Reasoning about Lists via List Interleaving

Pasquale Noce
Security Certification Specialist at Arjo Systems - Gep S.p.A.
pasquale dot noce dot lavoro at gmail dot com
pasquale dot noce at arjowiggins-it dot com

March 17, 2025

Abstract

Among the various mathematical tools introduced in his outstanding work on Communicating Sequential Processes, Hoare has defined "interleaves" as the predicate satisfied by any three lists such that the first list may be split into sublists alternately extracted from the other two ones, whatever is the criterion for extracting an item from either one list or the other in each step.

This paper enriches Hoare's definition by identifying such criterion with the truth value of a predicate taking as inputs the head and the tail of the first list. This enhanced "interleaves" predicate turns out to permit the proof of equalities between lists without the need of an induction. Some rules that allow to infer "interleaves" statements without induction, particularly applying to the addition or removal of a prefix to the input lists, are also proven. Finally, a stronger version of the predicate, named "Interleaves", is shown to fulfil further rules applying to the addition or removal of a suffix to the input lists.

Contents

1   List interleaving

theory *ListInterleaving*
imports *Main*
begin

Among the various mathematical tools introduced in his outstanding work on Communicating Sequential Processes [1], Hoare has defined *interleaves* as

the predicate satisfied by any three lists *s*, *t*, emphu such that *s* may be split into sublists alternately extracted from *t* and *u*, whatever is the criterion for extracting an item from either *t* or *u* in each step.

This paper enriches Hoare's definition by identifying such criterion with the truth value of a predicate taking as inputs the head and the tail of *s*. This enhanced *interleaves* predicate turns out to permit the proof of equalities between lists without the need of an induction. Some rules that allow to infer *interleaves* statements without induction, particularly applying to the addition of a prefix to the input lists, are also proven. Finally, a stronger version of the predicate, named *Interleaves*, is shown to fulfil further rules applying to the addition of a suffix to the input lists.

Throughout this paper, the salient points of definitions and proofs are commented; for additional information, cf. Isabelle documentation, particularly [5], [4], [3], and [2]. For a sample nontrivial application of the mathematical machinery developed in this paper, cf. [6].

1.1   A first version of interleaving

Here below is the definition of predicate *interleaves*, as well as of a convenient symbolic notation for it. As in the definition of predicate *interleaves* formulated in [1], the recursive decomposition of the input lists is performed by item prepending. In the case of a list *ws* constructed recursively by item appending rather than prepending, the statement that it satisfies predicate *interleaves* with two further lists can nevertheless be proven by induction using as input *rev ws*, rather than *ws* itself.

With respect to Hoare's homonymous predicate, *interleaves* takes as an additional input a predicate *P*, which is a function of a single item and a list. Then, for statement *interleaves P* (*x* # *xs*) (*y* # *ys*) (*z* # *zs*) to hold, the item picked for being *x* must be *y* if *P x xs*, *z* otherwise. On the contrary, in case either the second or the third list is empty, the truth value of *P x xs* does not matter and list *x* # *xs* just has to match the other nonempty one, if any.

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

interleaves - [] - (- # -) = False |
interleaves - [] [] [] = True

abbreviation *interleaves-syntax* ::
  'a list ⇒ 'a list ⇒ 'a list ⇒ ('a ⇒ 'a list ⇒ bool) ⇒ bool
  (‹(- ≃ {-, -, -})› [60, 60, 60] 51)
 where *xs ≃ {ys, zs, P} ≡ interleaves P xs ys zs*

The advantage provided by this enhanced *interleaves* predicate is that in case *xs ≃ {ys, zs, P}*, fixing the values of *xs* and either *ys* or *zs* has the effect of fixing the value of the remaining list, too. Namely, if *xs ≃ {ys', zs, P}* also holds, then *ys = ys'*, and likewise, if *xs ≃ {ys, zs', P}* also holds, then *zs = zs'*. Therefore, once two *interleaves* statements *xs ≃ {ys, zs, P}*, *xs' ≃ {ys', zs', P'}* have been proven along with equalities *xs = xs'*, *P = P'*, and either *zs = zs'* or *ys = ys'*, possibly by induction, the remaining equality, i.e. respectively *ys = ys'* and *zs = zs'*, can be inferred without the need of a further induction.

Here below is the proof of this property as well as of other ones. Particularly, it is also proven that in case *xs ≃ {ys, zs, P}*, lists *ys* and *zs* can be swapped by replacing predicate *P* with its negation.

It is worth noting that fixing the values of *ys* and *zs* does not fix the value of *xs* instead. A counterexample is *ys = [y]*, *zs = [z]* with *y ≠ z*, *P y [z] = True*, *P z [y] = False*, in which case both of the *interleaves* statements *[y, z] ≃ {ys, zs, P}* and *[z, y] ≃ {ys, zs, P}* hold.

lemma *interleaves-length* [*rule-format*]:
 xs ≃ {ys, zs, P} ⟶ length xs = length ys + length zs
proof (*induction P xs ys zs rule*: *interleaves.induct*, *simp-all*)
qed (*rule conjI*, (*rule-tac* [!] *impI*)+, *simp-all*)

lemma *interleaves-nil*:
 [] ≃ {ys, zs, P} ⟹ ys = [] ∧ zs = []
by (*rule interleaves.cases* [*of* (*P, [], ys, zs*)], *simp-all*)

lemma *interleaves-swap*:
 xs ≃ {ys, zs, P} = xs ≃ {zs, ys, λw ws. ¬ P w ws}
proof (*induction P xs ys zs rule*: *interleaves.induct*, *simp-all*)
  fix *y' :: 'a* **and** *ys' zs' P'*
  show *¬ [] ≃ {zs', y' # ys', λw ws. ¬ P' w ws}* **by** (*cases zs'*, *simp-all*)
qed

lemma *interleaves-equal-fst* [*rule-format*]:
 xs ≃ {ys, zs, P} ⟶ xs ≃ {ys', zs, P} ⟶ ys = ys'
proof (*induction xs arbitrary*: *ys ys' zs*, (*rule-tac* [!] *impI*)+)
  fix *ys ys' zs*
  assume *[] ≃ {ys, zs, P}*

3

hence $ys = [] \land zs = []$ **by** (*rule interleaves-nil*)
moreover assume $[] \simeq \{ys', zs, P\}$
hence $ys' = [] \land zs = []$ **by** (*rule interleaves-nil*)
ultimately show $ys = ys'$ **by** *simp*
next
  fix $x \; xs \; ys \; ys' \; zs$
  assume
    A: $\bigwedge ys \; ys' \; zs. \; xs \simeq \{ys, zs, P\} \longrightarrow xs \simeq \{ys', zs, P\} \longrightarrow ys = ys'$ **and**
    B: $x \; \# \; xs \simeq \{ys, zs, P\}$ **and**
    B': $x \; \# \; xs \simeq \{ys', zs, P\}$
  show $ys = ys'$
  proof (*cases zs, case-tac [2] ys, case-tac [2−3] ys', simp-all*)
    assume C: $zs = []$
    hence $\exists w \; ws. \; ys = w \; \# \; ws$ **using** B **by** (*cases ys, simp-all*)
    then obtain $w \; ws$ **where** Y: $ys = w \; \# \; ws$ **by** *blast*
    hence D: $w = x$ **using** B **and** C **by** *simp*
    have $xs \simeq \{ws, [], P\}$ **using** B **and** C **and** Y **by** *simp*
    moreover have $\exists w' \; ws'. \; ys' = w' \; \# \; ws'$
    using B' **and** C **by** (*cases ys', simp-all*)
    then obtain $w' \; ws'$ **where** Y': $ys' = w' \; \# \; ws'$ **by** *blast*
    hence D': $w' = x$ **using** B' **and** C **by** *simp*
    have $xs \simeq \{ws', [], P\}$ **using** B' **and** C **and** Y' **by** *simp*
    moreover have $xs \simeq \{ws, [], P\} \longrightarrow xs \simeq \{ws', [], P\} \longrightarrow ws = ws'$
    using A **.**
    ultimately have $ws = ws'$ **by** *simp*
    with Y **and** Y' **and** D **and** D' **show** *?thesis* **by** *simp*
  next
    fix $v \; vs \; w' \; ws'$
    assume C: $zs = v \; \# \; vs$ **and** $ys = []$
    hence D: $xs \simeq \{[], vs, P\}$ **using** B **by** *simp*
    assume E: $ys' = w' \; \# \; ws'$
    have $P \; x \; xs \lor \neg \; P \; x \; xs$ **by** *simp*
    moreover {
      assume $P \; x \; xs$
      hence $xs \simeq \{ws', v \; \# \; vs, P\}$ **using** B' **and** C **and** E **by** *simp*
      hence *length xs = Suc (length vs) + length ws'*
      by (*simp add: interleaves-length*)
      moreover have *length xs = length vs*
      using D **by** (*simp add: interleaves-length*)
      ultimately have *False* **by** *simp*
    }
    moreover {
      assume $\neg \; P \; x \; xs$
      hence $xs \simeq \{w' \; \# \; ws', vs, P\}$ **using** B' **and** C **and** E **by** *simp*
      moreover have $xs \simeq \{[], vs, P\} \longrightarrow xs \simeq \{w' \; \# \; ws', vs, P\} \longrightarrow$
        $[] = w' \; \# \; ws'$
      using A **.**
      ultimately have $[] = w' \; \# \; ws'$ **using** D **by** *simp*
      hence *False* **by** *simp*

