Formalized Burrows-Wheeler Transform

Louis Cheung and Christine Rizkallah

October 13, 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

theory *Nat-Mod-Helper*
  imports *Main*
begin

1 Nat Modulo Helper

lemma *nat-mod-add-neq-self*:
  $[\![a < (n :: nat);\ b < n;\ b \neq 0]\!] \Longrightarrow (a + b)\ mod\ n \neq a$
  by (*metis add-diff-cancel-left′ mod-if mod-mult-div-eq mod-mult-self1-is-0*)

lemma *nat-mod-a-pl-b-eq1*:
  $[\![n + b \leq a;\ a < (n :: nat)]\!] \Longrightarrow (a + b)\ mod\ n = b - (n - a)$
  using *order-le-less-trans* **by** *blast*

lemma *not-mod-a-pl-b-eq2*:
  $[\![n - a \leq b;\ a < n;\ b < (n :: nat)]\!] \Longrightarrow (a + b)\ mod\ n = b - (n - a)$
  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
$is\text{-}sublist\ xs\ ys = (\exists\, as\ bs.\ xs = as\ @\ ys\ @\ bs)$

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

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

lemma *inc-one-boundedD*:

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

lemma *inc-one-bounded-nth-plus*:
$\llbracket$*inc-one-bounded n xs*; *i* + *k* < *length xs*$\rrbracket \Longrightarrow$ *xs* ! (*i* + *k*) = (*xs* ! *i* + *k*) *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*:
$\llbracket$*inc-one-bounded n xs*; *length xs* ≤ *n*; *i* + *k* < *length xs*; *k* ≠ *0*$\rrbracket \Longrightarrow$ *xs* ! (*i* + *k*)
≠ *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 *length xs* ≤ *n*
and *i* < *length xs*
and *j* < *length xs*
and *i* ≠ *j*
shows *xs* ! *i* ≠ *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 ¬ *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*:
$\llbracket$*inc-one-bounded n xs*; *length xs* ≤ *n*$\rrbracket \Longrightarrow$ *distinct xs*
using *distinct-conv-nth inc-one-bounded-neq-nth* **by** *blast*

lemma *inc-one-bounded-subset-upt*:
$\llbracket$*inc-one-bounded n xs*; *length xs* ≤ *n*$\rrbracket \Longrightarrow$ *set xs* ⊆ {*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*) ⟹ *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*:
  ⟦*inc-one-bounded n xs*; *i* < *length xs*⟧ ⟹ *xs* ! *i* = ((λ*x*. *Suc x mod n*)^^*i*) (*xs* ! *0*)
proof (*induct i*)
  case *0*
  then show *?case*
    by *simp*
next
  case (*Suc i*)
  note *IH* = *this*

  from *IH*
  have *xs* ! *i* = ((λ*x*. *Suc x mod n*) ^^ *i*) (*xs* ! *0*)
    by *simp*
  hence *Suc* (*xs* ! *i*) *mod n* = ((λ*x*. *Suc x mod n*) ^^ *Suc i*) (*xs* ! *0*)
    by *force*
  moreover
  from *inc-one-boundedD*(*1*)[*OF IH*(*2*,*3*)]
  have *xs* ! *Suc i* = *Suc* (*xs* ! *i*) *mod n*.
  ultimately show *?case*
    by *presburger*
qed

lemma *inc-one-bounded-nth-le*:
  ⟦*inc-one-bounded n xs*; *i* < *length xs*; (*xs* ! *0*) + *i* < *n*⟧ ⟹
   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 *length xs* = *Suc k*
  and *Suc k* ≤ *n*
  and (*xs* ! *0*) + *k* < *n*
shows *xs* = [*xs* ! *0*..<(*xs* ! *0*) + *Suc k*]
proof (*intro list-eq-iff-nth-eq*[*THEN iffD2*] *conjI impI allI*)
  show *length xs* = *length* [*xs* ! *0*..<*xs* ! *0* + *Suc k*]
    using *assms*(*2*) **by** *force*
next
  fix *i*
  assume *i* < *length xs*
  hence [*xs* ! *0*..<*xs* ! *0* + *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..<xs ! 0 + Suc k] ! i
    by presburger
qed

lemma inc-one-bounded-upt2:
  assumes inc-one-bounded n xs
  and     length xs = Suc k
  and     Suc k ≤ n
  and     n ≤ (xs ! 0) + k
shows xs = [xs ! 0..<n] @ [0..<(xs ! 0) + Suc k − n]
proof (intro list-eq-iff-nth-eq[THEN iffD2] conjI impI allI)
  show length xs = length ([xs ! 0..<n] @ [0..<xs ! 0 + Suc k − n])
    using assms(1) assms(2) assms(4) inc-one-boundedD(2) less-or-eq-imp-le by
auto
next
  fix i
  assume i < length xs
  show xs ! i = ([xs ! 0..<n] @ [0..<xs ! 0 + Suc k − n]) ! i
  proof (cases i < length [xs ! 0..<n])
    assume i < length [xs ! 0..<n]
    hence ([xs ! 0..<n] @ [0..<xs ! 0 + Suc k − n]) ! i = [xs ! 0..<n] ! i
      by (meson nth-append)
    moreover
    have [xs ! 0..<n] ! i = xs ! 0 + i
      using ‹i < length [xs ! 0..<n]› by force
    moreover
    have xs ! 0 + i < n
      using ‹i < length [xs ! 0..<n]› by auto
    with inc-one-bounded-nth-le[OF assms(1) ‹i < length xs›]
    have xs ! i = xs ! 0 + i
      by blast
    ultimately show xs ! i = ([xs ! 0..<n] @ [0..<xs ! 0 + Suc k − n]) ! i
      by simp
  next
    assume ¬ i < length [xs ! 0..<n]
    hence ([xs ! 0..<n] @ [0..<xs ! 0 + Suc k − n]) ! i =
          [0..<xs ! 0 + Suc k − n] ! (i − length [xs ! 0..<n] )
      by (meson nth-append)
    moreover
    have [0..<xs ! 0 + Suc k − n] ! (i − length [xs ! 0..<n]) = i − (n − xs ! 0)
      using ‹i < length xs› add-0 assms(2) assms(4) by fastforce
    moreover
    {
      have i < n
        using ‹i < length xs› assms(2) assms(3) by linarith
```

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 ‹¬ $i < length\ [xs\ !\ 0..<n]$› **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* ‹$i <$
length xs›]
by *presburger*
}
ultimately show $xs\ !\ i = ([xs\ !\ 0..<n]\ @\ [0..<xs\ !\ 0 + Suc\ k - n])\ !\ i$
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*:
$m \leq n \Longrightarrow rotate\ m\ [0..<n] = [m..<n]\ @\ [0..<m]$
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 $\exists na\ as\ bs.\ rotate\ na\ [0..<n] = as\ @\ xs\ @\ bs$
using *append-Nil* **by** *blast*
next
case (*Suc k*)
hence $Suc\ k \leq n$
using *assms*(*2*) **by** *auto*

have $(xs\ !\ 0) + k < n \Longrightarrow \exists na\ as\ bs.\ rotate\ na\ [0..<n] = as\ @\ xs\ @\ bs$
proof −
assume $(xs\ !\ 0) + k < n$
with *inc-one-bounded-upt*(*1*)[*OF assms*(*1*) *Suc* ‹$Suc\ k \leq n$›]
have $xs = [xs\ !\ 0..<xs\ !\ 0 + Suc\ k]$
by *blast*
moreover
have $xs\ !\ 0 + Suc\ k \leq n$

by (*simp add*: *Suc-leI* ‹xs ! 0 + k < n›)
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 ‹xs ! 0 + *Suc* k ≤ n› **by** *fastforce*
ultimately show *?thesis*
by (*metis append.right-neutral append-Nil rotate-append*)
qed
moreover
have ¬ (xs ! 0) + k < n ⟹ ∃ *na as bs*. *rotate na* [0..<n] = *as* @ xs @ *bs*
proof −
assume ¬ (xs ! 0) + k < n
hence (xs ! 0) + k ≥ n
by *simp*
with *inc-one-bounded-upt*(2)[*OF assms*(1) *Suc* ‹*Suc* k ≤ n›]
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 ≤ 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 ‹*Suc* k ≤ n› ‹n ≤ xs ! 0 + k› **by** *auto*
ultimately have [0..<xs ! 0] = [0..<xs ! 0 + *Suc* k − n] @ [xs ! 0 + *Suc* k − n..<xs ! 0]
by *argo*
}
ultimately show *?thesis*
by (*metis append.assoc append-Nil*)
qed
ultimately show ∃ *na as bs*. *rotate na* [0..<n] = *as* @ xs @ *bs*
by *blast*
qed

lemma *is-rot-sublist-idx*:

is-rot-sublist [*0*..<*length xs*] *ys* ⟹ *is-rot-sublist xs* (*map* ((!) *xs*) *ys*)
unfolding *is-rot-sublist-def is-sublist-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-nth rotate-map*)
then show ∃ *n as bs*. *rotate n xs* = *as* @ *map* ((!) *xs*) *ys* @ *bs*
by *auto*
qed

lemma *is-rot-sublist-upt-eq-upt-hd*:
⟦*is-rot-sublist* [*0*..<*Suc n*] *ys*; *length ys* = *Suc n*; *ys* ! *0* = *0*⟧ ⟹ *ys* = [*0*..<*Suc n*]
unfolding *is-rot-sublist-def is-sublist-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-conv-mod*[*of m* [*0*..<*Suc n*]]
have *rotate* (*m mod length* [*0*..<*Suc n*]) [*0*..<*Suc n*] = *as* @ *ys* @ *bs*
by *simp*
with *rotate-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-zero le-Suc-eq length-upt mod-Suc-le-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-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
have *m mod Suc n* = *0*
by (*metis A add.right-neutral append.right-neutral calculation*(*2*,*3*) *diff-zero length-rotate mod-less-divisor nth-rotate nth-upt self-append-conv2 zero-le zero-less-Suc ordered-cancel-comm-monoid-diff-class.add-diff-inverse*)
ultimately show *ys* = [*0*..<*Suc n*]
by *simp*
qed

lemma *is-rot-sublist-upt-eq-upt-last*:
⟦*is-rot-sublist* [*0*..<*Suc n*] *ys*; *length ys* = *Suc n*; *ys* ! *n* = *n*⟧ ⟹ *ys* = [*0*..<*Suc*

```
n]
  unfolding is-rot-sublist-def is-sublist-def
proof (elim exE)
  fix m as bs
  assume A: length ys = Suc n ys ! n = n rotate m [0..<Suc n] = as @ ys @ bs
 with rotate-conv-mod[of m [0..<Suc n]]
  have rotate (m mod length [0..<Suc n]) [0..<Suc n] = as @ ys @ bs
    by simp
  with rotate-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-zero le-Suc-eq length-upt mod-Suc-le-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-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 Suc n = length ys ∀ i<Suc n. ([m mod Suc n..<n] @ n # [0..<m mod Suc
n]) ! i = ys ! i
    by blast+
  hence ([m mod Suc n..<n] @ n # [0..<m mod Suc n]) ! n = n
    by (simp add: A(2))
  with nth-append[of [m mod Suc n..<n] n # [0..<m mod Suc n] n]
  have n < length [m mod Suc n..<n] ∨
       (n # [0..<m mod Suc n]) ! (n − length [m mod Suc n..<n]) = n
    by argo
  hence m mod Suc n = 0
  proof
    assume n < length [m mod Suc n..<n]
    then show m mod Suc n = 0
      by simp
  next
    assume B: (n # [0..<m mod Suc n]) ! (n − length [m mod Suc n..<n]) = n
    show m mod Suc n = 0
    proof (cases n − length [m mod 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*:
$x \in set\ xs \Longrightarrow count\text{-}list\ xs\ x > 0$
by (*meson count-list-0-iff gr0I*)

