

# 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

February 6, 2026

## 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

<b>1 List interleaving</b>	<b>1</b>
1.1 A first version of interleaving . . . . .	2
1.2 A second, stronger version of interleaving . . . . .	5

## 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  $s$ ,  $t$ ,  $u$  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  $\wedge$  interleaves P xs ys (z # zs)
    else x = z  $\wedge$  interleaves P xs (y # ys) zs) |
  interleaves P (x # xs) (y # ys) [] =
    (x = y  $\wedge$  interleaves P xs ys []) |
  interleaves P (x # xs) [] (z # zs) =
    (x = z  $\wedge$  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**  $xs \simeq \{ys, zs, P\} \equiv \text{interleaves } P \text{ } xs \text{ } ys \text{ } zs$

The advantage provided by this enhanced *interleaves* predicate is that in case  $xs \simeq \{ys, zs, P\}$ , fixing the values of  $xs$  and either  $ys$  or  $zs$  has the effect of fixing the value of the remaining list, too. Namely, if  $xs \simeq \{ys', zs, P\}$  also holds, then  $ys = ys'$ , and likewise, if  $xs \simeq \{ys, zs', P\}$  also holds, then  $zs = zs'$ . Therefore, once two *interleaves* statements  $xs \simeq \{ys, zs, P\}$ ,  $xs' \simeq \{ys', zs', P'\}$  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  $xs \simeq \{ys, zs, P\}$ , 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  $y \neq z$ ,  $P \ y \ [z] = \text{True}$ ,  $P \ z \ [y] = \text{False}$ , in which case both of the *interleaves* statements  $[y, z] \simeq \{ys, zs, P\}$  and  $[z, y] \simeq \{ys, zs, P\}$  hold.

**lemma** *interleaves-length* [rule-format]:

$xs \simeq \{ys, zs, P\} \longrightarrow \text{length } xs = \text{length } ys + \text{length } zs$   
 ⟨proof⟩

**lemma** *interleaves-nil*:

$[] \simeq \{ys, zs, P\} \Longrightarrow ys = [] \wedge zs = []$   
 ⟨proof⟩

**lemma** *interleaves-swap*:

$xs \simeq \{ys, zs, P\} = xs \simeq \{zs, ys, \lambda w \ ws. \neg P \ w \ ws\}$   
 ⟨proof⟩

**lemma** *interleaves-equal-fst* [rule-format]:

$xs \simeq \{ys, zs, P\} \longrightarrow xs \simeq \{ys', zs, P\} \longrightarrow ys = ys'$   
 ⟨proof⟩

**lemma** *interleaves-equal-snd*:

$xs \simeq \{ys, zs, P\} \Longrightarrow xs \simeq \{ys, zs', P\} \Longrightarrow zs = zs'$   
 ⟨proof⟩

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*:

$xs \simeq \{xs, [], P\}$   
 $\langle proof \rangle$

**lemma** *interleaves-nil-all*:

$xs \simeq \{[], xs, P\}$   
 $\langle proof \rangle$

**lemma** *interleaves-equal-all-nil*:

$xs \simeq \{ys, [], P\} \implies xs = ys$   
 $\langle proof \rangle$

**lemma** *interleaves-equal-nil-all*:

$xs \simeq \{[], zs, P\} \implies xs = zs$   
 $\langle proof \rangle$

**lemma** *interleaves-filter* [rule-format]:

**assumes**  $A: \forall x xs. P x (\text{filter } Q xs) = P x xs$   
**shows**  $xs \simeq \{ys, zs, P\} \longrightarrow \text{filter } Q xs \simeq \{\text{filter } Q ys, \text{filter } Q zs, P\}$   
 $\langle proof \rangle$

**lemma** *interleaves-map* [rule-format]:

**assumes**  $A: \text{inj } f$   
**shows**  $xs \simeq \{ys, zs, P\} \longrightarrow$   
 $\text{map } f xs \simeq \{\text{map } f ys, \text{map } f zs, \lambda w ws. P (\text{inv } f w) (\text{map } (\text{inv } f) ws)\}$   
**(is**  $- \longrightarrow - \simeq \{-, -, ?P'\}$   
 $\langle proof \rangle$

**lemma** *interleaves-prefix-fst-1* [rule-format]:

**assumes**  $A: xs \simeq \{ys, zs, P\}$   
**shows**  $(\forall n < \text{length } ws. P (ws ! n) (\text{drop } (\text{Suc } n) ws @ xs)) \longrightarrow$   
 $ws @ xs \simeq \{ws @ ys, zs, P\}$   
 $\langle proof \rangle$

**lemma** *interleaves-prefix-fst-2* [rule-format]:

$ws @ xs \simeq \{ws @ ys, zs, P\} \longrightarrow$   
 $(\forall n < \text{length } ws. P (ws ! n) (\text{drop } (\text{Suc } n) ws @ xs)) \longrightarrow$   
 $xs \simeq \{ys, zs, P\}$   
 $\langle proof \rangle$