4

}
ultimately show *False* **..**
next
  fix *v vs w ws*
  assume *C*: $zs = v \# vs$ **and** $ys' = []$
  hence *D*: $xs \simeq \{[], vs, P\}$ **using** B' **by** *simp*
  assume *E*: $ys = w \# ws$
  have $P\ x\ xs \lor \neg\ P\ x\ xs$ **by** *simp*
  moreover {
    assume $P\ x\ xs$
    hence $xs \simeq \{ws, v \# vs, P\}$ **using** B **and** C **and** E **by** *simp*
    hence *length xs = Suc (length vs) + length ws*
     by (*simp add: interleaves-length*)
    moreover have *length xs = length vs*
     using D **by** (*simp add: interleaves-length*)
    ultimately have *False* **by** *simp*
  }
  moreover {
    assume $\neg\ P\ x\ xs$
    hence $xs \simeq \{w \# ws, vs, P\}$ **using** B **and** C **and** E **by** *simp*
    moreover have $xs \simeq \{[], vs, P\} \longrightarrow xs \simeq \{w \# ws, vs, P\} \longrightarrow [] = w \# ws$
     using A **.**
    ultimately have $[] = w \# ws$ **using** D **by** *simp*
    hence *False* **by** *simp*
  }
  ultimately show *False* **..**
next
  fix $v\ vs\ w\ ws\ w'\ ws'$
  assume *C*: $zs = v \# vs$ **and** *D*: $ys = w \# ws$ **and** *D'*: $ys' = w' \# ws'$
  have $P\ x\ xs \lor \neg\ P\ x\ xs$ **by** *simp*
  moreover {
    assume *E*: $P\ x\ xs$
    hence *F*: $w = x$ **using** B **and** C **and** D **by** *simp*
    have $xs \simeq \{ws, v \# vs, P\}$ **using** B **and** C **and** D **and** E **by** *simp*
    moreover have *F'*: $w' = x$ **using** B' **and** C **and** D' **and** E **by** *simp*
    have $xs \simeq \{ws', v \# vs, P\}$ **using** B' **and** C **and** D' **and** E **by** *simp*
    moreover have $xs \simeq \{ws, v \# vs, P\} \longrightarrow xs \simeq \{ws', v \# vs, P\} \longrightarrow$
     $ws = ws'$
     using A **.**
    ultimately have $ws = ws'$ **by** *simp*
    hence $w = w' \land ws = ws'$ **using** F **and** F' **by** *simp*
  }
  moreover {
    assume *E*: $\neg\ P\ x\ xs$
    hence $xs \simeq \{w \# ws, vs, P\}$ **using** B **and** C **and** D **by** *simp*
    moreover have $xs \simeq \{w' \# ws', vs, P\}$
     using B' **and** C **and** D' **and** E **by** *simp*
    moreover have $xs \simeq \{w \# ws, vs, P\} \longrightarrow xs \simeq \{w' \# ws', vs, P\} \longrightarrow$
     $w \# ws = w' \# ws'$

5

     using *A* **.**
     ultimately have $w \mathbin{\#} ws = w' \mathbin{\#} ws'$ **by** *simp*
     hence $w = w' \wedge ws = ws'$ **by** *simp*
    }
    ultimately show $w = w' \wedge ws = ws'$ **..**
  qed
qed

lemma *interleaves-equal-snd*:
  $xs \simeq \{ys,\ zs,\ P\} \implies xs \simeq \{ys,\ zs',\ P\} \implies zs = zs'$
by (*subst* (*asm*) (*1 2*) *interleaves-swap, rule interleaves-equal-fst*)

Since *interleaves* statements permit to prove equalities between lists without the need of an induction, it is useful to search for rules that allow to infer such statements themselves without induction, which is precisely what is done here below. Particularly, it is proven that under proper assumptions, predicate *interleaves* keeps being satisfied by applying a filter, a mapping, or the addition or removal of a prefix to the input lists.

lemma *interleaves-all-nil*:
  $xs \simeq \{xs,\ [],\ P\}$
by (*induction xs, simp-all*)

lemma *interleaves-nil-all*:
  $xs \simeq \{[],\ xs,\ P\}$
by (*induction xs, simp-all*)

lemma *interleaves-equal-all-nil*:
  $xs \simeq \{ys,\ [],\ P\} \implies xs = ys$
by (*insert interleaves-all-nil, rule interleaves-equal-fst*)

lemma *interleaves-equal-nil-all*:
  $xs \simeq \{[],\ zs,\ P\} \implies xs = zs$
by (*insert interleaves-nil-all, rule interleaves-equal-snd*)

lemma *interleaves-filter* [*rule-format*]:
  assumes *A*: $\forall x\ xs.\ P\ x\ (filter\ Q\ xs) = P\ x\ xs$
  shows $xs \simeq \{ys,\ zs,\ P\} \longrightarrow filter\ Q\ xs \simeq \{filter\ Q\ ys,\ filter\ Q\ zs,\ P\}$
proof (*induction xs arbitrary: ys zs, rule-tac* [!] *impI, simp*)
  fix *ys zs*
  assume $[] \simeq \{ys,\ zs,\ P\}$
  hence $ys = [] \wedge zs = []$ **by** (*rule interleaves-nil*)
  thus $[] \simeq \{filter\ Q\ ys,\ filter\ Q\ zs,\ P\}$ **by** *simp*
next
  fix *x xs ys zs*
  assume
    B: $\bigwedge ys'\ zs'.\ xs \simeq \{ys',\ zs',\ P\} \longrightarrow$

filter Q xs ≃ {*filter Q ys′*, *filter Q zs′*, *P*} **and**
  C: *x # xs* ≃ {*ys*, *zs*, *P*}
show *filter Q (x # xs)* ≃ {*filter Q ys*, *filter Q zs*, *P*}
proof (*cases ys, case-tac* [!] *zs, simp-all del: filter.simps, rule ccontr*)
  assume *ys* = [] **and** *zs* = []
  thus *False* **using** *C* **by** *simp*
next
  fix *z zs′*
  assume *ys* = [] **and** *zs* = *z # zs′*
  hence *D*: *x* = *z* ∧ *xs* ≃ {[], *zs′*, *P*} **using** *C* **by** *simp*
  moreover have *xs* ≃ {[], *zs′*, *P*} ⟶
    filter Q xs ≃ {*filter Q* [], *filter Q zs′*, *P*}
   using *B* **.**
  ultimately have *filter Q xs* ≃ {[], *filter Q zs′*, *P*} **by** *simp*
  thus *filter Q (x # xs)* ≃ {[], *filter Q (z # zs′)*, *P*} **using** *D* **by** *simp*
next
  fix *y ys′*
  assume *ys* = *y # ys′* **and** *zs* = []
  hence *D*: *x* = *y* ∧ *xs* ≃ {*ys′*, [], *P*} **using** *C* **by** *simp*
  moreover have *xs* ≃ {*ys′*, [], *P*} ⟶
    filter Q xs ≃ {*filter Q ys′*, *filter Q* [], *P*}
   using *B* **.**
  ultimately have *filter Q xs* ≃ {*filter Q ys′*, [], *P*} **by** *simp*
  thus *filter Q (x # xs)* ≃ {*filter Q (y # ys′)*, [], *P*} **using** *D* **by** *simp*
next
  fix *y ys′ z zs′*
  assume *ys* = *y # ys′* **and** *zs* = *z # zs′*
  hence *D*: *x # xs* ≃ {*y # ys′*, *z # zs′*, *P*} **using** *C* **by** *simp*
  show *filter Q (x # xs)* ≃ {*filter Q (y # ys′)*, *filter Q (z # zs′)*, *P*}
  proof (*cases P x xs*)
    case *True*
    hence *E*: *P x (filter Q xs)* **using** *A* **by** *simp*
    have *F*: *x* = *y* ∧ *xs* ≃ {*ys′*, *z # zs′*, *P*} **using** *D* **and** *True* **by** *simp*
    moreover have *xs* ≃ {*ys′*, *z # zs′*, *P*} ⟶
      filter Q xs ≃ {*filter Q ys′*, *filter Q (z # zs′)*, *P*}
     using *B* **.**
    ultimately have *G*: *filter Q xs* ≃ {*filter Q ys′*, *filter Q (z # zs′)*, *P*}
     by *simp*
    show *?thesis*
    proof (*cases Q x*)
      assume *Q x*
      hence *filter Q (x # xs)* = *x # filter Q xs* **by** *simp*
      moreover have *filter Q (y # ys′)* = *x # filter Q ys′*
       using ‹*Q x*› **and** *F* **by** *simp*
      ultimately show *?thesis* **using** *E* **and** *G*
       by (*cases filter Q (z # zs′), simp-all*)
     next
      assume ¬ *Q x*
      hence *filter Q (x # xs)* = *filter Q xs* **by** *simp*

      moreover have *filter Q (y # ys') = filter Q ys'*
       using ‹¬ *Q x*› **and** *F* **by** *simp*
      ultimately show *?thesis* **using** *E* **and** *G*
       by (*cases filter Q (z # zs'), simp-all*)
     qed
   next
    case *False*
    hence *E*: ¬ *P x (filter Q xs)* **using** *A* **by** *simp*
    have *F*: *x = z ∧ xs ≃ {y # ys', zs', P}* **using** *D* **and** *False* **by** *simp*
    moreover have *xs ≃ {y # ys', zs', P}* ⟶
     filter Q xs ≃ {filter Q (y # ys'), filter Q zs', P}
     using *B* .
    ultimately have *G*: *filter Q xs ≃ {filter Q (y # ys'), filter Q zs', P}*
     by *simp*
    show *?thesis*
    proof (*cases Q x*)
     assume *Q x*
     hence *filter Q (x # xs) = x # filter Q xs* **by** *simp*
     moreover have *filter Q (z # zs') = x # filter Q zs'*
      using ‹*Q x*› **and** *F* **by** *simp*
     ultimately show *?thesis* **using** *E* **and** *G*
      by (*cases filter Q (y # ys'), simp-all*)
    next
     assume ¬ *Q x*
     hence *filter Q (x # xs) = filter Q xs* **by** *simp*
     moreover have *filter Q (z # zs') = filter Q zs'*
      using ‹¬ *Q x*› **and** *F* **by** *simp*
     ultimately show *?thesis* **using** *E* **and** *G*
      by (*cases filter Q (z # zs'), simp-all*)
    qed
   qed
  qed
qed

