Suc } (xs ! i) \text{ mod } n$.

ultimately show *?case*

by *presburger*

qed

lemma *inc-one-bounded-nth-le*:

$\llbracket \text{inc-one-bounded } n \text{ } xs; i < \text{length } xs; (xs ! 0) + i < n \rrbracket \implies$

$xs ! i = (xs ! 0) + i$

by (*metis add-cancel-right-left inc-one-bounded-nth-plus mod-if*)

lemma *inc-one-bounded-upt1*:

assumes *inc-one-bounded* n xs

and $\text{length } xs = \text{Suc } k$

and $\text{Suc } k \leq n$

and $(xs ! 0) + k < n$

shows $xs = [xs ! 0 .. < (xs ! 0) + \text{Suc } k]$

proof (*intro list-eq-iff-nth-eq[THEN iffD2] conjI impI allI*)

show $\text{length } xs = \text{length } [xs ! 0 .. < xs ! 0 + \text{Suc } k]$

using *assms(2)* **by** *force*

next

fix i

assume $i < \text{length } xs$

hence $[xs ! 0 .. < xs ! 0 + \text{Suc } k] ! i = xs ! 0 + i$

by (*metis add-less-cancel-left assms(2) nth-upt*)

moreover

have $xs ! 0 + i < n$

```

    using ⟨i < length xs⟩ assms(2,4) by linarith
  with inc-one-bounded-nth-le[OF assms(1) ⟨i < length xs⟩]
  have xs ! i = xs ! 0 + i
    by simp
  ultimately show xs ! i = [xs ! 0..

```

```

moreover
from inc-one-boundedD(2)[OF assms(1), of 0]
have  $xs ! 0 < n$ 
  by (simp add: assms(2))
moreover
have  $n - xs ! 0 \leq i$ 
  using  $\langle \neg i < \text{length } [xs ! 0..<n] \rangle$  by force
ultimately have  $xs ! i = i - (n - xs ! 0)$ 
  using not-mod-a-pl-b-eq2[of n xs ! 0 i]
  inc-one-bounded-nth-plus[OF assms(1), of 0 i, simplified, OF  $\langle i <$ 
length xs $\rangle$ ]
  by presburger
}
ultimately show  $xs ! i = ([xs ! 0..<n] @ [0..<xs ! 0 + \text{Suc } k - n]) ! i$ 
  by argo
qed
qed

```

lemmas *inc-one-bounded-upt* = *inc-one-bounded-upt1 inc-one-bounded-upt2*

lemma *is-rot-sublist-nil*:

```

is-rot-sublist xs []
by (metis append-Nil is-rot-sublist-def is-sublist-def)

```

lemma *rotate-upt*:

```

 $m \leq n \implies \text{rotate } m [0..<n] = [m..<n] @ [0..<m]$ 
by (metis diff-zero le-Suc-ex length-upt rotate-append upt-add-eq-append zero-order(1))

```

lemma *inc-one-bounded-is-rot-sublist*:

```

assumes inc-one-bounded n xs length xs  $\leq n$ 
shows is-rot-sublist  $[0..<n]$  xs
unfolding is-rot-sublist-def is-sublist-def
proof (cases length xs)

```

```

  case 0
  then show  $\exists na as bs. \text{rotate } na [0..<n] = as @ xs @ bs$ 
    using append-Nil by blast

```

next

```

  case (Suc k)
  hence  $\text{Suc } k \leq n$ 
  using assms(2) by auto

```

```

have  $(xs ! 0) + k < n \implies \exists na as bs. \text{rotate } na [0..<n] = as @ xs @ bs$ 

```

proof –

```

  assume  $(xs ! 0) + k < n$ 
  with inc-one-bounded-upt(1)[OF assms(1) Suc  $\langle \text{Suc } k \leq n \rangle$ ]
  have  $xs = [xs ! 0..<xs ! 0 + \text{Suc } k]$ 
  by blast

```

moreover

```

have  $xs ! 0 + \text{Suc } k \leq n$ 

```

by (*simp add: Suc-leI* $\langle xs ! 0 + k < n \rangle$)
 with *upt-add-eq-append*[of $xs ! 0$ $xs ! 0 + Suc k$ $n - (xs ! 0 + Suc k)$]
 have $[xs ! 0..<n] = [xs ! 0..<xs ! 0 + Suc k] @ [xs ! 0 + Suc k..<n]$
 by (*metis le-add1 le-add-diff-inverse*)
 with *upt-add-eq-append*[of 0 $xs ! 0$ $n - xs ! 0$]
 have $[0..<n] = [0..<xs ! 0] @ [xs ! 0..<xs ! 0 + Suc k] @ [xs ! 0 + Suc k..<n]$
 using $\langle xs ! 0 + Suc k \leq n \rangle$ by *fastforce*
 ultimately show *?thesis*
 by (*metis append.right-neutral append-Nil rotate-append*)
qed
moreover
 have $\neg (xs ! 0) + k < n \implies \exists na as bs. rotate na [0..<n] = as @ xs @ bs$
proof -
 assume $\neg (xs ! 0) + k < n$
 hence $(xs ! 0) + k \geq n$
 by *simp*
 with *inc-one-bounded-upt*(2)[*OF* *assms*(1) *Suc* $\langle Suc k \leq n \rangle$]
 have $xs = [xs ! 0..<n] @ [0..<xs ! 0 + Suc k - n]$
 by *blast*
moreover
 from *inc-one-boundedD*(2)[*OF* *assms*(1), of 0]
 have $xs ! 0 < n$
 by (*simp add: Suc*)
 with *rotate-upt*[of $xs ! 0$ n]
 have $rotate (xs ! 0) [0..<n] = [xs ! 0..<n] @ [0..<xs ! 0]$
 by *linarith*
moreover
 {
 have $0 \leq xs ! 0 + Suc k - n$
 by *simp*
 hence $[0..<xs ! 0 + Suc k - n + (n - Suc k)] =$
 $[0..<xs ! 0 + Suc k - n] @ [xs ! 0 + Suc k - n..<xs ! 0 + Suc k -$
 $n + (n - Suc k)]$
 using *upt-add-eq-append*[of 0 $xs ! 0 + Suc k - n$ $n - Suc k$] by *blast*
moreover
 have $xs ! 0 = xs ! 0 + Suc k - n + (n - Suc k)$
 using $\langle Suc k \leq n \rangle \langle n \leq xs ! 0 + k \rangle$ by *auto*
 ultimately have $[0..<xs ! 0] = [0..<xs ! 0 + Suc k - n] @ [xs ! 0 + Suc k$
 $- n..<xs ! 0]$
 by *arg0*
 }
 ultimately show *?thesis*
 by (*metis append.assoc append-Nil*)
qed
ultimately show $\exists na as bs. rotate na [0..<n] = as @ xs @ bs$
 by *blast*
qed

lemma *is-rot-sublist-idx*:

$is\text{-rot}\text{-sublist } [0..<length\ xs] ys \implies is\text{-rot}\text{-sublist } xs (map\ (!)\ xs)\ ys$
unfolding $is\text{-rot}\text{-sublist}\text{-def } is\text{-sublist}\text{-def}$
proof ($elim\ exE$)
fix $n\ as\ bs$
assume $rotate\ n\ [0..<length\ xs] = as\ @\ ys\ @\ bs$
hence $rotate\ n\ xs = map\ (!)\ xs\ (as\ @\ ys\ @\ bs)$
by ($metis\ map\text{-}nth\ rotate\text{-}map$)
then show $\exists n\ as\ bs. rotate\ n\ xs = as\ @\ map\ (!)\ xs\ ys\ @\ bs$
by $auto$
qed

lemma $is\text{-rot}\text{-sublist}\text{-upt}\text{-eq}\text{-upt}\text{-hd}$:
 $\llbracket is\text{-rot}\text{-sublist } [0..<Suc\ n] ys; length\ ys = Suc\ n; ys\ !\ 0 = 0 \rrbracket \implies ys = [0..<Suc\ n]$
unfolding $is\text{-rot}\text{-sublist}\text{-def } is\text{-sublist}\text{-def}$
proof ($elim\ exE$)
fix $m\ as\ bs$
assume $A: length\ ys = Suc\ n\ ys\ !\ 0 = 0\ rotate\ m\ [0..<Suc\ n] = as\ @\ ys\ @\ bs$
with $rotate\text{-conv}\text{-mod}[of\ m\ [0..<Suc\ n]]$
have $rotate\ (m\ mod\ length\ [0..<Suc\ n])\ [0..<Suc\ n] = as\ @\ ys\ @\ bs$
by $simp$
with $rotate\text{-upt}[of\ m\ mod\ length\ [0..<Suc\ n]\ Suc\ n]$
have $[m\ mod\ length\ [0..<Suc\ n]..<Suc\ n]\ @\ [0..<m\ mod\ length\ [0..<Suc\ n]] =$
 $as\ @\ ys\ @\ bs$
by ($metis\ diff\text{-}zero\ le\text{-}Suc\text{-}eq\ length\text{-}upt\ mod\text{-}Suc\text{-}le\text{-}divisor$)
hence $[m\ mod\ Suc\ n..<Suc\ n]\ @\ [0..<m\ mod\ Suc\ n] = as\ @\ ys\ @\ bs$
by $simp$
moreover
have $as = []$
by ($metis\ A(1)\ A(3)\ diff\text{-}zero\ length\text{-}append\ length\text{-}greater\text{-}0\text{-}conv\ length\text{-}rotate\ length\text{-}upt$
 $less\text{-}add\text{-}same\text{-}cancel2\ not\text{-}add\text{-}less1$)
moreover
have $bs = []$
by ($metis\ A(1)\ A(3)\ append.\text{right}\text{-}neutral\ append\text{-}eq\text{-}append\text{-}conv\ calculation(2)$
 $diff\text{-}zero\ length\text{-}rotate\ length\text{-}upt\ self\text{-}append\text{-}conv2$)
moreover
have $m\ mod\ Suc\ n = 0$
by ($metis\ A\ add.\text{right}\text{-}neutral\ append.\text{right}\text{-}neutral\ calculation(2,3)\ diff\text{-}zero\ length\text{-}rotate$
 $mod\text{-}less\text{-}divisor\ nth\text{-}rotate\ nth\text{-}upt\ self\text{-}append\text{-}conv2\ zero\text{-}le\ zero\text{-}less\text{-}Suc$
 $ordered\text{-}cancel\text{-}comm\text{-}monoid\text{-}diff\text{-}class.\text{add}\text{-}diff\text{-}inverse$)
ultimately show $ys = [0..<Suc\ n]$
by $simp$
qed

lemma $is\text{-rot}\text{-sublist}\text{-upt}\text{-eq}\text{-upt}\text{-last}$:
 $\llbracket is\text{-rot}\text{-sublist } [0..<Suc\ n] ys; length\ ys = Suc\ n; ys\ !\ n = n \rrbracket \implies ys = [0..<Suc\ n]$

$n]$
unfolding *is-rot-sublist-def is-sublist-def*
proof (*elim exE*)
fix m *as* bs
assume A : $\text{length } ys = \text{Suc } n \text{ } ys ! n = n \text{ rotate } m [0..<\text{Suc } n] = as @ ys @ bs$
with *rotate-conv-mod*[of $m [0..<\text{Suc } n]$]
have $\text{rotate } (m \bmod \text{length } [0..<\text{Suc } n]) [0..<\text{Suc } n] = as @ ys @ bs$
by *simp*
with *rotate-upt*[of $m \bmod \text{length } [0..<\text{Suc } n] \text{Suc } n]$
have $[m \bmod \text{length } [0..<\text{Suc } n]..<\text{Suc } n] @ [0..<m \bmod \text{length } [0..<\text{Suc } n]] =$
 $as @ ys @ bs$
by (*metis diff-zero le-Suc-eq length-upt mod-Suc-le-divisor*)
hence $[m \bmod \text{Suc } n..<\text{Suc } n] @ [0..<m \bmod \text{Suc } n] = as @ ys @ bs$
by *simp*
moreover
have $as = []$
by (*metis A(1) A(3) diff-zero length-append length-greater-0-conv length-rotate*
length-upt
less-add-same-cancel2 not-add-less1)
moreover
have $bs = []$
by (*metis A(1) A(3) append.right-neutral append-eq-append-conv calculation(2)*
diff-zero
length-rotate length-upt self-append-conv2)
moreover
from *list-eq-iff-nth-eq*[*THEN iffD1, OF calculation(1), simplified,*
simplified calculation(2,3), simplified]
have $\text{Suc } n = \text{length } ys \forall i < \text{Suc } n. ([m \bmod \text{Suc } n..<n] @ n \# [0..<m \bmod \text{Suc}$
 $n]) ! i = ys ! i$
by *blast+*
hence $([m \bmod \text{Suc } n..<n] @ n \# [0..<m \bmod \text{Suc } n]) ! n = n$
by (*simp add: A(2)*)
with *nth-append*[of $[m \bmod \text{Suc } n..<n] n \# [0..<m \bmod \text{Suc } n] n]$
have $n < \text{length } [m \bmod \text{Suc } n..<n] \vee$
 $(n \# [0..<m \bmod \text{Suc } n]) ! (n - \text{length } [m \bmod \text{Suc } n..<n]) = n$
by *argo*
hence $m \bmod \text{Suc } n = 0$
proof
assume $n < \text{length } [m \bmod \text{Suc } n..<n]$
then show $m \bmod \text{Suc } n = 0$
by *simp*
next
assume B : $(n \# [0..<m \bmod \text{Suc } n]) ! (n - \text{length } [m \bmod \text{Suc } n..<n]) = n$
show $m \bmod \text{Suc } n = 0$
proof (*cases* $n - \text{length } [m \bmod \text{Suc } n..<n]$)
case 0
then show *?thesis*
by *simp*
next

```

    case (Suc x)
    then show ?thesis
    by (metis B One-nat-def add-Suc diff-diff-cancel length-upt lessI mod-Suc-le-divisor

        mod-less-divisor nless-le nth-Cons-Suc nth-upt plus-1-eq-Suc
zero-less-Suc)
    qed
    qed
    ultimately show  $ys = [0..<Suc\ n]$ 
    by simp
qed

end
theory Count-Util
imports HOL-Library.Multiset
        HOL-Combinatorics.List-Permutation
        SuffixArray.List-Util
        SuffixArray.List-Slice
begin

```

3 Counting

3.1 Count List

lemma *count-in*:

$x \in \text{set } xs \implies \text{count-list } xs\ x > 0$
by (*meson count-list-0-iff gr0I*)

lemma *in-count*:

$\text{count-list } xs\ x > 0 \implies x \in \text{set } xs$
by (*metis count-notin less-irrefl*)

lemma *notin-count*:

$\text{count-list } xs\ x = 0 \implies x \notin \text{set } xs$
by (*simp add: count-list-0-iff*)

lemma *count-list-eq-count*:

$\text{count-list } xs\ x = \text{count } (\text{mset } xs)\ x$
by (*induct xs; simp*)

lemma *count-list-perm*:

$xs <\sim\sim> ys \implies \text{count-list } xs\ x = \text{count-list } ys\ x$
by (*simp add: count-list-eq-count*)

lemma *in-count-nth-ex*:

$\text{count-list } xs\ x > 0 \implies \exists i < \text{length } xs. xs\ !\ i = x$
by (*meson in-count in-set-conv-nth*)

lemma *in-count-list-slice-nth-ex*:

count-list (list-slice xs i j) x > 0 $\implies \exists k < \text{length } xs. i \leq k \wedge k < j \wedge xs ! k = x$
by (*meson in-count nth-mem-list-slice*)

3.2 Cardinality

lemma *count-list-card*:

count-list xs x = card {j. j < length xs \wedge xs ! j = x}

proof (*induct xs rule: rev-induct*)

case *Nil*

then show *?case*

by *simp*

next

case (*snoc y xs*)

let *?A = {j. j < length xs \wedge xs ! j = x}*

let *?B = {j. j < length (xs @ [y]) \wedge (xs @ [y]) ! j = x}*

have *length xs \notin ?A*

by *simp*

have *?B - {length xs} = ?A*

by (*intro equalityI subsetI; clarsimp simp: nth-append*)

{

have *y = x \implies count-list (xs @ [y]) x = Suc (card ?A)*

by (*simp add: snoc*)

moreover

have *y = x \implies ?B = insert (length xs) ?A*

by (*metis (mono-tags, lifting) \langle ?B - {length xs} = ?A \rangle insert-Diff length-append-singleton
lessI mem-Collect-eq nth-append-length*)

with *card-insert-disjoint[OF - \langle length xs \notin - \rangle]*

have *y = x \implies card ?B = Suc (card ?A)*

by *simp*

ultimately have *y = x \implies ?case*

by *simp*

}

moreover

have *y \neq x \implies count-list (xs @ [y]) x = card ?A*

by (*simp add: snoc*)

hence *y \neq x \implies ?case*

using \langle ?B - {length xs} = ?A \rangle **by force**

ultimately show *?case*

by *blast*

qed

lemma *card-le-eq-card-less-pl-count-list*:

fixes *s :: 'a :: linorder list*

shows *card {k. k < length s \wedge s ! k \leq a} = card {k. k < length s \wedge s ! k < a}*
+ *count-list s a*

proof –

let $?A = \{k. k < \text{length } s \wedge s ! k \leq a\}$
let $?B = \{k. k < \text{length } s \wedge s ! k < a\}$
let $?C = \{k. k < \text{length } s \wedge s ! k = a\}$

have $?B \cap ?C = \{\}$
by *blast*

hence $\text{card } (?B \cup ?C) = \text{card } ?B + \text{count-list } s \ a$
by (*simp add: card-Un-disjoint count-list-card*)

moreover

have $?A = ?B \cup ?C$

proof *safe*

fix x
assume $s ! x \leq a \wedge s ! x \neq a$
then show $s ! x < a$
by *simp*

next

fix x
assume $s ! x < a$
then show $s ! x \leq a$
by *simp*

qed

hence $\text{card } ?A = \text{card } (?B \cup ?C)$
by *simp*

ultimately show *?thesis*
by *simp*

qed

lemma *card-less-idx-upper-strict*:

fixes $s :: 'a :: \text{linorder list}$
assumes $a \in \text{set } s$
shows $\text{card } \{k. k < \text{length } s \wedge s ! k < a\} < \text{length } s$

proof –

have $\exists i < \text{length } s. s ! i = a$
by (*meson assms in-set-conv-nth*)

then obtain i **where** P :

$i < \text{length } s \wedge s ! i = a$
by *blast*

have $\{k. k < \text{length } s \wedge s ! k < a\} \subseteq \{0..<\text{length } s\}$
using *atLeastLessThan-iff* **by** *blast*

moreover

have $i \in \{0..<\text{length } s\}$
by (*simp add: P(1)*)

moreover

have $i \notin \{k. k < \text{length } s \wedge s ! k < a\}$
by (*simp add: P(2)*)

ultimately have $\{k. k < \text{length } s \wedge s ! k < a\} \subset \{0..<\text{length } s\}$
by *blast*

then show *?thesis*
by (*metis card-upt finite-atLeastLessThan psubset-card-mono*)
qed

lemma *card-less-idx-upper*:
shows $\text{card } \{k. k < \text{length } s \wedge s ! k < a\} \leq \text{length } s$
by (*metis (no-types, lifting) atLeastLessThan-iff bot-nat-0.extremum mem-Collect-eq subsetI*
subset-eq-atLeast0-lessThan-card)

lemma *card-pl-count-list-strict-upper*:
fixes $s :: 'a :: \text{linorder list}$
shows $\text{card } \{i. i < \text{length } s \wedge s ! i < a\} + \text{count-list } s \ a \leq \text{length } s$
proof –
let $?X = \{i. i < \text{length } s \wedge s ! i < a\}$
let $?Y = \{i. i < \text{length } s \wedge s ! i = a\}$

have $?X \cap ?Y = \{\}$
by *blast*
hence $\text{card } (?X \cup ?Y) = \text{card } ?X + \text{card } ?Y$
by (*simp add: card-Un-disjoint*)
moreover
have $\text{card } ?Y = \text{count-list } s \ a$
by (*simp add: count-list-card*)
moreover
have $?X \cup ?Y \subseteq \{0..<\text{length } s\}$
by (*simp add: subset-iff*)
hence $\text{card } (?X \cup ?Y) \leq \text{length } s$
using *subset-eq-atLeast0-lessThan-card* **by** *blast*
ultimately show *?thesis*
by *presburger*
qed

3.3 Sorting

lemma *sorted-nth-le*:
assumes *sorted xs*
and $\text{card } \{k. k < \text{length } xs \wedge xs ! k < c\} < \text{length } xs$
shows $c \leq xs ! \text{card } \{k. k < \text{length } xs \wedge xs ! k < c\}$
using *assms*
proof (*induct xs*)
case *Nil*
then show *?case*
by *simp*
next
case (*Cons a xs*)
note *IH = this*

let $?A = \{k. k < \text{length } (a \# xs) \wedge (a \# xs) ! k < c\}$

```

let ?B = {k. k < length xs ∧ xs ! k < c}

have a < c ∨ c ≤ a
  by fastforce
then show ?case
proof
  assume a < c

  have finite ?B
    by auto
  hence finite (Suc ‹ ?B)
    by blast

  have card (Suc ‹ ?B) = card ?B
    using card-image inj-Suc by blast

  have {0} ∩ Suc ‹ ?B = {}
    by blast

  have ?A = {0} ∪ Suc ‹ ?B
proof (intro equalityI subsetI)
  fix x
  assume x ∈ {0} ∪ Suc ‹ ?B
  then show x ∈ ?A
  proof
    assume x ∈ {0}
    hence x = 0
      by simp
    then show ?thesis
      by (simp add: ‹a < c›)
  next
    assume x ∈ Suc ‹ ?B
    hence ∃y. x = Suc y ∧ xs ! y < c
      by blast
    then show ?thesis
      using ‹x ∈ Suc ‹ ?B› by force
  qed
next
  fix x
  assume x ∈ ?A
  hence x = 0 ∨ (∃y. x = Suc y ∧ xs ! y < c)
    using not0-implies-Suc by fastforce
  then show x ∈ {0} ∪ Suc ‹ ?B
  proof
    assume x = 0
    then show ?thesis
      by blast
  next
    assume ∃y. x = Suc y ∧ xs ! y < c

