Reasoning about Lists via List Interleaving

Pasquale Noce
Security Certification Specialist at Arjo Systems - Gep S.p.A.
pasquale dot noce dot lavoro at gmai1 dot com
pasquale dot noce at arjowiggins-it dot com

September 19, 2015

Abstract

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 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 1
   1.1 A first version of interleaving .............................. 2
   1.2 A second, stronger version of interleaving .................. 11

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 $s$, $t$, and $u$, 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 \textit{interleaves} predicate turns out to permit the proof of equalities between lists without the need of an induction. Some rules that allow to infer \textit{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 \textit{Interleaves}, is shown to fulfill 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 \textit{interleaves}, as well as of a convenient symbolic notation for it. As in the definition of predicate \textit{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 \textit{interleaves} with two further lists can nevertheless be proven by induction using as input $\text{rev } ws$, rather than $ws$ itself.

With respect to Hoare’s homonymous predicate, \textit{interleaves} takes as an additional input a predicate $P$, which is a function of a single item and a list. Then, for statement $\text{interleaves } P (x \# xs) (y \# ys) (z \# zs)$ 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.

\[
\begin{align*}
\text{fun } \text{interleaves } :: & \quad \text{('a list ⇒ bool) ⇒ 'a list ⇒ 'a list ⇒ 'a list ⇒ bool where} \\
\text{interleaves } P (x \# xs) (y \# ys) (z \# zs) & = (\text{if } P x xs \\
\text{then } x = y \land \text{interleaves } P xs ys (z \# zs) \\
\text{else } x = z \land \text{interleaves } P xs (y \# ys) zs) \\
\text{interleaves } P (x \# xs) (y \# ys) [] & = \\
(x = y \land \text{interleaves } P xs ys []) \\
\text{interleaves } P (x \# xs) [] (z \# zs) & = \\
(x = z \land \text{interleaves } P xs [] zs) \\
\text{interleaves } - (- \# -) [] & = \text{False} \\
\text{interleaves } - [] (- \# -) & = \text{False}
\end{align*}
\]
interleaves - [] - (· # ·) = False |
interleaves - [•] [] = True

syntax (xsymbols) -interleaves ::
't a list ⇒ 'a list ⇒ 'a list ⇒ ('a ⇒ 'a list ⇒ bool) ⇒ bool
((· ≃ {·, ·, ·}) [60, 60, 60] 51)

translations
xs ≃ {ys, zs, P} ⇔ CONST 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 zs 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 [!] impl)+, 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 [!] impl)+)

3
fix $ys$ $ys'$ $zs$
assume $[] \simeq \{ys, zs, P\}$
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 $A$: $\forall ys \ ys' \ zs. \ (ys \ zs) \simeq \{ys, zs, P\} \rightarrow x s \simeq \{ys', zs, P\} \rightarrow ys = ys'$ and
$B$: $x \neq xs \simeq \{ys, zs, P\}$ and
$B'$: $x \neq xs \simeq \{ys', zs, P\}$
show $ys = ys'$