lemma *interleaves-map* [*rule-format*]:
  assumes *A*: *inj f*
  shows *xs ≃ {ys, zs, P}* ⟶
   map f xs ≃ {map f ys, map f zs, λw ws. P (inv f w) (map (inv f) ws)}
   (**is** - ⟶ - ≃ {-, -, *?P'*})
proof (*induction xs arbitrary*: *ys zs, rule-tac* [!] *impI, simp-all*)
  fix *ys zs*
  assume *[] ≃ {ys, zs, P}*
  hence *ys = [] ∧ zs = []* **by** (*rule interleaves-nil*)
  thus *[] ≃ {map f ys, map f zs, ?P'}* **by** *simp*
next
  fix *x xs ys zs*
  assume
   B: ⋀*ys zs. xs ≃ {ys, zs, P}* ⟶ *map f xs ≃ {map f ys, map f zs, ?P'}* **and**
   C: *x # xs ≃ {ys, zs, P}*

show *f x # map f xs ≃ {map f ys, map f zs, ?P′}*
proof (*cases ys, case-tac* [!] *zs, simp-all del*: *interleaves.simps(1)*)
  assume *ys = [] **and** zs = []*
  thus *False* **using** *C* **by** *simp*
next
  fix *z zs′*
  assume *ys = [] **and** zs = z # zs′*
  hence *x = z ∧ xs ≃ {[], zs′, P}* **using** *C* **by** *simp*
  moreover have *xs ≃ {[], zs′, P} ⟶ map f xs ≃ {map f [], map f zs′, ?P′}*
  using *B* **.**
  ultimately show *f x = f z ∧ map f xs ≃ {[], map f zs′, ?P′}* **by** *simp*
next
  fix *y ys′*
  assume *ys = y # ys′ **and** zs = []*
  hence *x = y ∧ xs ≃ {ys′, [], P}* **using** *C* **by** *simp*
  moreover have *xs ≃ {ys′, [], P} ⟶ map f xs ≃ {map f ys′, map f [], ?P′}*
  using *B* **.**
  ultimately show *f x = f y ∧ map f xs ≃ {map f ys′, [], ?P′}* **by** *simp*
next
  fix *y ys′ z zs′*
  assume *ys = y # ys′ **and** zs = z # zs′*
  hence *D: x # xs ≃ {y # ys′, z # zs′, P}* **using** *C* **by** *simp*
  show *f x # map f xs ≃ {f y # map f ys′, f z # map f zs′, ?P′}*
  proof (*cases P x xs*)
    case *True*
    hence *E: ?P′ (f x) (map f xs)* **using** *A* **by** *simp*
    have *x = y ∧ xs ≃ {ys′, z # zs′, P}* **using** *D* **and** *True* **by** *simp*
    moreover have *xs ≃ {ys′, z # zs′, P} ⟶*
     map f xs ≃ {map f ys′, map f (z # zs′), ?P′}
    using *B* **.**
    ultimately have *f x = f y ∧ map f xs ≃ {map f ys′, map f (z # zs′), ?P′}*
    by *simp*
    thus *?thesis* **using** *E* **by** *simp*
  next
    case *False*
    hence *E: ¬ ?P′ (f x) (map f xs)* **using** *A* **by** *simp*
    have *x = z ∧ xs ≃ {y # ys′, zs′, P}* **using** *D* **and** *False* **by** *simp*
    moreover have *xs ≃ {y # ys′, zs′, P} ⟶*
     map f xs ≃ {map f (y # ys′), map f zs′, ?P′}
    using *B* **.**
    ultimately have *f x = f z ∧ map f xs ≃ {map f (y # ys′), map f zs′, ?P′}*
    by *simp*
    thus *?thesis* **using** *E* **by** *simp*
  qed
qed
qed

lemma *interleaves-prefix-fst-1* [*rule-format*]:
  assumes *A: xs ≃ {ys, zs, P}*

shows $(\forall\, n < length\ ws.\ P\ (ws\ !\ n)\ (drop\ (Suc\ n)\ ws\ @\ xs)) \longrightarrow$
  $ws\ @\ xs \simeq \{ws\ @\ ys,\ zs,\ P\}$
proof (*induction ws, simp-all add: A, rule impI*)
  fix *w ws*
  assume *B*: $\forall\, n < Suc\ (length\ ws).\ P\ ((w\ \#\ ws)\ !\ n)\ (drop\ n\ ws\ @\ xs)$
  assume $(\forall\, n < length\ ws.\ P\ (ws\ !\ n)\ (drop\ (Suc\ n)\ ws\ @\ xs)) \longrightarrow$
  $ws\ @\ xs \simeq \{ws\ @\ ys,\ zs,\ P\}$
  moreover have $\forall\, n < length\ ws.\ P\ (ws\ !\ n)\ (drop\ (Suc\ n)\ ws\ @\ xs)$
  proof (*rule allI, rule impI*)
    fix *n*
    assume $n < length\ ws$
    moreover have $Suc\ n < Suc\ (length\ ws) \longrightarrow$
    $P\ ((w\ \#\ ws)\ !\ (Suc\ n))\ (drop\ (Suc\ n)\ ws\ @\ xs)$
    using *B* **..**
    ultimately show $P\ (ws\ !\ n)\ (drop\ (Suc\ n)\ ws\ @\ xs)$ **by** *simp*
  qed
  ultimately have $ws\ @\ xs \simeq \{ws\ @\ ys,\ zs,\ P\}$ **..**
  moreover have $0 < Suc\ (length\ ws) \longrightarrow P\ ((w\ \#\ ws)\ !\ 0)\ (drop\ 0\ ws\ @\ xs)$
  using *B* **..**
  hence $P\ w\ (ws\ @\ xs)$ **by** *simp*
  ultimately show $w\ \#\ ws\ @\ xs \simeq \{w\ \#\ ws\ @\ ys,\ zs,\ P\}$ **by** (*cases zs, simp-all*)
qed

lemma *interleaves-prefix-fst-2* [*rule-format*]:
 $ws\ @\ xs \simeq \{ws\ @\ ys,\ zs,\ P\} \longrightarrow$
 $(\forall\, n < length\ ws.\ P\ (ws\ !\ n)\ (drop\ (Suc\ n)\ ws\ @\ xs)) \longrightarrow$
 $xs \simeq \{ys,\ zs,\ P\}$
proof (*induction ws, simp-all, (rule impI)+*)
  fix *w ws*
  assume *A*: $\forall\, n < Suc\ (length\ ws).\ P\ ((w\ \#\ ws)\ !\ n)\ (drop\ n\ ws\ @\ xs)$
  hence $0 < Suc\ (length\ ws) \longrightarrow P\ ((w\ \#\ ws)\ !\ 0)\ (drop\ 0\ ws\ @\ xs)$ **..**
  hence $P\ w\ (ws\ @\ xs)$ **by** *simp*
  moreover assume $w\ \#\ ws\ @\ xs \simeq \{w\ \#\ ws\ @\ ys,\ zs,\ P\}$
  ultimately have $ws\ @\ xs \simeq \{ws\ @\ ys,\ zs,\ P\}$ **by** (*cases zs, simp-all*)
  moreover assume $ws\ @\ xs \simeq \{ws\ @\ ys,\ zs,\ P\} \longrightarrow$
  $(\forall\, n < length\ ws.\ P\ (ws\ !\ n)\ (drop\ (Suc\ n)\ ws\ @\ xs)) \longrightarrow$
  $xs \simeq \{ys,\ zs,\ P\}$
  ultimately have $(\forall\, n < length\ ws.\ P\ (ws\ !\ n)\ (drop\ (Suc\ n)\ ws\ @\ xs)) \longrightarrow$
  $xs \simeq \{ys,\ zs,\ P\}$
  by *simp*
  moreover have $\forall\, n < length\ ws.\ P\ (ws\ !\ n)\ (drop\ (Suc\ n)\ ws\ @\ xs)$
  proof (*rule allI, rule impI*)
    fix *n*
    assume $n < length\ ws$
    moreover have $Suc\ n < Suc\ (length\ ws) \longrightarrow$
    $P\ ((w\ \#\ ws)\ !\ (Suc\ n))\ (drop\ (Suc\ n)\ ws\ @\ xs)$
    using *A* **..**
    ultimately show $P\ (ws\ !\ n)\ (drop\ (Suc\ n)\ ws\ @\ xs)$ **by** *simp*
  qed

ultimately show *xs* ≃ {*ys, zs, P*} **..**
qed

lemma *interleaves-prefix-fst* [*rule-format*]:
$\forall\, n < length\ ws.\ P\ (ws\ !\ n)\ (drop\ (Suc\ n)\ ws\ @\ xs) \Longrightarrow$
xs ≃ {*ys, zs, P*} = *ws* @ *xs* ≃ {*ws* @ *ys, zs, P*}
proof (*rule iffI*, *erule interleaves-prefix-fst-1*, *simp*)
qed (*erule interleaves-prefix-fst-2*, *simp*)

lemma *interleaves-prefix-snd* [*rule-format*]:
$\forall\, n < length\ ws.\ \neg\ P\ (ws\ !\ n)\ (drop\ (Suc\ n)\ ws\ @\ xs) \Longrightarrow$
xs ≃ {*ys, zs, P*} = *ws* @ *xs* ≃ {*ys, ws* @ *zs, P*}
proof (*subst* (*1 2*) *interleaves-swap*)
qed (*rule interleaves-prefix-fst*, *simp*)

1.2 A second, stronger version of interleaving

Simple counterexamples show that unlike prefixes, the addition or removal of suffixes to the input lists does not generally preserve the validity of predicate *interleaves*. In fact, if $P\ y\ [x] = True$ with $x \neq y$, then $[y, x] \simeq \{[x], [y], P\}$ does not hold although $[y] \simeq \{[], [y], \lambda w\ ws.\ P\ w\ (ws\ @\ [x])\}$ does, by virtue of lemma *?xs* ≃ {[], *?xs*, *?P*}. Similarly, $[x, y] \simeq \{[], [y, x], \lambda w\ ws.\ P\ w\ (ws\ @\ [x])\}$ does not hold for $x \neq y$ even though $[x, y, x] \simeq \{[x], [y, x], P\}$ does.