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

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

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

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

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

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

count-list (*list-slice xs i j*) $x > 0 \Longrightarrow \exists\, k <$ *length xs*. $i \le k \wedge k < j \wedge xs\ !\ k = x$
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 $y = x \Longrightarrow$ *count-list* (*xs* @ [*y*]) x = *Suc* (*card ?A*)
by (*simp add*: *snoc*)
moreover
have $y = x \Longrightarrow$ *?B* = *insert* (*length xs*) *?A*
by (*metis* (*mono-tags*, *lifting*) ‹*?B* $-$ {*length xs*} = *?A*› *insert-Diff length-append-singleton lessI mem-Collect-eq nth-append-length*)
with *card-insert-disjoint*[*OF* - ‹*length xs* $\notin$ -›]
have $y = x \Longrightarrow$ *card ?B* = *Suc* (*card ?A*)
by *simp*
ultimately have $y = x \Longrightarrow$ *?case*
by *simp*
}
moreover
have $y \ne x \Longrightarrow$ *count-list* (*xs* @ [*y*]) x = *card ?A*
by (*simp add*: *snoc*)
hence $y \ne x \Longrightarrow$ *?case*
using ‹*?B* $-$ {*length xs*} = *?A*› **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* ! $k \le a\}$ = *card* $\{k.\ k <$ *length s* $\wedge$ *s* ! $k < a\}$
+ *count-list s a*

```
proof -
  let ?A = {k. k < length s ∧ s ! k ≤ a}
  let ?B = {k. k < length s ∧ s ! k < a}
  let ?C = {k. k < length s ∧ s ! k = a}

  have ?B ∩ ?C = {}
    by blast
  hence card (?B ∪ ?C) = card ?B + count-list s a
    by (simp add: card-Un-disjoint count-list-card)
  moreover
  have ?A = ?B ∪ ?C
  proof safe
    fix x
    assume s ! x ≤ a s ! x ≠ a
    then show s ! x < a
      by simp
  next
    fix x
    assume s ! x < a
    then show s ! x ≤ a
      by simp
  qed
  hence card ?A = card (?B ∪ ?C)
    by simp
  ultimately show ?thesis
    by simp
qed

lemma card-less-idx-upper-strict:
  fixes s :: 'a :: linorder list
  assumes a ∈ set s
  shows card {k. k < length s ∧ s ! k < a} < length s
proof -
  have ∃ i < length s. s ! i = a
    by (meson assms in-set-conv-nth)
  then obtain i where P:
    i < length s s ! i = a
    by blast

  have {k. k < length s ∧ s ! k < a} ⊆ {0..<length s}
    using atLeastLessThan-iff by blast
  moreover
  have i ∈ {0..<length s}
    by (simp add: P(1))
  moreover
  have i ∉ {k. k < length s ∧ s ! k < a}
    by (simp add: P(2))
  ultimately have {k. k < length s ∧ s ! k < a} ⊂ {0..<length s}
    by blast
```

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

lemma *card-less-idx-upper*:
  shows *card* $\{k.\ k < \mathit{length}\ s \wedge s\ !\ k < a\} \le \mathit{length}\ s$
  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 $s :: 'a :: \mathit{linorder}\ \mathit{list}$
  shows *card* $\{i.\ i < \mathit{length}\ s \wedge s\ !\ i < a\} + \mathit{count\text{-}list}\ s\ a \le \mathit{length}\ s$
proof −
  let $?X = \{i.\ i < \mathit{length}\ s \wedge s\ !\ i < a\}$
  let $?Y = \{i.\ i < \mathit{length}\ s \wedge s\ !\ i = a\}$

  have $?X \cap ?Y = \{\}$
    by *blast*
  hence *card* $(?X \cup ?Y) = \mathit{card}\ ?X + \mathit{card}\ ?Y$
    by (*simp add*: *card-Un-disjoint*)
  moreover
  have *card* $?Y = \mathit{count\text{-}list}\ s\ a$
    by (*simp add*: *count-list-card*)
  moreover
  have $?X \cup ?Y \subseteq \{0..<\mathit{length}\ s\}$
    by (*simp add*: *subset-iff*)
  hence *card* $(?X \cup ?Y) \le \mathit{length}\ s$
    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 *card* $\{k.\ k < \mathit{length}\ xs \wedge xs\ !\ k < c\} < \mathit{length}\ xs$
shows $c \le xs\ !\ \mathit{card}\ \{k.\ k < \mathit{length}\ xs \wedge xs\ !\ k < c\}$
  using *assms*
proof (*induct xs*)
  case *Nil*
  then show *?case*
    by *simp*
next
  case (*Cons a xs*)
  note $IH = \mathit{this}$

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

```
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 $i < card\ \{k.\ k < length\ xs \wedge xs\ !\ k < c\}$
shows $xs\ !\ i < c$
proof (*rule ccontr*)
assume $\neg\ xs\ !\ i < c$
hence $i \notin \{k.\ k < length\ xs \wedge xs\ !\ k < c\}$
by *simp*
hence $\forall\, k < length\ xs.\ i \leq k \longrightarrow k \notin \{k.\ k < length\ xs \wedge xs\ !\ k < c\}$
using *assms*(*1*) *sorted-iff-nth-mono* **by** *fastforce*
hence $\{k.\ k < length\ xs \wedge xs\ !\ k < c\} \subseteq \{0..<i\}$
by *fastforce*
moreover
have $card\ \{0..<i\} = i$
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 $card\ \{k.\ k < length\ xs \wedge xs\ !\ k < c\} + i < length\ xs$
and $count\text{-}list\ xs\ c \leq i$
shows $xs\ !\ (card\ \{k.\ k < length\ xs \wedge xs\ !\ k < c\} + i) > c$
proof −
let $?A = \{k.\ k < length\ xs \wedge xs\ !\ k < c\}$
have $xs\ !\ (card\ ?A + i) \geq c$
using *assms*(*1*) *assms*(*2*) *sorted-nth-le-gen* **by** *blast*
hence $xs\ !\ (card\ ?A + i) = c \vee xs\ !\ (card\ ?A + i) > c$
by *force*
then show *?thesis*
proof
assume $xs\ !\ (card\ ?A + i) > c$
then show *?thesis* **.**
next
assume $xs\ !\ (card\ ?A + i) = c$

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

have *P2*: ∀ *k* < *length xs*. *k* < *card ?A* + *Suc i* ⟶ *xs* ! *k* ≤ *c*
  by (*metis* (*mono-tags*, *lifting*) *Suc-leI* ‹*xs* ! (*card ?A* + *i*) = *c*› *add-Suc-right*
add-le-cancel-left *assms*(*1*,*2*) *plus-1-eq-Suc*
sorted-nth-mono)

have *P3*: ∀ *x* ∈ {*card ?A*..<*card ?A* + *Suc i*}. *xs* ! *x* = *c*
proof *safe*
  fix *x*
  assume *x* ∈ {*card ?A*..<*card ?A* + *Suc i*}
  hence *A*: *card ?A* ≤ *x x* < *card ?A* + *Suc i*
    by *simp+*

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

have {*card ?A*..<*card ?A* + *Suc i*} ⊆ {*k*. *k* < *length xs* ∧ *xs* ! *k* = *c*}
proof
  fix *x*
  assume *A*: *x* ∈ {*card ?A*..<*card ?A* + *Suc i*}
  have *x* < *card ?A* + *Suc i*
    using *A* **by** *simp+*
  hence *x* < *length xs*
    using *assms*(*2*) **by** *linarith*
  moreover
  have *xs* ! *x* = *c*
    using *P3 A* **by** *blast*
  ultimately show *x* ∈ {*k*. *k* < *length xs* ∧ *xs* ! *k* = *c*}
    by *blast*
qed
hence *count-list xs c* ≥ *card* {*card ?A*..<*card ?A* + *Suc i*}
  using *count-list-card*[*of xs c*] *card-mono*
 by (*metis* (*mono-tags*, *lifting*) ‹*xs* ! (*card ?A* + *i*) = *c*› *assms*(*2*) *card-ge-0-finite*
count-in
nth-mem)

moreover
have *card* {*card ?A*..<*card ?A* + *Suc i*} = *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
$rank\ s\ x\ i \equiv count\text{-}list\ (take\ i\ s)\ x$

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*:
$a \neq x \Longrightarrow rank\ (a\ \#\ xs)\ x\ (Suc\ i) = rank\ xs\ x\ i$
by (*simp add*: *rank-def*)

5.2 Counting Properties

lemma *rank-length*:
$rank\ xs\ x\ (length\ xs) = count\text{-}list\ xs\ x$
by (*simp add*: *rank-def*)

lemma *rank-gre-length*:
$length\ xs \leq n \Longrightarrow rank\ xs\ x\ n = count\text{-}list\ xs\ x$
by (*simp add*: *rank-def*)

lemma *rank-not-in*:
$x \notin set\ xs \Longrightarrow rank\ xs\ x\ i = 0$
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*:
  $i \leq j \Longrightarrow$
   $rank\ xs\ x\ j = rank\ xs\ x\ i + card\ \{k.\ k < length\ xs \wedge i \leq k \wedge k < j \wedge xs\ !\ k = x\}$
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 ‹$i \leq j$› *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 $card\ (?Y \cup ?Z) = card\ ?Y + card\ ?Z$
by (*simp add*: *card-Un-disjoint*)
ultimately show *?thesis*
by *presburger*
qed

5.3 Bound Properties

lemma *rank-lower-bound*:
assumes $k < rank\ xs\ x\ i$
shows $k < i$
proof −
from *rank-card-spec*[*of xs x i*]
have $rank\ xs\ x\ i = card\ \{j.\ j < length\ xs \wedge j < i \wedge xs\ !\ j = x\}$ **.**
hence $k < card\ \{j.\ j < length\ xs \wedge j < i \wedge xs\ !\ j = x\}$
using *assms* **by** *presburger*
moreover
{
have $i \leq length\ xs \vee length\ xs < i$
using *linorder-not-less* **by** *blast*
moreover
have $i \leq length\ xs \Longrightarrow \{j.\ j < length\ xs \wedge j < i \wedge xs\ !\ j = x\} \subseteq \{0..<i\}$
using *atLeast0LessThan* **by** *blast*
hence $i \leq length\ xs \Longrightarrow card\ \{j.\ j < length\ xs \wedge j < i \wedge xs\ !\ j = x\} \leq i$
using *subset-eq-atLeast0-lessThan-card* **by** *presburger*
moreover
have $length\ xs < i \Longrightarrow \{j.\ j < length\ xs \wedge j < i \wedge xs\ !\ j = x\} \subseteq \{0..<length\ xs\}$
using *atLeast0LessThan* **by** *blast*
hence $length\ xs < i \Longrightarrow card\ \{j.\ j < length\ xs \wedge j < i \wedge xs\ !\ j = x\} \leq length\ xs$
using *subset-eq-atLeast0-lessThan-card* **by** *presburger*
hence $length\ xs < i \Longrightarrow card\ \{j.\ j < length\ xs \wedge j < i \wedge xs\ !\ j = x\} \leq i$
by *linarith*
ultimately have $card\ \{j.\ j < length\ xs \wedge j < i \wedge xs\ !\ j = x\} \leq i$
by *blast*
}
ultimately show *?thesis*
using *dual-order.strict-trans1* **by** *blast*
qed

corollary *rank-Suc-ex*:
assumes $k < rank\ xs\ x\ i$
shows $\exists l.\ i = Suc\ l$
by (*metis Nat.lessE assms rank-lower-bound*)

lemma *rank-upper-bound*:
$[\![i < length\ xs;\ xs\ !\ i = x]\!] \Longrightarrow rank\ xs\ x\ i < count\text{-}list\ xs\ x$

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.prems*(*2*) *count-in list.set-intros*(*1*) *nth-Cons-0 rank-0*)
  next
    case (*Suc n*)
    then show *?thesis*
    by (*metis Cons.hyps Cons.prems Suc-less-eq length-Cons nth-Cons-Suc rank-cons-diff*

        rank-cons-same rank-length)
  qed
qed

lemma *rank-idx-mono*:
  $i \le j \Longrightarrow$ *rank xs x i* $\le$ *rank xs x j*
proof (*cases* $i = j$)
  assume $i = j$
  then show *?thesis*
    by *simp*
next
  assume $i \le j$ $i \ne j$
  hence $i < j$
    using *antisym-conv2* **by** *blast*
  hence *prefix xs j* = *prefix xs i* @ *list-slice xs i j*
    by (*metis* ‹$i \le j$› *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*:
  ⟦$i <$ *length xs*; $i < j$; *xs* ! $i = x$⟧ $\Longrightarrow$ *rank xs x i* $<$ *rank xs x j*
proof −
  let *?X* = {*k*. $k <$ *length xs* $\wedge$ $i \le k$ $\wedge$ $k < j$ $\wedge$ *xs* ! $k = x$}
  assume $i <$ *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 *card ?X > 0*
  using *card-gt-0-iff* **by** *fastforce*
ultimately show *?thesis*
  by *linarith*
qed

lemma *rank-upper-bound-gen*:
rank xs x i $\leq$ *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 < length xs*
shows *i = card* {*j. j < length xs* $\wedge$ *xs ! j < xs ! i*} + *rank xs* (*xs ! i*) *i*
proof −

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

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

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

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

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

  show *xs ! k < xs ! i*
  proof (*rule ccontr*)
    assume ¬ *xs ! k < xs ! i*
    with ‹*xs ! k* $\leq$ *xs !* (*Min ?B*)› ‹*xs !* (*Min ?B*) = *xs ! i*›
    have *xs ! k = xs ! i*
      by *order*
    with ‹*k < Min ?B*› ‹*Min ?B < length xs*›
    have *k* $\in$ *?B*

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

have ?A = {0..<Min ?B}
proof (intro equalityI subsetI)
  fix x
  assume x ∈ ?A
  hence x < length xs xs ! x < xs ! i
    by blast+
  hence xs ! x < xs ! Min ?B
    using ‹xs ! Min ?B = xs ! i› by simp
  hence x < Min ?B
    using assms(1) ‹x < length xs› ‹Min ?B < length xs›
    by (meson dual-order.strict-iff-not not-le-imp-less sorted-nth-mono)
  then show x ∈ {0..<Min ?B}
    using atLeastLessThan-iff by blast
next
  fix x
  assume x ∈ {0..<Min ?B}
  with P ‹Min ?B < length xs›
  show x ∈ ?A
    by auto
qed
moreover
{
  let ?C = {j. j < length xs ∧ j < i ∧ xs ! j = xs ! i}
  from rank-card-spec[of xs xs ! i i]
  have rank xs (xs ! i) i = card ?C .
  moreover
  have ?C = {Min ?B..<i}
  proof (intro equalityI subsetI)
    fix x
    assume x ∈ ?C
    hence x < length xs x < i xs ! x = xs ! i
      by blast+
    hence Min ?B ≤ x
      by simp
    with ‹x < i›
    show x ∈ {Min ?B..<i}
      using atLeastLessThan-iff by blast
  next
    fix x
    assume x ∈ {Min ?B..<i}
    hence Min ?B ≤ x x < i
```

```
      using atLeastLessThan-iff by blast+
    moreover
    have xs ! x = xs ! i
    proof −
      have xs ! x ≤ xs ! i
        using assms(1,2) ‹x < i›
        by (simp add: sorted-wrt-nth-less)
      moreover
      have xs ! Min ?B ≤ xs ! x
        using assms(1,2) ‹Min ?B ≤ x› ‹x < i›
        by (meson order.strict-trans sorted-iff-nth-mono)
      ultimately show ?thesis
        using ‹xs ! Min ?B = xs ! i› by order
    qed
    ultimately show x ∈ ?C
      using assms(2) by fastforce
  qed
  ultimately have rank xs (xs ! i) i = card {Min ?B..<i}
    by presburger
 }
 ultimately show ?thesis
   by (simp add: ‹Min ?B ≤ i›)
qed

lemma sorted-rank:
 assumes sorted xs
 and     i < length xs
 and     xs ! i = a
shows rank xs a i = i − card {k. k < length xs ∧ 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 < length xs
 and     xs ! i < a
shows rank xs a i = 0
proof −
 have rank xs a i = card {k. k < length xs ∧ k < i ∧ xs ! k = a}
   by (simp add: rank-card-spec)
 moreover
 have {k. k < length xs ∧ k < i ∧ 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 < length xs
```