```

```

    then show ?thesis
      using ⟨x ∈ ?A⟩ by fastforce
    qed
  qed
  with card-Un-disjoint[OF - ⟨finite (Suc ‘ ?B)⟩ ⟨- ∩ - = -⟩]
  have card ?A = Suc (card ?B)
    by (simp add: ⟨card (Suc ‘ ?B) = card ?B⟩)
  hence (a # xs) ! card {k. k < length (a # xs) ∧ (a # xs) ! k < c} =
    xs ! card {k. k < length xs ∧ xs ! k < c}
    by simp
  then show ?case
    using Cons.hyps IH(2) IH(3) ⟨card ?A = Suc (card ?B)⟩ by auto
next
  assume c ≤ a
  have {k. k < length (a # xs) ∧ (a # xs) ! k < c} = {}
  proof safe
    fix x
    assume A: x < length (a # xs) (a # xs) ! x < c
    show x ∈ {}
    proof (cases x)
      case 0
      then show ?thesis
        using A(2) ⟨c ≤ a⟩ by auto
    next
      case (Suc n)
      hence a ≤ (a # xs) ! x
        using A(1) IH(2) by auto
      then show ?thesis
        using A(2) ⟨c ≤ a⟩ by auto
    qed
  qed
  then show ?thesis
    by (metis ⟨c ≤ a⟩ card.empty nth-Cons-0)
  qed
qed

```

lemma *sorted-nth-le-gen*:

```

  assumes sorted xs
  and card {k. k < length xs ∧ xs ! k < c} + i < length xs
  shows c ≤ xs ! (card {k. k < length xs ∧ xs ! k < c} + i)
  proof (cases i)
    case 0
    then show ?thesis
      using assms(1) assms(2) sorted-nth-le by auto
  next
    let ?x = card {k. k < length xs ∧ xs ! k < c}
    case (Suc n)
    with sorted-wrt-nth-less[OF assms(1), of ?x ?x + i]
    have xs ! ?x ≤ xs ! (?x + i)

```


using *assms(1) assms(2) le-add1 sorted-nth-mono* **by** *blast*
moreover
have $c \leq xs ! ?x$
using *add-lessD1 assms(1) assms(2) sorted-nth-le* **by** *blast*
ultimately show *?thesis*
by *order*
qed

lemma *sorted-nth-less-gen:*

assumes *sorted xs*
and $i < \text{card } \{k. k < \text{length } xs \wedge xs ! k < c\}$
shows $xs ! i < c$
proof (*rule ccontr*)
assume $\neg xs ! i < c$
hence $i \notin \{k. k < \text{length } xs \wedge xs ! k < c\}$
by *simp*
hence $\forall k < \text{length } xs. i \leq k \longrightarrow k \notin \{k. k < \text{length } xs \wedge xs ! k < c\}$
using *assms(1) sorted-iff-nth-mono* **by** *fastforce*
hence $\{k. k < \text{length } xs \wedge xs ! k < c\} \subseteq \{0..<i\}$
by *fastforce*
moreover
have $\text{card } \{0..<i\} = i$
by *auto*
ultimately show *False*
by (*metis assms(2) card-mono finite-atLeastLessThan verit-comp-simplify1(3)*)
qed

lemma *sorted-nth-gr-gen:*

assumes *sorted xs*
and $\text{card } \{k. k < \text{length } xs \wedge xs ! k < c\} + i < \text{length } xs$
and $\text{count-list } xs \ c \leq i$
shows $xs ! (\text{card } \{k. k < \text{length } xs \wedge xs ! k < c\} + i) > c$
proof –
let $?A = \{k. k < \text{length } xs \wedge xs ! k < c\}$
have $xs ! (\text{card } ?A + i) \geq c$
using *assms(1) assms(2) sorted-nth-le-gen* **by** *blast*
hence $xs ! (\text{card } ?A + i) = c \vee xs ! (\text{card } ?A + i) > c$
by *force*
then show *?thesis*
proof
assume $xs ! (\text{card } ?A + i) > c$
then show *?thesis* .
next
assume $xs ! (\text{card } ?A + i) = c$

from *sorted-nth-le-gen[OF assms(1)]*
have $P1: \forall k < \text{length } xs. \text{card } ?A \leq k \longrightarrow c \leq xs ! k$
by (*metis (mono-tags, lifting) assms(1) dual-order.strict-trans2 linorder-not-le sorted-iff-nth-mono sorted-nth-le*)

have $P2: \forall k < \text{length } xs. k < \text{card } ?A + \text{Suc } i \longrightarrow xs ! k \leq c$
by (*metis (mono-tags, lifting) Suc-leI* $\langle xs ! (\text{card } ?A + i) = c \rangle$ *add-Suc-right*
add-le-cancel-left *assms(1,2)* *plus-1-eq-Suc*
sorted-nth-mono)

have $P3: \forall x \in \{\text{card } ?A..<\text{card } ?A + \text{Suc } i\}. xs ! x = c$
proof *safe*
fix x
assume $x \in \{\text{card } ?A..<\text{card } ?A + \text{Suc } i\}$
hence $A: \text{card } ?A \leq x < \text{card } ?A + \text{Suc } i$
by *simp+*

have $c \leq xs ! x$
using $P1$ A *assms(2)* **by** *auto*
moreover
have $xs ! x \leq c$
using $A(2)$ $P2$ *assms(2)* **by** *force*
ultimately show $xs ! x = c$
by *simp*

qed

have $\{\text{card } ?A..<\text{card } ?A + \text{Suc } i\} \subseteq \{k. k < \text{length } xs \wedge xs ! k = c\}$

proof

fix x
assume $A: x \in \{\text{card } ?A..<\text{card } ?A + \text{Suc } i\}$
have $x < \text{card } ?A + \text{Suc } i$
using A **by** *simp+*
hence $x < \text{length } xs$
using *assms(2)* **by** *linarith*
moreover
have $xs ! x = c$
using $P3$ A **by** *blast*
ultimately show $x \in \{k. k < \text{length } xs \wedge xs ! k = c\}$
by *blast*

qed

hence $\text{count-list } xs \ c \geq \text{card } \{\text{card } ?A..<\text{card } ?A + \text{Suc } i\}$

using *count-list-card[of xs c]* *card-mono*

by (*metis (mono-tags, lifting)* $\langle xs ! (\text{card } ?A + i) = c \rangle$ *assms(2)* *card-ge-0-finite*
count-in
nth-mem)

moreover

have $\text{card } \{\text{card } ?A..<\text{card } ?A + \text{Suc } i\} = \text{Suc } i$

by *simp*

ultimately have *False*

using *assms(3)* **by** *linarith*

then show *?thesis*

by *blast*

```

    qed
  qed

end
theory Rank-Util
  imports HOL-Library.Multiset
           Count-Util
           SuffixArray.Prefix
begin

```

4 Rank Definition

Count how many occurrences of an element are in a certain index in the list

Definition 3.7 from [3]: Rank

```

definition rank :: 'a list  $\Rightarrow$  'a  $\Rightarrow$  nat  $\Rightarrow$  nat
  where
    rank s x i  $\equiv$  count-list (take i s) x

```

5 Rank Properties

5.1 List Properties

```

lemma rank-cons-same:
  rank (x # xs) x (Suc i) = Suc (rank xs x i)
by (simp add: rank-def)

```

```

lemma rank-cons-diff:
  a  $\neq$  x  $\implies$  rank (a # xs) x (Suc i) = rank xs x i
by (simp add: rank-def)

```

5.2 Counting Properties

```

lemma rank-length:
  rank xs x (length xs) = count-list xs x
by (simp add: rank-def)

```

```

lemma rank-gre-length:
  length xs  $\leq$  n  $\implies$  rank xs x n = count-list xs x
by (simp add: rank-def)

```

```

lemma rank-not-in:
  x  $\notin$  set xs  $\implies$  rank xs x i = 0
by (metis gr-zeroI in-count rank-def set-take-subset subset-code(1))

```

```

lemma rank-0:
  rank xs x 0 = 0
by (simp add: rank-def)

```

Theorem 3.11 from [3]: Rank Equivalence

lemma *rank-card-spec*:

$rank\ xs\ x\ i = card\ \{j. j < length\ xs \wedge j < i \wedge xs\ !\ j = x\}$

proof –

have $rank\ xs\ x\ i = count\text{-}list\ (take\ i\ xs)\ x$

by (*meson rank-def*)

moreover

have $count\text{-}list\ (take\ i\ xs)\ x = card\ \{j. j < length\ (take\ i\ xs) \wedge (take\ i\ xs)\ !\ j = x\}$

by (*metis count-list-card*)

moreover

have $\{j. j < length\ (take\ i\ xs) \wedge (take\ i\ xs)\ !\ j = x\} =$
 $\{j. j < length\ xs \wedge j < i \wedge xs\ !\ j = x\}$

by *fastforce*

ultimately show *?thesis*

by *simp*

qed

lemma *le-rank-plus-card*:

$i \leq j \implies$

$rank\ xs\ x\ j = rank\ xs\ x\ i + card\ \{k. k < length\ xs \wedge i \leq k \wedge k < j \wedge xs\ !\ k = x\}$

proof –

assume $i \leq j$

let $?X = \{k. k < length\ xs \wedge k < j \wedge xs\ !\ k = x\}$

have $rank\ xs\ x\ j = card\ ?X$

by (*simp add: rank-card-spec*)

moreover

let $?Y = \{k. k < length\ xs \wedge k < i \wedge xs\ !\ k = x\}$

have $rank\ xs\ x\ i = card\ ?Y$

by (*simp add: rank-card-spec*)

moreover

let $?Z = \{k. k < length\ xs \wedge i \leq k \wedge k < j \wedge xs\ !\ k = x\}$

have $?Y \cup ?Z = ?X$

proof *safe*

fix k

assume $k < i$

then show $k < j$

using $\langle i \leq j \rangle$ *order-less-le-trans* **by** *blast*

next

fix k

assume $\neg i \leq k$

then show $k < i$

using *linorder-le-less-linear* **by** *blast*

qed

moreover

have $?Y \cap ?Z = \{\}$

by *force*
 hence $\text{card } (?Y \cup ?Z) = \text{card } ?Y + \text{card } ?Z$
 by (*simp add: card-Un-disjoint*)
 ultimately show *?thesis*
 by *presburger*
 qed

5.3 Bound Properties

lemma *rank-lower-bound:*

assumes $k < \text{rank } xs \ x \ i$

shows $k < i$

proof –

from *rank-card-spec[of xs x i]*

have $\text{rank } xs \ x \ i = \text{card } \{j. j < \text{length } xs \wedge j < i \wedge xs ! j = x\}$.

hence $k < \text{card } \{j. j < \text{length } xs \wedge j < i \wedge xs ! j = x\}$

using *assms* by *presburger*

moreover

{

have $i \leq \text{length } xs \vee \text{length } xs < i$

using *linorder-not-less* by *blast*

moreover

have $i \leq \text{length } xs \implies \{j. j < \text{length } xs \wedge j < i \wedge xs ! j = x\} \subseteq \{0..<i\}$

using *atLeast0LessThan* by *blast*

hence $i \leq \text{length } xs \implies \text{card } \{j. j < \text{length } xs \wedge j < i \wedge xs ! j = x\} \leq i$

using *subset-eq-atLeast0-lessThan-card* by *presburger*

moreover

have $\text{length } xs < i \implies \{j. j < \text{length } xs \wedge j < i \wedge xs ! j = x\} \subseteq \{0..<\text{length } xs\}$

using *atLeast0LessThan* by *blast*

hence $\text{length } xs < i \implies \text{card } \{j. j < \text{length } xs \wedge j < i \wedge xs ! j = x\} \leq \text{length } xs$

xs

using *subset-eq-atLeast0-lessThan-card* by *presburger*

hence $\text{length } xs < i \implies \text{card } \{j. j < \text{length } xs \wedge j < i \wedge xs ! j = x\} \leq i$

by *linarith*

ultimately have $\text{card } \{j. j < \text{length } xs \wedge j < i \wedge xs ! j = x\} \leq i$

by *blast*

}

ultimately show *?thesis*

using *dual-order.strict-trans1* by *blast*

qed

corollary *rank-Suc-ex:*

assumes $k < \text{rank } xs \ x \ i$

shows $\exists l. i = \text{Suc } l$

by (*metis Nat.lessE assms rank-lower-bound*)

lemma *rank-upper-bound:*

$[i < \text{length } xs; xs ! i = x] \implies \text{rank } xs \ x \ i < \text{count-list } xs \ x$

```

proof (induct xs arbitrary: i)
  case Nil
  then show ?case
    by (simp add: rank-def)
next
  case (Cons a xs i)
  then show ?case
  proof (cases i)
    case 0
    then show ?thesis
      by (metis Cons.prem(2) count-in list.set-intros(1) nth-Cons-0 rank-0)
    next
    case (Suc n)
    then show ?thesis
      by (metis Cons.hyps Cons.prem(2) Suc-less-eq length-Cons nth-Cons-Suc rank-cons-diff
          rank-cons-same rank-length)
  qed
qed

```

```

lemma rank-idx-mono:
   $i \leq j \implies \text{rank } xs \ x \ i \leq \text{rank } xs \ x \ j$ 
proof (cases i = j)
  assume i = j
  then show ?thesis
    by simp
  next
  assume  $i \leq j \ i \neq j$ 
  hence  $i < j$ 
    using antisym-conv2 by blast
  hence prefix xs j = prefix xs i @ list-slice xs i j
    by (metis <i ≤ j> append-take-drop-id list-slice.elims min.absorb1 take-take)
  hence rank xs x j = rank xs x i + count-list (list-slice xs i j) x
    by (metis count-list-append rank-def)
  then show ?thesis
    by fastforce
qed

```

```

lemma rank-less:
   $\llbracket i < \text{length } xs; i < j; xs \ ! \ i = x \rrbracket \implies \text{rank } xs \ x \ i < \text{rank } xs \ x \ j$ 
proof -
  let ?X = {k. k < length xs ∧ i ≤ k ∧ k < j ∧ xs ! k = x}
  assume  $i < \text{length } xs \ i < j \ xs \ ! \ i = x$ 
  with le-rank-plus-card[of i j xs x]
  have rank xs x j = rank xs x i + card ?X
    using nless-le by blast
  moreover
  have  $i \in ?X$ 
    using <i < j> <i < length xs> <xs ! i = x> by blast

```

hence $\text{card } ?X > 0$
using *card-gt-0-iff* **by** *fastforce*
ultimately show *?thesis*
by *linarith*
qed

lemma *rank-upper-bound-gen*:
 $\text{rank } xs \ x \ i \leq \text{count-list } xs \ x$
by (*metis nat-le-linear rank-gre-length rank-idx-mono*)

5.4 Sorted Properties

lemma *sorted-card-rank-idx*:
assumes *sorted xs*
and $i < \text{length } xs$
shows $i = \text{card } \{j. j < \text{length } xs \wedge xs ! j < xs ! i\} + \text{rank } xs \ (xs ! i) \ i$
proof –

let $?A = \{j. j < \text{length } xs \wedge xs ! j < xs ! i\}$
let $?B = \{j. j < \text{length } xs \wedge xs ! j = xs ! i\}$

have $?B \neq \{\}$
using *assms(2)* **by** *blast*

have $\text{Min } ?B \in ?B$
by (*metis (no-types, lifting) Min-in <?B ≠ {}> finite-nat-set-iff-bounded mem-Collect-eq*)
hence $\text{Min } ?B < \text{length } xs \ xs ! (\text{Min } ?B) = xs ! i$
by *simp-all*

have $\text{Min } ?B \leq i$
by (*simp add: assms(2)*)

have $P: \forall k < \text{Min } ?B. xs ! k < xs ! i$
proof (*intro allI impI*)
fix k
assume $k < \text{Min } ?B$
with *sorted-nth-mono[OF assms(1) - <Min ?B < length xs>]*
have $xs ! k \leq xs ! (\text{Min } ?B)$
using *le-eq-less-or-eq* **by** *presburger*

show $xs ! k < xs ! i$
proof (*rule ccontr*)
assume $\neg xs ! k < xs ! i$
with $\langle xs ! k \leq xs ! (\text{Min } ?B) \rangle \langle xs ! (\text{Min } ?B) = xs ! i \rangle$
have $xs ! k = xs ! i$
by *order*
with $\langle k < \text{Min } ?B \rangle \langle \text{Min } ?B < \text{length } xs \rangle$
have $k \in ?B$

```

    by auto
  then show False
    by (metis (mono-tags, lifting) Min-gr-iff ‹k < Min ?B› ‹?B ≠ {}› fi-
nite-nat-set-iff-bounded
                                           less-irrefl-nat mem-Collect-eq)

qed
qed

have ?A = {0..

```



```

    using atLeastLessThan-iff by blast+
  moreover
  have  $xs ! x = xs ! i$ 
  proof -
    have  $xs ! x \leq xs ! i$ 
      using assms(1,2)  $\langle x < i \rangle$ 
      by (simp add: sorted-wrt-nth-less)
    moreover
    have  $xs ! \text{Min } ?B \leq xs ! x$ 
      using assms(1,2)  $\langle \text{Min } ?B \leq x \rangle \langle x < i \rangle$ 
      by (meson order.strict-trans sorted-iff-nth-mono)
    ultimately show ?thesis
      using  $\langle xs ! \text{Min } ?B = xs ! i \rangle$  by order
  qed
  ultimately show  $x \in ?C$ 
    using assms(2) by fastforce
  qed
  ultimately have  $\text{rank } xs (xs ! i) i = \text{card } \{\text{Min } ?B..<i\}$ 
    by presburger
}
ultimately show ?thesis
  by (simp add: \langle \text{Min } ?B \leq i \rangle)
qed

```

```

lemma sorted-rank:
  assumes sorted xs
  and  $i < \text{length } xs$ 
  and  $xs ! i = a$ 
shows  $\text{rank } xs a i = i - \text{card } \{k. k < \text{length } xs \wedge xs ! k < a\}$ 
  using assms(1) assms(2) assms(3) sorted-card-rank-idx by fastforce

```

```

lemma sorted-rank-less:
  assumes sorted xs
  and  $i < \text{length } xs$ 
  and  $xs ! i < a$ 
shows  $\text{rank } xs a i = 0$ 
proof -
  have  $\text{rank } xs a i = \text{card } \{k. k < \text{length } xs \wedge k < i \wedge xs ! k = a\}$ 
    by (simp add: rank-card-spec)
  moreover
  have  $\{k. k < \text{length } xs \wedge k < i \wedge xs ! k = a\} = \{\}$ 
    using assms sorted-wrt-nth-less by fastforce
  ultimately show ?thesis
    by fastforce
  qed

```

```

lemma sorted-rank-greater:
  assumes sorted xs
  and  $i < \text{length } xs$ 

```

```

and     $xs ! i > a$ 
shows  $rank\ xs\ a\ i = count\text{-}list\ xs\ a$ 
proof -
  let ?A = {k. k < length xs ∧ k < i ∧ xs ! k = a}
  have rank xs a i = card ?A
    by (simp add: rank-card-spec)
  moreover
  let ?B = {k. k < length xs ∧ k ≥ i ∧ xs ! k = a}
  let ?C = {k. k < length xs ∧ xs ! k = a}
  {
    have ?A ∪ ?B = ?C
    proof safe
      fix x
      assume ¬ i ≤ x
      then show x < i
        using linorder-le-less-linear by blast
    qed
    moreover
    have ?B = {}
    proof -
      have ∀ k < length xs. k ≥ i → xs ! k > a
        by (meson assms(1) assms(3) dual-order.strict-trans1 sorted-nth-mono)
      then show ?thesis
        by blast
    qed
    ultimately have ?A = ?C
      by blast
  }
  ultimately show ?thesis
    by (simp add: count-list-card)
qed

end
theory Select-Util
  imports Count-Util
           SuffixArray.Sorting-Util
begin

```

6 Select Definition

Find nth occurrence of an element in a list

Definition 3.8 from [3]: Select

```

fun select :: 'a list ⇒ 'a ⇒ nat ⇒ nat
  where
  select [] - - = 0 |
  select (a#xs) x 0 = (if x = a then 0 else Suc (select xs x 0)) |
  select (a#xs) x (Suc i) = (if x = a then Suc (select xs x i) else Suc (select xs x
(Suc i)))

```

7 Select Properties

7.1 Length Properties

lemma *notin-imp-select-length*:

$x \notin \text{set } xs \implies \text{select } xs \ x \ i = \text{length } xs$

proof (*induct xs arbitrary: i*)

case *Nil*

then show *?case*

by *simp*

next

case (*Cons a xs i*)

then show *?case*

proof (*cases i*)

case *0*

then show *?thesis*

using *Cons.hyps Cons.prem*s **by** *fastforce*

next

case (*Suc n*)

then show *?thesis*

using *Cons.hyps Cons.prem*s **by** *force*

qed

qed

lemma *select-length-imp-count-list-less*:

$\text{select } xs \ x \ i = \text{length } xs \implies \text{count-list } xs \ x \leq i$

by (*induct rule: select.induct[of - xs x i]; simp split: if-splits*)

lemma *select-Suc-length*:

$\text{select } xs \ x \ i = \text{length } xs \implies \text{select } xs \ x \ (\text{Suc } i) = \text{length } xs$

by (*induct rule: select.induct[of - xs x i]; clarsimp split: if-splits*)

7.2 List Properties

lemma *select-cons-neq*:

$\llbracket \text{select } xs \ x \ i = j; x \neq a \rrbracket \implies \text{select } (a \# xs) \ x \ i = \text{Suc } j$

by (*cases i; simp*)

lemma *cons-neq-select*:

$\llbracket \text{select } (a \# xs) \ x \ i = \text{Suc } j; x \neq a \rrbracket \implies \text{select } xs \ x \ i = j$

by (*cases i; simp*)

lemma *cons-eq-select*:

$\text{select } (x \# xs) \ x \ (\text{Suc } i) = \text{Suc } j \implies \text{select } xs \ x \ i = j$

by *simp*

lemma *select-cons-eq*:

$\text{select } xs \ x \ i = j \implies \text{select } (x \# xs) \ x \ (\text{Suc } i) = \text{Suc } j$

by *simp*

7.3 Bound Properties

lemma *select-max*:
 select xs x i ≤ length xs
 by (*induct rule: select.induct[of - xs x i]; simp*)

7.4 Nth Properties

lemma *nth-select*:
 $\llbracket j < \text{length } xs; \text{count-list } (\text{take } (\text{Suc } j) \text{ } xs) \text{ } x = \text{Suc } i; xs ! j = x \rrbracket$
 $\implies \text{select } xs \text{ } x \text{ } i = j$
proof (*induct arbitrary: j rule: select.induct[of - xs x i]*)
 case (1 *uu uv*)
 then show *?case*
 by *simp*
next
 case (2 *a xs x*)
 then show *?case*
 proof (*cases j*)
 case 0
 then show *?thesis*
 using 2.prem(3) **by** *auto*
 next
 case (*Suc n*)

 have $xs ! n = x$
 using 2.prem(3) *Suc* **by** *auto*
 moreover
 have $n < \text{length } xs$
 using 2.prem(1) *Suc* **by** *auto*
 moreover
 have $x \neq a$
 proof (*rule ccontr*)
 assume $\neg x \neq a$
 hence $x = a$
 by *blast*
 moreover
 have $\text{count-list } (\text{take } (\text{Suc } n) \text{ } xs) \text{ } x > 0$
 by (*simp add: <n < length xs> <xs ! n = x> take-Suc-conv-app-nth*)
 ultimately show *False*
 using 2.prem(2) *Suc* **by** *auto*
 qed
 moreover
 have $\text{count-list } (\text{take } (\text{Suc } n) \text{ } xs) \text{ } x = \text{Suc } 0$
 using 2.prem(2) *Suc calculation(3)* **by** *auto*
 ultimately have $\text{select } xs \text{ } x \text{ } 0 = n$
 using 2.hyps **by** *blast*
 then show *?thesis*
 by (*simp add: Suc <x ≠ a>*)
qed

```

next
  case (3 a xs x i)
  then show ?case
  proof (cases j)
    case 0
    then show ?thesis
    using 3.premis(2) 3.premis(3) by force
  next
  case (Suc n)
  then show ?thesis
  by (metis 3.hyps 3.premis Suc-inject Suc-less-eq add.right-neutral add-Suc-right
    count-list.simps(2) length-Cons nth-Cons-Suc plus-1-eq-Suc se-
lect.simps(3)
    take-Suc-Cons)

qed
qed

lemma nth-select-alt:
  [|j < length xs; count-list (take j xs) x = i; xs ! j = x|]
  ==> select xs x i = j
proof (induct arbitrary: j rule: select.induct[of - xs x i])
  case (1 uu uv)
  then show ?case
  by simp
next
  case (2 a xs x j)
  then show ?case
  proof (cases j)
    case 0
    then show ?thesis
    using 2.premis(3) by auto
  next
  case (Suc n)
  then show ?thesis
  by (metis 2.hyps 2.premis Suc-less-eq count-in count-list.simps(2) length-Cons
    list.set-intros(1) not-gr-zero nth-Cons-Suc select.simps(2) take-Suc-Cons)

qed
next
  case (3 a xs x i)
  then show ?case
  proof (cases j)
    case 0
    then show ?thesis
    using 3.premis(2) by auto
  next
  case (Suc n)
  then show ?thesis
  by (metis 3.hyps 3.premis One-nat-def Suc-inject Suc-less-eq add.right-neutral
    add-Suc-right count-list.simps(2) length-Cons nth-Cons-Suc se-

```

```

lect.simps(3)
  take-Suc-Cons)
qed
qed

lemma select-nth:
  [[select xs x i = j; j < length xs]]
  ==> count-list (take (Suc j) xs) x = Suc i ^ xs ! j = x
proof (induct arbitrary: j rule: select.induct[of - xs x i])
  case (1 uu uv)
  then show ?case
    by simp
next
  case (2 a xs x j)
  then show ?case
  proof (cases j)
    case 0
    then show ?thesis
    by (metis 2.prem1 One-nat-def add.right-neutral add-Suc-right count-list.simps
      nat.simps(3) nth-Cons-0 select-cons-neq take0 take-Suc-Cons)
  next
    case (Suc n)
    then show ?thesis
    using 2.hyps 2.prem1 2.prem2 by auto
  qed
next
  case (3 a xs x i j)
  then show ?case
  proof (cases j)
    case 0
    then show ?thesis
    by (metis 3.prem1 nat.simps(3) select-cons-eq select-cons-neq)
  next
    case (Suc n)
    then show ?thesis
    by (metis 3.hyps 3.prem One-nat-def Suc-le-eq add.right-neutral add-Suc-right
      count-list.simps(2) length-Cons less-Suc-eq-le nth-Cons-Suc select-cons-eq
      select-cons-neq take-Suc-Cons)
  qed
qed
qed

lemma select-nth-alt:
  [[select xs x i = j; j < length xs]]
  ==> count-list (take j xs) x = i ^ xs ! j = x
proof (induct arbitrary: j rule: select.induct[of - xs x i])
  case (1 uu uv)
  then show ?case
    by simp
next

```

```

case (2 a xs x j)
then show ?case
proof (cases j)
  case 0
  then show ?thesis
  using 2.prem(1) order.strict-iff-not by fastforce
next
case (Suc n)
then show ?thesis
  by (metis 2.prem(1) 2.prem(2) nat.inject nth-select-alt select-nth)
qed
next
case (3 a xs x i j)
then show ?case
proof (cases j)
  case 0
  then show ?thesis
  by (metis 3.prem(1) nat.simps(3) select-cons-eq select-cons-neq)
next
case (Suc n)
then show ?thesis
  by (metis 3.prem nat.inject nth-select-alt select-nth)
qed
qed

```

```

lemma select-less-0-nth:
  assumes i < length xs
  and i < select xs x 0
shows xs ! i ≠ x
proof (cases select xs x 0 < length xs)
  assume select xs x 0 < length xs
  with select-nth-alt[of xs x 0 select xs x 0]
  have count-list (take (select xs x 0) xs) x = 0 xs ! select xs x 0 = x
  by blast+
  with count-list-0-iff
  have x ∉ set (take (select xs x 0) xs)
  by metis
  then show ?thesis
  by (simp add: ⟨select xs x 0 < length xs⟩ assms(2) in-set-conv-nth)
next
assume ¬ select xs x 0 < length xs
hence length xs ≤ select xs x 0
  using linorder-le-less-linear by blast
with select-max[of xs x 0]
have select xs x 0 = length xs
  by simp
with select-length-imp-count-list-less
have count-list xs x = 0
  by (metis le-zero-eq)

```

```

with count-list-0-iff
have  $x \notin \text{set } xs$ 
  by fastforce
then show ?thesis
  using assms(1) nth-mem by blast
qed

```

7.5 Sorted Properties

Theorem 3.10 from [3]: Select Sorted Equivalence

```

lemma sorted-select:
  assumes sorted xs
  and  $i < \text{count-list } xs \ x$ 
shows  $\text{select } xs \ x \ i = \text{card } \{j. j < \text{length } xs \wedge xs ! j < x\} + i$ 
  using assms
proof (induct rule: select.induct[of - xs x i])
  case (1 uu ww)
  then show ?case
    by simp
next
  case (2 a xs x)
  note IH = this

  from IH(2)
  have sorted xs
    by simp

  have  $x = a \vee x \neq a$ 
    by blast
  moreover
  have  $x \neq a \implies ?case$ 
  proof -
    assume  $x \neq a$ 
    hence  $0 < \text{count-list } xs \ x$ 
      using IH(3) by fastforce
    with IH(1)[OF <x ≠ a> <sorted xs>]
    have  $\text{select } xs \ x \ 0 = \text{card } \{j. j < \text{length } xs \wedge xs ! j < x\}$ 
      by simp
    moreover
    {
      from in-count[OF <0 < count-list xs x>]
      have  $x \in \text{set } xs$  .
      with IH(2) <x ≠ a>
      have  $a < x$ 
        by (simp add: order-less-le)
      have  $\{j. j < \text{length } (a \# xs) \wedge (a \# xs) ! j < x\} =$ 
         $\{0\} \cup \text{Suc } \{j. j < \text{length } xs \wedge xs ! j < x\}$ 
      proof (safe)
        show  $(a \# xs) ! 0 < x$ 
    }
  
```



```

    by (simp add: ‹a < x›)
next
  fix y
  assume y < length xs
  then show Suc y < length (a # xs)
    by simp
next
  fix y
  assume y < length xs xs ! y < x
  then show (a # xs) ! Suc y < x
    by simp
next
  fix j
  assume A: j ∉ Suc ‘ {v. v < length xs ∧ xs ! v < x} j < length (a # xs)
    (a # xs) ! j < x

  have ∃ k. j = Suc k ⇒ False
  proof -
    assume ∃ k. j = Suc k
    then obtain k where
      j = Suc k
      by blast
    hence B: k < length xs xs ! k < x k ∉ {v. v < length xs ∧ xs ! v < x}
      using A by simp-all
    then show False
      by auto
  qed
  then show j = 0
    using not0-implies-Suc by blast
qed
moreover
{
  have finite {0}
    by blast
  moreover
  have finite (Suc ‘ {j. j < length xs ∧ xs ! j < x})
    by simp
  moreover
  have {0} ∩ Suc ‘ {j. j < length xs ∧ xs ! j < x} = {}
    by blast
  ultimately have
    card ({0} ∪ Suc ‘ {j. j < length xs ∧ xs ! j < x}) =
      Suc (card (Suc ‘ {j. j < length xs ∧ xs ! j < x}))
    using card-Un-disjoint[of {0} Suc ‘ {j. j < length xs ∧ xs ! j < x}] by
simp
}
ultimately have
  card {j. j < length (a # xs) ∧ (a # xs) ! j < x} =
    Suc (card (Suc ‘ {j. j < length xs ∧ xs ! j < x}))

```

```

    by presburger
  hence card {j. j < length (a # xs) ∧ (a # xs) ! j < x} =
    Suc (card {j. j < length xs ∧ xs ! j < x})
  by (simp add: card-image)
}
moreover
have select (a # xs) x 0 = Suc (select xs x 0)
  using ⟨x ≠ a⟩ select.simps(2)[of a xs x] by auto
ultimately show ?thesis
  by simp
qed
moreover
have x = a ⇒ ?case
proof -
  assume x = a
  with IH(2)
  have {j. j < length (a # xs) ∧ (a # xs) ! j < x} = {}
  by (metis (no-types, lifting) Collect-empty-eq less-nat-zero-code linorder-not-less
    neq0-conv
    nth-Cons-0 order-refl sorted-nth-less-mono)
  with ⟨x = a⟩
  show ?thesis
  by force
qed
ultimately show ?case
  by blast
next
case (3 a xs x i)
note IH = this

have sorted xs

  using IH(3) by auto
have a ≤ x
  by (metis IH(3-) Suc-less-eq2 count-list.simps(2) in-count order-refl sorted-simps(2)
    zero-less-Suc)

have x = a ∨ x ≠ a
  by blast
moreover
have x = a ⇒ ?case
proof -
  assume x = a
  with IH(4)
  have i < count-list xs x
    by auto
  with IH(1)[OF ⟨x = a⟩ ⟨sorted xs⟩]
  have select xs x i = card {j. j < length xs ∧ xs ! j < x} + i .
  moreover

```

```

from select.simps(3)[of a xs x i]  $\langle x = a \rangle$ 
have select (a # xs) x (Suc i) = Suc (select xs x i)
  by simp
moreover
from  $\langle a \leq x \rangle \langle x = a \rangle$  IH(3)
have  $\{j. j < \text{length } (a \# xs) \wedge (a \# xs) ! j < x\} = \{\}$ 
  by (metis (no-types, lifting) Collect-empty-eq length-Cons less-nat-zero-code
    linorder-not-less nth-Cons-0 sorted-nth-less-mono
zero-less-Suc)
hence card  $\{j. j < \text{length } (a \# xs) \wedge (a \# xs) ! j < x\} = 0$ 
  by simp
moreover
from  $\langle a \leq x \rangle \langle x = a \rangle$  IH(3)
have  $\{j. j < \text{length } xs \wedge xs ! j < x\} = \{\}$ 
  using nth-mem by fastforce
hence card  $\{j. j < \text{length } xs \wedge xs ! j < x\} = 0$ 
  by simp
ultimately show ?thesis
  by simp
qed
moreover
have  $x \neq a \implies ?case$ 
proof -
  assume  $x \neq a$ 
hence Suc i < count-list xs x
  using IH(4) by force
with IH(2)[OF  $\langle x \neq a \rangle \langle \text{sorted } xs \rangle$ ]
have select xs x (Suc i) = card  $\{j. j < \text{length } xs \wedge xs ! j < x\} + \text{Suc } i$  .
moreover
from  $\langle x \neq a \rangle$  select.simps(3)[of a xs x i]
have select (a # xs) x (Suc i) = Suc (select xs x (Suc i))
  by simp
moreover
{
  have  $\{j. j < \text{length } (a \# xs) \wedge (a \# xs) ! j < x\} =$ 
     $\{0\} \cup \text{Suc } \{j. j < \text{length } xs \wedge xs ! j < x\}$ 
  proof safe
    show  $(a \# xs) ! 0 < x$ 
    using  $\langle a \leq x \rangle \langle x \neq a \rangle$  by auto
  next
    fix y
    assume  $y < \text{length } xs$   $xs ! y < x$ 
    then show Suc y < length (a # xs)
      by simp
  next
    fix y
    assume  $y < \text{length } xs$   $xs ! y < x$ 
    then show  $(a \# xs) ! \text{Suc } y < x$ 
      by simp

```

```

next
  fix  $k$ 
  assume  $A: k \notin \text{Suc } \{j. j < \text{length } xs \wedge xs ! j < x\} \ k \notin \{\} \ k < \text{length } (a$ 
#  $xs)$ 
       $(a \# xs) ! k < x$ 

  have  $\exists l. k = \text{Suc } l \implies \text{False}$ 
  proof -
    assume  $\exists l. k = \text{Suc } l$ 
    then obtain  $l$  where
       $k = \text{Suc } l$ 
    by blast
    hence  $l \notin \{j. j < \text{length } xs \wedge xs ! j < x\} \ l < \text{length } xs \ xs ! l < x$ 
    using  $A$  by simp-all
    then show  $\text{False}$ 
    by blast
  qed
  then show  $k = 0$ 
    using not0-implies-Suc by blast
  qed
  moreover
  have finite  $\{0\}$ 
    by blast
  moreover
  have finite  $(\text{Suc } \{j. j < \text{length } xs \wedge xs ! j < x\})$ 
    by simp
  moreover
  have  $\{0\} \cap \text{Suc } \{j. j < \text{length } xs \wedge xs ! j < x\} = \{\}$ 
    by blast
  ultimately have
     $\text{card } (\{j. j < \text{length } (a \# xs) \wedge (a \# xs) ! j < x\}) =$ 
       $\text{Suc } (\text{card } (\text{Suc } \{j. j < \text{length } xs \wedge xs ! j < x\}))$ 
    by simp
  hence  $\text{card } (\{j. j < \text{length } (a \# xs) \wedge (a \# xs) ! j < x\}) =$ 
       $\text{Suc } (\text{card } \{j. j < \text{length } xs \wedge xs ! j < x\})$ 
    by (simp add: card-image)
  }
  ultimately show ?thesis
    by simp
  qed
  ultimately show ?case
    by blast
  qed

corollary sorted-select-0-plus:
  assumes sorted xs
  and  $i < \text{count-list } xs \ x$ 
shows  $\text{select } xs \ x \ i = \text{select } xs \ x \ 0 + i$ 
  using assms(1) assms(2) sorted-select by fastforce

```

corollary *select-sorted-0*:
assumes *sorted xs*
and $0 < \text{count-list } xs \ x$
shows $\text{select } xs \ x \ 0 = \text{card } \{j. j < \text{length } xs \wedge xs \ ! \ j < x\}$
by (*simp add: assms(1) assms(2) sorted-select*)

end
theory *Rank-Select*
imports *Main*
Rank-Util
Select-Util
begin

8 Rank and Select Properties

8.1 Correctness of Rank and Select

Correctness theorem statements based on [1].

8.1.1 Rank Correctness

lemma *rank-spec*:
 $\text{rank } s \ x \ i = \text{count } (\text{mset } (\text{take } i \ s)) \ x$
by (*simp add: count-list-eq-count rank-def*)

8.1.2 Select Correctness

lemma *select-spec*:
 $\text{select } s \ x \ i = j$
 $\implies (j < \text{length } s \wedge \text{rank } s \ x \ j = i) \vee (j = \text{length } s \wedge \text{count-list } s \ x \leq i)$
by (*metis le-eq-less-or-eq rank-def select-length-imp-count-list-less select-max select-nth-alt*)

Theorem 3.9 from [3]: Correctness of Select

lemma *select-correct*:
 $\text{select } s \ x \ i \leq \text{length } s \wedge$
 $(\text{select } s \ x \ i < \text{length } s \implies \text{rank } s \ x \ (\text{select } s \ x \ i) = i) \wedge$
 $(\text{select } s \ x \ i = \text{length } s \implies \text{count-list } s \ x \leq i)$

proof –

have $\text{select } s \ x \ i \leq \text{length } s$
by (*simp add: select-max*)

moreover

have $\text{select } s \ x \ i < \text{length } s \implies \text{rank } s \ x \ (\text{select } s \ x \ i) = i$
by (*metis rank-def select-nth-alt*)

moreover

have $\text{select } s \ x \ i = \text{length } s \implies \text{count-list } s \ x \leq i$
by (*simp add: select-length-imp-count-list-less*)

ultimately show *?thesis*
 by *blast*
 qed

8.2 Rank and Select

lemma *rank-select*:

select xs x i < length xs \implies rank xs x (select xs x i) = i

proof –

let *?j = select xs x i*

assume *select xs x i < length xs*

with *select-spec[of xs x i ?j]*

show *rank xs x (select xs x i) = i*

by *auto*

qed

lemma *select-upper-bound*:

i < rank xs x j \implies select xs x i < length xs

proof (*induct xs arbitrary: i j*)

case *Nil*

then show *?case*

by (*simp add: rank-def*)

next

case (*Cons a xs i j*)

note *IH = this*

from *rank-Suc-ex[OF Cons.premis]*

obtain *n* where

j = Suc n

by *blast*

show *?case*

proof (*cases a = x*)

assume *a = x*

show *?thesis*

proof (*cases i*)

case *0*

then show *?thesis*

by (*simp add: ‹a = x›*)

next

case (*Suc m*)

with *rank-cons-same[of a xs n] ‹j = Suc n› IH(2) ‹a = x›*

have *m < rank xs x n*

by *force*

with *IH(1)*

have *select xs x m < length xs*

by *simp*

then show *?thesis*

```

    by (simp add: Suc ⟨a = x⟩)
  qed
next
  assume a ≠ x
  with Cons.prem1 rank-cons-diff[of a x xs n] ⟨j = Suc n⟩
  have i < rank xs x n
    by force
  with Cons.hyps
  have select xs x i < length xs
    by simp
  then show ?thesis
    by (metis ⟨a ≠ x⟩ length-Cons not-less-eq select-cons-neq)
  qed
qed

```

lemma *select-out-of-range*:
 assumes *count-list xs a ≤ i*
 and *mset xs = mset ys*
 shows *select ys a i = length ys*
 by (metis *assms count-list-perm leD rank-select rank-upper-bound select-nth select-spec*)

8.3 Sorted Properties

lemma *sorted-nth-gen*:
 assumes *sorted xs*
 and *card {k. k < length xs ∧ xs ! k < c} < length xs*
 and *count-list xs c > i*
 shows *xs ! (card {k. k < length xs ∧ xs ! k < c} + i) = c*
proof –
 from *sorted-select[OF assms(1,3)]*
 have *select xs c i = card {j. j < length xs ∧ xs ! j < c} + i*.
 with *select-nth[of xs c i]*
 show ?thesis
 by (metis *assms(3) rank-length select-upper-bound*)
 qed

lemma *sorted-nth-gen-alt*:
 assumes *sorted xs*
 and *card {k. k < length xs ∧ xs ! k < a} ≤ i*
 and *i < card {k. k < length xs ∧ xs ! k < a} + card {k. k < length xs ∧ xs ! k = a}*
 shows *xs ! i = a*
proof (*cases a ∈ set xs*)
 assume *a ∉ set xs*
 hence *card {k. k < length xs ∧ xs ! k = a} = 0*
 by auto
 with *assms(2-)*
 show ?thesis

```

    by linarith
next
  assume  $a \in \text{set } xs$ 

  have  $\text{card } \{k. k < \text{length } xs \wedge xs ! k < a\} < \text{length } xs$ 
    using  $\langle a \in \text{set } xs \rangle \text{ card-less-idx-upper-strict}$  by blast
  moreover
  have  $\exists k. i = \text{card } \{k. k < \text{length } xs \wedge xs ! k < a\} + k$ 
    using assms(2) le-iff-add by blast
  then obtain  $k$  where
     $i = \text{card } \{k. k < \text{length } xs \wedge xs ! k < a\} + k$ 
    by blast
  moreover
  have  $k < \text{count-list } xs \ a$ 
    by (metis (mono-tags, lifting) count-list-card nat-add-left-cancel-less assms(3)
calculation(2))
  ultimately show ?thesis
    using sorted-nth-gen[OF assms(1), of a k]
    by blast
qed

end
theory SA-Util
  imports SuffixArray.Suffix-Array-Properties
           SuffixArray.Simple-SACA-Verification
           ../counting/Rank-Select
begin

```

9 Suffix Array Properties

9.1 Bijections

lemma *bij-betw-empty*:

```

bij-betw  $f \ \{\} \ \{\}$ 
using bij-betwI' by fastforce