assume $C$: $zs = []$
hence $\exists w \ ws. \ ys = w \neq ws$ using $B$ by (cases $ys$, simp-all)
then obtain $w \ ws$ where $Y$: $ys = w \neq ws$ 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 $\exists w'\ ws'. \ ys' = w' \neq ws'$
using $B'$ and $C$ by (cases $ys'$, simp-all)
then obtain $w' \ ws'$ where $Y'$: $ys' = w' \neq ws'$ 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 $xs \simeq \{ws, [], P\} \rightarrow x s \simeq \{ws', [], P\} \rightarrow ws = ws'$
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 $C$: $zs = v \neq vs$ and $ys = []$
hence $D$: $xs \simeq \{[], vs, P\}$ using $B$ by simp
assume $E$: $ys' = w' \neq ws'$
have $P \ x \ xs \lor \neg P \ x \ xs$ by simp
moreover {
assume $P \ x \ xs$
hence $xs \simeq \{ws', v \neq vs, P\}$ using $B'$ and $C$ and $E$ by simp
hence $\text{length} \ xs = \text{Suc} (\text{length} \ vs) + \text{length} \ ws'$
by (simp add: interleaves-length)
moreover have $\text{length} \ xs = \text{length} \ vs$
using $D$ by (simp add: interleaves-length)
ultimately have $\text{False}$ by simp
}
moreover {
assume $\neg P \ x \ xs$
hence $xs \simeq \{w' \neq ws', vs, P\}$ using $B'$ and $C$ and $E$ by simp
moreover have $xs \simeq \{[], vs, P\} \rightarrow x s \simeq \{w' \neq ws', vs, P\} \rightarrow
\[] = w' \neq ws'$
using $A$.
}
ultimately have $[] = w' \# ws'$ using $D$ by simp
hence False by simp }
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 $\text{length } \xs = \text{Suc (length } vs) + \text{length } ws$
  by (simp add: interleaves-length)
moreover have $\text{length } \xs = \text{length } vs$
  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 $D$ by simp
moreover have $\xs \simeq \{ [], \; vs, \; P \} \rightarrow \xs \simeq \{ w \# ws, \; vs, \; P \} \rightarrow [] = w \# ws$
  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 $\xs \simeq \{ ws, \; v \# vs, \; P \} \rightarrow \xs \simeq \{ ws', \; v \# vs, \; P \} \rightarrow \text{ws} = \text{ws}'$
  using $A$.
ultimately have $\text{ws} = \text{ws'}$ by simp
hence $w = w' \land w = w' \land w = w'$ 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 \( xs \simeq \{ w \# ws, vs, P \} \rightarrow xs \simeq \{ w' \# ws', vs, P \} \rightarrow \)
\[ w \# ws = w' \# ws' \]
using \( A \).
ultimately have \( w \# ws = w' \# ws' \) by simp
hence \( w = w' \land ws = ws' \) by simp

ultimately show \( w = w' \land ws = ws' \).

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 \( A : \forall x xs. P x \ (\text{filter Q} \ xs) = P x \ xs \)
shows \( xs \simeq \{ ys, zs, P \} \rightarrow \text{filter Q} \ xs \simeq \{ \text{filter Q} \ ys, \text{filter Q} \ zs, P \} \)
proof (induction xs arbitrary: ys zs, rule-tac [!] implI, simp)
fix \( ys zs \)
assume \( [] \simeq \{ ys, zs, P \} \)
hence \( ys = [] \land zs = [] \) by (rule interleaves-nil)
thus \( [] \simeq \{ \text{filter Q} \ ys, \text{filter Q} \ zs, P \} \) by simp
next
fix \( x \ xs \ ys zs \)
assume 
B: \(\forall ys', zs', xs \simeq \{ys', zs', P\} \rightarrow\)
filter Q xs \simeq \{filter Q ys', filter Q zs', P\} and
C: \(x \neq xs \simeq \{ys, zs, P\}\)
show filter Q (x \neq xs) \simeq \{filter Q ys, filter Q zs, P\}
proof (cases ys, case-tac \([\_]\)zs, simp-del: filter.simps, rule ccontr)
assume ys = [\_] and zs = [\_]
thus False using C by simp
next
fix y ys'
assume ys = y \# ys' and zs = [\_]