and $xs \,!\, i > a$
shows $rank\ xs\ a\ i = count\text{-}list\ xs\ a$
proof −
  let $?A = \{k.\ k < length\ xs \land k < i \land xs \,!\, k = a\}$
  have $rank\ xs\ a\ i = card\ ?A$
    by (*simp add*: *rank-card-spec*)
  moreover
  let $?B = \{k.\ k < length\ xs \land k \geq i \land xs \,!\, k = a\}$
  let $?C = \{k.\ k < length\ xs \land xs \,!\, k = a\}$
  {
    have $?A \cup ?B = ?C$
    proof *safe*
      fix x
      assume $\neg\ i \leq x$
      then show $x < i$
        using *linorder-le-less-linear* **by** *blast*
    qed
    moreover
    have $?B = \{\}$
    proof −
      have $\forall k < length\ xs.\ k \geq i \longrightarrow xs \,!\, k > a$
        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 $select :: 'a\ list \Rightarrow 'a \Rightarrow nat \Rightarrow nat$
  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*:
  $x \notin set\ xs \Longrightarrow select\ xs\ x\ i = length\ xs$
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.prems* **by** *fastforce*
  next
    case (*Suc n*)
    then show *?thesis*
      using *Cons.hyps Cons.prems* **by** *force*
  qed
qed

lemma *select-length-imp-count-list-less*:
  $select\ xs\ x\ i = length\ xs \Longrightarrow count\text{-}list\ xs\ x \leq i$
  by (*induct rule*: *select.induct*[*of - xs x i*]; *simp split*: *if-splits*)

lemma *select-Suc-length*:
  $select\ xs\ x\ i = length\ xs \Longrightarrow select\ xs\ x\ (Suc\ i) = length\ xs$
  by (*induct rule*: *select.induct*[*of - xs x i*]; *clarsimp split*: *if-splits*)

7.2 List Properties

lemma *select-cons-neq*:
  $[\![select\ xs\ x\ i = j;\ x \neq a]\!] \Longrightarrow select\ (a\ \#\ xs)\ x\ i{=}\ Suc\ j$
  by (*cases i*; *simp*)

lemma *cons-neq-select*:
  $[\![select\ (a\ \#\ xs)\ x\ i = Suc\ j;\ x \neq a]\!] \Longrightarrow select\ xs\ x\ i = j$
  by (*cases i*; *simp*)

lemma *cons-eq-select*:
  $select\ (x\ \#\ xs)\ x\ (Suc\ i) = Suc\ j \Longrightarrow select\ xs\ x\ i = j$
  by *simp*

lemma *select-cons-eq*:
  $select\ xs\ x\ i = j \Longrightarrow select\ (x\ \#\ xs)\ x\ (Suc\ i) = Suc\ j$
  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:
  ⟦j < length xs; count-list (take (Suc j) xs) x = Suc 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)
  then show ?case
  proof (cases j)
    case 0
    then show ?thesis
      using 2.prems(3) by auto
  next
    case (Suc n)

    have xs ! n = x
      using 2.prems(3) Suc by auto
    moreover
    have n < length xs
      using 2.prems(1) Suc by auto
    moreover
    have x ≠ a
    proof (rule ccontr)
      assume ¬ x ≠ a
      hence x = a
        by blast
      moreover
      have count-list (take (Suc n) xs) x > 0
        by (simp add: ‹n < length xs› ‹xs ! n = x› take-Suc-conv-app-nth)
      ultimately show False
        using 2.prems(2) Suc by auto
    qed
    moreover
    have count-list (take (Suc n) xs) x = Suc 0
      using 2.prems(2) Suc calculation(3) by auto
    ultimately have select xs x 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.prems*(*2*) *3.prems*(*3*) **by** *force*
  next
    case (*Suc n*)
    then show *?thesis*
     by (*metis 3.hyps 3.prems Suc-inject Suc-less-eq add.right-neutral add-Suc-right count-list.simps*(*2*) *length-Cons nth-Cons-Suc plus-1-eq-Suc select.simps*(*3*)
       take-Suc-Cons)
  qed
qed

lemma *nth-select-alt*:
  $[\![j < \textit{length xs}; \textit{count-list} (\textit{take j xs})\ x = i; \textit{xs} ! j = x]\!]$
   $\Longrightarrow$ *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.prems*(*3*) **by** *auto*
  next
    case (*Suc n*)
    then show *?thesis*
     by (*metis 2.hyps 2.prems 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.prems*(*2*) **by** *auto*
  next
    case (*Suc n*)
    then show *?thesis*
     by (*metis 3.hyps 3.prems 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.prems*(*1*) *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.prems*(*1*) *2.prems*(*2*) **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.prems*(*1*) *nat.simps*(*3*) *select-cons-eq select-cons-neq*)
  next
    case (*Suc n*)
    then show *?thesis*
    by (*metis 3.hyps 3.prems 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