```

lemma *bij-betw-sort-idx-ex*:

```

assumes  $xs = \text{sort } ys$ 
shows  $\exists f. \text{bij-betw } f \ \{j. j < \text{length } ys \wedge ys ! j < x\} \ \{j. j < \text{length } xs \wedge xs ! j < x\}$ 

```

proof –

```

let  $?A = \{j. j < \text{length } ys \wedge ys ! j < x\}$ 
let  $?B = \{j. j < \text{length } xs \wedge xs ! j < x\}$ 

```

```

have  $\text{mset } ys = \text{mset } xs$ 
  by (simp add: assms)
with permutation-Ex-bij[of ys xs]
obtain  $f$  where

```



```

    bij-betw  $f \{..<length\ ys\} \{..<length\ xs\}$ 
     $(\forall i < length\ ys. ys\ !\ i = xs\ !\ f\ i)$ 
    by blast
  moreover
  have  $?A \subseteq \{..<length\ ys\}$ 
    by blast
  moreover
  have  $f\ ' \ ?A = ?B$ 
  proof safe
    fix  $a$ 
    assume  $a < length\ ys\ ys\ !\ a < x$ 
    then show  $f\ a < length\ xs$ 
      by (meson bij-betw-apply calculation(1) lessThan-iff)
    next
    fix  $a$ 
    assume  $a < length\ ys\ ys\ !\ a < x$ 
    then show  $xs\ !\ f\ a < x$ 
      by (simp add: calculation(2))
    next
    fix  $a$ 
    assume  $A: a < length\ xs\ xs\ !\ a < x$ 
    from bij-betw-iff-bijections[THEN iffD1, OF calculation(1)]
    obtain  $g$  where
       $\forall x \in \{..<length\ ys\}. f\ x \in \{..<length\ xs\} \wedge g\ (f\ x) = x$ 
       $\forall y \in \{..<length\ xs\}. g\ y \in \{..<length\ ys\} \wedge f\ (g\ y) = y$ 
      by blast
    then show  $a \in f\ ' \ ?A$ 
      by (metis (no-types, lifting) A calculation(2) imageI lessThan-iff mem-Collect-eq)
  qed
  ultimately show ?thesis
    using bij-betw-subset
    by blast
  qed

```

9.2 Suffix Properties

lemma *suffix-hd-set-eq*:

```

 $\{k. k < length\ s \wedge s\ !\ k = c\} = \{k. k < length\ s \wedge (\exists xs. suffix\ s\ k = c\ \#\ xs)\}$ 
  using suffix-cons-ex by fastforce

```

lemma *suffix-hd-set-less*:

```

 $\{k. k < length\ s \wedge s\ !\ k < c\} = \{k. k < length\ s \wedge suffix\ s\ k < [c]\}$ 
  using suffix-cons-ex by fastforce

```

lemma *select-nth-suffix-start1*:

```

  assumes  $i < card\ \{k. k < length\ s \wedge (\exists as. suffix\ s\ k = a\ \#\ as)\}$ 
  and  $xs = sort\ s$ 
  shows  $select\ xs\ a\ i = card\ \{k. k < length\ s \wedge suffix\ s\ k < [a]\} + i$ 
  proof -

```

```

let ?A = {k. k < length s ∧ (∃ as. suffix s k = a # as)}
let ?A' = {k. k < length s ∧ s ! k = a}

have ?A = ?A'
  using suffix-cons-Suc by fastforce
with assms(1)
have i < count-list s a
  by (simp add: count-list-card)
hence i < count-list xs a
  by (metis assms(2) count-list-perm mset-sort)
moreover
let ?B = {k. k < length s ∧ suffix s k < [a]}
let ?B' = {k. k < length s ∧ s ! k < a}
let ?B'' = {k. k < length xs ∧ xs ! k < a}
{
  have ?B = ?B'
    using suffix-cons-ex by fastforce
  moreover
  have card ?B' = card ?B''
    using bij-betw-sort-idx-ex[OF assms(2), of a] bij-betw-same-card
    by blast
  ultimately have card ?B = card ?B''
    by presburger
}
ultimately show ?thesis
  using sorted-select assms(2) by force
qed

lemma select-nth-suffix-start2:
  assumes card {k. k < length s ∧ (∃ as. suffix s k = a # as)} ≤ i
  and xs = sort s
shows select xs a i = length xs
proof (rule select-out-of-range[of s])
  show mset s = mset xs
    by (simp add: assms(2))
next
  let ?A = {k. k < length s ∧ (∃ as. suffix s k = a # as)}
  let ?A' = {k. k < length s ∧ s ! k = a}
  have ?A = ?A'
    using suffix-cons-Suc by fastforce
  with assms(1)
  show count-list s a ≤ i
    by (simp add: count-list-card)
qed

context Suffix-Array-General begin

```

9.3 General Properties

lemma *sa-subset-upt*:
 $set (sa\ s) \subseteq \{0..< length\ s\}$
by (*simp add: sa-set-upt*)

lemma *sa-suffix-sorted*:
 $sorted (map (suffix\ s) (sa\ s))$
using *sa-g-sorted strict-sorted-imp-sorted* **by** *blast*

9.4 Nth Properties

lemma *sa-nth-suc-le*:
assumes $j < length\ s$
and $i < j$
and $s ! (sa\ s ! i) = s ! (sa\ s ! j)$
and $Suc (sa\ s ! i) < length\ s$
and $Suc (sa\ s ! j) < length\ s$
shows $s ! Suc (sa\ s ! i) \leq s ! (Suc (sa\ s ! j))$
proof –
from *sorted-wrt-nth-less[OF sa-g-sorted[of s] assms(2)] assms(1,2)*
have $suffix\ s (sa\ s ! i) < suffix\ s (sa\ s ! j)$
using *sa-length* **by** *auto*
with *assms(3-)*
have $suffix\ s (Suc (sa\ s ! i)) < suffix\ s (Suc (sa\ s ! j))$
by (*metis Cons-less-Cons Cons-nth-drop-Suc Suc-lessD order-less-imp-not-less*)
then show *?thesis*
by (*metis Cons-less-Cons assms(4,5) dual-order.asym suffix-cons-Suc verit-comp-simplify1(3)*)
qed

lemma *sa-nth-suc-le-ex*:
assumes $j < length\ s$
and $i < j$
and $s ! (sa\ s ! i) = s ! (sa\ s ! j)$
and $Suc (sa\ s ! i) < length\ s$
and $Suc (sa\ s ! j) < length\ s$
shows $\exists k\ l. k < l \wedge sa\ s ! k = Suc (sa\ s ! i) \wedge sa\ s ! l = Suc (sa\ s ! j)$
proof –
from *sorted-wrt-nth-less[OF sa-g-sorted[of s] assms(2)] assms(1,2)*
have $suffix\ s (sa\ s ! i) < suffix\ s (sa\ s ! j)$
using *sa-length* **by** *auto*
with *assms(3-)*
have $suffix\ s (Suc (sa\ s ! i)) < suffix\ s (Suc (sa\ s ! j))$
by (*metis Cons-less-Cons Cons-nth-drop-Suc Suc-lessD order-less-imp-not-less*)
moreover
from *ex-sa-nth[OF assms(4)]*
obtain k **where**
 $k < length\ s$
 $sa\ s ! k = Suc (sa\ s ! i)$
by *blast*

moreover
from $ex\text{-}sa\text{-}nth[OF\ assms(5)]$
obtain l **where**
 $l < length\ s$
 $sa\ s\ !\ l = Suc\ (sa\ s\ !\ j)$
by $blast$
ultimately have $k < l$
using $sorted\text{-}nth\text{-}less\text{-}mono[OF\ strict\text{-}sorted\text{-}imp\text{-}sorted[OF\ sa\text{-}g\text{-}sorted[of\ s]]]$
by $(metis\ length\text{-}map\ not\text{-}less\text{-}iff\text{-}gr\text{-}or\text{-}eq\ nth\text{-}map\ sa\text{-}length)$
with $\langle sa\ s\ !\ k = \rightarrow \rangle \langle sa\ s\ !\ l = \rightarrow \rangle$
show $?thesis$
by $blast$
qed

lemma $sorted\text{-}map\text{-}nth\text{-}sa$:
 $sorted\ (map\ (nth\ s)\ (sa\ s))$
proof $(intro\ sorted\text{-}wrt\text{-}mapI)$
fix $i\ j$
assume $i < j\ j < length\ (sa\ s)$
hence $suffix\ s\ (sa\ s\ !\ i) < suffix\ s\ (sa\ s\ !\ j)$
using $sa\text{-}g\text{-}sorted\ sorted\text{-}wrt\text{-}mapD$ **by** $blast$
moreover
have $suffix\ s\ (sa\ s\ !\ i) = s\ !\ (sa\ s\ !\ i) \# suffix\ s\ (Suc\ (sa\ s\ !\ i))$
by $(metis\ \langle i < j \rangle \langle j < length\ (sa\ s) \rangle order.\text{strict}\text{-}trans\ sa\text{-}length\ sa\text{-}nth\text{-}ex\ suffix\text{-}cons\text{-}Suc)$
moreover
have $suffix\ s\ (sa\ s\ !\ j) = s\ !\ (sa\ s\ !\ j) \# suffix\ s\ (Suc\ (sa\ s\ !\ j))$
by $(metis\ \langle j < length\ (sa\ s) \rangle sa\text{-}length\ sa\text{-}nth\text{-}ex\ suffix\text{-}cons\text{-}Suc)$
ultimately show $s\ !\ (sa\ s\ !\ i) \leq s\ !\ (sa\ s\ !\ j)$
by $fastforce$
qed

lemma $perm\text{-}map\text{-}nth\text{-}sa$:
 $s < \sim \sim > map\ (nth\ s)\ (sa\ s)$
by $(metis\ map\text{-}nth\ mset\text{-}map\ sa\text{-}g\text{-}permutation)$

lemma $sort\text{-}eq\text{-}map\text{-}nth\text{-}sa$:
 $sort\ s = map\ (nth\ s)\ (sa\ s)$
by $(metis\ perm\text{-}map\text{-}nth\text{-}sa\ properties\text{-}for\text{-}sort\ sorted\text{-}map\text{-}nth\text{-}sa)$

lemma $sort\text{-}sa\text{-}nth$:
 $i < length\ s \implies sort\ s\ !\ i = s\ !\ (sa\ s\ !\ i)$
by $(simp\ add:\ sa\text{-}length\ sort\text{-}eq\text{-}map\text{-}nth\text{-}sa)$

lemma $inj\text{-}on\text{-}nth\text{-}sa\text{-}upt$:
assumes $j \leq length\ s\ l \leq length\ s$
shows $inj\text{-}on\ (nth\ (sa\ s))\ (\{i..<j\} \cup \{k..<l\})$
proof
fix $x\ y$

```

assume  $x \in \{i..<j\} \cup \{k..<l\}$   $y \in \{i..<j\} \cup \{k..<l\}$   $sa\ s !\ x = sa\ s !\ y$ 

have  $x < length\ s$ 
  using  $\langle x \in \{i..<j\} \cup \{k..<l\} \rangle$  assms(1) assms(2) by auto
moreover
have  $y < length\ s$ 
  using  $\langle y \in \{i..<j\} \cup \{k..<l\} \rangle$  assms(1) assms(2) by auto
ultimately show  $x = y$ 
  by (metis  $\langle sa\ s !\ x = sa\ s !\ y \rangle$  nth-eq-iff-index-eq sa-distinct sa-length)
qed

```

```

lemma nth-sa-upt-set:
   $nth\ (sa\ s)\ '\{0..<length\ s\} = \{0..<length\ s\}$ 
proof safe
  fix  $x$ 
  assume  $x \in \{0..<length\ s\}$ 
  then show  $sa\ s !\ x \in \{0..<length\ s\}$ 
    using sa-nth-ex by force
next
  fix  $x$ 
  assume  $x \in \{0..<length\ s\}$ 
  then show  $x \in (!)\ (sa\ s)\ '\{0..<length\ s\}$ 
    by (metis ex-sa-nth image-iff in-set-conv-nth sa-length sa-set-upt)
qed

```

9.5 Valid List Properties

```

lemma valid-list-sa-hd:
  assumes valid-list s
  shows  $\exists n. length\ s = Suc\ n \wedge sa\ s !\ 0 = n$ 
proof -
  from valid-list-ex-def[THEN iffD1, OF assms]
  obtain  $xs$  where
     $s = xs\ @\ [bot]$ 
    by blast
  hence valid-list  $(xs\ @\ [bot])$ 
    using assms by simp
  with valid-list-bot-min[of xs sa, OF - sa-g-permutation sa-g-sorted]
  obtain  $ys$  where
     $sa\ (xs\ @\ [bot]) = length\ xs\ \#\ ys$ 
    by blast
  with  $\langle s = xs\ @\ [bot] \rangle$ 
  show ?thesis
    by simp
qed

```

```

lemma valid-list-not-last:
  assumes valid-list s
  and  $i < length\ s$ 

```

```

  and   j < length s
  and   i ≠ j
  and   s ! i = s ! j
shows i < length s - 1 ∧ j < length s - 1
  by (metis One-nat-def Suc-pred assms hd-drop-conv-nth last-suffix-index less-Suc-eq
      valid-list-length)

```

end

```

lemma Suffix-Array-General-ex:
  ∃ sa. Suffix-Array-General sa
  using simple-saca.Suffix-Array-General-axioms by auto

```

end

```

theory SA-Count
  imports Rank-Select
          ../util/SA-Util
begin

```

10 Counting Properties on Suffix Arrays

```
context Suffix-Array-General begin
```

10.1 Counting Properties

```

lemma sa-card-index:
  assumes i < length s
  shows i = card {j. j < length s ∧ suffix s (sa s ! j) < suffix s (sa s ! i)}
        (is i = card ?A)
proof -
  let ?P = λj. j < length s ∧ suffix s (sa s ! j) < suffix s (sa s ! i)
  have P: ∀ j < i. ?P j
  proof (safe)
    fix j
    assume j < i
    with assms
    show j < length s
      by simp
  next
    fix j
    assume j < i
    with sorted-wrt-nth-less[OF sa-g-sorted[of s] ⟨j < i⟩] assms
    show suffix s (sa s ! j) < suffix s (sa s ! i)
      using assms sa-length by auto
  qed

  have ?A = {j. j < i}
  proof (safe)
    fix x

```

```

    assume  $x < i$ 
    then show  $x < \text{length } s$ 
      using assms by simp
  next
  fix  $x$ 
  assume  $x < i$ 
  then show  $\text{suffix } s (sa\ s\ !\ x) < \text{suffix } s (sa\ s\ !\ i)$ 
    using  $P$  by auto
  next
  fix  $x$ 
  assume  $Q: x < \text{length } s \ \text{suffix } s (sa\ s\ !\ x) < \text{suffix } s (sa\ s\ !\ i)$ 
  hence  $x \neq i$ 
    by blast
  with sorted-nth-less-mono[OF strict-sorted-imp-sorted[OF sa-g-sorted],
    simplified length-map sa-length,
    OF Q(1) assms]
     $Q$  assms
  show  $x < i$ 
    by (simp add: sa-length)
  qed
  then show ?thesis
    using card-Collect-less-nat by presburger
  qed

corollary sa-card-s-index:
  assumes  $i < \text{length } s$ 
  shows  $i = \text{card } \{j. j < \text{length } s \wedge \text{suffix } s\ j < \text{suffix } s (sa\ s\ !\ i)\}$ 
    (is  $i = \text{card } ?A$ )
proof –
  let  $?i = sa\ s\ !\ i$ 
  let  $?v = s\ !\ ?i$ 
  let  $?B = \{j. j < \text{length } s \wedge \text{suffix } s (sa\ s\ !\ j) < \text{suffix } s\ ?i\}$ 

  from sa-card-index[OF assms]
  have  $i = \text{card } ?B$  .
  moreover
  have bij-betw  $(\lambda x. sa\ s\ !\ x)\ ?B\ ?A$ 
  proof (intro bij-betwI'; safe)
    fix  $x\ y$ 
    assume  $x < \text{length } s\ y < \text{length } s\ sa\ s\ !\ x = sa\ s\ !\ y$ 
    then show  $x = y$ 
      by (simp add: nth-eq-iff-index-eq sa-distinct sa-length)
  next
  fix  $x$ 
  assume  $x < \text{length } s$ 
  then show  $sa\ s\ !\ x < \text{length } s$ 
    using sa-nth-ex by fastforce
  next
  fix  $x$ 

```

assume $x < \text{length } s \text{ suffix } s \ x < \text{suffix } s \ ?i$
then show $\exists y \in ?B. x = sa \ s \ ! \ y$
using *ex-sa-nth* **by** *blast*
qed
hence $\text{card } ?B = \text{card } ?A$
using *bij-betw-same-card* **by** *blast*
ultimately show *?thesis*
by *simp*
qed

lemma *sa-card-s-idx*:
assumes $i < \text{length } s$
shows $i = \text{card } \{j. j < \text{length } s \wedge s \ ! \ j < s \ ! \ (sa \ s \ ! \ i)\} +$
 $\text{card } \{j. j < \text{length } s \wedge s \ ! \ j = s \ ! \ (sa \ s \ ! \ i) \wedge \text{suffix } s \ j < \text{suffix } s \ (sa \ s \ !$
 $i)\}$
proof –
let $?i = sa \ s \ ! \ i$
let $?v = s \ ! \ ?i$
let $?A = \{j. j < \text{length } s \wedge s \ ! \ j < ?v\}$
let $?B = \{j. j < \text{length } s \wedge s \ ! \ j = ?v \wedge \text{suffix } s \ j < \text{suffix } s \ ?i\}$
let $?C = \{j. j < \text{length } s \wedge \text{suffix } s \ j < \text{suffix } s \ ?i\}$

from *sa-card-s-index[OF assms]*
have $i = \text{card } ?C$
by *simp*
moreover
have $?A \cap ?B = \{\}$
by *fastforce*
moreover
have $?C = ?A \cup ?B$
proof (*safe*)
fix x
assume $x < \text{length } s \ \text{suffix } s \ x < \text{suffix } s \ ?i \ \neg s \ ! \ x < s \ ! \ ?i$
then show $s \ ! \ x = s \ ! \ ?i$
by (*metis Cons-less-Cons sa-nth-ex assms suffix-cons-Suc*)
next
fix x
assume $x < \text{length } s \ s \ ! \ x < s \ ! \ ?i$
then show $\text{suffix } s \ x < \text{suffix } s \ ?i$
by (*metis Cons-less-Cons sa-nth-ex assms suffix-cons-Suc*)
qed
ultimately show *?thesis*
by (*simp add: card-Un-disjoint*)
qed

lemma *sa-card-index-lower-bound*:
assumes $i < \text{length } s$
shows $\text{card } \{j. j < \text{length } s \wedge s \ ! \ (sa \ s \ ! \ j) < s \ ! \ (sa \ s \ ! \ i)\} \leq i$
(is $\text{card } ?A \leq i$ **)**

proof –
let $?B = \{j. j < \text{length } s \wedge \text{suffix } s (sa\ s\ !\ j) < \text{suffix } s (sa\ s\ !\ i)\}$
have $?A \subseteq ?B$
proof safe
fix x
assume $x < \text{length } s\ s\ !\ (sa\ s\ !\ x) < s\ !\ (sa\ s\ !\ i)$
then show $\text{suffix } s (sa\ s\ !\ x) < \text{suffix } s (sa\ s\ !\ i)$
by (*metis Cons-less-Cons Cons-nth-drop-Suc assms sa-nth-ex*)
qed
hence $\text{card } ?A \leq \text{card } ?B$
by (*simp add: card-mono*)
then show *?thesis*
using *sa-card-index[OF assms]* **by** *simp*
qed

lemma *sa-card-rank-idx*:
assumes $i < \text{length } s$
shows $i = \text{card } \{j. j < \text{length } s \wedge s\ !\ (sa\ s\ !\ j) < s\ !\ (sa\ s\ !\ i)\}$
 $+ \text{rank } (\text{sort } s) (s\ !\ (sa\ s\ !\ i))\ i$
proof –
from *sorted-card-rank-idx[of sort s i]*
have $i = \text{card } \{j. j < \text{length } (\text{sort } s) \wedge \text{sort } s\ !\ j < \text{sort } s\ !\ i\} + \text{rank } (\text{sort } s)$
 $(\text{sort } s\ !\ i)\ i$
using *assms* **by** *fastforce*
moreover
have $\text{sort } s\ !\ i = s\ !\ (sa\ s\ !\ i)$
using *assms sort-sa-nth* **by** *auto*
moreover
have $\text{length } (\text{sort } s) = \text{length } s$
by *simp*
ultimately show *?thesis*
using *sort-sa-nth[of -s]*
by (*metis (no-types, lifting) Collect-cong*)
qed

corollary *sa-card-rank-s-idx*:
assumes $i < \text{length } s$
shows $i = \text{card } \{j. j < \text{length } s \wedge s\ !\ j < s\ !\ (sa\ s\ !\ i)\}$
 $+ \text{rank } (\text{sort } s) (s\ !\ (sa\ s\ !\ i))\ i$
proof –
let $?A = \{j. j < \text{length } s \wedge s\ !\ j < s\ !\ (sa\ s\ !\ i)\}$
and $?B = \{j. j < \text{length } s \wedge s\ !\ (sa\ s\ !\ j) < s\ !\ (sa\ s\ !\ i)\}$
from *sa-card-rank-idx[OF assms]*
have $i = \text{card } \{j. j < \text{length } s \wedge s\ !\ (sa\ s\ !\ j) < s\ !\ (sa\ s\ !\ i)\} +$
 $\text{rank } (\text{sort } s) (s\ !\ (sa\ s\ !\ i))\ i .$
moreover
have *bij-betw* $(\lambda x. sa\ s\ !\ x)$
 $\{j. j < \text{length } s \wedge s\ !\ (sa\ s\ !\ j) < s\ !\ (sa\ s\ !\ i)\}$
 $\{j. j < \text{length } s \wedge s\ !\ j < s\ !\ (sa\ s\ !\ i)\}$

proof (*rule bij-betwI'*; *safe*)
fix $x\ y$
assume $x < \text{length } s\ y < \text{length } s\ sa\ s!\ x = sa\ s!\ y$
then show $x = y$
by (*simp add: nth-eq-iff-index-eq sa-distinct sa-length*)
next
fix x
assume $x < \text{length } s$
then show $sa\ s!\ x < \text{length } s$
using *sa-nth-ex* **by** *auto*
next
fix x
assume $x < \text{length } s\ s!\ x < s!\ (sa\ s!\ i)$
then show $\exists xa \in \{j. j < \text{length } s \wedge s!\ (sa\ s!\ j) < s!\ (sa\ s!\ i)\}. x = sa\ s!\$
 xa
using *ex-sa-nth* **by** *blast*
qed
hence $\text{card } ?B = \text{card } ?A$
using *bij-betw-same-card* **by** *blast*
ultimately show *?thesis*
by *simp*
qed

lemma *sa-rank-nth*:

assumes $i < \text{length } s$
shows $\text{rank } (\text{sort } s)\ (s!\ (sa\ s!\ i))\ i =$
 $\text{card } \{j. j < \text{length } s \wedge s!\ j = s!\ (sa\ s!\ i) \wedge$
 $\text{suffix } s\ j < \text{suffix } s\ (sa\ s!\ i)\}$

proof –

let $?i = sa\ s!\ i$
let $?v = s!\ ?i$
let $?A = \{j. j < \text{length } s \wedge s!\ j < ?v\}$
let $?B = \{j. j < \text{length } s \wedge s!\ j = ?v \wedge \text{suffix } s\ j < \text{suffix } s\ ?i\}$

from *sa-card-rank-s-idx[OF assms]*
have $i = \text{card } ?A + \text{rank } (\text{sort } s)\ ?v\ i$.
moreover
from *sa-card-s-idx[OF assms]*
have $i = \text{card } ?A + \text{card } ?B$.
ultimately show *?thesis*
by *linarith*

qed

lemma *sa-suffix-nth*:

assumes $\text{card } \{k. k < \text{length } s \wedge s!\ k < c\} + i < \text{length } s$
and $i < \text{count-list } s\ c$
shows $\exists as. \text{suffix } s\ (sa\ s!\ (\text{card } \{k. k < \text{length } s \wedge s!\ k < c\} + i)) = c \# as$
proof –
let $?A = \{k. k < \text{length } s \wedge s!\ k < c\}$

```

let ?i = card ?A
let ?A' = {k. k < length (sort s) ∧ (sort s) ! k < c}

have ∃ as. suffix s (sa s ! (?i + i)) = (s ! (sa s ! (?i + i))) # as
  using assms sa-nth-ex suffix-cons-ex by blast
moreover
have s ! (sa s ! (?i + i)) = sort s ! (?i + i)
  using assms(1) sort-sa-nth by presburger
moreover
{
  have i < count-list (sort s) c
    by (metis assms(2) count-list-perm sort-perm)
  moreover
  have card ?A = card ?A'
  proof –
    have ∃ f. bij-betw f {n. n < length s ∧ s ! n < c} {n. n < length (sort s) ∧
sort s ! n < c}
      using bij-betw-sort-idx-ex by blast
      then show ?thesis
      using bij-betw-same-card by blast
    qed
    ultimately have sort s ! (?i + i) = c
      using sorted-nth-gen[of sort s c i] assms(1) by auto
  }
ultimately show ?thesis
  by force
qed