hence D: \(x = y \land xs \simeq \{ys', [\_], P\}\) using C by simp
moreover have xs \simeq \{ys', [\_], P\} \rightarrow
filter Q xs \simeq \{filter Q ys', filter Q [\_], P\}
using B .
ultimately have filter Q xs \simeq \{filter Q ys', [\_], P\} by simp
thus filter Q (x \neq xs) \simeq \{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: \(x \# xs \simeq \{y \# ys', z \# zs', P\}\) using C by simp
show filter Q (x \# xs) \simeq \{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: \(x = y \land xs \simeq \{ys', z \# zs', P\}\) using D and True by simp
moreover have xs \simeq \{ys', z \# zs', P\} \rightarrow
filter Q xs \simeq \{filter Q ys', filter Q (z \# zs'), P\}
using B .
ultimately have G: filter Q xs \simeq \{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
\textbf{lemma} interleaves-map [rule-format]:
\textbf{assumes} A: inj f
\textbf{shows} \(xs \simeq \{ys, zs, P\} \longrightarrow\)
\(\text{map f} \ xy \simeq \{\text{map f} \ y, \text{map f} \ z, \lambda w \ ws. \ P (\text{inv f} \ w) (\text{map (inv f)} \ ws)\}\)
\textbf{proof} (induction \(xs\) arbitrary: \(ys \ zs\), rule-tac \([\!]\) \(\text{impl}\), \(\text{simp-all}\))
\textbf{fix} \(ys \ zs\)
\textbf{assume} \(\| \simeq \{ys, zs, P\}\)
\textbf{hence} \(ys = \| \wedge zs = \|\) by (rule interleaves-nil)
\textbf{thus} \(\| \simeq \{\text{map f} \ y, \text{map f} \ z, \| P'\}\) by simp
\textbf{next}
\textbf{fix} \(x \ xs \ ys \ zs\)
\textbf{assume} 

\begin{verbatim}
assume \(\neg Q\ x\)
hence \(\text{filter} \ Q \ (x \# xs) = \text{filter} \ Q \ xs\) by simp
moreover have \(\text{filter} \ Q \ (y \# ys') = \text{filter} \ Q \ ys'\)
  using \(\neg Q\ x\) and \(F\) by simp
ultimately show \(?thesis\) using \(E\) and \(G\)
  by (cases \(\text{filter} \ Q \ (z \# zs')\), simp-all)
qed
next
case \(False\)
hence \(E:\neg P \ x \ (\text{filter} \ Q \ xs)\) using \(A\) by simp
have \(F: x = z \wedge xs \simeq \{y \# ys', zs', P\}\) using \(D\) and \(False\) by simp
moreover have \(zs \simeq \{\text{filter} \ Q \ (y \# ys'), \text{filter} \ Q \ zs', P\}\)
  using \(B\)
ultimately have \(G: \text{filter} \ Q \ xs \simeq \{\text{filter} \ Q \ (y \# ys'), \text{filter} \ Q \ zs', P\}\)
  by simp
show \(?thesis\)
proof (cases \(Q\ x\))
assume \(Q\ x\)
hence \(\text{filter} \ Q \ (x \# xs) = x \# \text{filter} \ Q \ xs\) by simp
moreover have \(\text{filter} \ Q \ (z \# zs') = x \# \text{filter} \ Q \ zs'\)
  using \(\neg Q\ x\) and \(F\) by simp
ultimately show \(?thesis\) using \(E\) and \(G\)
  by (cases \(\text{filter} \ Q \ (z \# zs')\), simp-all)
qed
qed
qed

definition \(\text{interleaves-map}\) [rule-format]:
\textbf{assumes} A: inj f
\textbf{shows} \(xs \simeq \{ys, zs, P\} \longrightarrow\)
\(\text{map f} \ xy \simeq \{\text{map f} \ y, \text{map f} \ z, \lambda w \ ws. \ P (\text{inv f} \ w) (\text{map (inv f)} \ ws)\}\)
\textbf{proof} (induction \(xs\) arbitrary: \(ys \ zs\), rule-tac \([\!]\) \(\text{impl}\), \(\text{simp-all}\))
\textbf{fix} \(ys \ zs\)
\textbf{assume} \(\| \simeq \{ys, zs, P\}\)
\textbf{hence} \(ys = \| \wedge zs = \|\) by (rule interleaves-nil)
\textbf{thus} \(\| \simeq \{\text{map f} \ y, \text{map f} \ z, \| P'\}\) by simp
\textbf{next}
\textbf{fix} \(x \ xs \ ys \ zs\)
\textbf{assume} 

\begin{verbatim}
"
B: \( \forall ys \; zs. \; xs \simeq \{ys, zs, P\} \rightarrow \operatorname{map} f \; xs \simeq \{map \; f \; ys, map \; f \; zs, ?P\} \) and