Both counterexamples would not work any longer if the truth value of the input predicate were significant even if either the second or the third list is empty. In fact, in the former case, condition $P\ y\ [x] = True$ would entail the falseness of statement $[y] \simeq \{[], [y], \lambda w\ ws.\ P\ w\ (ws\ @\ [x])\}$, so that the validity of rule $[y] \simeq \{[], [y], \lambda w\ ws.\ P\ w\ (ws\ @\ [x])\} \Longrightarrow [y, x] \simeq \{[x], [y], P\}$ would be preserved. In the latter case, statement $[x, y, x] \simeq \{[x], [y, x], P\}$ may only hold provided the last item x of the first list is extracted from the third one, which would require that $\neg\ P\ x\ []$; thus, subordinating rule $[x, y, x] \simeq \{[x], [y, x], P\} \Longrightarrow [x, y] \simeq \{[], [y, x], \lambda w\ ws.\ P\ w\ (ws\ @\ [x])\}$ to condition $P\ x\ []$ would preserve its validity.

This argument suggests that in order to obtain an *interleaves* predicate whose validity is also preserved upon the addition or removal of a suffix to the input lists, the truth value of the input predicate must matter until both the second and the third list are empty. In what follows, such a stronger version of the predicate, named *Interleaves*, is defined along with a convenient symbolic notation for it.

fun *Interleaves* ::
($'a \Rightarrow 'a\ list \Rightarrow bool$) $\Rightarrow 'a\ list \Rightarrow 'a\ list \Rightarrow 'a\ list \Rightarrow bool$ **where**
Interleaves P ($x\ \#\ xs$) ($y\ \#\ ys$) ($z\ \#\ zs$) = (*if P x xs*
  then $x = y \land$ *Interleaves P xs ys* ($z\ \#\ zs$)

else x = z ∧ Interleaves P xs (y # ys) zs) |
Interleaves P (x # xs) (y # ys) [] =
  (P x xs ∧ x = y ∧ Interleaves P xs ys []) |
Interleaves P (x # xs) [] (z # zs) =
  (¬ P x xs ∧ x = z ∧ Interleaves P xs [] zs) |
Interleaves - (- # -) [] [] = False |
Interleaves - [] (- # -) - = False |
Interleaves - [] - (- # -) = False |
Interleaves - [] [] [] = True

abbreviation *Interleaves-syntax* ::
  ′a list ⇒ ′a list ⇒ ′a list ⇒ (′a ⇒ ′a list ⇒ bool) ⇒ bool
  (‹(- ≅ {-, -, -})› [60, 60, 60] 51)
 where *xs ≅ {ys, zs, P} ≡ Interleaves P xs ys zs*

In what follows, it is proven that predicate *Interleaves* is actually not weaker
than, viz. is a sufficient condition for, predicate *interleaves*. Moreover, the
former predicate is shown to fulfil the same rules as the latter, although
sometimes under more stringent assumptions (cf. lemmas *Interleaves-all-nil*,
Interleaves-nil-all with lemmas *?xs ≃ { ?xs, [], ?P}*, *?xs ≃ {[], ?xs, ?P}*),
and to have the further property that under proper assumptions, its validity
is preserved upon the addition or removal of a suffix to the input lists.

lemma *Interleaves-interleaves* [*rule-format*]:
 xs ≅ {ys, zs, P} ⟶ xs ≃ {ys, zs, P}
proof (*induction P xs ys zs rule*: *interleaves.induct, simp-all*)
qed (*rule conjI*, (*rule-tac* [!] *impI*)+, *simp-all*)

lemma *Interleaves-length*:
 xs ≅ {ys, zs, P} ⟹ length xs = length ys + length zs
by (*drule Interleaves-interleaves, rule interleaves-length*)

lemma *Interleaves-nil*:
 [] ≅ {ys, zs, P} ⟹ ys = [] ∧ zs = []
by (*drule Interleaves-interleaves, rule interleaves-nil*)

lemma *Interleaves-swap*:
 xs ≅ {ys, zs, P} = xs ≅ {zs, ys, λw ws. ¬ P w ws}
proof (*induction P xs ys zs rule*: *Interleaves.induct, simp-all*)
  fix *y′* :: *′a* **and** *ys′ zs′ P′*
  show *¬ [] ≅ {zs′, y′ # ys′, λw ws. ¬ P′ w ws}* **by** (*cases zs′, simp-all*)
qed

lemma *Interleaves-equal-fst*:
 xs ≅ {ys, zs, P} ⟹ xs ≅ {ys′, zs, P} ⟹ ys = ys′
by ((*drule Interleaves-interleaves*)+, *rule interleaves-equal-fst*)

lemma *Interleaves-equal-snd*:
  $xs \cong \{ys,\ zs,\ P\} \implies xs \cong \{ys,\ zs',\ P\} \implies zs = zs'$
by ((*drule Interleaves-interleaves*)+, *rule interleaves-equal-snd*)

lemma *Interleaves-equal-all-nil*:
  $xs \cong \{ys,\ [],\ P\} \implies xs = ys$
by (*drule Interleaves-interleaves*, *rule interleaves-equal-all-nil*)

lemma *Interleaves-equal-nil-all*:
  $xs \cong \{[],\ zs,\ P\} \implies xs = zs$
by (*drule Interleaves-interleaves*, *rule interleaves-equal-nil-all*)

lemma *Interleaves-filter* [*rule-format*]:
  assumes *A*: $\forall\, x\ xs.\ P\ x\ (\textit{filter}\ Q\ xs) = P\ x\ xs$
  shows $xs \cong \{ys,\ zs,\ P\} \longrightarrow \textit{filter}\ Q\ xs \cong \{\textit{filter}\ Q\ ys,\ \textit{filter}\ Q\ zs,\ P\}$
proof (*induction xs arbitrary: ys zs, rule-tac* [!] *impI, simp*)
  fix *ys zs*
  assume $[] \cong \{ys,\ zs,\ P\}$
  hence $ys = [] \land zs = []$ **by** (*rule Interleaves-nil*)
  thus $[] \cong \{\textit{filter}\ Q\ ys,\ \textit{filter}\ Q\ zs,\ P\}$ **by** *simp*
next
  fix *x xs ys zs*
  assume
    B: $\bigwedge ys'\ zs'.\ xs \cong \{ys',\ zs',\ P\} \longrightarrow$
      $\textit{filter}\ Q\ xs \cong \{\textit{filter}\ Q\ ys',\ \textit{filter}\ Q\ zs',\ P\}$ **and**
    C: $x\ \#\ xs \cong \{ys,\ zs,\ P\}$
  show $\textit{filter}\ Q\ (x\ \#\ xs) \cong \{\textit{filter}\ Q\ ys,\ \textit{filter}\ Q\ zs,\ P\}$
  proof (*cases ys, case-tac* [!] *zs, simp-all del: filter.simps, rule ccontr*)
    assume $ys = []$ **and** $zs = []$
    thus *False* **using** *C* **by** *simp*
  next
    fix *z zs'*
    assume $ys = []$ **and** $zs = z\ \#\ zs'$
    hence *D*: $\neg\ P\ x\ xs \land x = z \land xs \cong \{[],\ zs',\ P\}$ **using** *C* **by** *simp+*
    moreover have $xs \cong \{[],\ zs',\ P\} \longrightarrow$
      $\textit{filter}\ Q\ xs \cong \{\textit{filter}\ Q\ [],\ \textit{filter}\ Q\ zs',\ P\}$
      using *B* .
    ultimately have $\textit{filter}\ Q\ xs \cong \{[],\ \textit{filter}\ Q\ zs',\ P\}$ **by** *simp*
    moreover have $\neg\ P\ x\ (\textit{filter}\ Q\ xs)$ **using** *A* **and** *D* **by** *simp+*
    ultimately show $\textit{filter}\ Q\ (x\ \#\ xs) \cong \{[],\ \textit{filter}\ Q\ (z\ \#\ zs'),\ P\}$
      using *D* **by** *simp*
  next
    fix *y ys'*
    assume $ys = y\ \#\ ys'$ **and** $zs = []$
    hence *D*: $P\ x\ xs \land x = y \land xs \cong \{ys',\ [],\ P\}$ **using** *C* **by** *simp+*
    moreover have $xs \cong \{ys',\ [],\ P\} \longrightarrow$
      $\textit{filter}\ Q\ xs \cong \{\textit{filter}\ Q\ ys',\ \textit{filter}\ Q\ [],\ P\}$
      using *B* .
    ultimately have $\textit{filter}\ Q\ xs \cong \{\textit{filter}\ Q\ ys',\ [],\ P\}$ **by** *simp*