```

10.2 Ordering Properties

lemma *sa-suffix-order-le*:

assumes card {k. k < length s ∧ s ! k < c} < length s

shows [c] ≤ suffix s (sa s ! (card {k. k < length s ∧ s ! k < c}))

proof –

let ?A = {k. k < length s ∧ s ! k < c}

let ?A' = {k. k < length (sort s) ∧ (sort s) ! k < c}

let ?i = card ?A

let ?i' = card ?A'

have ∃ as. suffix s (sa s ! ?i) = (s ! (sa s ! ?i)) # as

using *assms sa-nth-ex suffix-cons-ex* **by** *blast*

then obtain as **where**

suffix s (sa s ! ?i) = (s ! (sa s ! ?i)) # as

by *blast*

moreover

from *sort-sa-nth[of ?i s]*

have sort s ! ?i = s ! (sa s ! ?i)

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

moreover

have $?i = ?i'$
proof –
have $\exists f. \text{bij-betw } f \{n. n < \text{length } s \wedge s ! n < c\} \{n. n < \text{length } (\text{sort } s) \wedge \text{sort } s ! n < c\}$
using *bij-betw-sort-idx-ex* **by** *blast*
then show *?thesis*
using *bij-betw-same-card* **by** *blast*
qed
hence $c \leq \text{sort } s ! ?i$
using *sorted-nth-le*[of *sort s c*] *assms* **by** *auto*
ultimately show *?thesis*
by *fastforce*
qed

lemma *sa-suffix-order-le-gen*:
assumes $\text{card } \{k. k < \text{length } s \wedge s ! k < c\} + i < \text{length } s$
shows $[c] \leq \text{suffix } s (sa \ s ! (\text{card } \{k. k < \text{length } s \wedge s ! k < c\} + i))$
proof (*cases i*)
case 0
then show *?thesis*
using *assms sa-suffix-order-le* **by** *auto*
next
let $?x = \text{card } \{k. k < \text{length } s \wedge s ! k < c\}$
case (*Suc m*)
with *sorted-wrt-mapD*[*OF sa-g-sorted*, of $?x \ ?x + i \ s$]
have $\text{suffix } s (sa \ s ! ?x) < \text{suffix } s (sa \ s ! (?x + i))$
using *assms sa-length* **by** *auto*
moreover
have $[c] \leq \text{suffix } s (sa \ s ! ?x)$
using *add-lessD1 assms sa-suffix-order-le* **by** *blast*
ultimately show *?thesis*
by *order*
qed

lemma *sa-suffix-nth-less*:
assumes $i < \text{card } \{k. k < \text{length } s \wedge s ! k < c\}$
shows $\forall as. \text{suffix } s (sa \ s ! i) < c \ \# \ as$
proof –
have $i < \text{length } s$
using *assms card-less-idx-upper dual-order.strict-trans1* **by** *blast*
hence $\exists as. \text{suffix } s (sa \ s ! i) = s ! (sa \ s ! i) \ \# \ as$
using *sa-nth-ex suffix-cons-Suc* **by** *blast*
moreover
have $i < \text{card } \{k. k < \text{length } (\text{sort } s) \wedge (\text{sort } s) ! k < c\}$
using *bij-betw-sort-idx-ex*[of *sort s s c*] *assms bij-betw-same-card* **by** *force*
with *sorted-nth-less-gen*[of *sort s i c*]
have $s ! (sa \ s ! i) < c$
using *sorted-nth-less-gen*[of *sort s i c*] $\langle i < \text{length } s \rangle$ *sort-sa-nth* **by** *force*
ultimately show *?thesis*

```

    by fastforce
qed

lemma sa-suffix-nth-gr:
  assumes card {k. k < length s ∧ s ! k < c} + i < length s
  and count-list s c ≤ i
shows ∀ as. c ≠ as < suffix s (sa s ! (card {k. k < length s ∧ s ! k < c} + i))
proof -
  let ?x = card {k. k < length s ∧ s ! k < c}
  let ?i = ?x + i
  let ?y = card {k. k < length (sort s) ∧ sort s ! k < c}
  have ∃ as. suffix s (sa s ! ?i) = s ! (sa s ! ?i) ≠ as
    using assms(1) sa-nth-ex suffix-cons-Suc by blast
  moreover
  {
    have ?y = ?x
      using bij-betw-sort-idx-ex[of sort s s c] bij-betw-same-card by force
    moreover
    have ?y + i < length (sort s)
      using assms(1) calculation(1) by auto
    moreover
    have count-list (sort s) c ≤ i
      by (metis assms(2) count-list-perm mset-sort)
    ultimately have s ! (sa s ! ?i) > c
      using sorted-nth-gr-gen[of sort s c i] sort-sa-nth by fastforce
  }
  ultimately show ?thesis
    by fastforce
qed

end

end
theory BWT
  imports ../util/SA-Util

begin

```

11 Burrows-Wheeler Transform

Based on [2]

Definition 3.3 from [3]: Canonical BWT

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

bwt-canon s = map last (sort (map (λx. rotate x s) [0..<length s]))

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

Definition 3.4 from [3]: Suffix Array Version of the BWT

definition *bwt-sa* :: ('a :: {linorder, order-bot}) list ⇒ 'a list
where
bwt-sa s = map (λi. s ! ((i + length s - Suc 0) mod (length s))) (sa s)
end

12 BWT Verification

12.1 List Rotations

lemma *rotate-suffix-prefix*:
assumes $i < \text{length } xs$
shows $\text{rotate } i \ xs = \text{suffix } xs \ i \ @ \ \text{prefix } xs \ i$
by (*simp add: assms rotate-drop-take*)

lemma *rotate-last*:
assumes $i < \text{length } xs$
shows $\text{last } (\text{rotate } i \ xs) = xs \ ! \ ((i + \text{length } xs - \text{Suc } 0) \ \text{mod} \ (\text{length } xs))$
by (*metis Nat.add-diff-assoc One-nat-def Suc-leI assms diff-less last-conv-nth length-greater-0-conv length-rotate list.size(3) not-less-zero nth-rotate zero-less-one*)

lemma (*in Suffix-Array-General*) *map-last-rotations*:

$\text{map last } (\text{map } (\lambda i. \text{rotate } i \ s) \ (sa \ s)) = \text{bwt-sa } s$

proof –

have $\forall x \in \text{set } (sa \ s). \ \text{last } (\text{rotate } x \ s) = s \ ! \ ((x + \text{length } s - \text{Suc } 0) \ \text{mod} \ \text{length } s)$

by (*meson atLeastLessThan-iff rotate-last sa-subset-upt subset-code(1)*)

then show *?thesis*

unfolding *bwt-sa-def* **by** *simp*

qed

lemma *distinct-rotations*:

assumes *valid-list* s

and $i < \text{length } s$

and $j < \text{length } s$

and $i \neq j$

shows $\text{rotate } i \ s \neq \text{rotate } j \ s$

proof –

from *rotate-suffix-prefix*[*OF assms(2)*]

rotate-suffix-prefix[*OF assms(3)*]

suffix-has-no-prefix-suffix[*OF assms, simplified*]

suffix-has-no-prefix-suffix[*OF assms(1,3,2) assms(4)[symmetric], simplified*]

show *?thesis*

by (*metis append-eq-append-conv2*)

qed

12.2 Ordering

lemma *list-less-suffix-app-prefix-1*:

assumes *valid-list xs*
and $i < \text{length } xs$
and $j < \text{length } xs$
and $\text{suffix } xs \ i < \text{suffix } xs \ j$

shows $\text{suffix } xs \ i \ @ \ \text{prefix } xs \ i < \text{suffix } xs \ j \ @ \ \text{prefix } xs \ j$

proof –

from *suffix-less-ex*[*OF assms*]

obtain $b \ c \ as \ bs \ cs$ **where**

$\text{suffix } xs \ i = as \ @ \ b \ \# \ bs$
 $\text{suffix } xs \ j = as \ @ \ c \ \# \ cs$
 $b < c$

by *blast*

hence $\text{suffix } xs \ i \ @ \ \text{prefix } xs \ i = as \ @ \ b \ \# \ bs \ @ \ \text{prefix } xs \ i$

$\text{suffix } xs \ j \ @ \ \text{prefix } xs \ j = as \ @ \ c \ \# \ cs \ @ \ \text{prefix } xs \ j$

by *simp-all*

with $\langle b < c \rangle$

show *?thesis*

by (*metis list-less-ex*)

qed

lemma *list-less-suffix-app-prefix-2*:

assumes *valid-list xs*

and $i < \text{length } xs$

and $j < \text{length } xs$

and $\text{suffix } xs \ i \ @ \ \text{prefix } xs \ i < \text{suffix } xs \ j \ @ \ \text{prefix } xs \ j$

shows $\text{suffix } xs \ i < \text{suffix } xs \ j$

by (*metis assms list-less-suffix-app-prefix-1 not-less-iff-gr-or-eq suffixes-neq*)

corollary *list-less-suffix-app-prefix*:

assumes *valid-list xs*

and $i < \text{length } xs$

and $j < \text{length } xs$

shows $\text{suffix } xs \ i < \text{suffix } xs \ j \longleftrightarrow$

$\text{suffix } xs \ i \ @ \ \text{prefix } xs \ i < \text{suffix } xs \ j \ @ \ \text{prefix } xs \ j$

using *assms list-less-suffix-app-prefix-1 list-less-suffix-app-prefix-2* **by** *blast*

Theorem 3.5 from [3]: Same Suffix and Rotation Order

lemma *list-less-suffix-rotate*:

assumes *valid-list xs*

and $i < \text{length } xs$

and $j < \text{length } xs$

shows $\text{suffix } xs \ i < \text{suffix } xs \ j \longleftrightarrow \text{rotate } i \ xs < \text{rotate } j \ xs$

by (*simp add: assms list-less-suffix-app-prefix rotate-suffix-prefix*)

lemma (in *Suffix-Array-General*) *sorted-rotations*:

assumes *valid-list s*

shows *strict-sorted* ($\text{map } (\lambda i. \text{rotate } i \ s) \ (sa \ s)$)

```

proof (intro sorted-wrt-mapI)
  fix i j
  assume i < j j < length (sa s)
  with sorted-wrt-nth-less[OF sa-g-sorted ⟨i < j⟩, simplified, OF ⟨j < -⟩]
  have suffix s (sa s ! i) < suffix s (sa s ! j)
    by force
  with list-less-suffix-rotate[THEN iffD1, OF assms, of sa s ! i sa s ! j]
  show rotate (sa s ! i) s < rotate (sa s ! j) s
    by (metis ⟨i < j⟩ ⟨j < length (sa s)⟩ dual-order.strict-trans sa-length sa-nth-ex)
qed

```

12.3 BWT Equivalence

Theorem 3.6 from [3]: BWT and Suffix Array Correspondence Canonical
 BWT and BWT via Suffix Array Correspondence

theorem (in *Suffix-Array-General*) *bwt-canon-eq-bwt-sa*:

assumes *valid-list s*
shows *bwt-canon s = bwt-sa s*

proof –

let *?xs = map (λx. rotate x s) [0..<length s]*

have *distinct ?xs*

by (intro distinct-conv-nth[THEN iffD2] allI impI; simp add: distinct-rotations[OF *assms*])

hence *strict-sorted (sort ?xs)*

using *distinct-sort sorted-sort strict-sorted-iff* **by** *blast*

hence *sort ?xs = map (λi. rotate i s) (sa s)*

using *sorted-rotations*[OF *assms*]

by (*simp add: strict-sorted-equal sa-set-upt*)

with *map-last-rotations*[of *s*]

have *map last (sort ?xs) = bwt-sa s*

by *presburger*

then show *?thesis*

by (*metis bwt-canon-def*)

qed

end

theory *BWT-SA-Corres*

imports *BWT*

../.. / counting / SA-Count

../.. / util / Rotated-Substring

begin

13 BWT and Suffix Array Correspondence

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

Definition 3.12 from [3]: BWT Permutation

definition $bwt\text{-perm} :: ('a :: \{linorder, order\text{-}bot\}) list \Rightarrow nat list$
where
 $bwt\text{-perm } s = map (\lambda i. (i + length\ s - Suc\ 0) \bmod (length\ s)) (sa\ s)$

13.1 BWT Using Suffix Arrays

lemma $map\text{-}bwt\text{-}indexes$:
fixes $s :: ('a :: \{linorder, order\text{-}bot\}) list$
shows $bwt\text{-}sa\ s = map (\lambda i. s\ !\ i) (bwt\text{-}perm\ s)$
by ($simp\ add: bwt\text{-}perm\text{-}def\ bwt\text{-}sa\text{-}def$)

lemma $map\text{-}bwt\text{-}indexes\text{-}perm$:
fixes $s :: ('a :: \{linorder, order\text{-}bot\}) list$
shows $bwt\text{-}perm\ s <\sim\sim> [0..<length\ s]$
proof ($intro\ distinct\text{-}set\text{-}imp\text{-}perm$)
show $distinct\ [0..<length\ s]$
by $simp$

next
show $set\ (bwt\text{-}perm\ s) = set\ [0..<length\ s]$
unfolding $bwt\text{-}perm\text{-}def$
proof $safe$
fix x
assume $x \in set\ (map\ (\lambda i. (i + length\ s - Suc\ 0) \bmod length\ s) (sa\ s))$
hence $x < length\ s$
by ($metis\ (no\text{-}types, lifting)\ ex\text{-}map\text{-}conv\ length\text{-}map\ length\text{-}pos\text{-}if\text{-}in\text{-}set\ mod\text{-}less\text{-}divisor$)

$sa\text{-}length$)

then show $x \in set\ [0..<length\ s]$
by $simp$

next
fix x
assume $x \in set\ [0..<length\ s]$
hence $x \in \{0..<length\ s\}$
using $atLeastLessThan\text{-}upt$ **by** $blast$

have $x \in (\lambda i. (i + length\ s - Suc\ 0) \bmod length\ s) \text{ ' } \{0..<length\ s\}$

proof ($cases\ Suc\ x < length\ s$)
assume $Suc\ x < length\ s$
hence $(\lambda i. (i + length\ s - Suc\ 0) \bmod length\ s) (Suc\ x) = x$
by $simp$
then show $?thesis$
using $\langle Suc\ x < length\ s \rangle$ **by** $force$

next
assume $\neg\ Suc\ x < length\ s$
with $\langle x \in \{0..<length\ s\} \rangle$
have $Suc\ x = length\ s$
by $simp$
hence $(\lambda i. (i + length\ s - Suc\ 0) \bmod length\ s) 0 = x$
using $diff\text{-}Suc\text{-}1'$ $lessI\ mod\text{-}less$ **by** $presburger$

```

then show ?thesis
  by (metis (mono-tags, lifting) ‹Suc x = length s› atLeastLessThan-iff imageI
zero-le
                                zero-less-Suc)

qed
then show  $x \in \text{set } (\text{map } (\lambda i. (i + \text{length } s - \text{Suc } 0) \bmod \text{length } s) (sa \ s))$ 
  by (simp add: sa-set-upt)
qed
next
show distinct (bwt-perm s)
proof (intro distinct-conv-nth[THEN iffD2] allI impI)
  fix i j
  assume A:  $i < \text{length } (bwt\text{-perm } s) \ j < \text{length } (bwt\text{-perm } s) \ i \neq j$ 

  have  $bwt\text{-perm } s ! i = (sa \ s ! i + \text{length } s - \text{Suc } 0) \bmod (\text{length } s)$ 
    using A(1) bwt-perm-def by force
  moreover
  have  $bwt\text{-perm } s ! j = (sa \ s ! j + \text{length } s - \text{Suc } 0) \bmod (\text{length } s)$ 
    using A(2) bwt-perm-def by force
  moreover
  have  $sa \ s ! i \neq sa \ s ! j$ 
    by (metis A bwt-perm-def length-map nth-eq-iff-index-eq sa-distinct)

  have  $(sa \ s ! i + \text{length } s - \text{Suc } 0) \bmod (\text{length } s) \neq$ 
     $(sa \ s ! j + \text{length } s - \text{Suc } 0) \bmod (\text{length } s)$ 
  proof (cases sa s ! i)
    case 0
    hence  $(sa \ s ! i + \text{length } s - \text{Suc } 0) \bmod (\text{length } s) = \text{length } s - \text{Suc } 0$ 
    by (metis diff-Suc-less gen-length-def length-code length-greater-0-conv
list.size(3)
                                mod-by-0 mod-less)
  moreover
  have  $\exists m. sa \ s ! j = \text{Suc } m$ 
    using 0 ‹sa s ! i ≠ sa s ! j› not0-implies-Suc by force
  then obtain m where
     $sa \ s ! j = \text{Suc } m$ 
    by blast
  hence  $(sa \ s ! j + \text{length } s - \text{Suc } 0) \bmod (\text{length } s) = m$ 
    using A(2) bwt-perm-def sa-length sa-nth-ex by force
  moreover
  have  $\text{Suc } m \leq \text{length } s - \text{Suc } 0$ 
    by (metis 0 A(1) A(2) Suc-pred ‹sa s ! j = Suc m› bwt-perm-def length-map
less-Suc-eq-le
                                sa-length sa-nth-ex)
  hence  $m < \text{length } s - \text{Suc } 0$ 
    using Suc-le-eq by blast
  ultimately show ?thesis
    by (metis not-less-iff-gr-or-eq)
next

```

```

case (Suc n)
assume sa s ! i = Suc n
hence B: (sa s ! i + length s - Suc 0) mod (length s) = n
  using A(1) bwt-perm-def sa-length sa-nth-ex by force
show ?thesis
proof (cases sa s ! j)
  case 0
  hence (sa s ! j + length s - Suc 0) mod (length s) = length s - Suc 0
    by (metis add-eq-if diff-Suc-less length-greater-0-conv list.size(3) mod-by-0
mod-less)
  moreover
  have Suc n ≤ length s - Suc 0
    by (metis 0 A(1,2) Suc Suc-pred bwt-perm-def length-map less-Suc-eq-le
sa-length
sa-nth-ex)
  hence n < length s - Suc 0
    using Suc-le-eq by blast
  ultimately show ?thesis
    by (simp add: B)
next
case (Suc m)
hence (sa s ! j + length s - Suc 0) mod (length s) = m
  using A(2) add-Suc bwt-perm-def sa-length sa-nth-ex by force
moreover
have m ≠ n
  using Suc ⟨sa s ! i = Suc n⟩ ⟨sa s ! i ≠ sa s ! j⟩ by auto
  ultimately show ?thesis
    using B by presburger
qed
qed
ultimately show bwt-perm s ! i ≠ bwt-perm s ! j
  by presburger
qed
qed