**lemma** *interleaves-prefix-fst* [rule-format]:

$\forall n < \text{length } ws. P (ws ! n) (\text{drop } (\text{Suc } n) ws @ xs) \implies$   
 $xs \simeq \{ys, zs, P\} = ws @ xs \simeq \{ws @ ys, zs, P\}$   
 <proof>

**lemma** *interleaves-prefix-snd* [rule-format]:  
 $\forall n < \text{length } ws. \neg P (ws ! n) (\text{drop } (\text{Suc } n) ws @ xs) \implies$   
 $xs \simeq \{ys, zs, P\} = ws @ xs \simeq \{ys, ws @ zs, P\}$   
 <proof>

## 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  $P y [x] = \text{True}$  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 \simeq \{\[], ?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  $P y [x] = \text{True}$  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])\} \implies [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\} \implies [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  $\wedge$  *Interleaves* P xs ys (z # zs)  
   else x = z  $\wedge$  *Interleaves* P xs (y # ys) zs) |  
*Interleaves* P (x # xs) (y # ys) [] =  
 (P x xs  $\wedge$  x = y  $\wedge$  *Interleaves* P xs ys []) |  
*Interleaves* P (x # xs) [] (z # zs) =  
 ( $\neg$  P x xs  $\wedge$  x = z  $\wedge$  *Interleaves* P xs [] zs) |  
*Interleaves* - (- # -) [] [] = False |

$Interleaves - [] (- \# -) - = False \mid$   
 $Interleaves - [] - (- \# -) = False \mid$   
 $Interleaves - [] [] = True$

**abbreviation** *Interleaves-syntax* ::

$'a list \Rightarrow 'a list \Rightarrow 'a list \Rightarrow ('a \Rightarrow 'a list \Rightarrow bool) \Rightarrow bool$

$\langle (- \cong \{-, -, -\}) \rangle [60, 60, 60] 51$

**where**  $xs \cong \{ys, zs, P\} \equiv Interleaves P xs ys zs$

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  $?xs \simeq \{?xs, [], ?P\}$ ,  $?xs \simeq \{[], ?xs, ?P\}$ ), 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*]:

$xs \cong \{ys, zs, P\} \longrightarrow xs \simeq \{ys, zs, P\}$   
 $\langle proof \rangle$

**lemma** *Interleaves-length*:

$xs \cong \{ys, zs, P\} \Longrightarrow length xs = length ys + length zs$   
 $\langle proof \rangle$

**lemma** *Interleaves-nil*:

$[] \cong \{ys, zs, P\} \Longrightarrow ys = [] \wedge zs = []$   
 $\langle proof \rangle$

**lemma** *Interleaves-swap*:

$xs \cong \{ys, zs, P\} = xs \cong \{zs, ys, \lambda w ws. \neg P w ws\}$   
 $\langle proof \rangle$

**lemma** *Interleaves-equal-fst*:

$xs \cong \{ys, zs, P\} \Longrightarrow xs \cong \{ys', zs, P\} \Longrightarrow ys = ys'$   
 $\langle proof \rangle$

**lemma** *Interleaves-equal-snd*:

$xs \cong \{ys, zs, P\} \Longrightarrow xs \cong \{ys, zs', P\} \Longrightarrow zs = zs'$   
 $\langle proof \rangle$

**lemma** *Interleaves-equal-all-nil*:

$xs \cong \{ys, [], P\} \Longrightarrow xs = ys$   
 $\langle proof \rangle$

**lemma** *Interleaves-equal-nil-all*:

$xs \cong \{[], zs, P\} \Longrightarrow xs = zs$

$\langle proof \rangle$

**lemma** *Interleaves-filter* [rule-format]:

**assumes**  $A: \forall x xs. P x (filter Q xs) = P x xs$

**shows**  $xs \cong \{ys, zs, P\} \longrightarrow filter Q xs \cong \{filter Q ys, filter Q zs, P\}$

$\langle proof \rangle$

**lemma** *Interleaves-map* [rule-format]:

**assumes**  $A: inj f$

**shows**  $xs \cong \{ys, zs, P\} \longrightarrow$

$map f xs \cong \{map f ys, map f zs, \lambda w ws. P (inv f w) (map (inv f) ws)\}$