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.prems(1) order.strict-iff-not by fastforce
  next
    case (Suc n)
    then show ?thesis
      by (metis 2.prems(1) 2.prems(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.prems(1) nat.simps(3) select-cons-eq select-cons-neq)
  next
    case (Suc n)
    then show ?thesis
      by (metis 3.prems 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 $x \notin$ *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 $i <$ *count-list xs x*
shows *select xs x i* $=$ *card* $\{j.\ j <$ *length xs* $\wedge$ *xs* ! $j < x\} + i$
using *assms*
proof (*induct rule*: *select.induct*[*of - xs x i*])
case (*1 uu uv*)
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 $x \neq a \Longrightarrow$ *?case*
proof −
assume $x \neq a$
hence $0 <$ *count-list xs x*
using *IH*(*3*) **by** *fastforce*
with *IH*(*1*)[*OF* ‹$x \neq a$› ‹*sorted xs*›]
have *select xs x 0* $=$ *card* $\{j.\ j <$ *length xs* $\wedge$ *xs* ! $j < x\}$
by *simp*
moreover
{
from *in-count*[*OF* ‹$0 <$ *count-list xs x*›]
have $x \in$ *set xs* **.**
with *IH*(*2*) ‹$x \neq a$›
have $a < x$
by (*simp add*: *order-less-le*)
have $\{j.\ j <$ *length* (*a* # *xs*) $\wedge$ (*a* # *xs*) ! $j < x\} =$
$\{0\} \cup$ *Suc* ‘ $\{j.\ j <$ *length xs* $\wedge$ *xs* ! $j < x\}$
proof (*safe*)
show (*a* # *xs*) ! $0 < x$

by (*simp add*: ‹$a < x$›)
next
fix y
assume $y < \mathit{length}\ xs$
then show $\mathit{Suc}\ y < \mathit{length}\ (a \# xs)$
by *simp*
next
fix y
assume $y < \mathit{length}\ xs\ xs\ !\ y < x$
then show $(a \# xs)\ !\ \mathit{Suc}\ y < x$
by *simp*
next
fix j
assume A: $j \notin \mathit{Suc}\ `\ \{v.\ v < \mathit{length}\ xs \wedge xs\ !\ v < x\}\ j < \mathit{length}\ (a \# xs)$
$(a \# xs)\ !\ j < x$

have $\exists k.\ j = \mathit{Suc}\ k \Longrightarrow \mathit{False}$
proof −
assume $\exists k.\ j = \mathit{Suc}\ k$
then obtain k **where**
$j = \mathit{Suc}\ k$
by *blast*
hence B: $k < \mathit{length}\ xs\ xs\ !\ k < x\ k \notin \{v.\ v < \mathit{length}\ xs \wedge xs\ !\ v < x\}$
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* $(\mathit{Suc}\ `\ \{j.\ j < \mathit{length}\ xs \wedge xs\ !\ j < x\})$
by *simp*
moreover
have $\{0\} \cap \mathit{Suc}\ `\ \{j.\ j < \mathit{length}\ xs \wedge xs\ !\ j < x\} = \{\}$
by *blast*
ultimately have
$\mathit{card}\ (\{0\} \cup \mathit{Suc}\ `\ \{j.\ j < \mathit{length}\ xs \wedge xs\ !\ j < x\}) =$
$\mathit{Suc}\ (\mathit{card}\ (\mathit{Suc}\ `\ \{j.\ j < \mathit{length}\ xs \wedge xs\ !\ j < x\}))$
using *card-Un-disjoint*[*of* $\{0\}$ $\mathit{Suc}\ `\ \{j.\ j < \mathit{length}\ xs \wedge xs\ !\ j < x\}$] **by**
simp
}
ultimately have
$\mathit{card}\ \{j.\ j < \mathit{length}\ (a \# xs) \wedge (a \# xs)\ !\ j < x\} =$
$\mathit{Suc}\ (\mathit{card}\ (\mathit{Suc}\ `\{j.\ j < \mathit{length}\ xs \wedge xs\ !\ j < x\}))$

by *presburger*
hence *card* $\{j.\ j < length\ (a \# xs) \wedge (a \# xs)\ !\ j < x\} =$
Suc $(card\ \{j.\ j < length\ xs \wedge xs\ !\ j < x\})$
by (*simp add*: *card-image*)
}
moreover
have *select* $(a \# xs)\ x\ 0 = Suc\ (select\ xs\ x\ 0)$
using ‹$x \neq a$› *select.simps*(*2*)[*of a xs x*] **by** *auto*
ultimately show *?thesis*
by *simp*
qed
moreover
have $x = a \Longrightarrow$ *?case*
proof −
assume $x = a$
with *IH*(*2*)
have $\{j.\ j < length\ (a \# xs) \wedge (a \# xs)\ !\ j < x\} = \{\}$
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 $a \leq x$
by (*metis IH*(*3*−) *Suc-less-eq2 count-list.simps*(*2*) *in-count order-refl sorted-simps*(*2*)
zero-less-Suc)

have $x = a \vee x \neq a$
by *blast*
moreover
have $x = a \Longrightarrow$ *?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* $\{j.\ j < length\ xs \wedge xs\ !\ j < x\} + i$ **.**
moreover

from *select.simps(3)[of a xs x i]* ‹$x = a$›
have *select* $(a \# xs)$ x $(Suc\ i) = Suc\ (select\ xs\ x\ i)$
by *simp*
moreover
from ‹$a \leq x$› ‹$x = a$› *IH(3)*
have $\{j.\ j < length\ (a \# xs) \wedge (a \# xs) ! j < x\} = \{\}$
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 $card\ \{j.\ j < length\ (a \# xs) \wedge (a \# xs) ! j < x\} = 0$
by *simp*
moreover
from ‹$a \leq x$› ‹$x = a$› *IH(3)*
have $\{j.\ j < length\ xs \wedge xs ! j < x\} = \{\}$
using *nth-mem* **by** *fastforce*
hence $card\ \{j.\ j < length\ xs \wedge xs ! j < x\} = 0$
by *simp*
ultimately show *?thesis*
by *simp*
qed
moreover
have $x \neq a \Longrightarrow$ *?case*
proof −
assume $x \neq a$
hence $Suc\ i < count\text{-}list\ xs\ x$
using *IH(4)* **by** *force*
with *IH(2)[OF* ‹$x \neq a$› ‹*sorted xs*›*]*
have $select\ xs\ x\ (Suc\ i) = card\ \{j.\ j < length\ xs \wedge xs ! j < x\} + Suc\ i$ **.**
moreover
from ‹$x \neq a$› *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 $\{j.\ j < length\ (a \# xs) \wedge (a \# xs) ! j < x\} =$
$\{0\} \cup Suc\ `\ \{j.\ j < length\ xs \wedge xs ! j < x\}$
proof *safe*
show $(a \# xs) ! 0 < x$
using ‹$a \leq x$› ‹$x \neq a$› **by** *auto*
next
fix y
assume $y < length\ xs$ $xs ! y < x$
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 *k*
assume *A*: $k \notin Suc$ ` $\{j.\ j < length\ xs \wedge xs\ !\ j < x\}$ $k \notin \{\}$ $k < length\ (a$
$\#\ xs)$
$(a\ \#\ xs)\ !\ k < x$

have $\exists\, l.\ k = Suc\ l \Longrightarrow False$
proof −
assume $\exists\, l.\ k = Suc\ l$
then obtain *l* **where**
$k = Suc\ l$
by *blast*
hence $l \notin \{j.\ j < length\ xs \wedge xs\ !\ j < x\}$ $l < length\ xs$ $xs\ !\ l < x$
using *A* **by** *simp-all*
then show *False*
by *blast*
qed
then show $k = 0$
using *not0-implies-Suc* **by** *blast*
qed
moreover
have *finite* $\{0\}$
by *blast*
moreover
have *finite* $(Suc$ ` $\{j.\ j < length\ xs \wedge xs\ !\ j < x\})$
by *simp*
moreover
have $\{0\} \cap Suc$ ` $\{j.\ j < length\ xs \wedge xs\ !\ j < x\} = \{\}$
by *blast*
ultimately have
$card\ (\{j.\ j < length\ (a\ \#\ xs) \wedge (a\ \#\ xs)\ !\ j < x\}) =$
$Suc\ (card\ (Suc$ ` $\{j.\ j < length\ xs \wedge xs\ !\ j < x\}))$
by *simp*
hence $card\ (\{j.\ j < length\ (a\ \#\ xs) \wedge (a\ \#\ xs)\ !\ j < x\}) =$
$Suc\ (card\ \{j.\ j < length\ xs \wedge xs\ !\ j < x\})$
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 $i < count\text{-}list\ xs\ x$
shows $select\ xs\ x\ i = select\ xs\ x\ 0 + i$
using *assms*(*1*) *assms*(*2*) *sorted-select* **by** *fastforce*

corollary *select-sorted-0*:
  assumes *sorted xs*
  and $0 < $ *count-list xs x*
shows *select xs x 0* = *card* {*j*. *j* < *length xs* ∧ *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*:
  rank s x i = *count* (*mset* (*take i s*)) *x*
  by (*simp add*: *count-list-eq-count rank-def*)

8.1.2 Select Correctness

lemma *select-spec*:
  select s x i = *j*
    ⟹ (*j* < *length s* ∧ *rank s x j* = *i*) ∨ (*j* = *length s* ∧ *count-list s x* ≤ *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*:
  select s x i ≤ *length s* ∧
  (*select s x i* < *length s* ⟶ *rank s x* (*select s x i*) = *i*) ∧
  (*select s x i* = *length s* ⟶ *count-list s x* ≤ *i*)
proof −
  have *select s x i* ≤ *length s*
    by (*simp add*: *select-max*)
  moreover
  have *select s x i* < *length s* ⟶ *rank s x* (*select s x i*) = *i*
    by (*metis rank-def select-nth-alt*)
  moreover
  have *select s x i* = *length s* ⟶ *count-list s x* ≤ *i*
    by (*simp add*: *select-length-imp-count-list-less*)

ultimately show *?thesis*
  by *blast*
qed

8.2 Rank and Select

lemma *rank-select*:
  $select\ xs\ x\ i < length\ xs \Longrightarrow rank\ xs\ x\ (select\ xs\ x\ i) = i$
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*:
  $i < rank\ xs\ x\ j \Longrightarrow select\ xs\ x\ i < length\ xs$
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.prems*]
  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.prems 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 $a \in$ *set xs*

have *card* $\{k.\ k <$ *length xs* $\wedge$ *xs* ! $k < a\} <$ *length xs*
using ‹$a \in$ *set xs*› *card-less-idx-upper-strict* **by** *blast*
moreover
have $\exists k.\ i =$ *card* $\{k.\ k <$ *length xs* $\wedge$ *xs* ! $k < a\} + k$
using *assms*(*2*) *le-iff-add* **by** *blast*
then obtain k **where**
$i =$ *card* $\{k.\ k <$ *length xs* $\wedge$ *xs* ! $k < a\} + k$
by *blast*
moreover
have $k <$ *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* = *sort ys*
shows $\exists f.$ *bij-betw f* $\{j.\ j <$ *length ys* $\wedge$ *ys* ! $j < x\}$ $\{j.\ j <$ *length xs* $\wedge$ *xs* ! $j < x\}$
proof −