```

lemma *bwt-sa-perm*:
fixes s :: ('a :: {linorder, order-bot}) list
shows bwt-sa s <~> s
by (metis map-bwt-indices-perm map-bwt-indices map-nth mset-map)

lemma *bwt-sa-nth*:
fixes s :: ('a :: {linorder, order-bot}) list
fixes i :: nat
assumes i < length s
shows bwt-sa s ! i = s ! (((sa s ! i) + length s - 1) mod (length s))
using assms sa-length bwt-sa-def **by** force

lemma *bwt-perm-nth*:
fixes s :: ('a :: {linorder, order-bot}) list

fixes $i :: \text{nat}$
assumes $i < \text{length } s$
shows $\text{bwt-perm } s ! i = ((\text{sa } s ! i) + \text{length } s - 1) \bmod (\text{length } s)$
using *assms sa-length bwt-perm-def* **by force**

lemma *bwt-perm-s-nth*:
fixes $s :: ('a :: \{\text{linorder}, \text{order-bot}\}) \text{list}$
fixes $i :: \text{nat}$
assumes $i < \text{length } s$
shows $\text{bwt-sa } s ! i = s ! (\text{bwt-perm } s ! i)$
using *assms bwt-perm-nth bwt-sa-nth* **by presburger**

lemma *bwt-sa-length*:
fixes $s :: ('a :: \{\text{linorder}, \text{order-bot}\}) \text{list}$
shows $\text{length } (\text{bwt-sa } s) = \text{length } s$
using *sa-length bwt-sa-def* **by force**

lemma *bwt-perm-length*:
fixes $s :: ('a :: \{\text{linorder}, \text{order-bot}\}) \text{list}$
shows $\text{length } (\text{bwt-perm } s) = \text{length } s$
using *sa-length bwt-perm-def* **by force**

lemma *ex-bwt-perm-nth*:
fixes $s :: ('a :: \{\text{linorder}, \text{order-bot}\}) \text{list}$
fixes $k :: \text{nat}$
assumes $k < \text{length } s$
shows $\exists i < \text{length } s. \text{bwt-perm } s ! i = k$
using *assms ex-perm-nth map-bwt-indexes-perm* **by blast**

lemma *valid-list-sa-index-helper*:
fixes $s :: ('a :: \{\text{linorder}, \text{order-bot}\}) \text{list}$
fixes $i j :: \text{nat}$
assumes *valid-list s*
and $i < \text{length } s$
and $j < \text{length } s$
and $i \neq j$
and $s ! (\text{bwt-perm } s ! i) = s ! (\text{bwt-perm } s ! j)$

shows $\text{sa } s ! i \neq 0$
proof (*rule ccontr*)
assume $\neg \text{sa } s ! i \neq 0$
hence $\text{sa } s ! i = 0$
by *clarsimp*

from *valid-list-length-ex[OF assms(1)]*
obtain n **where**
 $\text{length } s = \text{Suc } n$
by *blast*

```

let ?i = (sa s ! i + length s - 1) mod length s
and ?j = (sa s ! j + length s - 1) mod length s

from bwt-perm-nth[OF assms(2)]
have bwt-perm s ! i = ?i .
moreover
from bwt-perm-nth[OF assms(3)]
have bwt-perm s ! j = ?j .
moreover
have ?i = n
  by (simp add: ⟨length s = Suc n⟩ ⟨sa s ! i = 0⟩)
hence s ! ?i = bot
  by (metis One-nat-def ⟨length s = Suc n⟩ assms(1) diff-Suc-Suc diff-zero
last-conv-nth
      list.size(3) nat.distinct(1) valid-list-def)
moreover
have ∃ k. sa s ! j = Suc k
  by (metis ⟨length s = Suc n⟩ ⟨sa s ! i = 0⟩ assms(2-4) less-Suc-eq-0-disj
nth-eq-iff-index-eq
      sa-distinct sa-length sa-nth-ex)
then obtain k where
  sa s ! j = Suc k
  by blast
hence ?j = k ∧ k < n
  by (metis ⟨length s = Suc n⟩ add-Suc-right add-Suc-shift add-diff-cancel-left'
assms(3)
      dual-order.strict-trans lessI mod-add-self2 mod-less not-less-eq plus-1-eq-Suc
      sa-nth-ex)
hence s ! ?j ≠ bot
  by (metis ⟨length s = Suc n⟩ assms(1) diff-Suc-1 valid-list-def)
ultimately show False
  by (metis assms(5))
qed

```

Theorem 3.13 from [3]: Suffix Relative Order Preservation Relative order of the suffixes is maintained by the BWT permutation

```

lemma bwt-relative-order:
  fixes s :: ('a :: {linorder, order-bot}) list
  fixes i j :: nat
  assumes valid-list s
  and i < j
  and j < length s
  and s ! (bwt-perm s ! i) = s ! (bwt-perm s ! j)
shows suffix s (bwt-perm s ! i) < suffix s (bwt-perm s ! j)
proof –
  from valid-list-length-ex[OF assms(1)]
  obtain n where
    length s = Suc n
  by blast

```

```

let ?i = (sa s ! i + length s - 1) mod length s
and ?j = (sa s ! j + length s - 1) mod length s

from bwt-perm-nth[of i s] assms(2-3)
have bwt-perm s ! i = ?i
  using dual-order.strict-trans by blast
moreover
from bwt-perm-nth[OF assms(3)]
have bwt-perm s ! j = ?j .
moreover
from sorted-wrt-nth-less[OF sa-g-sorted assms(2)] assms(2,3)
have suffix s (sa s ! i) < suffix s (sa s ! j)
  using sa-length by force
moreover
have ∃ k. sa s ! i = Suc k
  using valid-list-sa-index-helper[OF assms(1) - assms(3) - assms(4)] assms(2,3)
    dual-order.strict-trans not0-implies-Suc by blast
then obtain k where
  sa s ! i = Suc k
  by blast
moreover
from calculation(4)
have ?i = k
  by (metis Suc-lessD add.assoc assms(2,3) diff-Suc-1 dual-order.strict-trans
    mod-add-self2
    mod-less plus-1-eq-Suc sa-nth-ex)
moreover
have ∃ l. sa s ! j = Suc l
  using valid-list-sa-index-helper[OF assms(1) assms(3) - - assms(4)][symmetric]]
  assms(2,3)
    dual-order.strict-trans not0-implies-Suc by blast
then obtain l where
  sa s ! j = Suc l
  by blast
moreover
from calculation(6)
have ?j = l
  using assms(3) sa-nth-ex by force
ultimately show ?thesis
  by (metis Cons-less-Cons Cons-nth-drop-Suc assms(1,4) mod-less-divisor valid-list-length)
qed

```

```

lemma bwt-sa-card-s-idx:
  fixes s :: ('a :: {linorder, order-bot}) list
  fixes i :: nat
  assumes valid-list s
  and i < length s
  shows i = card {j. j < length s ∧ j < i ∧ bwt-sa s ! j ≠ bwt-sa s ! i} +

```

$$\text{card } \{j. j < \text{length } s \wedge s ! j = \text{bwt-sa } s ! i \wedge \text{suffix } s j < \text{suffix } s (\text{bwt-perm } s ! i)\}$$

proof –

let $?bwt = \text{bwt-sa } s$
let $?idx = \text{bwt-perm } s$
let $?i = ?idx ! i$
let $?v = ?bwt ! i$
let $?A = \{j. j < \text{length } s \wedge j < i \wedge ?bwt ! j \neq ?v\}$
let $?B = \{j. j < \text{length } s \wedge s ! j = ?v \wedge \text{suffix } s j < \text{suffix } s ?i\}$
let $?C = \{j. j < \text{length } s \wedge j < i \wedge ?bwt ! j = ?v\}$

have $P: \bigwedge x. \llbracket x < i; \neg x < \text{length } s \rrbracket \implies \text{False}$
using $\text{assms}(2)$ *dual-order.strict-trans* **by** *blast*

have $?A \cap ?C = \{\}$
by *blast*
moreover
have $?A \cup ?C = \{0..<i\}$
by (*safe; clarsimp dest!; P*)
ultimately have $i = \text{card } ?A + \text{card } ?C$
by (*metis (no-types, lifting) List.finite-set atLeastLessThan-upt card-Un-disjnt card-upt*
disjnt-def finite-Un)

moreover
have *bij-betw* $(\lambda x. ?idx ! x) ?C ?B$
proof (*intro bij-betwI'; safe*)
fix $x y$
assume $x < \text{length } s \wedge y < \text{length } s \wedge ?idx ! x = ?idx ! y$
with *perm-distinct-iff* [*OF map-bwt-indexes-perm, of s*]
show $x = y$
by (*simp add: bwt-perm-length nth-eq-iff-index-eq*)

next
fix x
assume $x < \text{length } s$
with *map-bwt-indexes-perm* [*of s*]
show $?idx ! x < \text{length } s$
using *perm-nth-ex* **by** *blast*

next
fix x
assume $x < \text{length } s \wedge \text{bwt-sa } s ! x = ?v$
then show $s ! (?idx ! x) = ?v$
using *bwt-perm-s-nth* **by** *auto*

next
fix x
assume $x < \text{length } s \wedge x < i \wedge \text{bwt-sa } s ! x = ?v$
then show $\text{suffix } s (?idx ! x) < \text{suffix } s ?i$
using *bwt-relative-order* [*OF assms(1) - assms(2), of x*] *assms(2)* *bwt-perm-s-nth*

by *fastforce*
next

```

fix  $x$ 
assume  $Q: x < \text{length } s \mid x = ?v \text{ suffix } s \mid x < \text{suffix } s \mid ?i$ 

from  $\text{perm-nth}[OF \text{ map-bwt-indexes-perm}[of \ s, \ \text{symmetric}],$ 
       $\text{simplified length-map sa-length length-upt}]$ 
have  $\exists y < \text{length } s. x = ?idx \mid y$ 
      using  $Q(1) \text{ bwt-perm-length}$  by  $\text{auto}$ 
then obtain  $y$  where
   $y < \text{length } s$ 
   $x = ?idx \mid y$ 
  by  $\text{blast}$ 
moreover
from  $Q(2) \text{ calculation}$ 
have  $?bwt \mid y = ?v$ 
      by  $(\text{simp add: bwt-perm-s-nth})$ 
moreover
have  $y < i$ 
proof  $(\text{rule ccontr})$ 
  assume  $\neg y < i$ 
  hence  $i \leq y$ 
      by  $\text{simp}$ 
moreover
from  $Q(3) \langle x = ?idx \mid y \rangle$ 
have  $i = y \implies \text{False}$ 
      by  $\text{blast}$ 
moreover
have  $i < y \implies \text{False}$ 
proof  $-$ 
  assume  $i < y$ 
from  $\text{bwt-relative-order}[OF \ \text{assms}(1) \langle i < y \rangle \langle y < - \rangle]$ 
       $Q(2) \langle x = ?idx \mid y \rangle$ 
have  $\text{suffix } s \mid ?i < \text{suffix } s \mid (?idx \mid y)$ 
      by  $(\text{simp add: bwt-perm-s-nth assms}(2))$ 
with  $Q(3) \langle x = ?idx \mid y \rangle$ 
show  $\text{False}$ 
      using  $\text{order.asym}$  by  $\text{blast}$ 
qed
ultimately show  $\text{False}$ 
      using  $\text{nat-less-le}$  by  $\text{blast}$ 
qed
ultimately show  $\exists y \in ?C. x = \text{bwt-perm } s \mid y$ 
      by  $\text{blast}$ 
qed
hence  $\text{card } ?C = \text{card } ?B$ 
      using  $\text{bij-betw-same-card}$  by  $\text{blast}$ 
ultimately
show  $?thesis$ 
      by  $\text{presburger}$ 
qed

```


lemma *bwt-perm-to-sa-idx*:
assumes *valid-list s*
and $i < \text{length } s$
shows $\exists k < \text{length } s. \text{sa } s ! k = \text{bwt-perm } s ! i \wedge$
 $k = \text{card } \{j. j < \text{length } s \wedge s ! j < \text{bwt-sa } s ! i\} +$
 $\text{card } \{j. j < \text{length } s \wedge s ! j = \text{bwt-sa } s ! i \wedge$
 $\text{suffix } s j < \text{suffix } s (\text{bwt-perm } s ! i)\}$
proof –
let $?bwt = \text{bwt-sa } s$
let $?v = ?bwt ! i$
let $?i = \text{bwt-perm } s ! i$
let $?A = \{j. j < \text{length } s \wedge s ! j < ?v\}$
let $?B = \{j. j < \text{length } s \wedge s ! j = ?v \wedge \text{suffix } s j < \text{suffix } s ?i\}$

have $\exists k < \text{length } s. \text{sa } s ! k = ?i$
by (*metis assms bwt-perm-nth ex-sa-nth mod-less-divisor valid-list-length*)
then obtain k **where**
 $k < \text{length } s$
 $\text{sa } s ! k = ?i$
by *blast*
moreover
have $s ! (\text{sa } s ! k) = ?v$
using *assms(2) bwt-perm-s-nth calculation(2)* **by** *presburger*
with *sa-card-s-idx[OF calculation(1)]*
have $k = \text{card } ?A + \text{card } ?B$
by (*metis calculation(2)*)
ultimately show *?thesis*
by *blast*
qed

corollary *bwt-perm-eq*:
fixes $s :: ('a :: \{\text{linorder}, \text{order-bot}\}) \text{list}$
fixes $i :: \text{nat}$
assumes *valid-list s*
and $i < \text{length } s$
shows $\text{bwt-perm } s ! i =$
 $\text{sa } s ! (\text{card } \{j. j < \text{length } s \wedge s ! j < \text{bwt-sa } s ! i\} +$
 $\text{card } \{j. j < \text{length } s \wedge s ! j = \text{bwt-sa } s ! i \wedge$
 $\text{suffix } s j < \text{suffix } s (\text{bwt-perm } s ! i)\})$
using *assms bwt-perm-to-sa-idx* **by** *presburger*

13.2 BWT Rank Properties

lemma *bwt-perm-rank-nth*:
fixes $s :: ('a :: \{\text{linorder}, \text{order-bot}\}) \text{list}$
fixes $i :: \text{nat}$
assumes *valid-list s*
and $i < \text{length } s$

shows $\text{rank } (bwt\text{-sa } s) (bwt\text{-sa } s ! i) i =$
 $\text{card } \{j. j < \text{length } s \wedge s ! j = bwt\text{-sa } s ! i \wedge$
 $\text{suffix } s j < \text{suffix } s (bwt\text{-perm } s ! i)\}$

proof –

let $?bwt = bwt\text{-sa } s$
let $?idx = bwt\text{-perm } s$
let $?i = ?idx ! i$
let $?v = ?bwt ! i$
let $?A = \{j. j < \text{length } s \wedge s ! j = ?v \wedge \text{suffix } s j < \text{suffix } s ?i\}$
let $?B = \{j. j < \text{length } ?bwt \wedge j < i \wedge ?bwt ! j = ?v\}$
let $?C = \{j. j < \text{length } s \wedge j < i \wedge ?bwt ! j = ?v\}$

from $\text{valid-list-length-ex}[OF \text{ assms}(1)]$
obtain n **where**
 $\text{length } s = \text{Suc } n$
by blast

from $\text{rank-card-spec}[of ?bwt ?v i]$
have $\text{rank } ?bwt ?v i = \text{card } ?B$.
moreover
have $?B = ?C$
by $(\text{simp add: bwt-sa-length sa-length})$
moreover
have $\text{bij-betw } (\lambda x. ?idx ! x) ?C ?A$
proof $(\text{rule bij-betwI'; safe})$
fix $x y$
assume $x < \text{length } s \wedge y < \text{length } s \wedge ?idx ! x = ?idx ! y$
then show $x = y$
by $(\text{metis map-bwt-indexes-perm bwt-perm-length nth-eq-iff-index-eq perm-distinct-set-of-upt-iff})$

next
fix x
assume $x < \text{length } s$
then show $?idx ! x < \text{length } s$
using $\text{map-bwt-indexes-perm perm-nth-ex}$ **by** blast

next
fix x
assume $x < \text{length } s \wedge x < i \wedge ?bwt ! x = ?v$
then show $s ! (?idx ! x) = ?v$
using bwt-perm-s-nth **by** auto

next
fix x
assume $x < \text{length } s \wedge x < i \wedge ?bwt ! x = ?v$
then show $\text{suffix } s (?idx ! x) < \text{suffix } s ?i$
by $(\text{simp add: assms}(1,2) \text{ bwt-relative-order bwt-perm-s-nth})$

next
fix x
assume $x < \text{length } s \wedge s ! x = ?v \wedge \text{suffix } s x < \text{suffix } s ?i$

```

from perm-nth[OF map-bwt-indices-perm[of s, symmetric],
  simplified length-map sa-length length-upt, of x]
have  $\exists y < \text{length } s. x = ?idx ! y$ 
  using  $\langle x < \text{length } s \rangle$  bwt-perm-length by auto
then obtain y where
  y < length s
  x = ?idx ! y
  by blast
moreover
from calculation  $\langle s ! x = ?v \rangle$ 
have ?bwt ! y = ?v
  using bwt-perm-s-nth by presburger
moreover
have y < i
proof (rule ccontr)
  assume  $\neg y < i$ 
  hence i  $\leq$  y
  by simp
moreover
from  $\langle \text{suffix } s \ x < \text{suffix } s \ ?i \rangle$   $\langle x = ?idx ! y \rangle$ 
have y = i  $\implies$  False
  by blast
moreover
have i < y  $\implies$  False
proof -
  assume i < y
  with bwt-relative-order[OF assms(1)  $\langle i < y \rangle$   $\langle y < - \rangle$ ]  $\langle x = ?idx ! y \rangle$   $\langle s ! x$ 
= bwt-sa s ! i  $\rangle$ 
  have suffix s ?i < suffix s x
  using assms(2) bwt-perm-s-nth by presburger
  with  $\langle \text{suffix } s \ x < \text{suffix } s \ ?i \rangle$ 
  show False
  using less-not-sym by blast
qed
ultimately show False
  by linarith
qed
ultimately show  $\exists y \in ?C. x = \text{bwt-perm } s ! y$ 
  by blast
qed
hence card ?C = card ?A
  using bij-betw-same-card by blast
ultimately show ?thesis
  by presburger
qed

lemma bwt-sa-card-rank-s-idx:
fixes s :: ('a :: {linorder, order-bot}) list
fixes i :: nat

```

```

assumes valid-list s
and  $i < \text{length } s$ 
shows  $i = \text{card } \{j. j < \text{length } s \wedge j < i \wedge \text{bwt-sa } s ! j \neq \text{bwt-sa } s ! i\} +$ 
 $\text{rank } (\text{bwt-sa } s) (\text{bwt-sa } s ! i) i$ 
using assms bwt-sa-card-s-idx bwt-perm-rank-nth by presburger

```

13.3 Suffix Array and BWT Rank

```

lemma sa-bwt-perm-same-rank:
fixes  $s :: ('a :: \{\text{linorder, order-bot}\}) \text{ list}$ 
fixes  $i j :: \text{nat}$ 
assumes valid-list s
and  $i < \text{length } s$ 
and  $j < \text{length } s$ 
and  $sa \ s ! i = \text{bwt-perm } s ! j$ 
shows  $\text{rank } (\text{sort } s) (s ! (sa \ s ! i)) i = \text{rank } (\text{bwt-sa } s) (\text{bwt-sa } s ! j) j$ 
proof –
let  $?i = sa \ s ! i$ 
let  $?v = s ! ?i$ 
let  $?A = \{j. j < \text{length } s \wedge s ! j = ?v \wedge \text{suffix } s \ j < \text{suffix } s \ ?i\}$ 

have  $\text{bwt-sa } s ! j = ?v$ 
using bwt-perm-s-nth[OF assms(3)] assms(4) by presburger

from sa-rank-nth[OF assms(2)]
have  $\text{rank } (\text{sort } s) ?v i = \text{card } ?A .$ 
moreover
from bwt-perm-rank-nth[OF assms(1,3), simplified assms(4)[symmetric]]  $\langle \text{bwt-sa } s ! j = ?v \rangle$ 
have  $\text{rank } (\text{bwt-sa } s) (\text{bwt-sa } s ! j) j = \text{card } ?A$ 
by simp
ultimately show ?thesis
by simp
qed

```

Theorem 3.17 from [3]: Same Rank Rank for each symbol is the same in the BWT and suffix array

```

lemma rank-same-sa-bwt-perm:
fixes  $s :: ('a :: \{\text{linorder, order-bot}\}) \text{ list}$ 
fixes  $i j :: \text{nat}$ 
fixes  $v :: 'a$ 
assumes valid-list s
and  $i < \text{length } s$ 
and  $j < \text{length } s$ 
and  $s ! (sa \ s ! i) = v$ 
and  $\text{bwt-sa } s ! j = v$ 
and  $\text{rank } (\text{sort } s) v i = \text{rank } (\text{bwt-sa } s) v j$ 
shows  $sa \ s ! i = \text{bwt-perm } s ! j$ 
proof –