13

moreover have *P x* (*filter Q xs*) **using** *A* **and** *D* **by** *simp+*
ultimately show *filter Q* (*x # xs*) ≅ {*filter Q* (*y # ys'*), [], *P*}
 using *D* **by** *simp*
next
 fix *y ys' z zs'*
 assume *ys = y # ys'* **and** *zs = z # zs'*
 hence *D*: *x # xs* ≅ {*y # ys', z # zs', P*} **using** *C* **by** *simp*
 show *filter Q* (*x # xs*) ≅ {*filter Q* (*y # ys'*), *filter Q* (*z # zs'*), *P*}
 proof (*cases P x xs*)
  case *True*
  hence *E*: *P x* (*filter Q xs*) **using** *A* **by** *simp*
  have *F*: *x = y ∧ xs* ≅ {*ys', z # zs', P*} **using** *D* **and** *True* **by** *simp*
  moreover have *xs* ≅ {*ys', z # zs', P*} ⟶
   filter Q xs ≅ {*filter Q ys', filter Q* (*z # zs'*), *P*}
   using *B* **.**
  ultimately have *G*: *filter Q xs* ≅ {*filter Q ys', filter Q* (*z # zs'*), *P*}
   by *simp*
  show *?thesis*
  proof (*cases Q x*)
   assume *Q x*
   hence *filter Q* (*x # xs*) = *x # filter Q xs* **by** *simp*
   moreover have *filter Q* (*y # ys'*) = *x # filter Q ys'*
    using ⟨*Q x*⟩ **and** *F* **by** *simp*
   ultimately show *?thesis* **using** *E* **and** *G*
    by (*cases filter Q* (*z # zs'*), *simp-all*)
  next
   assume ¬ *Q x*
   hence *filter Q* (*x # xs*) = *filter Q xs* **by** *simp*
   moreover have *filter Q* (*y # ys'*) = *filter Q ys'*
    using ⟨¬ *Q x*⟩ **and** *F* **by** *simp*
   ultimately show *?thesis* **using** *E* **and** *G*
    by (*cases filter Q* (*z # zs'*), *simp-all*)
  qed
 next
  case *False*
  hence *E*: ¬ *P x* (*filter Q xs*) **using** *A* **by** *simp*
  have *F*: *x = z ∧ xs* ≅ {*y # ys', zs', P*} **using** *D* **and** *False* **by** *simp*
  moreover have *xs* ≅ {*y # ys', zs', P*} ⟶
   filter Q xs ≅ {*filter Q* (*y # ys'*), *filter Q zs', P*}
   using *B* **.**
  ultimately have *G*: *filter Q xs* ≅ {*filter Q* (*y # ys'*), *filter Q zs', P*}
   by *simp*
  show *?thesis*
  proof (*cases Q x*)
   assume *Q x*
   hence *filter Q* (*x # xs*) = *x # filter Q xs* **by** *simp*
   moreover have *filter Q* (*z # zs'*) = *x # filter Q zs'*
    using ⟨*Q x*⟩ **and** *F* **by** *simp*
   ultimately show *?thesis* **using** *E* **and** *G*

14

by (*cases filter Q (y # ys′), simp-all*)
   next
    assume ¬ *Q x*
    hence *filter Q (x # xs) = filter Q xs* **by** *simp*
    moreover have *filter Q (z # zs′) = filter Q zs′*
     using ‹¬ *Q x*› **and** *F* **by** *simp*
    ultimately show *?thesis* **using** *E* **and** *G*
     by (*cases filter Q (z # zs′), simp-all*)
   qed
  qed
 qed
qed

lemma *Interleaves-map* [*rule-format*]:
  assumes *A*: *inj f*
  shows $xs \cong \{ys,\ zs,\ P\} \longrightarrow$
   map f xs $\cong$ {*map f ys, map f zs, λw ws. P (inv f w) (map (inv f) ws)*}
   (**is** $\text{-} \longrightarrow \text{-} \cong \{\text{-},\ \text{-},\ ?P'\}$)
proof (*induction xs arbitrary: ys zs, rule-tac* [!] *impI, simp-all*)
  fix *ys zs*
  assume $[] \cong \{ys,\ zs,\ P\}$
  hence $ys = [] \wedge zs = []$ **by** (*rule Interleaves-nil*)
  thus $[] \cong \{map\ f\ ys,\ map\ f\ zs,\ ?P'\}$ **by** *simp*
next
  fix *x xs ys zs*
  assume
   B: $\bigwedge ys\ zs.\ xs \cong \{ys,\ zs,\ P\} \longrightarrow map\ f\ xs \cong \{map\ f\ ys,\ map\ f\ zs,\ ?P'\}$ **and**
   C: $x\ \#\ xs \cong \{ys,\ zs,\ P\}$
  show $f\ x\ \#\ map\ f\ xs \cong \{map\ f\ ys,\ map\ f\ zs,\ ?P'\}$
  proof (*cases ys, case-tac* [!] *zs, simp-all del: Interleaves.simps(1−3)*)
   assume $ys = []$ **and** $zs = []$
   thus *False* **using** *C* **by** *simp*
  next
   fix *z zs′*
   assume $ys = []$ **and** $zs = z\ \#\ zs′$
   hence *D*: $\neg\ P\ x\ xs \wedge x = z \wedge xs \cong \{[],\ zs′,\ P\}$ **using** *C* **by** *simp+*
   moreover have $xs \cong \{[],\ zs′,\ P\} \longrightarrow map\ f\ xs \cong \{map\ f\ [],\ map\ f\ zs′,\ ?P'\}$
    using *B* .
   ultimately have $map\ f\ xs \cong \{[],\ map\ f\ zs′,\ ?P'\}$ **by** *simp*
   moreover have $\neg\ ?P'\ (f\ x)\ (map\ f\ xs)$ **using** *A* **and** *D* **by** *simp+*
   ultimately show $f\ x\ \#\ map\ f\ xs \cong \{[],\ f\ z\ \#\ map\ f\ zs′,\ ?P'\}$
    using *D* **by** *simp*
  next
   fix *y ys′*
   assume $ys = y\ \#\ ys′$ **and** $zs = []$
   hence *D*: $P\ x\ xs \wedge x = y \wedge xs \cong \{ys′,\ [],\ P\}$ **using** *C* **by** *simp+*
   moreover have $xs \cong \{ys′,\ [],\ P\} \longrightarrow map\ f\ xs \cong \{map\ f\ ys′,\ map\ f\ [],\ ?P'\}$
    using *B* .
   ultimately have $map\ f\ xs \cong \{map\ f\ ys′,\ [],\ ?P'\}$ **by** *simp*

moreover have *?P′ (f x) (map f xs)* **using** *A* **and** *D* **by** *simp+*
ultimately show *f x # map f xs ≅ {f y # map f ys′, [], ?P′}*
 using *D* **by** *simp*
next
 fix *y ys′ z zs′*
 assume *ys = y # ys′* **and** *zs = z # zs′*
 hence *D*: *x # xs ≅ {y # ys′, z # zs′, P}* **using** *C* **by** *simp*
 show *f x # map f xs ≅ {f y # map f ys′, f z # map f zs′, ?P′}*
 proof (*cases P x xs*)
  case *True*
  hence *E*: *?P′ (f x) (map f xs)* **using** *A* **by** *simp*
  have *x = y ∧ xs ≅ {ys′, z # zs′, P}* **using** *D* **and** *True* **by** *simp*
  moreover have *xs ≅ {ys′, z # zs′, P} ⟶*
   map f xs ≅ {map f ys′, map f (z # zs′), ?P′}
   using *B* **.**
  ultimately have *f x = f y ∧ map f xs ≅ {map f ys′, map f (z # zs′), ?P′}*
   by *simp*
  thus *?thesis* **using** *E* **by** *simp*
 next
  case *False*
  hence *E*: *¬ ?P′ (f x) (map f xs)* **using** *A* **by** *simp*
  have *x = z ∧ xs ≅ {y # ys′, zs′, P}* **using** *D* **and** *False* **by** *simp*
  moreover have *xs ≅ {y # ys′, zs′, P} ⟶*
   map f xs ≅ {map f (y # ys′), map f zs′, ?P′}
   using *B* **.**
  ultimately have *f x = f z ∧ map f xs ≅ {map f (y # ys′), map f zs′, ?P′}*
   by *simp*
  thus *?thesis* **using** *E* **by** *simp*
 qed
 qed
qed

lemma *Interleaves-prefix-fst-1* [*rule-format*]:
 assumes *A*: *xs ≅ {ys, zs, P}*
 shows *(∀ n < length ws. P (ws ! n) (drop (Suc n) ws @ xs)) ⟶*
  ws @ xs ≅ {ws @ ys, zs, P}
proof (*induction ws, simp-all add*: *A, rule impI*)
 fix *w ws*
 assume *B*: *∀ n < Suc (length ws). P ((w # ws) ! n) (drop n ws @ xs)*
 assume *(∀ n < length ws. P (ws ! n) (drop (Suc n) ws @ xs)) ⟶*
  ws @ xs ≅ {ws @ ys, zs, P}
 moreover have *∀ n < length ws. P (ws ! n) (drop (Suc n) ws @ xs)*
 proof (*rule allI, rule impI*)
  fix *n*
  assume *n < length ws*
  moreover have *Suc n < Suc (length ws) ⟶*
   P ((w # ws) ! (Suc n)) (drop (Suc n) ws @ xs)
   using *B* **..**
  ultimately show *P (ws ! n) (drop (Suc n) ws @ xs)* **by** *simp*