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

have *mset ys* = *mset xs*
by (*simp add*: *assms*)
with *permutation-Ex-bij*[*of ys xs*]
obtain f **where**

```
    bij-betw f {..<length ys} {..<length xs}
    (∀ i<length ys. ys ! i = xs ! f i)
    by blast
  moreover
  have ?A ⊆ {..<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
      ∀ x∈{..<length ys}. f x ∈ {..<length xs} ∧ g (f x) = x
      ∀ y∈{..<length xs}. g y ∈ {..<length ys} ∧ f (g y) = y
      by blast
    then show a ∈ 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 ∧ s ! k = c } = {k. k < length s ∧ (∃ xs. suffix s k = c # xs)}
  using suffix-cons-ex by fastforce

lemma suffix-hd-set-less:
  {k. k < length s ∧ s ! k < c } = {k. k < length s ∧ suffix s k < [c]}
  using suffix-cons-ex by fastforce

lemma select-nth-suffix-start1:
  assumes i < card {k. k < length s ∧ (∃ as. suffix s k = a # as)}
  and     xs = sort s
shows select xs a i = card {k. k < length s ∧ suffix s k < [a]} + i
proof −
```

let $?A = \{k.\ k < length\ s \wedge (\exists as.\ suffix\ s\ k = a\ \#\ as)\}$
let $?A' = \{k.\ k < length\ s \wedge s\ !\ k = a\}$

have $?A = ?A'$
using *suffix-cons-Suc* **by** *fastforce*
with *assms(1)*
have $i < count\text{-}list\ s\ a$
by (*simp add*: *count-list-card*)
hence $i < count\text{-}list\ xs\ a$
by (*metis assms(2) count-list-perm mset-sort*)
moreover
let $?B = \{k.\ k < length\ s \wedge suffix\ s\ k < [a]\}$
let $?B' = \{k.\ k < length\ s \wedge s\ !\ k < a\}$
let $?B'' = \{k.\ k < length\ xs \wedge xs\ !\ k < a\}$
{
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*
by (*simp add*: ‹$xs = sort\ s$› *sorted-select*)
qed

lemma *select-nth-suffix-start2*:
assumes $card\ \{k.\ k < length\ s \wedge (\exists as.\ suffix\ s\ k = a\ \#\ as)\} \leq i$
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 $?A = \{k.\ k < length\ s \wedge (\exists as.\ suffix\ s\ k = a\ \#\ as)\}$
let $?A' = \{k.\ k < length\ s \wedge s\ !\ k = a\}$
have $?A = ?A'$
using *suffix-cons-Suc* **by** *fastforce*
with *assms(1)*
show $count\text{-}list\ s\ a \leq i$
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*) ≤ *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 ∃ *k l*. *k* < *l* ∧ *sa s* ! *k* = *Suc* (*sa s* ! *i*) ∧ *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-sa-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-nth-less-mono[OF strict-sorted-imp-sorted[OF sa-g-sorted[of s]]]
    by (metis length-map not-less-iff-gr-or-eq nth-map sa-length)
  with ‹sa s ! k = -› ‹sa s ! l = -›
  show ?thesis
    by blast
qed

lemma sorted-map-nths-sa:
  sorted (map (nth s) (sa s))
proof (intro sorted-wrt-mapI)
  fix i j
  assume i < j j < length (sa s)
  hence suffix s (sa s ! i) < suffix s (sa s ! j)
    using sa-g-sorted sorted-wrt-mapD by blast
  moreover
  have suffix s (sa s ! i) = s ! (sa s ! i) # suffix s (Suc (sa s ! i))
     by (metis ‹i < j› ‹j < length (sa s)› order.strict-trans sa-length sa-nth-ex
suffix-cons-Suc)
  moreover
  have suffix s (sa s ! j) = s ! (sa s ! j) # suffix s (Suc (sa s ! j))
    by (metis ‹j < length (sa s)› sa-length sa-nth-ex suffix-cons-Suc)
  ultimately show s ! (sa s ! i) ≤ s ! (sa s ! j)
    by fastforce
qed

lemma perm-map-nths-sa:
  s <~~> map (nth s) (sa s)
  by (metis map-nth mset-map sa-g-permutation)

lemma sort-eq-map-nths-sa:
  sort s = map (nth s) (sa s)
  by (metis perm-map-nths-sa properties-for-sort sorted-map-nths-sa)

lemma sort-sa-nth:
  i < length s ⟹ sort s ! i = s ! (sa s ! i)
  by (simp add: sa-length sort-eq-map-nths-sa)

lemma inj-on-nth-sa-upt:
  assumes j ≤ length s l ≤ length s
shows inj-on (nth (sa s)) ({i..<j} ∪ {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 ‹$x \in \{i..<j\} \cup \{k..<l\}$› *assms(1) assms(2)* **by** *auto*
moreover
have $y <$ *length s*
  using ‹$y \in \{i..<j\} \cup \{k..<l\}$› *assms(1) assms(2)* **by** *auto*
ultimately show $x = y$
  by (*metis* ‹$sa\ s\ !\ x = sa\ s\ !\ y$› *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 ‹*s* = *xs* @ [*bot*]›
  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 Arays

```
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 $x < \mathit{length}\ s$
using *assms* **by** *simp*
next
fix x
assume $x < i$
then show $\mathit{suffix}\ s\ (\mathit{sa}\ s\ !\ x) < \mathit{suffix}\ s\ (\mathit{sa}\ s\ !\ i)$
using P **by** *auto*
next
fix x
assume Q: $x < \mathit{length}\ s\ \mathit{suffix}\ s\ (\mathit{sa}\ s\ !\ x) < \mathit{suffix}\ s\ (\mathit{sa}\ s\ !\ i)$
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 $i < \mathit{length}\ s$
shows $i = \mathit{card}\ \{j.\ j < \mathit{length}\ s \wedge \mathit{suffix}\ s\ j < \mathit{suffix}\ s\ (\mathit{sa}\ s\ !\ i)\}$
(**is** $i = \mathit{card}\ ?A$)
proof −
let $?i = \mathit{sa}\ s\ !\ i$
let $?v = s\ !\ ?i$
let $?B = \{j.\ j < \mathit{length}\ s \wedge \mathit{suffix}\ s\ (\mathit{sa}\ s\ !\ j) < \mathit{suffix}\ s\ ?i\}$

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

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

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

from *sa-card-s-index*[*OF assms*]
have $i = \textit{card } ?C$
by *simp*
moreover
have $?A \cap ?B = \{\}$
by *fastforce*
moreover
have $?C = ?A \cup ?B$
proof (*safe*)
fix *x*
assume $x < \textit{length } s$ $\textit{suffix } s\ x < \textit{suffix } s\ ?i$ $\neg s\ !\ x < s\ !\ ?i$
then show $s\ !\ x = s\ !\ ?i$
by (*metis Cons-less-Cons sa-nth-ex assms suffix-cons-Suc*)
next
fix *x*
assume $x < \textit{length } s$ $s\ !\ x < s\ !\ ?i$
then show $\textit{suffix } s\ x < \textit{suffix } s\ ?i$
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 $i < \textit{length } s$
shows $\textit{card } \{j.\ j < \textit{length } s \wedge s\ !\ (\textit{sa } s\ !\ j) < s\ !\ (\textit{sa } s\ !\ i)\} \le i$
(**is** $\textit{card } ?A \le i$)

```
proof –
  let ?B = {j. j < length s ∧ suffix s (sa s ! j) < suffix s (sa s ! i)}
  have ?A ⊆ ?B
  proof safe
    fix x
    assume x < length s s ! (sa s ! x) < s ! (sa s ! i)
    then show suffix s (sa s ! x) < suffix s (sa s ! i)
      by (metis Cons-less-Cons Cons-nth-drop-Suc assms sa-nth-ex)
  qed
  hence card ?A ≤ 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 < length s
  shows i = card {j. j < length s ∧ s ! (sa s ! j) < s ! (sa s ! i)}
            + rank (sort s) (s ! (sa s ! i)) i
proof –
  from sorted-card-rank-idx[of sort s i]
  have i = card {j. j < length (sort s) ∧ sort s ! j < sort s ! i} + rank (sort s)
(sort s ! i) i
    using assms by fastforce
  moreover
  have sort s ! i = s ! (sa s ! i)
    using assms sort-sa-nth by auto
  moreover
  have length (sort s) = 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 < length s
  shows i = card {j. j < length s ∧ s ! j < s ! (sa s ! i)}
            + rank (sort s) (s ! (sa s ! i)) i
proof –
  let ?A = {j. j < length s ∧ s ! j < s ! (sa s ! i)}
  and ?B = {j. j < length s ∧ s ! (sa s ! j) < s ! (sa s ! i)}
  from sa-card-rank-idx[OF assms]
  have i = card {j. j < length s ∧ s ! (sa s ! j) < s ! (sa s ! i)} +
          rank (sort s) (s ! (sa s ! i)) i .
  moreover
  have bij-betw (λx. sa s ! x)
          {j. j < length s ∧ s ! (sa s ! j) < s ! (sa s ! i)}
          {j. j < length s ∧ s ! j < s ! (sa s ! i)}
```

```
  proof (rule bij-betwI'; safe)
    fix x y
    assume x < length s y < 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 < length s
    then show sa s ! x < length s
      using sa-nth-ex by auto
  next
    fix x
    assume x < length s s ! x < s ! (sa s ! i)
    then show ∃xa ∈ {j. j < length s ∧ s ! (sa s ! j) < s ! (sa s ! i)}.  x = sa s !
xa
      using ex-sa-nth by blast
  qed
  hence card ?B = card ?A
    using bij-betw-same-card by blast
  ultimately show ?thesis
    by simp
qed

lemma sa-rank-nth:
  assumes i < length s
  shows rank (sort s) (s ! (sa s ! i)) i =
          card {j. j < length s ∧ s ! j = s ! (sa s ! i) ∧
                   suffix s j < suffix s (sa s ! i)}
proof −
  let ?i = sa s ! i
  let ?v = s ! ?i
  let ?A = {j. j < length s ∧ s ! j < ?v}
  let ?B = {j. j < length s ∧ s ! j = ?v ∧ suffix s j < suffix s ?i}

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

lemma sa-suffix-nth:
  assumes card {k. k < length s ∧ s ! k < c } + i < length s
  and     i < count-list s c
shows ∃as. suffix s (sa s ! (card {k. k < length s ∧ s ! k < c} + i)) = c # as
proof −
  let ?A = {k. k < length s ∧ 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 ∃ *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
hence *c* ≤ *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 *card* {*k*. *k* < *length s* ∧ *s* ! *k* < *c* } + *i* < *length s*
shows [*c*] ≤ *suffix s* (*sa s* ! (*card* {*k*. *k* < *length s* ∧ *s* ! *k* < *c*} + *i*))
proof (*cases i*)
case *0*
then show *?thesis*
using *assms sa-suffix-order-le* **by** *auto*
next
let *?x* = *card* {*k*. *k* < *length s* ∧ *s* ! *k* < *c* }
case (*Suc m*)
with *sorted-wrt-mapD*[*OF sa-g-sorted*, *of ?x ?x* + *i s*]
have *suffix s* (*sa s* ! *?x*) < *suffix s* (*sa s* ! (*?x* + *i*))
using *assms sa-length* **by** *auto*
moreover
have [*c*] ≤ *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* < *card* {*k*. *k* < *length s* ∧ *s* ! *k* < *c*}
shows ∀ *as*. *suffix s* (*sa s* ! *i*) < *c* # *as*
proof −
have *i* < *length s*
using *assms card-less-idx-upper dual-order.strict-trans1* **by** *blast*
hence ∃ *as*. *suffix s* (*sa s* ! *i*) = *s* ! (*sa s* ! *i*) # *as*
using *sa-nth-ex suffix-cons-Suc* **by** *blast*
moreover
have *i* < *card* {*k*. *k* < *length* (*sort s*) ∧ (*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*] ‹*i* < *length s*› *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* < *length xs*
  shows *rotate i xs* = *suffix xs i* @ *prefix xs i*
  by (*simp add*: *assms rotate-drop-take*)