```

let $?A = \{j. j < \text{length } s \wedge s ! j < v\}$
from $\text{sa-card-rank-s-idx}[OF \text{ assms}(2), \text{ simplified assms}(4)]$
have $i = \text{card } ?A + \text{rank } (\text{sort } s) v i .$
moreover
from $\text{bwt-perm-rank-nth}[OF \text{ assms}(1,3), \text{ simplified assms}(5)]$
 $\text{bwt-perm-eq}[OF \text{ assms}(1,3), \text{ simplified assms}(5)]$
have $\text{bwt-perm } s ! j = \text{sa } s ! (\text{card } ?A + \text{rank } (\text{bwt-sa } s) v j)$
by presburger
with $\text{assms}(6)$
have $\text{bwt-perm } s ! j = \text{sa } s ! (\text{card } ?A + \text{rank } (\text{sort } s) v i)$
by simp
ultimately show $?thesis$
by simp
qed

lemma $\text{rank-bwt-card-suffix}$:

fixes $s :: ('a :: \{\text{linorder}, \text{order-bot}\}) \text{ list}$
fixes $i :: \text{nat}$
fixes $a :: 'a$

assumes $i < \text{length } s$

shows $\text{rank } (\text{bwt-sa } s) a i =$

$$\text{card } \{k. k < \text{length } s \wedge k < i \wedge \text{bwt-sa } s ! k = a \wedge a \# \text{suffix } s (\text{sa } s ! k) < a \# \text{suffix } s (\text{sa } s ! i)\}$$

proof –

let $?X = \{j. j < \text{length } (\text{bwt-sa } s) \wedge j < i \wedge \text{bwt-sa } s ! j = a\}$

let $?Y = \{k. k < \text{length } s \wedge k < i \wedge \text{bwt-sa } s ! k = a \wedge a \# \text{suffix } s (\text{sa } s ! k) < a \# \text{suffix } s (\text{sa } s ! i)\}$

from $\text{rank-card-spec}[of \text{ bwt-sa } s a i]$

have $\text{rank } (\text{bwt-sa } s) a i = \text{card } ?X .$

moreover

have $?Y \subseteq ?X$

using bwt-sa-length **by** auto

moreover

have $?X \subseteq ?Y$

proof safe

fix x

assume $x < \text{length } (\text{bwt-sa } s)$

then show $x < \text{length } s$

by $(\text{simp add: bwt-sa-length})$

next

fix x

assume $x < \text{length } (\text{bwt-sa } s) x < i a = \text{bwt-sa } s ! x$

with $\text{sorted-wrt-mapD}[OF \text{ sa-g-sorted}, of x i s]$

show $\text{bwt-sa } s ! x \# \text{suffix } s (\text{sa } s ! x) < \text{bwt-sa } s ! x \# \text{suffix } s (\text{sa } s ! i)$

by $(\text{simp add: assms sa-length})$

qed

ultimately show $?thesis$

by force

qed

lemma *sa-to-bwt-perm-idx*:

fixes $s :: ('a :: \{\text{linorder}, \text{order-bot}\}) \text{list}$

fixes $i :: \text{nat}$

assumes *valid-list s*

and $i < \text{length } s$

shows $sa\ s\ !\ i =$

$bwt\text{-perm } s\ !\ (\text{select } (bwt\text{-sa } s)\ (s\ !\ (sa\ s\ !\ i))\ (\text{rank } (\text{sort } s)\ (s\ !\ (sa\ s\ !\ i))\ i))$

proof –

let $?a = s\ !\ (sa\ s\ !\ i)$

let $?r1 = \text{rank } (\text{sort } s)\ ?a\ i$

let $?i = \text{select } (bwt\text{-sa } s)\ ?a\ ?r1$

let $?r2 = \text{rank } (bwt\text{-sa } s)\ ?a\ ?i$

have $?r1 < \text{count-list } (\text{sort } s)\ ?a$

by (*simp add: assms(2) rank-upper-bound sort-sa-nth*)

hence $?r1 < \text{count-list } (bwt\text{-sa } s)\ ?a$

by (*metis bwt-sa-perm count-list-perm mset-sort*)

hence $?i < \text{length } (bwt\text{-sa } s)$

by (*metis rank-length select-upper-bound*)

hence $?r1 = ?r2 \wedge bwt\text{-sa } s\ !\ ?i = ?a$

by (*metis rank-select select-nth-alt*)

with *rank-same-sa-bwt-perm[OF assms, of ?i ?a]*

show *?thesis*

using $\langle ?i < \text{length } (bwt\text{-sa } s) \rangle\ bwt\text{-sa-length}$ **by** *fastforce*

qed

lemma *suffix-bwt-perm-sa*:

fixes $s :: ('a :: \{\text{linorder}, \text{order-bot}\}) \text{list}$

fixes $i :: \text{nat}$

assumes *valid-list s*

and $i < \text{length } s$

and $bwt\text{-sa } s\ !\ i \neq \text{bot}$

shows $\text{suffix } s\ (bwt\text{-perm } s\ !\ i) = bwt\text{-sa } s\ !\ i \# \text{suffix } s\ (sa\ s\ !\ i)$

proof –

from *bwt-sa-nth[OF assms(2)]*

have $bwt\text{-sa } s\ !\ i = s\ !\ ((sa\ s\ !\ i + \text{length } s - 1) \bmod \text{length } s)$.

moreover

have $sa\ s\ !\ i \neq 0$

by (*metis add-diff-cancel-left' assms(1,3) calculation diff-less diff-zero last-conv-nth*)

length-greater-0-conv less-one mod-less valid-list-def)

ultimately have $bwt\text{-sa } s\ !\ i = s\ !\ (sa\ s\ !\ i - 1)$

by (*metis Nat.add-diff-assoc2 One-nat-def Suc-lessD Suc-pred assms(2) bot-nat-0.not-eq-extremum*)

less-Suc-eq-le linorder-not-less mod-add-self2 mod-if sa-nth-ex)

hence $bwt\text{-sa } s\ !\ i \# \text{suffix } s\ (sa\ s\ !\ i) = \text{suffix } s\ (sa\ s\ !\ i - 1)$

by (*metis Suc-lessD $\langle sa\ s\ !\ i \neq 0 \rangle$ add-diff-inverse-nat assms(2) less-one*)

```

plus-1-eq-Suc
  sa-nth-ex suffix-cons-Suc)
moreover
have bwt-perm s ! i = sa s ! i - 1
  by (metis Nat.add-diff-assoc2 One-nat-def Suc-leI Suc-lessD Suc-pred <sa s ! i
≠ 0> assms(2)
      bwt-perm-nth mod-add-self2 mod-less not-gr-zero sa-nth-ex)
ultimately show ?thesis
  by presburger
qed

end

end
theory IBWT
  imports BWT-SA-Corres
begin

```

14 Inverse Burrows-Wheeler Transform

Inverse BWT algorithm obtained from [6]

14.1 Abstract Versions

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

These are abstract because they use additional information about the original string and its suffix array.

Definition 3.15 from [3]: Abstract LF-Mapping

```

fun lf-map-abs :: 'a list ⇒ nat ⇒ nat
where
lf-map-abs s i = select (sort s) (bwt-sa s ! i) (rank (bwt-sa s) (bwt-sa s ! i) i)

```

Definition 3.16 from [3]: Inverse BWT Permutation

```

fun ibwt-perm-abs :: nat ⇒ 'a list ⇒ nat ⇒ nat list
where
ibwt-perm-abs 0 - - = [] |
ibwt-perm-abs (Suc n) s i = ibwt-perm-abs n s (lf-map-abs s i) @ [i]

```

end

14.2 Concrete Versions

These are concrete because they only rely on the BWT-transformed sequence without any additional information.

Definition 3.14 from [3]: Inverse BWT - LF-mapping

```

fun lf-map-conc :: ('a :: {linorder, order-bot}) list ⇒ 'a list ⇒ nat ⇒ nat
  where
    lf-map-conc ss bs i = (select ss (bs ! i) 0) + (rank bs (bs ! i) i)

```

```

fun ibwt-perm-conc :: nat ⇒ ('a :: {linorder, order-bot}) list ⇒ 'a list ⇒ nat ⇒
nat list
  where
    ibwt-perm-conc 0 - - - = [] |
    ibwt-perm-conc (Suc n) ss bs i = ibwt-perm-conc n ss bs (lf-map-conc ss bs i)
@ [i]

```

Definition 3.14 from [3]: Inverse BWT - Inverse BWT Rotated Subsequence

```

fun ibwtn :: nat ⇒ ('a :: {linorder, order-bot}) list ⇒ 'a list ⇒ nat ⇒ 'a list
  where
    ibwtn 0 - - - = [] |
    ibwtn (Suc n) ss bs i = ibwtn n ss bs (lf-map-conc ss bs i) @ [bs ! i]

```

Definition 3.14 from [3]: Inverse BWT

```

fun ibwt :: ('a :: {linorder, order-bot}) list ⇒ 'a list
  where
    ibwt bs = ibwtn (length bs) (sort bs) bs (select bs bot 0)

```

15 List Filter

lemma filter-nth-app-upt:

```

filter (λi. P (xs ! i)) [0..by (induct xs arbitrary: ys rule: rev-induct; simp)

```

lemma filter-eq-nth-upt:

```

filter P xs = map (λi. xs ! i) (filter (λi. P (xs ! i)) [0..

```

proof (induct xs rule: rev-induct)

case Nil

then show ?case

by simp

next

case (snoc x xs)

have ?case \longleftrightarrow

```

map (!) xs (filter (λi. P (xs ! i)) [0..

```

using snoc **by** simp

moreover

```

have map (!) (xs @ [x]) (filter (λi. P ((xs @ [x]) ! i)) [0..

```

using filter-nth-app-upt[of P xs [x]] **by** simp

moreover

```

have map (!) xs (filter (λi. P (xs ! i)) [0..

```



```

      map (!) (xs @ [x]) (filter (λi. P (xs ! i)) [0..

```

```

lemma distinct-filter-nth-upt:
  distinct (filter (λi. P (xs ! i)) [0..

```

```

lemma filter-nth-upt-set:
  set (filter (λi. P (xs ! i)) [0..

```

```

lemma filter-length-upt:
  length (filter (λi. P (xs ! i)) [0..

```

```

lemma perm-filter-length:
  xs <~> ys ⇒
  length (filter (λi. P (xs ! i)) [0..

```

16 Verification of the Inverse Burrows-Wheeler Transform

```

context Suffix-Array-General begin

```

16.1 LF-Mapping Simple Properties

```

lemma lf-map-abs-less-length:
  fixes s :: 'a list
  fixes i j :: nat
  assumes i < length s
shows lf-map-abs s i < length s
proof –
  let ?v = bwt-sa s ! i
  let ?r = rank (bwt-sa s) ?v i
  let ?i = lf-map-abs s i

  have ?i = select (sort s) ?v ?r
    by (metis lf-map-abs.simps)

  have ?r < count-list (bwt-sa s) ?v
    by (simp add: assms bwt-sa-length rank-upper-bound)
moreover

```

```

have bwt-sa  $s < \sim \sim >$  sort  $s$ 
  using bwt-sa-perm by auto
ultimately have  $?r < \text{count-list } (\text{sort } s) ?v$ 
  by (metis (no-types, lifting) count-list-perm)
with rank-length[of sort  $s$   $?v$ , symmetric]
have  $?r < \text{rank } (\text{sort } s) ?v (\text{length } s)$ 
  by simp
with select-upper-bound
have select (sort  $s$ )  $?v ?r < \text{length } (\text{sort } s)$ 
  by metis
with  $\langle ?i = \text{select } (\text{sort } s) ?v ?r \rangle$ 
show ?thesis
  by (metis length-sort)
qed

```

```

corollary lf-map-abs-less-length-funpow:
  fixes  $s :: 'a \text{ list}$ 
  fixes  $i \ j :: \text{nat}$ 
  assumes  $i < \text{length } s$ 
shows  $((\text{lf-map-abs } s) \sim k) \ i < \text{length } s$ 
proof (induct  $k$ )
  case  $0$ 
  then show ?case
    using assms by auto
next
  case (Suc  $k$ )
  then show ?case
    by (metis comp-apply funpow.simps(2) lf-map-abs-less-length)
qed

```

```

lemma lf-map-abs-equiv:
  fixes  $s :: ('a :: \{\text{linorder}, \text{order-bot}\}) \text{ list}$ 
  fixes  $i \ r :: \text{nat}$ 
  fixes  $v :: 'a$ 
  assumes  $i < \text{length } (\text{bwt-sa } s)$ 
  and  $v = \text{bwt-sa } s ! i$ 
  and  $r = \text{rank } (\text{bwt-sa } s) \ v \ i$ 
shows  $\text{lf-map-abs } s \ i = \text{card } \{j. j < \text{length } (\text{bwt-sa } s) \wedge \text{bwt-sa } s ! j < v\} + r$ 
proof -

```

```

  have  $\exists k. \text{length } s = \text{Suc } k$ 
    by (metis assms(1) bwt-sa-length less-nat-zero-code not0-implies-Suc)
  then obtain  $n$  where
     $\text{length } s = \text{Suc } n$ 
    by blast

```

```

let  $?P = (\lambda x. x < v)$ 

```

```

have  $\text{lf-map-abs } s \ i = \text{select } (\text{sort } s) \ v \ r$ 

```

```

  by (metis assms(2) assms(3) lf-map-abs.simps)
moreover
from rank-upper-bound[OF assms(1) assms(2)[symmetric]] assms(3)
have r < count-list (bwt-sa s) v
  by simp
hence r < count-list (sort s) v
  using count-list-perm[OF trans[OF bwt-sa-perm sort-perm]] by simp
with sorted-select[of sort s r v]
have select (sort s) v r = card {j. j < length (sort s) ∧ sort s ! j < v} + r
  by simp
moreover
have length (filter (λx. ?P (sort s ! x)) [0..

```

16.2 LF-Mapping Correctness

```

lemma sa-lf-map-abs:
  assumes valid-list s
  and i < length s
shows sa s ! (lf-map-abs s i) = (sa s ! i + length s - Suc 0) mod (length s)
proof -
  let ?v = bwt-sa s ! i
  let ?r = rank (bwt-sa s) ?v i
  let ?i = lf-map-abs s i

  have ?i = select (sort s) ?v ?r
    by (metis lf-map-abs.simps)

  from lf-map-abs-less-length[OF assms(2)]
  have ?i < length s .
  hence select (sort s) ?v ?r < length (sort s)
    by (metis length-sort lf-map-abs.simps)
  with rank-select
  have rank (sort s) ?v (select (sort s) ?v ?r) = ?r
    by metis
  with ⟨?i = select (sort s) ?v ?r⟩
  have rank (sort s) ?v ?i = ?r
    by simp
  moreover
  have ?i < length s

```

```

    using ⟨select (sort s) ?v ?r < length (sort s)⟩ ⟨?i = select (sort s) ?v ?r⟩ by
auto
moreover
{
  from select-nth[of sort s ?v ?r ?i]
  have sort s ! lf-map-abs s i = bwt-sa s ! i
    by (metis ⟨?i = select (sort s) ?v ?r⟩ calculation(2) length-sort)
  moreover
  have s ! (sa s ! ?i) = sort s ! ?i
    using ⟨?i < length s⟩ sort-sa-nth by presburger
  ultimately have s ! (sa s ! ?i) = ?v
    by presburger
}
ultimately have sa s ! ?i = bwt-perm s ! i
  using rank-same-sa-bwt-perm[OF assms(1)- assms(2), of ?i ?v]
  by blast
then show ?thesis
  using bwt-perm-nth[OF assms(2)]
  by simp
qed

```

Theorem 3.18 from [3]: Abstract LF-Mapping Correctness

corollary *bwt-perm-lf-map-abs*:

```

  fixes s :: ('a :: {linorder, order-bot}) list
  fixes i :: nat
  assumes valid-list s
  and i < length s
  shows bwt-perm s ! (lf-map-abs s i) = (bwt-perm s ! i + length s - Suc 0) mod
(length s)
  by (metis One-nat-def bwt-perm-nth assms(1,2) lf-map-abs-less-length sa-lf-map-abs)

```

16.3 Backwards Inverse BWT Simple Properties

lemma *ibwt-perm-abs-length*:

```

  fixes s :: 'a list
  fixes n i :: nat
  shows length (ibwt-perm-abs n s i) = n
  by (induct n arbitrary: i; simp)

```

lemma *ibwt-perm-abs-nth*:

```

  fixes s :: 'a list
  fixes k n i :: nat
  assumes k ≤ n
  shows (ibwt-perm-abs (Suc n) s i) ! k = ((lf-map-abs s)  $\sim$ (n-k)) i
  using assms
  proof (induct n arbitrary: i k)
  case 0
  then show ?case
    by simp
  next

```

```

case (Suc n i k)
note IH = this

have A: ibwt-perm-abs (Suc (Suc n)) s i = ibwt-perm-abs (Suc n) s (lf-map-abs
s i) @ [i]
  by simp

have k ≤ n ⇒ ?case
proof -
  assume k ≤ n
  with IH(1)[of k lf-map-abs s i]
  have ibwt-perm-abs (Suc n) s (lf-map-abs s i) ! k = (lf-map-abs s  $\widehat{\sim}$  (Suc n -
k)) i
    by (metis Suc-diff-le comp-apply funpow.simps(2) funpow-swap1)
  then show ?thesis
    by (metis ⟨k ≤ n⟩ A ibwt-perm-abs-length le-imp-less-Suc nth-append)
qed
moreover
have k = Suc n ⇒ ?case
proof -
  assume k = Suc n
  with ibwt-perm-abs-length[of Suc (Suc n) s i] A
  have ibwt-perm-abs (Suc (Suc n)) s i ! k = i
    by (metis ibwt-perm-abs-length nth-append-length)
  moreover
  have (lf-map-abs s  $\widehat{\sim}$  (Suc n - k)) i = i
    by (simp add: ⟨k = Suc n⟩)
  ultimately show ?thesis
    by presburger
qed
ultimately show ?case
  using Suc.prems le-Suc-eq by blast
qed

corollary ibwt-perm-abs-alt-nth:
  fixes s :: 'a list
  fixes n i k :: nat
  assumes k < n
  shows (ibwt-perm-abs n s i) ! k = ((lf-map-abs s)  $\widehat{\sim}$  (n - Suc k)) i
  by (metis assms add-diff-cancel-left' diff-diff-left le-add1 less-imp-Suc-add plus-1-eq-Suc
ibwt-perm-abs-nth)

lemma ibwt-perm-abs-nth-le-length:
  fixes s :: 'a list
  fixes n i k :: nat
  assumes i < length s
  assumes k < n
  shows (ibwt-perm-abs n s i) ! k < length s
  using assms ibwt-perm-abs-alt-nth lf-map-abs-less-length-funpow by force

```

lemma *ibwt-perm-abs-map-ver*:
 $ibwt\text{-perm-abs } n \ s \ i = \text{map } (\lambda x. ((lf\text{-map-abs } s) \sim x) \ i) \ (\text{rev } [0..<n])$
proof (*intro list-eq-iff-nth-eq*[*THEN iffD2*] *conjI allI impI*)
show $\text{length } (ibwt\text{-perm-abs } n \ s \ i) = \text{length } (\text{map } (\lambda x. (lf\text{-map-abs } s \ \sim x) \ i) \ (\text{rev } [0..<n]))$
by (*simp add: ibwt-perm-abs-length*)
next
fix *j*
assume $j < \text{length } (ibwt\text{-perm-abs } n \ s \ i)$
hence $j < n$
by (*simp add: ibwt-perm-abs-length*)

have $\text{map } (\lambda x. (lf\text{-map-abs } s \ \sim x) \ i) \ (\text{rev } [0..<n]) \ ! \ j =$
 $(\lambda x. (lf\text{-map-abs } s \ \sim x) \ i) \ (\text{rev } [0..<n]) \ ! \ j$
by (*simp add: <j < n>*)
moreover
have $(\lambda x. (lf\text{-map-abs } s \ \sim x) \ i) \ (\text{rev } [0..<n]) \ ! \ j = (lf\text{-map-abs } s \ \sim (n - \text{Suc } j)) \ i$
by (*metis <j < n> add-cancel-right-left diff-Suc-less diff-zero length-greater-0-conv length-upt*
less-nat-zero-code nth-upt rev-nth)
ultimately show $ibwt\text{-perm-abs } n \ s \ i \ ! \ j = \text{map } (\lambda x. (lf\text{-map-abs } s \ \sim x) \ i) \ (\text{rev } [0..<n]) \ ! \ j$
using *ibwt-perm-abs-alt-nth*[*OF <j < n>, of s i*] **by** *presburger*
qed

16.4 Backwards Inverse BWT Correctness

lemma *inc-one-bounded-sa-ibwt-perm-abs*:
fixes $s :: ('a :: \{\text{linorder, order-bot}\}) \ \text{list}$
fixes $i \ n :: \text{nat}$
assumes *valid-list s*
and $i < \text{length } s$
shows $inc\text{-one-bounded } (\text{length } s) \ (\text{map } (!) \ (sa \ s)) \ (ibwt\text{-perm-abs } n \ s \ i)$
 $(\text{is } inc\text{-one-bounded } ?n \ ?xs)$
unfolding *inc-one-bounded-def*
proof (*safe*)
fix *j*
assume $\text{Suc } j < \text{length } (\text{map } (!) \ (sa \ s)) \ (ibwt\text{-perm-abs } n \ s \ i)$
hence $\text{Suc } j < n$
by (*simp add: ibwt-perm-abs-length*)
hence $\exists k. n = \text{Suc } k$
using *less-imp-Suc-add* **by** *blast*
then obtain *k* **where**
 $n = \text{Suc } k$
by *blast*

let $?i = ((lf\text{-map-abs } s) \sim (k - \text{Suc } j)) \ i$

```

have ibwt-perm-abs  $n\ s\ i\ !\ \text{Suc}\ j = ?i$ 
  by (metis  $\langle \text{Suc}\ j < n \rangle \langle n = \text{Suc}\ k \rangle$  less-Suc-eq-le ibwt-perm-abs-nth)
moreover
{
  have ibwt-perm-abs  $n\ s\ i\ !\ j = ((\text{lf-map-abs}\ s) \sim^{(k-j)} i)$ 
    by (metis Suc-less-SucD  $\langle \text{Suc}\ j < n \rangle \langle n = \text{Suc}\ k \rangle$  nless-le ibwt-perm-abs-nth)
  moreover
  have  $((\text{lf-map-abs}\ s) \sim^{(k-j)} i) = \text{lf-map-abs}\ s\ ?i$ 
    using  $\langle \text{Suc}\ j < n \rangle \langle n = \text{Suc}\ k \rangle$  less-imp-Suc-add by fastforce
  ultimately have ibwt-perm-abs  $n\ s\ i\ !\ j = \text{lf-map-abs}\ s\ ?i$ 
    by presburger
}
moreover
{
  have  $?i < \text{length}\ s$ 
    by (simp add: assms lf-map-abs-less-length-funpow)
  with sa- $\text{lf-map-abs}$ [OF assms(1), of ?i]
  have  $\text{sa}\ s\ !\ \text{lf-map-abs}\ s\ ?i = (\text{sa}\ s\ !\ ?i + \text{length}\ s - \text{Suc}\ 0) \bmod \text{length}\ s$ 
    by fastforce
  hence Suc  $(\text{sa}\ s\ !\ \text{lf-map-abs}\ s\ ?i) \bmod \text{length}\ s$ 
    = Suc  $((\text{sa}\ s\ !\ ?i + \text{length}\ s - \text{Suc}\ 0) \bmod \text{length}\ s) \bmod \text{length}\ s$ 
    by simp
  moreover
  have Suc  $((\text{sa}\ s\ !\ ?i + \text{length}\ s - \text{Suc}\ 0) \bmod \text{length}\ s) \bmod \text{length}\ s = \text{sa}\ s\ !\ ?i$ 
    using  $\langle ?i < \text{length}\ s \rangle$  assms(1) mod-Suc-eq sa-nth-ex valid-list-length by
fastforce
  ultimately have  $\text{sa}\ s\ !\ ?i = \text{Suc}\ (\text{sa}\ s\ !\ \text{lf-map-abs}\ s\ ?i) \bmod \text{length}\ s$ 
    by presburger
}
ultimately have
 $\text{sa}\ s\ !\ (\text{ibwt-perm-abs}\ n\ s\ i\ !\ \text{Suc}\ j) = \text{Suc}\ (\text{sa}\ s\ !\ (\text{ibwt-perm-abs}\ n\ s\ i\ !\ j)) \bmod$ 
 $\text{length}\ s$ 
  by presburger
then show
 $\text{map}\ (!)\ (\text{sa}\ s) (\text{ibwt-perm-abs}\ n\ s\ i) !\ \text{Suc}\ j =$ 
 $\text{Suc}\ (\text{map}\ (!)\ (\text{sa}\ s) (\text{ibwt-perm-abs}\ n\ s\ i) !\ j) \bmod \text{length}\ s$ 
  using  $\langle \text{Suc}\ j < \text{length}\ (\text{map}\ (!)\ (\text{sa}\ s) (\text{ibwt-perm-abs}\ n\ s\ i)) \rangle$  by auto
next
fix j
assume  $j < \text{length}\ (\text{map}\ (!)\ (\text{sa}\ s) (\text{ibwt-perm-abs}\ n\ s\ i))$ 
hence  $j < n$ 
  by (simp add: ibwt-perm-abs-length)
hence ibwt-perm-abs  $n\ s\ i\ !\ j = ((\text{lf-map-abs}\ s) \sim^{(n-\text{Suc}\ j)} i)$ 
  using ibwt-perm-abs-alt-nth by blast
moreover
have  $((\text{lf-map-abs}\ s) \sim^{(n-\text{Suc}\ j)} i) < \text{length}\ s$ 
  using assms lf-map-abs-less-length-funpow by blast
hence  $\text{sa}\ s\ !\ (((\text{lf-map-abs}\ s) \sim^{(n-\text{Suc}\ j)} i) < \text{length}\ s$ 
  using sa-nth-ex by blast