C: \( x \neq xs \simeq \{ys, zs, P\} \)

show \( f \; x \neq map \; f \; xs \simeq \{map \; f \; ys, map \; f \; zs, ?P'\} \)

proof (cases \( ys \), case-tac \([\square] \) \( zs \), simp-all del: interleaves.simps(1))

assume \( ys = \square \) and \( zs = \square \)

thus False using C by simp

next

fix \( z \; zs' \)

assume \( ys = \square \) and \( zs = z \neq zs' \)

hence \( x = z \land xs \simeq \{zs', P\} \) using C by simp

moreover have \( xs \simeq \{ys', [], P\} \rightarrow \operatorname{map} f \; xs \simeq \{map \; f \; [], map \; f \; zs', ?P'\} \)

using B.

ultimately show \( f \; x = f \; z \land \operatorname{map} f \; xs \simeq \{[], map \; f \; zs', ?P'\} \) by simp

next

fix \( y \; ys' \)

assume \( ys = \square \) and \( zs = z \neq zs' \)

hence \( x = y \land xs \simeq \{ys', [], P\} \) using C by simp

moreover have \( xs \simeq \{ys, zs', P\} \rightarrow \operatorname{map} f \; xs \simeq \{map f \; ys', map f \; zs', ?P'\} \)

using B.

ultimately show \( f \; x = f \; y \land \operatorname{map} f \; xs \simeq \{map f \; ys', [], ?P'\} \) by simp

next

fix \( y \; ys' \; z \; zs' \)

assume \( ys = y \neq ys' \) and \( zs = z \neq zs' \)

hence \( D: \; x \neq xs \simeq \{y \neq ys', z \neq zs', P\} \) using C by simp

show \( f \; x \neq map f \; xs \simeq \{f \; y \neq map f \; ys', f \; z \neq map f \; zs', ?P'\} \)

proof (cases \( P \; x \; xs \))

  case True

  hence \( E: \; ?P' \; (f \; x) \; (map f \; xs) \) using A by simp

  have \( x = y \land xs \simeq \{ys', z \neq zs', P\} \) using D and True by simp

  moreover have \( xs \simeq \{ys', z \neq zs', P\} \rightarrow \operatorname{map} f \; xs \simeq \{map f \; ys', map f (z \neq zs'), ?P'\} \)

  using B.

  ultimately have \( f \; x = f \; y \land \operatorname{map} f \; xs \simeq \{map f \; ys', map f (z \neq zs'), ?P'\} \)

  by simp

  thus \( \text{thesis using E by simp} \)

next

  case False

  hence \( E: \; \neg \; ?P' \; (f \; x) \; (map f \; xs) \) using A by simp

  have \( x = z \land xs \simeq \{y \neq ys', zs', P\} \) using D and False by simp

  moreover have \( xs \simeq \{y \neq ys', zs', P\} \rightarrow \operatorname{map} f \; xs \simeq \{map f \; (y \neq ys'), map f \; zs', ?P'\} \)

  using B.

  ultimately have \( f \; x = f \; z \land \operatorname{map} f \; xs \simeq \{map f \; (y \neq ys'), map f \; zs', ?P'\} \)

  by simp

  thus \( \text{thesis using E by simp} \)