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

$\langle proof \rangle$

**lemma** *Interleaves-prefix-fst-1* [rule-format]:

**assumes**  $A: xs \cong \{ys, zs, P\}$

**shows**  $(\forall n < length ws. P (ws ! n) (drop (Suc n) ws @ xs)) \longrightarrow$

$ws @ xs \cong \{ws @ ys, zs, P\}$

$\langle proof \rangle$

**lemma** *Interleaves-prefix-fst-2* [rule-format]:

$ws @ xs \cong \{ws @ ys, zs, P\} \longrightarrow$

$(\forall n < length ws. P (ws ! n) (drop (Suc n) ws @ xs)) \longrightarrow$

$xs \cong \{ys, zs, P\}$

$\langle proof \rangle$

**lemma** *Interleaves-prefix-fst* [rule-format]:

$\forall n < length ws. P (ws ! n) (drop (Suc n) ws @ xs) \implies$

$xs \cong \{ys, zs, P\} = ws @ xs \cong \{ws @ ys, zs, P\}$

$\langle proof \rangle$

**lemma** *Interleaves-prefix-snd* [rule-format]:

$\forall n < length ws. \neg P (ws ! n) (drop (Suc n) ws @ xs) \implies$

$xs \cong \{ys, zs, P\} = ws @ xs \cong \{ys, ws @ zs, P\}$

$\langle proof \rangle$

**lemma** *Interleaves-all-nil-1* [rule-format]:

$xs \cong \{xs, [], P\} \longrightarrow (\forall n < length xs. P (xs ! n) (drop (Suc n) xs))$

$\langle proof \rangle$

**lemma** *Interleaves-all-nil-2* [rule-format]:

$\forall n < length xs. P (xs ! n) (drop (Suc n) xs) \implies xs \cong \{xs, [], P\}$

$\langle proof \rangle$

**lemma** *Interleaves-all-nil*:

$xs \cong \{xs, [], P\} = (\forall n < length xs. P (xs ! n) (drop (Suc n) xs))$

$\langle proof \rangle$

**lemma** *Interleaves-nil-all*:

$xs \cong \{[], xs, P\} = (\forall n < \text{length } xs. \neg P (xs ! n) (\text{drop } (Suc\ n) xs))$   
 ⟨proof⟩

**lemma** *Interleaves-suffix-one-aux*:

**assumes**  $A: P\ x\ []$   
**shows**  $\neg xs @ [x] \cong \{[], zs, P\}$   
 ⟨proof⟩

**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  $- \longrightarrow - \cong \{-, -, ?P'\}$ )  
 ⟨proof⟩

**lemma** *Interleaves-suffix-fst-1* [rule-format]:

**assumes**  $A: \forall n < \text{length } ws. P (ws ! n) (\text{drop } (Suc\ n) ws)$   
**shows**  $xs \cong \{ys, zs, \lambda v\ vs. P\ v\ (vs @ ws)\} \longrightarrow xs @ ws \cong \{ys @ ws, zs, P\}$   
 (is  $- \cong \{-, -, ?P'\} \longrightarrow -$ )  
 ⟨proof⟩

**lemma** *Interleaves-suffix-one-fst-1* [rule-format]:

$P\ x\ [] \implies$   
 $xs \cong \{ys, zs, \lambda w\ ws. P\ w\ (ws @ [x])\} \implies xs @ [x] \cong \{ys @ [x], zs, P\}$   
 ⟨proof⟩

**lemma** *Interleaves-suffix-one-fst*:

$P\ x\ [] \implies$   
 $xs \cong \{ys, zs, \lambda w\ ws. P\ w\ (ws @ [x])\} = xs @ [x] \cong \{ys @ [x], zs, P\}$   
 ⟨proof⟩

**lemma** *Interleaves-suffix-one-snd*:

$\neg P\ x\ [] \implies$   
 $xs \cong \{ys, zs, \lambda w\ ws. P\ w\ (ws @ [x])\} = xs @ [x] \cong \{ys, zs @ [x], P\}$   
 ⟨proof⟩

**lemma** *Interleaves-suffix-aux* [rule-format]:

$(\forall n < \text{length } ws. P (ws ! n) (\text{drop } (Suc\ n) ws)) \longrightarrow$   
 $x \# xs @ ws \cong \{ws, zs, P\} \longrightarrow$   
 $\neg P\ x\ (xs @ ws)$   
 ⟨proof⟩

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

**assumes**  $A: \forall n < \text{length } ws. P (ws ! n) (\text{drop } (Suc\ n) ws)$   
**shows**  $xs @ ws \cong \{ys @ ws, zs, P\} \longrightarrow xs \cong \{ys, zs, \lambda v\ vs. P\ v\ (vs @ ws)\}$   
 (is  $- \longrightarrow - \cong \{-, -, ?P'\}$ )  
 ⟨proof⟩

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

$\forall n < \text{length } ws. P (ws ! n) (\text{drop } (Suc\ n) ws) \implies$

$xs \cong \{ys, zs, \lambda v vs. P v (vs @ ws)\} = xs @ ws \cong \{ys @ ws, zs, P\}$   
(proof)

**lemma** *Interleaves-suffix-snd* [rule-format]:

$\forall n < \text{length } ws. \neg P (ws ! n) (\text{drop } (Suc\ n) ws) \implies$   
 $xs \cong \{ys, zs, \lambda v vs. P v (vs @ ws)\} = xs @ ws \cong \{ys, zs @ ws, P\}$   
(proof)

**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](http://isa-afp.org/entries/Noninterference_Ipurge_Unwinding.shtml), Formal proof development.