```

ultimately have $sa\ s\ !\ (ibwt\text{-perm-abs}\ n\ s\ i\ !\ j) < length\ s$
by *presburger*
then show $map\ (!)\ (sa\ s)\ (ibwt\text{-perm-abs}\ n\ s\ i)\ !\ j < length\ s$
by (*simp add: <j < n> ibwt-perm-abs-length*)
qed

corollary *is-rot-sublist-sa-ibwt-perm-abs:*
fixes $s :: ('a :: \{linorder, order-bot\})\ list$
fixes $i\ n :: nat$
assumes *valid-list s*
and $i < length\ s$
and $n \leq length\ s$
shows *is-rot-sublist [0..<length s] (map (!) (sa s) (ibwt-perm-abs n s i))*
by (*simp add: asms inc-one-bounded-is-rot-sublist inc-one-bounded-sa-ibwt-perm-abs ibwt-perm-abs-length*)

lemma *inc-one-bounded-bwt-perm-ibwt-perm-abs:*
fixes $s :: ('a :: \{linorder, order-bot\})\ list$
fixes $i\ n :: nat$
assumes *valid-list s*
and $i < length\ s$
shows *inc-one-bounded (length s) (map (!) (bwt-perm s) (ibwt-perm-abs n s i))*
unfolding *inc-one-bounded-def*
proof *safe*
fix j
assume $Suc\ j < length\ (map\ (!)\ (bwt\text{-perm}\ s)\ (ibwt\text{-perm-abs}\ n\ s\ i))$
hence $Suc\ j < n$
by (*simp add: ibwt-perm-abs-length*)
hence $\exists k. n = Suc\ k$
using *less-imp-Suc-add* **by** *auto*
then obtain k **where**
 $n = Suc\ k$
by *blast*

let $?i = ((lf\text{-map-abs}\ s) \sim^{(k - Suc\ j)})\ i$
from *ibwt-perm-abs-nth[of Suc j k s i]*
have $ibwt\text{-perm-abs}\ n\ s\ i\ !\ Suc\ j = ?i$
using $\langle Suc\ j < n \rangle \langle n = Suc\ k \rangle$ *less-Suc-eq-le* **by** *blast*
moreover
{
have $ibwt\text{-perm-abs}\ n\ s\ i\ !\ j = ((lf\text{-map-abs}\ s) \sim^{(k - j)})\ i$
by (*metis Suc-less-SucD <Suc j < n> <n = Suc k> nless-le ibwt-perm-abs-nth*)
moreover
have $((lf\text{-map-abs}\ s) \sim^{(k - j)})\ i = lf\text{-map-abs}\ s\ ?i$
using $\langle Suc\ j < n \rangle \langle n = Suc\ k \rangle$ *less-imp-Suc-add* **by** *fastforce*
ultimately have $ibwt\text{-perm-abs}\ n\ s\ i\ !\ j = lf\text{-map-abs}\ s\ ?i$
by *presburger*
}
moreover


```

{
  have ?i < length s
    by (simp add: assms lf-map-abs-less-length-funpow)
  with bwt-perm-lf-map-abs[OF assms(1), of ?i]
  have bwt-perm s ! lf-map-abs s ?i = (bwt-perm s ! ?i + length s - Suc 0) mod
length s
    by blast
  hence Suc (bwt-perm s ! lf-map-abs s ?i) mod length s =
    Suc ((bwt-perm s ! ?i + length s - Suc 0) mod length s) mod length s
    by presburger
  moreover
  from valid-list-length-ex[OF assms(1)]
  obtain n where
    length s = Suc n
    by blast
  hence Suc ((bwt-perm s ! ?i + length s - Suc 0) mod length s) mod length s =
    bwt-perm s ! ?i
    by (metis (no-types, lifting) Suc-pred bwt-perm-nth ‹?i < length s› add-gr-0
assms(1)
      mod-Suc-eq mod-add-self2 mod-mod-trivial valid-list-length)
  ultimately have bwt-perm s ! ?i = Suc (bwt-perm s ! lf-map-abs s ?i) mod
length s
    by presburger
}
ultimately have bwt-perm s ! (ibwt-perm-abs n s i ! Suc j) =
  Suc (bwt-perm s ! (ibwt-perm-abs n s i ! j)) mod length s
  by presburger
then show map (!) (bwt-perm s) (ibwt-perm-abs n s i) ! Suc j =
  Suc (map (!) (bwt-perm s) (ibwt-perm-abs n s i) ! j) mod length s
  using ‹Suc j < length (map (!) (bwt-perm s) (ibwt-perm-abs n s i))› by auto
next
fix j
assume j < length (map (!) (bwt-perm s) (ibwt-perm-abs n s i))
hence j < n
  by (simp add: ibwt-perm-abs-length)
hence ∃ k. n = Suc k
  using less-imp-Suc-add by blast
then obtain k where
  n = Suc k
  by blast
hence ibwt-perm-abs n s i ! j = ((lf-map-abs s) ^ (k - j)) i
  by (metis ‹j < n› less-Suc-eq-le ibwt-perm-abs-nth)
moreover
have ((lf-map-abs s) ^ (k - j)) i < length s
  using assms lf-map-abs-less-length-funpow by blast
hence bwt-perm s ! ((lf-map-abs s) ^ (k - j)) i < length s
  using map-bwt-indexes-perm perm-nth-ex by blast
ultimately have bwt-perm s ! (ibwt-perm-abs n s i ! j) < length s
  by presburger

```

then show $\text{map } (!) (bwt\text{-perm } s) (ibwt\text{-perm-abs } n \ s \ i) ! j < \text{length } s$
by (*simp add: $\langle j < n \rangle$ ibwt-perm-abs-length*)
qed

Theorem 3.19 from [3]: Abstract Inverse BWT Permutation Rotated Sub-list

corollary *is-rot-sublist-bwt-perm-ibwt-perm-abs*:
fixes $s :: ('a :: \{\text{linorder, order-bot}\}) \text{ list}$
fixes $i \ n :: \text{nat}$
assumes *valid-list s*
and $i < \text{length } s$
and $n \leq \text{length } s$
shows *is-rot-sublist* $[0..<\text{length } s]$ $(\text{map } (!) (bwt\text{-perm } s) (ibwt\text{-perm-abs } n \ s \ i))$
by (*simp add: asms inc-one-bounded-is-rot-sublist inc-one-bounded-bwt-perm-ibwt-perm-abs ibwt-perm-abs-length*)

lemma *bwt-ibwt-perm-sa-lookup-idx*:
assumes *valid-list s*
shows $\text{map } (!) (bwt\text{-perm } s) (ibwt\text{-perm-abs } (\text{length } s) \ s \ (\text{select } (bwt\text{-sa } s) \ \text{bot} \ 0))$
 $= [0..<\text{length } s]$

proof –

from *valid-list-length-ex[OF asms]*

obtain n **where**

$\text{length } s = \text{Suc } n$

by *blast*

let $?i = \text{select } (bwt\text{-sa } s) \ \text{bot} \ 0$

let $?xs = ibwt\text{-perm-abs } (\text{length } s) \ s \ ?i$

have $\text{bot} \in \text{set } s$

by (*metis asms in-set-conv-decomp valid-list-ex-def*)

hence $\text{bot} \in \text{set } (bwt\text{-sa } s)$

by (*metis bwt-sa-perm perm-set-eq*)

hence $\text{count-list } (bwt\text{-sa } s) \ \text{bot} > 0$

by (*meson count-in*)

hence $0 < \text{rank } (bwt\text{-sa } s) \ \text{bot} \ (\text{length } (bwt\text{-sa } s))$

by (*metis rank-length*)

hence $?i < \text{length } (bwt\text{-sa } s)$

by (*meson select-upper-bound*)

hence $?i < \text{length } s$

by (*metis bwt-sa-length*)

with *is-rot-sublist-bwt-perm-ibwt-perm-abs[OF asms, of ?i length s] $\langle \text{length } s = \text{Suc } n \rangle$*

have *is-rot-sublist* $[0..<\text{Suc } n]$ $(\text{map } (!) (bwt\text{-perm } s) \ ?xs)$

by (*metis nle-le*)

moreover

have $\text{length } (\text{map } (!) (bwt\text{-perm } s) \ ?xs) = \text{Suc } n$

by (*metis $\langle \text{length } s = \text{Suc } n \rangle$ length-map ibwt-perm-abs-length*)

```

moreover
{
  have (map (!) (bwt-perm s)) ?xs ! n = bwt-perm s ! ?i
    by (simp add: ⟨length s = Suc n⟩ nth-append ibwt-perm-abs-length)
  moreover
  have bwt-sa s ! ?i = bot
    by (simp add: ⟨?i < length (bwt-sa s)⟩ select-nth-alt)
  hence bwt-perm s ! ?i = n
    by (metis ⟨length s = Suc n⟩ ⟨?i < length s⟩ antisym-conv3 assms bwt-perm-nth
      bwt-perm-s-nth diff-Suc-1 mod-less-divisor not-less-eq valid-list-def)
  ultimately
  have (map (!) (bwt-perm s)) ?xs ! n = n
    by blast
}
ultimately show ?thesis
using is-rot-sublist-upt-eq-upt-last[of n map (!) (bwt-perm s)] ?xs
by (metis ⟨length s = Suc n⟩)
qed

```

lemma *map-bwt-sa-bwt-perm*:

```

∀ x ∈ set xs. x < length s ⇒
  map (!) (bwt-sa s) xs = map (!) s (map (!) (bwt-perm s) xs)
by (simp add: bwt-perm-s-nth)

```

theorem *ibwt-perm-abs-bwt-sa-lookup-correct*:

```

fixes s :: ('a :: {linorder, order-bot}) list
assumes valid-list s
shows map (!) (bwt-sa s) (ibwt-perm-abs (length s) s (select (bwt-sa s) bot 0))
= s

```

proof –

```

let ?i = select (bwt-sa s) bot 0
let ?xs = map (!) (bwt-perm s) (ibwt-perm-abs (length s) s ?i)

```

```

have bot ∈ set s
  by (metis assms in-set-conv-decomp valid-list-ex-def)
hence bot ∈ set (bwt-sa s)
  by (metis bwt-sa-perm perm-set-eq)
hence count-list (bwt-sa s) bot > 0
  by (meson count-in)
hence 0 < rank (bwt-sa s) bot (length (bwt-sa s))
  by (metis rank-length)
hence ?i < length (bwt-sa s)
  by (meson select-upper-bound)
hence ?i < length s
  by (metis bwt-sa-length)

```

```

have map (!) (bwt-sa s) (ibwt-perm-abs (length s) s ?i) = map (!) s ?xs

```

proof (*intro map-bwt-sa-bwt-perm ballI*)

```

fix x

```

assume $x \in \text{set } (\text{ibwt-perm-abs } (\text{length } s) s ?i)$
from $\text{in-set-conv-nth}[\text{THEN } \text{iffD1}, \text{OF } \langle x \in \cdot \rangle]$
obtain i **where**
 $i < \text{length } (\text{ibwt-perm-abs } (\text{length } s) s ?i)$
 $\text{ibwt-perm-abs } (\text{length } s) s ?i ! i = x$
by blast
with $\text{ibwt-perm-abs-alt-nth}[\text{of } i \text{ length } s s ?i]$
have $x = (\text{lf-map-abs } s \sim (\text{length } s - \text{Suc } i)) ?i$
by $(\text{metis } \text{ibwt-perm-abs-length})$
moreover
have $(\text{lf-map-abs } s \sim (\text{length } s - \text{Suc } i)) ?i < \text{length } s$
using $\langle ?i < \text{length } s \rangle \text{ assms } \text{lf-map-abs-less-length-funpow}$ **by** presburger
ultimately show $x < \text{length } s$
by blast
qed
then show $?thesis$
using $\text{bwt-ibwt-perm-sa-lookup-idx}[\text{OF } \text{assms}] \text{ map-nth}$ **by** auto
qed

16.5 Concretization

lemma $\text{lf-map-abs-eq-conc}$:

$i < \text{length } s \implies \text{lf-map-abs } s i = \text{lf-map-conc } (\text{sort } (\text{bwt-sa } s)) (\text{bwt-sa } s) i$

proof –

let $?v = \text{bwt-sa } s ! i$

let $?r = \text{rank } (\text{bwt-sa } s) ?v i$

let $?ss = \text{sort } (\text{bwt-sa } s)$

assume $i < \text{length } s$

hence $\text{rank } (\text{bwt-sa } s) ?v i < \text{count-list } (\text{sort } s) ?v$

using $\text{rank-upper-bound}[\text{of } i \text{ bwt-sa } s ?v]$

by $(\text{metis } \text{bwt-sa-length } \text{bwt-sa-perm } \text{count-list-perm } \text{mset-sort})$

with $\text{sorted-select}[\text{of } ?ss ?r ?v]$

have $\text{select } ?ss ?v ?r = \text{card } \{j. j < \text{length } ?ss \wedge ?ss ! j < ?v\} + ?r$

by $(\text{metis } (\text{full-types}) \text{bwt-sa-perm } \text{sorted-list-of-multiset-mset } \text{sorted-sort})$

moreover

have $\text{sort } s = \text{sort } ?ss$

by $(\text{simp add: } \text{bwt-sa-perm } \text{properties-for-sort})$

moreover

have $\text{select } (\text{sort } s) ?v ?r = \text{card } \{j. j < \text{length } (\text{sort } s) \wedge (\text{sort } s) ! j < ?v\} + ?r$

by $(\text{simp add: } \langle \text{rank } (\text{bwt-sa } s) ?v i < \text{count-list } (\text{sort } s) ?v \rangle \text{sorted-select})$

ultimately show $?thesis$

by $(\text{metis } (\text{full-types}) \langle \text{rank } (\text{bwt-sa } s) ?v i < \text{count-list } (\text{sort } s) ?v \rangle \text{bwt-sa-perm}$

$\text{lf-map-abs.simps } \text{lf-map-conc.simps } \text{sorted-list-of-multiset-mset}$

$\text{sorted-select-0-plus } \text{sorted-sort})$

qed

lemma *ibwt-perm-abs-conc-eq*:
 $i < \text{length } s \implies \text{ibwt-perm-abs } n \ s \ i = \text{ibwt-perm-conc } n \ (\text{sort } (\text{bwt-sa } s)) \ (\text{bwt-sa } s) \ i$

proof (*induct n arbitrary: i*)
case 0
then show ?case
by auto

next
case (Suc n)

let ?ss = sort (bwt-sa s)
let ?bs = bwt-sa s

have $\text{ibwt-perm-abs } (\text{Suc } n) \ s \ i = \text{ibwt-perm-abs } n \ s \ (\text{lf-map-abs } s \ i) \ @ \ [i]$
by simp

moreover
have $\text{ibwt-perm-conc } (\text{Suc } n) \ ?ss \ ?bs \ i = \text{ibwt-perm-conc } n \ ?ss \ ?bs \ (\text{lf-map-conc } ?ss \ ?bs \ i) \ @ \ [i]$
by simp

moreover
have $\text{lf-map-abs } s \ i = \text{lf-map-conc } ?ss \ ?bs \ i$
using Suc.premis lf-map-abs-eq-conc **by** blast

moreover
have $\text{lf-map-abs } s \ i < \text{length } s$
using Suc.premis lf-map-abs-less-length **by** blast

ultimately show ?case
using Suc.hyps **by** presburger

qed

theorem *ibwt-n-bwt-sa-lookup-correct*:
fixes $s \ xs \ ys :: ('a :: \{\text{linorder}, \text{order-bot}\}) \ \text{list}$
assumes valid-list s
and $xs = \text{sort } (\text{bwt-sa } s)$
and $ys = \text{bwt-sa } s$

shows $\text{map } (!) \ ys \ (\text{ibwt-perm-conc } (\text{length } ys) \ xs \ ys \ (\text{select } ys \ \text{bot } 0)) = s$

proof –
from *ibwt-perm-abs-bwt-sa-lookup-correct*[OF *assms*(1)]
have $\text{map } (!) \ (\text{bwt-sa } s) \ (\text{ibwt-perm-abs } (\text{length } s) \ s \ (\text{select } (\text{bwt-sa } s) \ \text{bot } 0)) = s$.

moreover
have $\text{select } (\text{bwt-sa } s) \ \text{bot } 0 < \text{length } s$
by (*metis* (no-types, lifting) *assms*(1) *bot-nat-0.extremum-uniqueI* *bwt-sa-length* *bwt-sa-perm*

count-list-perm *diff-Suc-1* *last-conv-nth* *length-greater-0-conv*
less-nat-zero-code *rank-upper-bound* *sa-nth-ex* *select-spec*
valid-list-def *valid-list-sa-hd*)

with *ibwt-perm-abs-conc-eq*
have $\text{ibwt-perm-abs } (\text{length } s) \ s \ (\text{select } (\text{bwt-sa } s) \ \text{bot } 0) =$

```

      ibwt-perm-conc (length ys) xs ys (select ys bot 0)
    using assms(2) assms(3) bwt-sa-length by presburger
  ultimately show ?thesis
    using assms(3) by auto
qed

```

```

lemma ibwtn-eq-map-ibwt-perm-conc:
  shows ibwtn n ss bs i = map (!) bs (ibwt-perm-conc n ss bs i)
  by (induct n arbitrary: i; simp)

```

```

theorem ibwtn-correct:
  fixes s xs ys :: ('a :: {linorder, order-bot}) list
  assumes valid-list s
  and xs = sort (bwt-sa s)
  and ys = bwt-sa s
shows ibwtn (length ys) xs ys (select ys bot 0) = s
  by (metis ibwtn-eq-map-ibwt-perm-conc ibwtn-bwt-sa-lookup-correct assms)

```

16.6 Inverse BWT Correctness

BWT (suffix array version) is invertible

```

theorem ibwt-correct:
  fixes s :: ('a :: {linorder, order-bot}) list
  assumes valid-list s
  shows ibwt (bwt-sa s) = s
  by (simp add: assms ibwtn-correct)

```

end

Theorem 3.20 from [3]: Correctness of the Inverse BWT

```

theorem ibwt-correct-canon:
  fixes s :: ('a :: {linorder, order-bot}) list
  assumes valid-list s
  shows ibwt (bwt-canon s) = s
  by (metis Suffix-Array-General.bwt-canon-eq-bwt-sa Suffix-Array-General.ibwt-correct
      Suffix-Array-General-ex assms)

```

end

References

- [1] R. Affeldt, J. Garrigue, X. Qi, and K. Tanaka. Proving tree algorithms for succinct data structures. In *Proc. Interactive Theorem Proving*, volume 141 of *LIPICs*, pages 5:1–5:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
- [2] M. Burrows and D. Wheeler. A block-sorting lossless data compression algorithm. Technical report, Digital SRC Research Report, 1994.

- [3] L. Cheung, A. Moffat, and C. Rizkallah. Formalized Burrows-Wheeler Transform. In *Proc. Certified Programs and Proofs*. ACM, 2025. To appear.
- [4] L. Cheung and C. Rizkallah. Formalized Burrows-Wheeler Transform (artefact), December 2024.
- [5] L. Cheung and C. Rizkallah. Formally verified suffix array construction. *Archive of Formal Proofs*, September 2024. <https://isa-afp.org/entries/SuffixArray.html>, Formal proof development.
- [6] P. Ferragina and G. Manzini. Opportunistic data structures with applications. In *Foundations of Computer Science*, pages 390–398. IEEE Computer Society, 2000.
- [7] 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.