qed

qed

qed
lemma interleaves-prefix-fst-1 [rule-format]:
assumes A: $xs \simeq \{ys, zs, P\}$
shows $(\forall n < \text{length } ws. \ P ((w \# ws) ! n) (\text{drop } (\text{Suc } n) ws @ xs))$ $\rightarrow$
\hspace{1cm} $ws @ zs \simeq \{ws @ ys, zs, P\}$
proof (induction ws, simp-all add: A, rule impI)
\hspace{1cm} fix w ws
\hspace{1cm} assume $n < \text{Suc } (\text{length } ws)$. $P ((w \# ws) ! n) (\text{drop } n ws @ zs)$
\hspace{1cm} assume $(\forall n < \text{length } ws. \ P (ws ! n) (\text{drop } (\text{Suc } n) ws @ xs))$ $\rightarrow$
\hspace{2cm} $ws @ zs \simeq \{ws @ ys, zs, P\}$
moreover have $\forall n < \text{length } ws. \ P (ws ! n) (\text{drop } (\text{Suc } n) ws @ xs)$
proof (rule allI, rule impI)
\hspace{1cm} fix n
\hspace{1cm} assume $n < \text{length } ws$
\hspace{2cm} moreover have $\text{Suc } n < \text{Suc } (\text{length } ws) \rightarrow$
\hspace{3cm} $P ((w \# ws) ! (\text{Suc } n)) (\text{drop } (\text{Suc } n) ws @ xs)$
\hspace{2cm} using $B$ ..
\hspace{2cm} ultimately show $P (ws ! n) (\text{drop } (\text{Suc } n) ws @ xs)$ by simp
qed
ultimately have $ws @ xs \simeq \{ws @ ys, zs, P\}$ ..
moreover have $0 < \text{Suc } (\text{length } ws) \rightarrow$ $P ((w \# ws) ! 0) (\text{drop } 0 ws @ xs)$
using $B$ ..
\hspace{2cm} 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 @ zs $\simeq \{ws @ ys, zs, P\}$ $\rightarrow$
$(\forall n < \text{length } ws. \ P (ws ! n) (\text{drop } (\text{Suc } n) ws @ xs))$ $\rightarrow$
\hspace{1cm} $xs \simeq \{ys, zs, P\}$
proof (induction ws, simp-all, (rule impI)+)
\hspace{1cm} fix w ws
\hspace{1cm} assume $A: \forall n < \text{Suc } (\text{length } ws). \ P ((w \# ws) ! n) (\text{drop } n ws @ xs)$
\hspace{2cm} hence $0 < \text{Suc } (\text{length } ws) \rightarrow$ $P ((w \# ws) ! 0) (\text{drop } 0 ws @ xs)$ ..
\hspace{2cm} hence $P (w (ws @ xs)$ by simp
\hspace{2cm} moreover assume $w \# ws @ xs \simeq \{w \# ws @ ys, zs, P\}$
\hspace{2cm} ultimately have $ws @ xs \simeq \{ws @ ys, zs, P\}$ by (cases zs, simp-all)
\hspace{2cm} moreover assume $ws @ xs \simeq \{ws @ ys, zs, P\} \rightarrow$
\hspace{3cm} $(\forall n < \text{length } ws. \ P (ws ! n) (\text{drop } (\text{Suc } n) ws @ xs))$ $\rightarrow$
\hspace{4cm} $xs \simeq \{ys, zs, P\}$
\hspace{2cm} ultimately have $\forall n < \text{length } ws. \ P (ws ! n) (\text{drop } (\text{Suc } n) ws @ xs)$ $\rightarrow$
\hspace{3cm} $xs \simeq \{ys, zs, P\}$
\hspace{3cm} by simp
\hspace{2cm} moreover have $\forall n < \text{length } ws. \ P (ws ! n) (\text{drop } (\text{Suc } n) ws @ xs)$
proof (rule allI, rule impI)
\hspace{3cm} fix n
\hspace{4cm} assume $n < \text{length } ws$
\hspace{5cm} moreover have $\text{Suc } n < \text{Suc } (\text{length } ws) \rightarrow$
\hspace{6cm} $P ((w \# ws) ! (\text{Suc } n)) (\text{drop } (\text{Suc } n) ws @ xs)$
\hspace{5cm} using $A$ ..
ultimately show $P (\text{ws ! n}) (\text{drop (Suc n)} \text{ ws @ xs})$ by simp

qed

ultimately show $\text{xs \simeq \{ys, zs, P\} ..}$

qed

lemma interleave-prefix-fst [rule-format]:
\[$n < \text{length ws, } P (\text{ws ! n}) (\text{drop (Suc n)} \text{ ws @ xs}) \implies \text{xs \simeq \{ys, zs, P\} = ws @ zs \simeq \{ws @ ys, zs, P\}}$

proof (rule iffI, erule interleave-prefix-fst-1, simp)

qed (erule interleave-prefix-fst-2, simp)