qed
ultimately have *ws @ xs ≅ {ws @ ys, zs, P}* **..**
moreover have *0 < Suc (length ws) ⟶ P ((w # ws) ! 0) (drop 0 ws @ xs)*
  using *B* **..**
hence *P w (ws @ xs)* **by** *simp*
ultimately show *w # ws @ xs ≅ {w # ws @ ys, zs, P}* **by** (*cases zs, simp-all*)
qed

lemma *Interleaves-prefix-fst-2* [*rule-format*]:
 ws @ xs ≅ {ws @ ys, zs, P} ⟶
 (∀ n < length ws. P (ws ! n) (drop (Suc n) ws @ xs)) ⟶
 xs ≅ {ys, zs, P}
proof (*induction ws, simp-all, (rule impI)+*)
  fix *w ws*
  assume *A*: *∀ n < Suc (length ws). P ((w # ws) ! n) (drop n ws @ xs)*
  hence *0 < Suc (length ws) ⟶ P ((w # ws) ! 0) (drop 0 ws @ xs)* **..**
  hence *P w (ws @ xs)* **by** *simp*
  moreover assume *w # ws @ xs ≅ {w # ws @ ys, zs, P}*
  ultimately have *ws @ xs ≅ {ws @ ys, zs, P}* **by** (*cases zs, simp-all*)
  moreover assume *ws @ xs ≅ {ws @ ys, zs, P} ⟶*
    (∀ n < length ws. P (ws ! n) (drop (Suc n) ws @ xs)) ⟶
    xs ≅ {ys, zs, P}
  ultimately have *(∀ n < length ws. P (ws ! n) (drop (Suc n) ws @ xs)) ⟶*
    xs ≅ {ys, zs, P}
   by *simp*
  moreover have *∀ n < length ws. P (ws ! n) (drop (Suc n) ws @ xs)*
  proof (*rule allI, rule impI*)
    fix *n*
    assume *n < length ws*
    moreover have *Suc n < Suc (length ws) ⟶*
      P ((w # ws) ! (Suc n)) (drop (Suc n) ws @ xs)
     using *A* **..**
    ultimately show *P (ws ! n) (drop (Suc n) ws @ xs)* **by** *simp*
  qed
  ultimately show *xs ≅ {ys, zs, P}* **..**
qed

lemma *Interleaves-prefix-fst* [*rule-format*]:
 ∀ n < length ws. P (ws ! n) (drop (Suc n) ws @ xs) ⟹
 xs ≅ {ys, zs, P} = ws @ xs ≅ {ws @ ys, zs, P}
proof (*rule iffI, erule Interleaves-prefix-fst-1, simp*)
qed (*erule Interleaves-prefix-fst-2, simp*)

lemma *Interleaves-prefix-snd* [*rule-format*]:
 ∀ n < length ws. ¬ P (ws ! n) (drop (Suc n) ws @ xs) ⟹
 xs ≅ {ys, zs, P} = ws @ xs ≅ {ys, ws @ zs, P}
proof (*subst (1 2) Interleaves-swap*)
qed (*rule Interleaves-prefix-fst, simp*)

lemma *Interleaves-all-nil-1* [*rule-format*]:
 $xs \cong \{xs, [], P\} \longrightarrow (\forall\, n < length\ xs.\ P\ (xs\ !\ n)\ (drop\ (Suc\ n)\ xs))$
proof (*induction xs, simp-all, rule impI, erule conjE, rule allI, rule impI*)
  fix *x xs n*
  assume
    $xs \cong \{xs, [], P\} \longrightarrow (\forall\, n < length\ xs.\ P\ (xs\ !\ n)\ (drop\ (Suc\ n)\ xs))$ **and**
    $xs \cong \{xs, [], P\}$
  hence *A*: $\forall\, n < length\ xs.\ P\ (xs\ !\ n)\ (drop\ (Suc\ n)\ xs)$ **..**
  assume
    B: $P\ x\ xs$ **and**
    C: $n < Suc\ (length\ xs)$
  show $P\ ((x\ \#\ xs)\ !\ n)\ (drop\ n\ xs)$
  proof (*cases n, simp-all add*: *B*)
    case (*Suc m*)
    have $m < length\ xs \longrightarrow P\ (xs\ !\ m)\ (drop\ (Suc\ m)\ xs)$ **using** *A* **..**
    moreover have $m < length\ xs$ **using** *C* **and** *Suc* **by** *simp*
    ultimately show $P\ (xs\ !\ m)\ (drop\ (Suc\ m)\ xs)$ **..**
  qed
qed

lemma *Interleaves-all-nil-2* [*rule-format*]:
 $\forall\, n < length\ xs.\ P\ (xs\ !\ n)\ (drop\ (Suc\ n)\ xs) \Longrightarrow xs \cong \{xs, [], P\}$
by (*insert Interleaves-prefix-fst* [*of xs P* [] [] []], *simp*)

lemma *Interleaves-all-nil*:
 $xs \cong \{xs, [], P\} = (\forall\, n < length\ xs.\ P\ (xs\ !\ n)\ (drop\ (Suc\ n)\ xs))$
proof (*rule iffI, rule allI, rule impI, rule Interleaves-all-nil-1, assumption+*)
qed (*rule Interleaves-all-nil-2, simp*)

lemma *Interleaves-nil-all*:
 $xs \cong \{[], xs, P\} = (\forall\, n < length\ xs.\ \neg\, P\ (xs\ !\ n)\ (drop\ (Suc\ n)\ xs))$
by (*subst Interleaves-swap, simp add*: *Interleaves-all-nil*)

lemma *Interleaves-suffix-one-aux*:
  assumes *A*: $P\ x\ []$
  shows $\neg\, xs\ @\ [x] \cong \{[], zs, P\}$
using [[*simproc del*: *defined-all*]]
proof (*induction xs arbitrary*: *zs, simp-all, rule-tac* [!] *notI*)
  fix *zs*
  assume $[x] \cong \{[], zs, P\}$
  thus *False* **by** (*cases zs, simp-all add*: *A*)
next
  fix *w xs zs*
  assume *B*: $\bigwedge zs.\ \neg\, xs\ @\ [x] \cong \{[], zs, P\}$
  assume $w\ \#\ xs\ @\ [x] \cong \{[], zs, P\}$
  thus *False* **proof** (*cases zs, simp-all, (erule-tac conjE*)+)
    fix zs'
    assume $xs\ @\ [x] \cong \{[], zs', P\}$
    moreover have $\neg\, xs\ @\ [x] \cong \{[], zs', P\}$ **using** *B* **.**

ultimately show *False* **by** *contradiction*

  qed

qed

lemma *Interleaves-suffix-one-fst-2* [*rule-format*]:

  assumes *A*: *P x* []

  shows xs @ $[x] \cong \{ys$ @ $[x], zs, P\} \longrightarrow xs \cong \{ys, zs, \lambda w\ ws.\ P\ w\ (ws$ @ $[x])\}$

    (**is** - $\longrightarrow$ - $\cong \{$-, -, *?P′*$\}$)

using [[*simproc del*: *defined-all*]]

proof (*induction xs arbitrary*: *ys zs*, *rule-tac* [!] *impI*, *simp-all*)

  fix *ys zs*

  assume $[x] \cong \{ys$ @ $[x], zs, P\}$

  hence *B*: *length* $[x]$ = *length* (ys @ $[x]$) + *length zs*

   by (*rule Interleaves-length*)

  have *ys*: $ys = []$ **by** (*cases ys*, *simp*, *insert B*, *simp*)

  then have $zs = []$ **by** (*cases zs*, *simp*, *insert B*, *simp*)

  with *ys* **show** $[] \cong \{ys, zs, ?P′\}$ **by** *simp*

next

  fix *w xs ys zs*

  assume *B*: $\bigwedge ys\ zs.\ xs$ @ $[x] \cong \{ys$ @ $[x], zs, P\} \longrightarrow xs \cong \{ys, zs, ?P′\}$

  assume w # xs @ $[x] \cong \{ys$ @ $[x], zs, P\}$

  thus w # $xs \cong \{ys, zs, ?P′\}$

  proof (*cases zs*, *case-tac* [!] *ys*, *simp-all del*: *Interleaves.simps(1,3)*,

   (*erule-tac* [1−2] *conjE*)+)

    assume xs @ $[x] \cong \{[], [], P\}$

    thus *False* **by** (*cases xs*, *simp-all*)

  next

    fix *ys′*

    have xs @ $[x] \cong \{ys′$ @ $[x], [], P\} \longrightarrow xs \cong \{ys′, [], ?P′\}$ **using** *B* .