lemma *rotate-last*:
  assumes *i* < *length xs*
  shows *last* (*rotate i xs*) = *xs* ! ((*i* + *length xs* − *Suc 0*) *mod* (*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*:
  map last (*map* (*λi*. *rotate i s*) (*sa s*)) = *bwt-sa s*
proof −
  have ∀ *x*∈*set* (*sa s*). *last* (*rotate x s*) = *s* ! ((*x* + *length s* − *Suc 0*) *mod 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* < *length s*
  and *j* < *length s*
  and *i* ≠ *j*
shows *rotate i s* ≠ *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 $i < length\ xs$
  and $j < length\ xs$
  and $suffix\ xs\ i < suffix\ xs\ j$
shows $suffix\ xs\ i\ @\ prefix\ xs\ i < suffix\ xs\ j\ @\ prefix\ xs\ j$
proof −
  from *suffix-less-ex*[*OF assms*]
  obtain *b c as bs cs* **where**
    $suffix\ xs\ i = as\ @\ b\ \#\ bs$
    $suffix\ xs\ j = as\ @\ c\ \#\ cs$
    $b < c$
    by *blast*
  hence $suffix\ xs\ i\ @\ prefix\ xs\ i = as\ @\ b\ \#\ bs\ @\ prefix\ xs\ i$
      $suffix\ xs\ j\ @\ prefix\ xs\ j = as\ @\ c\ \#\ cs\ @\ prefix\ xs\ j$
    by *simp-all*
  with ‹$b < c$›
  show *?thesis*
    by (*metis list-less-ex*)
qed

lemma *list-less-suffix-app-prefix-2*:
  assumes *valid-list xs*
  and $i < length\ xs$
  and $j < length\ xs$
  and $suffix\ xs\ i\ @\ prefix\ xs\ i < suffix\ xs\ j\ @\ prefix\ xs\ j$
shows $suffix\ xs\ i < suffix\ xs\ j$
  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 $i < length\ xs$
  and $j < length\ xs$
shows $suffix\ xs\ i < suffix\ xs\ j \longleftrightarrow$
    $suffix\ xs\ i\ @\ prefix\ xs\ i < suffix\ xs\ j\ @\ prefix\ xs\ j$
  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 $i < length\ xs$
  and $j < length\ xs$
shows $suffix\ xs\ i < suffix\ xs\ j \longleftrightarrow rotate\ i\ xs < rotate\ j\ xs$
  by (*simp add*: *assms list-less-suffix-app-prefix rotate-suffix-prefix*)

lemma (**in** *Suffix-Array-General*) *sorted-rotations*:
  assumes *valid-list s*
  shows $strict\text{-}sorted\ (map\ (\lambda i.\ rotate\ i\ s)\ (sa\ s))$

```
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 Canoncial 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-perm* :: (*'a* :: {*linorder*, *order-bot*}) *list* ⇒ *nat list*
where
bwt-perm s = *map* (λ*i*. (*i* + *length s* − *Suc 0*) *mod* (*length s*)) (*sa s*)

13.1 BWT Using Suffix Arrays

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

lemma *map-bwt-indexes-perm*:
fixes *s* :: (*'a* :: {*linorder*, *order-bot*}) *list*
shows *bwt-perm s* <~~> [*0*..<*length s*]
proof (*intro distinct-set-imp-perm*)
show *distinct* [*0*..<*length s*]
by *simp*
next
show *set* (*bwt-perm s*) = *set* [*0*..<*length s*]
unfolding *bwt-perm-def*
proof *safe*
fix *x*
assume *x* ∈ *set* (*map* (λ*i*. (*i* + *length s* − *Suc 0*) *mod length s*) (*sa s*))
hence *x* < *length s*
by (*metis* (*no-types*, *lifting*) *ex-map-conv length-map length-pos-if-in-set mod-less-divisor*
sa-length)
then show *x* ∈ *set* [*0*..<*length s*]
by *simp*
next
fix *x*
assume *x* ∈ *set* [*0*..<*length s*]
hence *x* ∈ {*0*..<*length s*}
using *atLeastLessThan-upt* **by** *blast*

have *x* ∈ (λ*i*. (*i* + *length s* − *Suc 0*) *mod length s*) ‘ {*0*..<*length s*}
proof (*cases Suc x* < *length s*)
assume *Suc x* < *length s*
hence (λ*i*. (*i* + *length s* − *Suc 0*) *mod length s*) (*Suc x*) = *x*
by *simp*
then show *?thesis*
using ‹*Suc x* < *length s*› **by** *force*
next
assume ¬ *Suc x* < *length s*
with ‹*x* ∈ {*0*..<*length s*}›
have *Suc x* = *length s*
by *simp*
hence (λ*i*. (*i* + *length s* − *Suc 0*) *mod length s*) *0* = *x*
using *diff-Suc-1'* *lessI mod-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 $x \in$ *set* (*map* (λi. (i + *length s* − *Suc 0*) *mod 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* < *length* (*bwt-perm s*) *j* < *length* (*bwt-perm s*) *i* ≠ *j*

have *bwt-perm s* ! *i* = (*sa s* ! *i* + *length s* − *Suc 0*) *mod* (*length s*)
using *A*(*1*) *bwt-perm-def* **by** *force*
moreover
have *bwt-perm s* ! *j* = (*sa s* ! *j* + *length s* − *Suc 0*) *mod* (*length s*)
using *A*(*2*) *bwt-perm-def* **by** *force*
moreover
have *sa s* ! *i* ≠ *sa s* ! *j*
by (*metis A bwt-perm-def length-map nth-eq-iff-index-eq sa-distinct*)

have (*sa s* ! *i* + *length s* − *Suc 0*) *mod* (*length s*) ≠
(*sa s* ! *j* + *length s* − *Suc 0*) *mod* (*length s*)
proof (*cases sa s* ! *i*)
case *0*
moreover have ‹*length s* > *0*›
using *A*(*2*) *bwt-perm-def sa-length* **by** *fastforce*
ultimately have (*sa s* ! *i* + *length s* − *Suc 0*) *mod* (*length s*) = *length s* − *Suc 0*
by *simp*
moreover
have ∃ *m*. *sa s* ! *j* = *Suc m*
using *0* ‹*sa s* ! *i* ≠ *sa s* ! *j*› *not0-implies-Suc* **by** *force*
then obtain *m* **where**
sa s ! *j* = *Suc m*
by *blast*
hence (*sa s* ! *j* + *length s* − *Suc 0*) *mod* (*length s*) = *m*
using *A*(*2*) *bwt-perm-def sa-length sa-nth-ex* **by** *force*
moreover
have *Suc m* ≤ *length s* − *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* < *length s* − *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-indexes-perm map-bwt-indexes 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 :: nat
  assumes i < length s
  shows bwt-perm s ! i = ((sa s ! i) + length s − 1) mod (length s)
  using assms sa-length bwt-perm-def by force

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

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

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

lemma ex-bwt-perm-nth:
  fixes s :: ('a :: {linorder, order-bot}) list
  fixes k :: nat
  assumes k < length s
  shows ∃ i < length s. 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 :: {linorder, order-bot}) list
  fixes i j :: nat
  assumes valid-list s
  and     i < length s
  and     j < length s
  and     i ≠ j
  and     s ! (bwt-perm s ! i) = s ! (bwt-perm s ! j)

shows sa s ! i ≠ 0
proof (rule ccontr)
  assume ¬ sa s ! i ≠ 0
  hence sa s ! i = 0
    by clarsimp

  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 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 $i = card\ \{j.\ j < length\ s \wedge j < i \wedge bwt\text{-}sa\ s\ !\ j \neq bwt\text{-}sa\ s\ !\ i\} +$
$card\ \{j.\ j < length\ s \wedge s\ !\ j = bwt\text{-}sa\ s\ !\ i \wedge$
$suffix\ s\ j < suffix\ s\ (bwt\text{-}perm\ s\ !\ i)\}$
proof −
let *?bwt* = *bwt-sa s*
let *?idx* = *bwt-perm s*
let *?i* = *?idx* ! *i*
let *?v* = *?bwt* ! *i*
let $?A = \{j.\ j < length\ s \wedge j < i \wedge ?bwt\ !\ j \neq ?v\}$
let $?B = \{j.\ j < length\ s \wedge s\ !\ j = ?v \wedge suffix\ s\ j < suffix\ s\ ?i\}$
let $?C = \{j.\ j < length\ s \wedge j < i \wedge ?bwt\ !\ j = ?v\}$

have P: $\bigwedge x.\ [\![x < i; \neg x < length\ s]\!] \Longrightarrow False$
using *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 $i = card\ ?A + card\ ?C$
by (*metis* (*no-types*, *lifting*) *List.finite-set atLeastLessThan-upt card-Un-disjnt card-upt*
disjnt-def finite-Un)
moreover
have *bij-betw* (λ*x*. *?idx* ! *x*) *?C ?B*
proof (*intro bij-betwI'*; *safe*)
fix *x y*
assume $x < length\ s\ y < length\ s\ ?idx\ !\ x = ?idx\ !\ y$
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 $x < length\ s$
with *map-bwt-indexes-perm*[*of s*]
show $?idx\ !\ x < length\ s$
using *perm-nth-ex* **by** *blast*
next
fix *x*
assume $x < length\ s\ bwt\text{-}sa\ s\ !\ x = ?v$
then show $s\ !\ (?idx\ !\ x) = ?v$
using *bwt-perm-s-nth* **by** *auto*
next
fix *x*
assume $x < length\ s\ x < i\ bwt\text{-}sa\ s\ !\ x = ?v$
then show $suffix\ s\ (?idx\ !\ x) < suffix\ s\ ?i$
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 < length s s ! x = ?v suffix s x < suffix s ?i

  from perm-nth[OF map-bwt-indexes-perm[of s, symmetric],
          simplified length-map sa-length length-upt]
  have ∃ y < length s. x = ?idx ! y
    using Q(1) bwt-perm-length by auto
  then obtain y where
    y < length s
    x = ?idx ! y
    by blast
  moreover
  from Q(2) calculation
  have ?bwt ! y = ?v
    by (simp add: bwt-perm-s-nth)
  moreover
  have y < i
  proof (rule ccontr)
    assume ¬ y < i
    hence i ≤ y
      by simp
    moreover
    from Q(3) ‹x = ?idx ! y›
    have i = y ⟹ False
      by blast
    moreover
    have i < y ⟹ False
    proof −
      assume i < y
      from bwt-relative-order[OF assms(1) ‹i < y› ‹y < -›]
          Q(2) ‹x = ?idx ! y›
      have suffix s ?i < suffix s (?idx ! y)
        by (simp add: bwt-perm-s-nth assms(2))
      with Q(3) ‹x = ?idx ! y›
      show False
        using order.asym by blast
    qed
    ultimately show False
      using nat-less-le by blast
  qed
  ultimately show ∃ y ∈ ?C. x = bwt-perm s ! y
    by blast
qed
hence card ?C = card ?B
  using bij-betw-same-card by blast
ultimately
show ?thesis
  by presburger
```