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

proof (subst (1 2) interleave-swap)

qed (rule interleave-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 \textit{interleaves}. In fact, if $P \ y \ [x] = \text{True}$ with $x \neq y$, then $[y, x] \simeq \{[x], [y], P\}$ does not hold although $[y] \simeq \{[], [y]\}, \lambda w \text{ ws. } P \ w \ (\text{ws @ [x]})$ does, by virtue of lemma $\exists x \ s : \{[], [x], P\}$. Similarly, $[x, y] \simeq \{[], [y, x]\}, \lambda w \text{ ws. } P \ w \ (\text{ws @ [x]})$ 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 $[y] \simeq \{[], [y]\}, \lambda w \text{ ws. } P \ w \ (\text{ws @ [x]})$, so that the validity of rule $[y] \simeq \{[], [y]\}, \lambda w \text{ ws. } P \ w \ (\text{ws @ [x]}) \implies [y, x] \simeq \{[x], [y], P\}$ 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 $[x, y, x] \simeq \{[x], [y, x], P\} \implies [x, y] \simeq \{[], [y, x]\}, \lambda w \text{ ws. } P \ w \ (\text{ws @ [x]})$ to condition $P \ x \ []$ would preserve its validity.

This argument suggests that in order to obtain an \textit{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 \textit{Interleaves}, is defined along with a convenient symbolic notation for it.

fun Interleaves ::
\((\forall a \Rightarrow \forall \text{list} \Rightarrow \text{bool}) \Rightarrow \forall \text{list} \Rightarrow \forall \text{list} \Rightarrow \text{bool} \text{ where}

11
Interleaves $P (x \not\equiv xs) (y \not\equiv ys) (z \not\equiv zs) = (if P x xs$ 
then $x = y \land Interleaves P xs ys (z \not\equiv zs)$ 
else $x = z \land Interleaves P xs (y \not\equiv ys) zs |$

$Interleaves P (x \not\equiv xs) (y \not\equiv ys) [] =$
$(P x xs \land x = y \land Interleaves P xs ys []) |$
$Interleaves P (x \not\equiv xs) (y \not\equiv ys) [z \not\equiv zs] =$
$(\neg P x xs \land x = z \land Interleaves P xs [] zs) |$
$Interleaves - (\not\equiv -) [] [] = False |$
$Interleaves - - (\not\equiv -) [] = False |$
$Interleaves - - [] [] = True$

syntax (symbols) - Interleaves ::
't a list ⇒ 'a list ⇒ ('a ⇒ 'a list ⇒ bool) ⇒ bool
((- ≡ {,-,-}) [60, 60, 60] 51)

translations
$xs ≡ \{ys, zs, P\} ⇩ CONSTRAINT Interleaves P xs ys zs$

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 fulfill 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]:
$x_1 ≡ \{y, z, P\} \longrightarrow x_1 ≃ \{y, z, P\}$
proof (induction $P x_1 y_1 z_1$ rule: interleaves.induct, simp-all)
qed (rule conjI, (rule-tac [!] impl)+, simp-all)

lemma Interleaves-length:
$x_1 ≡ \{y, z, P\} \Longrightarrow length x_1 = length y_1 + length z_1$
by (drule Interleaves-interleaves, rule interleaves-length)

lemma Interleaves-nil:
[] ≡ \{y, z, P\} \Longrightarrow y = [] \land z = []
by (drule Interleaves-interleaves, rule interleaves-nil)

lemma Interleaves-swap:
$x_1 ≡ \{y, z, P\} = x_1 ≡ \{z, y, λw ws. \neg P w ws\}$
proof (induction $P x_1 y_1 z_1$ rule: Interleaves.induct, simp-all)
fix $y' :: 'a and y_1 zs' P'$
show $\neg [!] ≡ \{zs', y' \not\equiv ys', λw ws. \neg P' w ws\}$ by (cases zs', simp-all)
qed
lemma Interleaves-equal-fst:
\[ xs \equiv \{ys, zs, P\} \implies xs \equiv \{ys', zs, P\} \implies ys = ys' \]
by ((drule Interleaves-interleaves)+, rule Interleaves-equal-fst)