    moreover assume xs @ $[x] \cong \{ys′$ @ $[x], [], P\}$

    ultimately show $xs \cong \{ys′, [], ?P′\}$ **..**

  next

    fix *z′ zs′*

    assume w # xs @ $[x] \cong \{[x], z′$ # $zs′, P\}$

    thus w # $xs \cong \{[], z′$ # $zs′, ?P′\}$

    proof (*cases P w* (xs @ $[x]$), *simp-all*, *erule-tac* [!] *conjE*)

      assume xs @ $[x] \cong \{[], z′$ # $zs′, P\}$

      moreover have $\neg\ xs$ @ $[x] \cong \{[], z′$ # $zs′, P\}$

       using *A* **by** (*rule Interleaves-suffix-one-aux*)

      ultimately show *False* **by** *contradiction*

    next

      have xs @ $[x] \cong \{[x], zs′, P\} \longrightarrow xs \cong \{[], zs′, ?P′\}$ **using** *B* **by** *simp*

      moreover assume xs @ $[x] \cong \{[x], zs′, P\}$

      ultimately show $xs \cong \{[], zs′, ?P′\}$ **..**

    qed

  next

    fix *y′ ys′ z′ zs′*

    assume w # xs @ $[x] \cong \{y′$ # $ys′$ @ $[x], z′$ # $zs′, P\}$

    thus w # $xs \cong \{y′$ # $ys′, z′$ # $zs′, ?P′\}$

proof (*cases P w (xs @ [x]), simp-all, erule-tac [!] conjE*)
   have $xs @ [x] \cong \{ys' @ [x], z' \# zs', P\} \longrightarrow xs \cong \{ys', z' \# zs', ?P'\}$
    using B **.**
   moreover assume $xs @ [x] \cong \{ys' @ [x], z' \# zs', P\}$
   ultimately show $xs \cong \{ys', z' \# zs', ?P'\}$ **..**
  next
   have $xs @ [x] \cong \{y' \# ys' @ [x], zs', P\} \longrightarrow xs \cong \{y' \# ys', zs', ?P'\}$
    using B **by** *simp*
   moreover assume $xs @ [x] \cong \{y' \# ys' @ [x], zs', P\}$
   ultimately show $xs \cong \{y' \# ys', zs', ?P'\}$ **..**
  qed
 qed
qed

lemma *Interleaves-suffix-fst-1* [*rule-format*]:
  assumes A: $\forall n < length\ ws.\ P\ (ws\ !\ n)\ (drop\ (Suc\ n)\ ws)$
  shows $xs \cong \{ys, zs, \lambda v\ vs.\ P\ v\ (vs @ ws)\} \longrightarrow xs @ ws \cong \{ys @ ws, zs, P\}$
  (**is** $\text{-} \cong \{\text{-, -,}\ ?P'\} \longrightarrow \text{-}$)
using [[*simproc del*: *defined-all*]]
proof (*induction xs arbitrary*: *ys zs, rule-tac* [!] *impI, simp-all*)
 fix *ys zs*
 assume $[] \cong \{ys, zs, ?P'\}$
 hence $ys = [] \land zs = []$ **by** (*rule Interleaves-nil*)
 thus $ws \cong \{ys @ ws, zs, P\}$ **using** A **by** (*simp add*: *Interleaves-all-nil*)
next
 fix *x xs ys zs*
 assume A: $\bigwedge ys\ zs.\ xs \cong \{ys, zs, ?P'\} \longrightarrow xs @ ws \cong \{ys @ ws, zs, P\}$
 assume $x \# xs \cong \{ys, zs, ?P'\}$
 thus $x \# xs @ ws \cong \{ys @ ws, zs, P\}$
 proof (*rule-tac Interleaves.cases* [*of* (*?P', x \# xs, ys, zs*)],
 simp-all del: *Interleaves.simps*(*1*),
 (*erule-tac conjE*)+, (*erule-tac* [*2*] *conjE*)+, (*erule-tac* [*3*] *conjE*)+)
  fix $P'\ x'\ xs'\ y'\ ys'\ z'\ zs'$
  assume
   B: $x' \# xs' \cong \{y' \# ys', z' \# zs', P'\}$ **and**
   C: $?P' = P'$ **and**
   D: $xs = xs'$
  show $x' \# xs' @ ws \cong \{y' \# ys' @ ws, z' \# zs', P\}$
  proof (*cases P' x' xs'*)
   have $xs \cong \{ys', z' \# zs', ?P'\} \longrightarrow xs @ ws \cong \{ys' @ ws, z' \# zs', P\}$
    using A **.**
   moreover case *True*
   hence $xs \cong \{ys', z' \# zs', ?P'\}$ **using** B **and** C **and** D **by** *simp*
   ultimately have $xs @ ws \cong \{ys' @ ws, z' \# zs', P\}$ **..**
   moreover have $P\ x'\ (xs' @ ws)$ **using** C [*symmetric*] **and** *True* **by** *simp*
   moreover have $x' = y'$ **using** B **and** *True* **by** *simp*
   ultimately show *?thesis* **using** D **by** *simp*
  next
   have $xs \cong \{y' \# ys', zs', ?P'\} \longrightarrow xs @ ws \cong \{(y' \# ys') @ ws, zs', P\}$

using A .
          moreover case *False*
          hence $xs \cong \{y' \mathbin{\#} ys',\ zs',\ ?P'\}$ **using** B **and** C **and** D **by** *simp*
          ultimately have $xs \mathbin{@} ws \cong \{(y' \mathbin{\#} ys') \mathbin{@} ws,\ zs',\ P\}$ **..**
          moreover have $\neg\ P\ x'\ (xs' \mathbin{@} ws)$ **using** C [*symmetric*] **and** *False* **by** *simp*
          moreover have $x' = z'$ **using** B **and** *False* **by** *simp*
          ultimately show *?thesis* **using** D **by** *simp*
      qed
   next
      fix $P'\ x'\ xs'\ y'\ ys'$
      have $xs \cong \{ys',\ [],\ ?P'\} \longrightarrow xs \mathbin{@} ws \cong \{ys' \mathbin{@} ws,\ [],\ P\}$ **using** A .
      moreover assume
         $xs' \cong \{ys',\ [],\ P'\}$ **and**
         B: $?P' = P'$ **and**
         C: $xs = xs'$
      hence $xs \cong \{ys',\ [],\ ?P'\}$ **by** *simp*
      ultimately have $xs' \mathbin{@} ws \cong \{ys' \mathbin{@} ws,\ [],\ P\}$ **using** C **by** *simp*
      moreover assume
         $P'\ x'\ xs'$ **and**
         $x' = y'$
      hence $P\ y'\ (xs' \mathbin{@} ws)$ **using** B [*symmetric*] **by** *simp*
      ultimately show $P\ y'\ (xs' \mathbin{@} ws) \wedge xs' \mathbin{@} ws \cong \{ys' \mathbin{@} ws,\ [],\ P\}$ **by** *simp*
   next
      fix $P'\ x'\ xs'\ z'\ zs'$
      have $xs \cong \{[],\ zs',\ ?P'\} \longrightarrow xs \mathbin{@} ws \cong \{[] \mathbin{@} ws,\ zs',\ P\}$ **using** A .
      moreover assume
         $xs' \cong \{[],\ zs',\ P'\}$ **and**
         B: $?P' = P'$ **and**
         C: $xs = xs'$
      hence $xs \cong \{[],\ zs',\ ?P'\}$ **by** *simp*
      ultimately have $xs' \mathbin{@} ws \cong \{ws,\ zs',\ P\}$ **using** C **by** *simp*
      moreover assume
         $\neg\ P'\ x'\ xs'$ **and**
         $x' = z'$
      hence $\neg\ P\ z'\ (xs' \mathbin{@} ws)$ **using** B [*symmetric*] **by** *simp*
      ultimately show $z' \mathbin{\#} xs' \mathbin{@} ws \cong \{ws,\ z' \mathbin{\#} zs',\ P\}$ **by** (*cases ws*, *simp-all*)
   qed
qed

lemma *Interleaves-suffix-one-fst-1* [*rule-format*]:
  $P\ x\ [] \implies$
  $xs \cong \{ys,\ zs,\ \lambda w\ ws.\ P\ w\ (ws \mathbin{@} [x])\} \implies xs \mathbin{@} [x] \cong \{ys \mathbin{@} [x],\ zs,\ P\}$
by (*rule Interleaves-suffix-fst-1*, *simp*)

lemma *Interleaves-suffix-one-fst*:
  $P\ x\ [] \implies$
  $xs \cong \{ys,\ zs,\ \lambda w\ ws.\ P\ w\ (ws \mathbin{@} [x])\} = xs \mathbin{@} [x] \cong \{ys \mathbin{@} [x],\ zs,\ P\}$
proof (*rule iffI*, *rule Interleaves-suffix-one-fst-1*, *assumption+*)
qed (*rule Interleaves-suffix-one-fst-2*)

lemma *Interleaves-suffix-one-snd*:
 ¬ *P x* [] ⟹
 xs ≅ {*ys, zs, λw ws. P w* (*ws* @ [*x*])} = *xs* @ [*x*] ≅ {*ys, zs* @ [*x*], *P*}
by (*subst* (*1 2*) *Interleaves-swap, rule Interleaves-suffix-one-fst*)