qed

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

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

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

13.2 BWT Rank Properties

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

```
  and      i < length s
shows rank (bwt-sa s) (bwt-sa s ! i) i =
        card {j. j < length s ∧ s ! j = bwt-sa s ! i ∧
                 suffix s j < suffix s (bwt-perm s ! i)}
proof −
  let ?bwt = bwt-sa s
  let ?idx = bwt-perm s
  let ?i = ?idx ! i
  let ?v = ?bwt ! i
  let ?A = {j. j < length s ∧ s ! j = ?v ∧ suffix s j < suffix s ?i}
  let ?B = {j. j < length ?bwt ∧ j < i ∧ ?bwt ! j = ?v}
  let ?C = {j. j < length s ∧ j < i ∧ ?bwt ! j = ?v}

  from valid-list-length-ex[OF assms(1)]
  obtain n where
    length s = Suc n
    by blast

  from rank-card-spec[of ?bwt ?v i]
  have rank ?bwt ?v i = card ?B .
  moreover
  have ?B = ?C
    by (simp add: bwt-sa-length sa-length)
  moreover
  have bij-betw (λx. ?idx ! x) ?C ?A
  proof (rule bij-betwI'; safe)
    fix x y
    assume x < length s y < length s ?idx ! x = ?idx ! y
    then show x = y
      by (metis map-bwt-indexes-perm  bwt-perm-length nth-eq-iff-index-eq
              perm-distinct-set-of-upt-iff)
  next
    fix x
    assume x < length s
    then show ?idx ! x < length s
      using map-bwt-indexes-perm perm-nth-ex by blast
  next
    fix x
    assume x < length s x < i ?bwt ! x = ?v
    then show s ! (?idx ! x) = ?v
      using bwt-perm-s-nth by auto
  next
    fix x
    assume x < length s x < i ?bwt ! x = ?v
    then show suffix s (?idx ! x) < suffix s ?i
      by (simp add: assms(1,2) bwt-relative-order bwt-perm-s-nth)
  next
    fix x
    assume x < length s s ! x = ?v suffix s x < suffix s ?i
```

from *perm-nth*[*OF map-bwt-indexes-perm*[*of s*, *symmetric*],
simplified length-map sa-length length-upt, *of x*]
have $\exists\, y < $ *length s*. $x = $ *?idx* ! y
using ‹$x <$ *length s*› *bwt-perm-length* **by** *auto*
then obtain y **where**
$y <$ *length s*
$x = $ *?idx* ! y
by *blast*
moreover
from *calculation* ‹s ! $x = $ *?v*›
have *?bwt* ! $y = $ *?v*
using *bwt-perm-s-nth* **by** *presburger*
moreover
have $y < i$
proof (*rule ccontr*)
assume $\neg\ y < i$
hence $i \le y$
by *simp*
moreover
from ‹*suffix s x* < *suffix s ?i*› ‹$x = $ *?idx* ! y›
have $y = i \Longrightarrow$ *False*
by *blast*
moreover
have $i < y \Longrightarrow$ *False*
proof −
assume $i < y$
with *bwt-relative-order*[*OF assms*(*1*) ‹$i < y$› ‹$y <$ -›] ‹$x = $ *?idx* ! y› ‹s ! x
= *bwt-sa s* ! i›
have *suffix s ?i* < *suffix s x*
using *assms*(*2*) *bwt-perm-s-nth* **by** *presburger*
with ‹*suffix s x* < *suffix s ?i*›
show *False*
using *less-not-sym* **by** *blast*
qed
ultimately show *False*
by *linarith*
qed
ultimately show $\exists\, y \in$ *?C*. $x = $ *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 $i <$ *length s*
shows $i =$ *card* $\{j.\ j <$ *length s* $\wedge\ j < i\ \wedge$ *bwt-sa s* ! $j \neq$ *bwt-sa s* ! $i\}$ +
rank (*bwt-sa s*) (*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 :: ($'a$ :: {*linorder*, *order-bot*}) *list*
fixes $i\ j$ :: *nat*
assumes *valid-list s*
and $i <$ *length s*
and $j <$ *length s*
and *sa s* ! i = *bwt-perm s* ! j
shows *rank* (*sort s*) (s ! (*sa s* ! i)) i = *rank* (*bwt-sa s*) (*bwt-sa s* ! j) j
proof −
let $?i$ = *sa s* ! i
let $?v = s$! $?i$
let $?A = \{j.\ j <$ *length s* $\wedge\ s$! $j = ?v\ \wedge$ *suffix s j* $<$ *suffix s ?i*$\}$

have *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 *rank* (*sort s*) $?v\ i$ = *card ?A* .
moreover
from *bwt-perm-rank-nth*[*OF assms*(*1*,*3*), *simplified assms*(*4*)[*symmetric*]] ‹*bwt-sa s* ! $j = ?v$›
have *rank* (*bwt-sa s*) (*bwt-sa s* ! j) j = *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 :: ($'a$ :: {*linorder*, *order-bot*}) *list*
fixes $i\ j$ :: *nat*
fixes v :: $'a$
assumes *valid-list s*
and $i <$ *length s*
and $j <$ *length s*
and s ! (*sa s* ! i) = v
and *bwt-sa s* ! $j = v$
and *rank* (*sort s*) $v\ i$ = *rank* (*bwt-sa s*) $v\ j$
shows *sa s* ! i = *bwt-perm s* ! j

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

lemma rank-bwt-card-suffix:
  fixes s :: ('a :: {linorder, order-bot}) list
  fixes i :: nat
  fixes a :: 'a
  assumes i < length s
  shows rank (bwt-sa s) a i =
        card {k. k < length s ∧ k < i ∧ bwt-sa s ! k = a ∧
                 a # suffix s (sa s ! k) < a # suffix s (sa s ! i)}
proof –
  let ?X = {j. j < length (bwt-sa s) ∧ j < i ∧ bwt-sa s ! j = a}
  let ?Y = {k. k < length s ∧ k < i ∧ bwt-sa s ! k = a ∧
               a # suffix s (sa s ! k) < a # suffix s (sa s ! i)}