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

lemma Interleaves-equal-nil:
\[ xs \equiv \{ys, zs, P\} \implies xs = ys \]
by (drule Interleaves-interleaves, rule Interleaves-equal-nil)

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

lemma Interleaves-filter [rule-format]:
assumes \( A: \forall x \, xs. \, P \, x \, (\text{filter } Q \, xs) = P \, x \, xs \)
shows \( zs \equiv \{ys, zs, P\} \implies \text{filter } Q \, xs \equiv \{\text{filter } Q \, ys, \text{filter } Q \, zs, P\} \)
proof (induction xs arbitrary: ys zs, rule-tac \[\text{ infographic} \, \text{implI}, \, \text{simpl} \]
fix ys zs
assume \( [] \equiv \{ys, zs, P\} \)
hence \( ys = [] \wedge zs = [] \) by (rule Interleaves-nil)
thus \( [] \equiv \{\text{filter } Q \, ys, \text{filter } Q \, zs, P\} \) by simp
next
fix x xs ys zs
assume \( B: \forall y \, y'. \, zs', \, xs \equiv \{ys', zs', P\} \implies \text{filter } Q \, xs \equiv \{\text{filter } Q \, ys', \text{filter } Q \, zs', P\} \) and
\( C: x \neq zs \equiv \{ys, zs, P\} \)
show \( \text{filter } Q \, (x \neq zs) \equiv \{\text{filter } Q \, ys, \text{filter } Q \, zs, P\} \)
proof (cases ys, case-tac \[\text{ infographic} \, \text{filter} \, \text{simps}, \, \text{rule ccontr} \]
assume \( ys = [] \) and \( zs = [] \)
thus \( False \) using \( C \) by simp
next
fix z zs'
assume \( ys = [] \) and \( zs = z \neq zs' \)
hence \( D: \neg P \, x \, zs \wedge x = z \wedge xs \equiv [] \wedge zs' \equiv \{[], zs', P\} \) using \( C \) by simp+
moreover have \( xs \equiv [] \wedge zs' \equiv \{[], zs', P\} \implies \text{filter } Q \, xs \equiv \{\text{filter } Q \, [], \text{filter } Q \, zs', P\} \)
using \( B \).
ultimately have \( \text{filter } Q \, xs \equiv [] \wedge zs' \equiv \{[], \text{filter } Q \, zs', P\} \) by simp
moreover have \( \neg P \, x \, (\text{filter } Q \, xs) \) using \( A \) and \( D \) by simp+
ultimately show \( \text{filter } Q \, (x \neq zs) \equiv [] \wedge zs \equiv [] \)
using \( D \) by simp
next
fix y ys'
assume \( ys = y \neq ys' \) and \( zs = [] \)
hence \( D: P \, x \, xs \wedge x = y \wedge xs \equiv \{ys', [], P\} \) using \( C \) by simp+
moreover have \(xs \cong \{ys', [], P\} \rightarrow\)
\[filter Q xs \cong \{filter Q ys', filter Q [], P\}\]
using \(B\).
ultimately have \(filter Q xs \cong \{filter Q ys', [], P\}\) by simp
moreover have \(P x (filter Q xs)\) using \(A\) and \(D\) by simp
ultimately show \(filter Q (x \# xs) \cong \{filter Q (y \# ys'), [], P\}\)
using \(D\) by simp
next
fix \(y\) \(ys'\) \(z\) \(zs'\)
assume \(ys = y \# ys'\) and \(zs = z \# zs'\)
hence \(D: x \# xs \cong \{y \# ys', z \# zs', P\}\) using \(C\) by simp
show \(filter Q (x \# xs) \cong \{filter Q (y \# ys'), filter Q (z \# zs'), P\}\)
proof (cases \(P x xs\))
case \(True\)
hence \(E: P x (filter Q xs)\) using \(A\) by simp
have \(F: x = y \land xs \cong \{ys', z \# zs', P\}\) using \(D\) and \(True\) by simp
moreover have \(xs \cong \{ys', z \# zs', P\} \rightarrow\)
\[filter Q xs \cong \{filter Q ys', filter Q (z \# zs'), P\}\]
using \(B\).
ultimately have \(G: filter Q xs \cong \{filter Q ys', filter Q (z \# zs'), P\}\)
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 \(\neg Q x\)
hence \(filter Q (x \# xs) = filter Q xs\) by simp
moreover have \(filter Q (y \# ys') = filter Q ys'\)
using \((\neg Q x)\) and \(F\) by simp
ultimately show \(?thesis\) using \(E\) and \(G\)
by (cases filter Q (z \# zs'), simp-all)
qed
next
case \(False\)
hence \(E: \neg P x (filter Q xs)\) using \(A\) by simp
have \(F: x = y \land xs \cong \{ys', zs', P\}\) using \(D\) and \(False\) by simp
moreover have \(xs \cong \{y \# ys', zs', P\} \rightarrow\)
\[filter Q xs \cong \{filter Q (y \# ys'), filter Q zs', P\}\]
using \(B\).
ultimately have \(G: filter Q xs \cong \{filter Q (y \# ys'), filter Q zs', P\}\)
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