lemma *Interleaves-suffix-aux* [*rule-format*]:
 (∀ *n* < *length ws. P* (*ws* ! *n*) (*drop* (*Suc n*) *ws*)) ⟶
 x # *xs* @ *ws* ≅ {*ws, zs, P*} ⟶
 ¬ *P x* (*xs* @ *ws*)
proof (*induction ws arbitrary*: *P rule*: *rev-induct, simp-all,*
 rule impI, (*rule-tac* [*2*] *impI*)+)
  fix *P*
  assume *x* # *xs* ≅ {[], *zs, P*}
  thus ¬ *P x xs* **by** (*cases zs, simp-all*)
next
 fix *w ws P*
 assume
   A: ⋀*P'*. (∀ *n* < *length ws. P'* (*ws* ! *n*) (*drop* (*Suc n*) *ws*)) ⟶
     x # *xs* @ *ws* ≅ {*ws, zs, P'*} ⟶ ¬ *P' x* (*xs* @ *ws*) **and**
   B: ∀ *n* < *Suc* (*length ws*). *P* ((*ws* @ [*w*]) ! *n*)
     (*drop* (*Suc n*) *ws* @ *drop* (*Suc n* − *length ws*) [*w*])
  assume *x* # *xs* @ *ws* @ [*w*] ≅ {*ws* @ [*w*], *zs, P*}
  hence *C*: (*x* # *xs* @ *ws*) @ [*w*] ≅ {*ws* @ [*w*], *zs, P*} **by** *simp*
  let *?P'* = *λv vs. P v* (*vs* @ [*w*])
  have (∀ *n* < *length ws. ?P'* (*ws* ! *n*) (*drop* (*Suc n*) *ws*)) ⟶
    x # *xs* @ *ws* ≅ {*ws, zs, ?P'*} ⟶ ¬ *?P' x* (*xs* @ *ws*)
   using *A* **.**
  moreover have ∀ *n* < *length ws. ?P'* (*ws* ! *n*) (*drop* (*Suc n*) *ws*)
  proof (*rule allI, rule impI*)
    fix *n*
    assume *D*: *n* < *length ws*
    moreover have *n* < *Suc* (*length ws*) ⟶ *P* ((*ws* @ [*w*]) ! *n*)
      (*drop* (*Suc n*) *ws* @ *drop* (*Suc n* − *length ws*) [*w*])
     using *B* **..**
    ultimately have *P* ((*ws* @ [*w*]) ! *n*) (*drop* (*Suc n*) *ws* @ [*w*]) **by** *simp*
    moreover have *n* < *length* (*butlast* (*ws* @ [*w*])) **using** *D* **by** *simp*
    hence *butlast* (*ws* @ [*w*]) ! *n* = (*ws* @ [*w*]) ! *n* **by** (*rule nth-butlast*)
    ultimately show *P* (*ws* ! *n*) (*drop* (*Suc n*) *ws* @ [*w*]) **by** *simp*
  qed
  ultimately have *x* # *xs* @ *ws* ≅ {*ws, zs, ?P'*} ⟶ ¬ *?P' x* (*xs* @ *ws*) **..**
  moreover have *length ws* < *Suc* (*length ws*) ⟶ *P* ((*ws* @ [*w*]) ! *length ws*)
    (*drop* (*Suc* (*length ws*)) *ws* @ *drop* (*Suc* (*length ws*) − *length ws*) [*w*])
   using *B* **..**
  hence *P w* [] **by** *simp*
  hence *x* # *xs* @ *ws* ≅ {*ws, zs, ?P'*}
   using *C* **by** (*rule Interleaves-suffix-one-fst-2*)
  ultimately have ¬ *?P' x* (*xs* @ *ws*) **..**
  thus ¬ *P x* (*xs* @ *ws* @ [*w*]) **by** *simp*

qed

lemma *Interleaves-suffix-fst-2* [*rule-format*]:
  assumes *A*: ∀ *n* < *length ws. P* (*ws* ! *n*) (*drop* (*Suc n*) *ws*)
  shows *xs* @ *ws* ≅ {*ys* @ *ws, zs, P*} ⟶ *xs* ≅ {*ys, zs, λv vs. P v* (*vs* @ *ws*)}
    (**is** - ⟶ - ≅ {-, -, *?P′*})
using [[*simproc del*: *defined-all*]]
proof (*induction xs arbitrary*: *ys zs, rule-tac* [!] *impI, simp-all*)
  fix *ys zs*
  assume *ws* ≅ {*ys* @ *ws, zs, P*}
  hence *B*: *length ws* = *length* (*ys* @ *ws*) + *length zs*
   by (*rule Interleaves-length*)
  have *ys*: *ys* = [] **by** (*cases ys, simp, insert B, simp*)
  then have *zs* = [] **by** (*cases zs, simp, insert B, simp*)
  with *ys* **show** [] ≅ {*ys, zs, ?P′*} **by** *simp*
next
  fix *x xs ys zs*
  assume *B*: ⋀*ys zs. xs* @ *ws* ≅ {*ys* @ *ws, zs, P*} ⟶ *xs* ≅ {*ys, zs, ?P′*}
  assume *x* # *xs* @ *ws* ≅ {*ys* @ *ws, zs, P*}
  thus *x* # *xs* ≅ {*ys, zs, ?P′*}
  proof (*cases zs, case-tac* [!] *ys, simp-all del*: *Interleaves.simps*(*1,3*),
  (*erule-tac* [*2*] *conjE*)+)
    assume *C*: *x* # *xs* @ *ws* ≅ {*ws*, [], *P*}
    have *length* (*x* # *xs* @ *ws*) = *length ws* + *length* []
     by (*insert Interleaves-length* [*OF C*], *simp*)
    thus *False* **by** *simp*
  next
    fix *ys′*
    have *xs* @ *ws* ≅ {*ys′* @ *ws*, [], *P*} ⟶ *xs* ≅ {*ys′*, [], *?P′*} **using** *B* .
    moreover assume *xs* @ *ws* ≅ {*ys′* @ *ws*, [], *P*}
    ultimately show *xs* ≅ {*ys′*, [], *?P′*} ..
  next
    fix *z′ zs′*
    assume *x* # *xs* @ *ws* ≅ {*ws, z′* # *zs′, P*}
    thus *x* # *xs* ≅ {[], *z′* # *zs′, ?P′*}
    proof (*cases P x* (*xs* @ *ws*), *simp-all*)
      case *True*
      moreover assume *x* # *xs* @ *ws* ≅ {*ws, z′* # *zs′, P*}
      with *A* [*rule-format*] **have** ¬ *P x* (*xs* @ *ws*)
       by (*rule Interleaves-suffix-aux*)
      ultimately show *False* **by** *contradiction*
    next
      case *False*
      moreover assume *x* # *xs* @ *ws* ≅ {*ws, z′* # *zs′, P*}
      ultimately have *x* = *z′* ∧ *xs* @ *ws* ≅ {*ws, zs′, P*} **by** (*cases ws, simp-all*)
      moreover have *xs* @ *ws* ≅ {[] @ *ws, zs′, P*} ⟶ *xs* ≅ {[], *zs′, ?P′*}
       using *B* .
      ultimately show *x* = *z′* ∧ *xs* ≅ {[], *zs′, ?P′*} **by** *simp*
    qed

next
 fix y' ys' z' zs'
 assume $x \# xs @ ws \cong \{y' \# ys' @ ws, z' \# zs', P\}$
 thus $x \# xs \cong \{y' \# ys', z' \# zs', ?P'\}$
 proof (*cases P x (xs @ ws), simp-all, erule-tac* [!] *conjE*)
  have $xs @ ws \cong \{ys' @ ws, z' \# zs', P\} \longrightarrow xs \cong \{ys', z' \# zs', ?P'\}$
   using B .
  moreover assume $xs @ ws \cong \{ys' @ ws, z' \# zs', P\}$
  ultimately show $xs \cong \{ys', z' \# zs', ?P'\}$ ..
 next
  have $xs @ ws \cong \{y' \# ys' @ ws, zs', P\} \longrightarrow xs \cong \{y' \# ys', zs', ?P'\}$
   using B **by** *simp*
  moreover assume $xs @ ws \cong \{y' \# ys' @ ws, zs', P\}$
  ultimately show $xs \cong \{y' \# ys', zs', ?P'\}$ ..
 qed
 qed
qed

lemma *Interleaves-suffix-fst* [*rule-format*]:
 $\forall\, n < length\ ws.\ P\ (ws\ !\ n)\ (drop\ (Suc\ n)\ ws) \Longrightarrow$
 $xs \cong \{ys, zs, \lambda v\ vs.\ P\ v\ (vs @ ws)\} = xs @ ws \cong \{ys @ ws, zs, P\}$
proof (*rule iffI, rule Interleaves-suffix-fst-1, simp-all*)
qed (*rule Interleaves-suffix-fst-2, simp*)

lemma *Interleaves-suffix-snd* [*rule-format*]:
 $\forall\, n < length\ ws.\ \neg\ P\ (ws\ !\ n)\ (drop\ (Suc\ n)\ ws) \Longrightarrow$
 $xs \cong \{ys, zs, \lambda v\ vs.\ P\ v\ (vs @ ws)\} = xs @ ws \cong \{ys, zs @ ws, P\}$
by (*subst* (*1 2*) *Interleaves-swap, rule Interleaves-suffix-fst, simp*)

end

References

[1] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, Inc., 1985.

[2] A. Krauss. *Defining Recursive Functions in Isabelle/HOL.* http://isabelle.in.tum.de/website-Isabelle2015/dist/Isabelle2015/doc/functions.pdf.

[3] T. Nipkow. *A Tutorial Introduction to Structured Isar Proofs.* http://isabelle.in.tum.de/website-Isabelle2011/dist/Isabelle2011/doc/isar-overview.pdf.

[4] T. Nipkow. *Programming and Proving in Isabelle/HOL,* May 2015. http://isabelle.in.tum.de/website-Isabelle2015/dist/Isabelle2015/doc/prog-prove.pdf.

[5] T. Nipkow, L. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, May 2015. http://isabelle.in.tum.de/website-Isabelle2015/dist/Isabelle2015/doc/tutorial.pdf.

[6] P. Noce. The ipurge unwinding theorem for csp noninterference security. *Archive of Formal Proofs*, June 2015. http://isa-afp.org/entries/Noninterference_Ipurge_Unwinding.shtml, Formal proof development.