  from rank-card-spec[of bwt-sa s a i]
  have rank (bwt-sa s) a i = card ?X .
  moreover
  have ?Y ⊆ ?X
    using bwt-sa-length by auto
  moreover
  have ?X ⊆ ?Y
  proof safe
    fix x
    assume x < length (bwt-sa s)
    then show x < length s
      by (simp add: bwt-sa-length)
  next
    fix x
    assume x < length (bwt-sa s) x < i a = bwt-sa s ! x
    with sorted-wrt-mapD[OF sa-g-sorted, of x i s]
    show bwt-sa s ! x # suffix s (sa s ! x) < bwt-sa s ! x # suffix s (sa s ! i)
      by (simp add: assms sa-length)
  qed
  ultimately show ?thesis
```

by *force*
qed

lemma *sa-to-bwt-perm-idx*:
fixes s :: ($'a$ :: {*linorder*, *order-bot*}) *list*
fixes i :: *nat*
assumes *valid-list s*
and $i <$ *length s*
shows *sa s* ! i =
bwt-perm s ! (*select* (*bwt-sa s*) (*s* ! (*sa s* ! *i*)) (*rank* (*sort s*) (*s* ! (*sa s* ! *i*)) *i*))
proof −
let *?a* = *s* ! (*sa s* ! *i*)
let *?r1* = *rank* (*sort s*) *?a i*
let *?i* = *select* (*bwt-sa s*) *?a ?r1*
let *?r2* = *rank* (*bwt-sa s*) *?a ?i*

have *?r1* < *count-list* (*sort s*) *?a*
by (*simp add*: *assms*(*2*) *rank-upper-bound sort-sa-nth*)
hence *?r1*< *count-list* (*bwt-sa s*) *?a*
by (*metis bwt-sa-perm count-list-perm mset-sort*)
hence *?i* < *length* (*bwt-sa s*)
by (*metis rank-length select-upper-bound*)
hence *?r1* = *?r2* ∧ *bwt-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 ‹*?i* < *length* (*bwt-sa s*)› *bwt-sa-length* **by** *fastforce*
qed

lemma *suffix-bwt-perm-sa*:
fixes s :: ($'a$ :: {*linorder*, *order-bot*}) *list*
fixes i :: *nat*
assumes *valid-list s*
and $i <$ *length s*
and *bwt-sa s* ! $i \neq$ *bot*
shows *suffix s* (*bwt-perm s* ! *i*) = *bwt-sa s* ! *i* # *suffix s* (*sa s* ! *i*)
proof −
from *bwt-sa-nth*[*OF assms*(*2*)]
have *bwt-sa s* ! *i* = *s* ! ((*sa s* ! *i* + *length s* − *1*) *mod 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-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-sa s* ! *i* # *suffix s* (*sa s* ! *i*) = *suffix s* (*sa s* ! *i* − *1*)

by (*metis Suc-lessD ‹sa s ! i ≠ 0› 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 (λ*i*. *P* (*xs* ! *i*)) [*0*..<*length xs*] = *filter* (λ*i*. *P* ((*xs* @ *ys*) ! *i*)) [*0*..<*length xs*]
by (*induct xs arbitrary*: *ys rule*: *rev-induct*; *simp*)

lemma *filter-eq-nth-upt*:
filter P xs = *map* (λ*i*. *xs* ! *i*) (*filter* (λ*i*. *P* (*xs* ! *i*)) [*0*..<*length xs*])
proof (*induct xs rule*: *rev-induct*)
case *Nil*
then show *?case*
by *simp*
next
case (*snoc x xs*)
have *?case* ⟷
map ((!) *xs*) (*filter* (λ*i*. *P* (*xs* ! *i*)) [*0*..<*length xs*]) =
map ((!) (*xs* @ [*x*])) (*filter* (λ*i*. *P* ((*xs* @ [*x*]) ! *i*)) [*0*..<*length xs*])
using *snoc* **by** *simp*
moreover
have *map* ((!) (*xs* @ [*x*])) (*filter* (λ*i*. *P* ((*xs* @ [*x*]) ! *i*)) [*0*..<*length xs*]) =
map ((!) (*xs* @ [*x*])) (*filter* (λ*i*. *P* (*xs* ! *i*)) [*0*..<*length xs*])
using *filter-nth-app-upt*[*of P xs* [*x*]] **by** *simp*
moreover
have *map* ((!) *xs*) (*filter* (λ*i*. *P* (*xs* ! *i*)) [*0*..<*length xs*]) =

    map ((!) (xs @ [x])) (filter (λi. P (xs ! i)) [0..<length xs])
  by (clarsimp simp: nth-append)
ultimately show ?case
  by argo
qed

lemma distinct-filter-nth-upt:
  distinct (filter (λi. P (xs ! i)) [0..<length xs])
  by simp

lemma filter-nth-upt-set:
  set (filter (λi. P (xs ! i)) [0..<length xs]) = {i. i < length xs ∧ P (xs ! i)}
  using set-filter **by** simp

lemma filter-length-upt:
  length (filter (λi. P (xs ! i)) [0..<length xs]) = card {i. i < length xs ∧ P (xs ! i)}
  by (metis distinct-card distinct-filter-nth-upt filter-nth-upt-set)

lemma perm-filter-length:
  xs <~~> ys ⟹
   length (filter (λi. P (xs ! i)) [0..<length xs])
    = length (filter (λi. P (ys ! i)) [0..<length ys])
  by (metis filter-eq-nth-upt length-map mset-filter perm-length)

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 <~~> sort s
    using bwt-sa-perm by auto
  ultimately have ?r < count-list (sort s) ?v
    by (metis (no-types, lifting) count-list-perm)
  with rank-length[of sort s ?v, symmetric]
  have ?r < rank (sort s) ?v (length s)
    by simp
  with select-upper-bound
  have select (sort s) ?v ?r < length (sort s)
    by metis
  with ‹?i = select (sort s) ?v ?r›
  show ?thesis
    by (metis length-sort)
qed

corollary lf-map-abs-less-length-funpow:
  fixes s :: 'a list
  fixes i j :: nat
  assumes i < length s
shows ((lf-map-abs s)^^k) i < 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 :: {linorder, order-bot}) list
  fixes i r :: nat
  fixes v :: 'a
  assumes i < length (bwt-sa s)
  and     v = bwt-sa s ! i
  and     r = rank (bwt-sa s) v i
shows lf-map-abs s i = card {j. j < length (bwt-sa s) ∧ bwt-sa s ! j < v} + r
proof –

  have ∃k. length s = Suc k
    by (metis assms(1) bwt-sa-length less-nat-zero-code not0-implies-Suc)
  then obtain n where
    length s = Suc n
    by blast

  let ?P = (λx. x < v)

  have lf-map-abs s i = select (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* (λ*x*. *?P* (*sort s* ! *x*)) [*0*..<*length* (*sort s*)])
= *card* {*j*. *j* < *length* (*sort s*) ∧ *sort s* ! *j* < *v*}
using *filter-length-upt*[*of ?P sort s*] **by** *simp*
moreover
have *length* (*filter* (λ*x*. *?P* (*bwt-sa s* ! *x*)) [*0*..<*length* (*bwt-sa s*)])
= *card* {*j*. *j* < *length* (*bwt-sa s*) ∧ *bwt-sa s* ! *j* < *v*}
using *filter-length-upt*[*of ?P bwt-sa s*] **by** *simp*
ultimately show *?thesis*
using *perm-filter-length*[*OF trans*[*OF bwt-sa-perm sort-perm*], *of ?P s*]
by *presburger*
qed

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 $k \leq n$
shows (*ibwt-perm-abs* (*Suc n*) *s i*) ! *k* = ((*lf-map-abs s*)$^{\frown\frown}$(*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 $k \leq n \Longrightarrow$ *?case*
proof −
assume $k \leq n$
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* ^^ (*Suc n* − *k*)) *i*
by (*metis Suc-diff-le comp-apply funpow.simps*(*2*) *funpow-swap1*)
then show *?thesis*
by (*metis* ‹$k \leq n$› *A ibwt-perm-abs-length le-imp-less-Suc nth-append*)
qed
moreover
have *k* = *Suc n* $\Longrightarrow$ *?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* ^^ (*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*)^^(*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-perm-abs n s i = *map* (λ*x*. ((*lf-map-abs s*)^^*x*) *i*) (*rev* [*0* ..<*n*])
proof (*intro list-eq-iff-nth-eq*[*THEN iffD2*] *conjI allI impI*)
show *length* (*ibwt-perm-abs n s i*) = *length* (*map* (λ*x*. (*lf-map-abs s* ^^ *x*) *i*) (*rev* [*0* ..<*n*]))
by (*simp add*: *ibwt-perm-abs-length*)
next
fix *j*
assume *j* < *length* (*ibwt-perm-abs n s i*)
hence *j* < *n*
by (*simp add*: *ibwt-perm-abs-length*)

have *map* (λ*x*. (*lf-map-abs s* ^^ *x*) *i*) (*rev* [*0* ..<*n*]) ! *j* =
(λ*x*. (*lf-map-abs s* ^^ *x*) *i*) (*rev* [*0* ..<*n*] ! *j*)
by (*simp add*: ‹*j* < *n*›)
moreover
have (λ*x*. (*lf-map-abs s* ^^ *x*) *i*) (*rev* [*0* ..<*n*] ! *j*) = (*lf-map-abs s* ^^ (*n* − *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-perm-abs n s i* ! *j* = *map* (λ*x*. (*lf-map-abs s* ^^ *x*) *i*) (*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* :: {*linorder*, *order-bot*}) *list*
fixes *i n* :: *nat*
assumes *valid-list s*
and *i* < *length s*
shows *inc-one-bounded* (*length s*) (*map* ((!) (*sa s*)) (*ibwt-perm-abs n s i*))
(**is** *inc-one-bounded ?n ?xs*)
unfolding *inc-one-bounded-def*
proof (*safe*)
fix *j*
assume *Suc j* < *length* (*map* ((!) (*sa s*)) (*ibwt-perm-abs n s i*))
hence *Suc 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*

let *?i* = ((*lf-map-abs s*)^^(*k* − *Suc j*)) *i*

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

ultimately have *sa s* ! (*ibwt-perm-abs n s i* ! *j*) < *length s*
  by *presburger*
then show *map* ((!) (*sa s*)) (*ibwt-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* ≤ *length s*
shows *is-rot-sublist* [*0*..<*length s*] (*map* ((!) (*sa s*)) (*ibwt-perm-abs n s i*))
  by (*simp add*: *assms 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-perm s*)) (*ibwt-perm-abs n s i*))
  hence *Suc j* < *n*
    by (*simp add*: *ibwt-perm-abs-length*)
  hence ∃ *k*. *n* = *Suc k*
    using *less-imp-Suc-add* **by** *auto*
  then obtain *k* **where**
    n = *Suc k*
    by *blast*

  let *?i* = ((*lf-map-abs s*)^^(*k* − *Suc j*)) *i*
  from *ibwt-perm-abs-nth*[*of Suc j k s i*]
  have *ibwt-perm-abs n s i* ! *Suc j* = *?i*
    using ‹*Suc j* < *n*› ‹*n* = *Suc k*› *less-Suc-eq-le* **by** *blast*
  moreover
  {
    have *ibwt-perm-abs n s i* ! *j* = ((*lf-map-abs s*)^^(*k* − *j*)) *i*
      by (*metis Suc-less-SucD* ‹*Suc j* < *n*› ‹*n* = *Suc k*› *nless-le ibwt-perm-abs-nth*)
    moreover
    have ((*lf-map-abs s*)^^(*k* − *j*)) *i* = *lf-map-abs s ?i*
      using ‹*Suc j* < *n*› ‹*n* = *Suc k*› *less-imp-Suc-add* **by** *fastforce*
    ultimately have *ibwt-perm-abs n s i* ! *j* = *lf-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 *map* ((!) (*bwt-perm s*)) (*ibwt-perm-abs n s i*) ! *j* < *length s*
by (*simp add*: ‹*j* < *n*› *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* :: {*linorder*, *order-bot*}) *list*
fixes *i n* :: *nat*
assumes *valid-list s*
and *i* < *length s*
and *n* ≤ *length s*
shows *is-rot-sublist* [*0* ..<*length s*] (*map* ((!) (*bwt-perm s*)) (*ibwt-perm-abs n s i*))
by (*simp add*: *assms 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 *map* ((!) (*bwt-perm s*)) (*ibwt-perm-abs* (*length s*) *s* (*select* (*bwt-sa s*) *bot* *0*))
= [*0* ..<*length s*]
proof −
from *valid-list-length-ex*[*OF assms*]
obtain *n* **where**
length s = *Suc n*
by *blast*

let *?i* = *select* (*bwt-sa s*) *bot 0*
let *?xs* = *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*)
with *is-rot-sublist-bwt-perm-ibwt-perm-abs*[*OF assms*, *of ?i length s*] ‹*length s* = *Suc n*›
have *is-rot-sublist* [*0* ..<*Suc n*] (*map* ((!) (*bwt-perm s*)) *?xs*)
by (*metis nle-le*)
moreover
have *length* (*map* ((!) (*bwt-perm s*)) *?xs*) = *Suc n*
by (*metis* ‹*length s* = *Suc n*› *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 $x \in set\ (ibwt\text{-}perm\text{-}abs\ (length\ s)\ s\ ?i)$

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

16.5 Concretization

lemma *lf-map-abs-eq-conc*:
$i < length\ s \Longrightarrow lf\text{-}map\text{-}abs\ s\ i = lf\text{-}map\text{-}conc\ (sort\ (bwt\text{-}sa\ s))\ (bwt\text{-}sa\ s)\ i$
proof −
let $?v = bwt\text{-}sa\ s\ !\ i$
let $?r = rank\ (bwt\text{-}sa\ s)\ ?v\ i$
let $?ss = sort\ (bwt\text{-}sa\ s)$
assume $i < length\ s$
hence $rank\ (bwt\text{-}sa\ s)\ ?v\ i < count\text{-}list\ (sort\ s)\ ?v$
using *rank-upper-bound*[*of i bwt-sa s ?v*]
by (*metis bwt-sa-length bwt-sa-perm count-list-perm mset-sort*)
with *sorted-select*[*of ?ss ?r ?v*]
have $select\ ?ss\ ?v\ ?r = card\ \{j.\ j < length\ ?ss \wedge ?ss\ !\ j < ?v\} + ?r$
by (*metis* (*full-types*) *bwt-sa-perm sorted-list-of-multiset-mset sorted-sort*)
moreover
have $sort\ s = sort\ ?ss$
by (*simp add*: *bwt-sa-perm properties-for-sort*)
moreover
have $select\ (sort\ s)\ ?v\ ?r = card\ \{j.\ j < length\ (sort\ s) \wedge (sort\ s)\ !\ j < ?v\} +$
$?r$
by (*simp add*: ‹$rank\ (bwt\text{-}sa\ s)\ ?v\ i < count\text{-}list\ (sort\ s)\ ?v$› *sorted-select*)
ultimately show *?thesis*
by (*metis* (*full-types*) ‹$rank\ (bwt\text{-}sa\ s)\ ?v\ i < count\text{-}list\ (sort\ s)\ ?v$› *bwt-sa-perm*
lf-map-abs.simps lf-map-conc.simps sorted-list-of-multiset-mset
sorted-select-0-plus sorted-sort)
qed

lemma *ibwt-perm-abs-conc-eq*:
i < *length s* ⟹ *ibwt-perm-abs n s i* = *ibwt-perm-conc n* (*sort* (*bwt-sa s*)) (*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 *ibwt-perm-abs* (*Suc n*) *s i* = *ibwt-perm-abs n s* (*lf-map-abs s i*) @ [*i*]
by *simp*
moreover
have *ibwt-perm-conc* (*Suc n*) *?ss ?bs i* = *ibwt-perm-conc n ?ss ?bs* (*lf-map-conc ?ss ?bs i*) @ [*i*]
by *simp*
moreover
have *lf-map-abs s i* = *lf-map-conc ?ss ?bs i*
using *Suc.prems lf-map-abs-eq-conc* **by** *blast*
moreover
have *lf-map-abs s i* < *length s*
using *Suc.prems lf-map-abs-less-length* **by** *blast*
ultimately show *?case*
using *Suc.hyps* **by** *presburger*
qed

theorem *ibwtn-bwt-sa-lookup-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 *map* ((!) *ys*) (*ibwt-perm-conc* (*length ys*) *xs ys* (*select ys bot 0*)) = *s*
proof −
from *ibwt-perm-abs-bwt-sa-lookup-correct*[*OF assms*(*1*)]
have *map* ((!) (*bwt-sa s*)) (*ibwt-perm-abs* (*length s*) *s* (*select* (*bwt-sa s*) *bot 0*)) = *s* **.**
moreover
have *select* (*bwt-sa s*) *bot 0* < *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 *ibwt-perm-abs* (*length s*) *s* (*select* (*bwt-sa s*) *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. Ceritifed 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.