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 − − − − − → ⊆ {y, x, ?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(1–3))
assume ys = [] and zs = []
thus False using C by simp
next
fix z zs'
assume ys = [] and zs = z ≠ zs'
hence D: ¬ P x xs ∧ 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 have map f xs ⊆ {[], map f zs', ?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 z ≠ map f zs', ?P'}
  using D by simp
next
fix y ys'
assume ys = y ≠ ys' and zs = []
hence $D$: $P \ x \ xs \land x = y \land zs \equiv \{ys', [], P\}$ using $C$ by simp+

moreover have $xs \cong \{ys', [], P\} \rightarrow map f xs \cong \{map f ys', map f [], ?P'\}$

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 $f x \# map f xs \cong \{f y \# map f ys', [], ?P'\}$

using $D$ by simp

next

fix $y$ $ys'$ $z$ $zs'$

assume $ys = y \# ys'$ and $zs = z \# zs'$

hence $D$: $x \# xs \cong \{y \# ys', z \# zs', P\}$ using $C$ by simp

show $f x \# map f xs \cong \{f y \# map f ys', f z \# map f zs', ?P'\}$

proof (cases $P \ x \ xs$)

case True

hence $E$: $?P' (f x) (map f xs)$ using $A$ by simp

have $x = y \land zs \cong \{ys', z \# zs', P\}$ using $D$ and $True$ by simp

moreover have $xs \cong \{ys', z \# zs', P\} \rightarrow$

$map f xs \cong \{map f ys', map f (z \# zs'), ?P'\}$

using $B$.

ultimately have $f x = f y \land map f xs \cong \{map f ys', map f (z \# zs'), ?P'\}$

by simp

thus $\text{thesis using } E$ by simp

dqd

dqd

dqd

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

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

shows $(\forall n < \text{length } ws. \ P (ws ! n) (\text{drop } (\text{Suc } n) ws @ xs)) \rightarrow$ $\ P (ws \@ ws @ ws @ xs)$

proof (induction $ws$, simp-all add: $A$, rule impI)

fix $w$ $ws$

assume $B$: $\forall n < \text{Suc } (\text{length } ws). \ P ((w \# ws) ! n) (\text{drop } n ws @ xs)$

assume $(\forall n < \text{length } ws. \ P (ws ! n) (\text{drop } (\text{Suc } n) ws @ xs)) \rightarrow$ $\ P (ws \@ ws @ ws @ xs)$

moreover have $\forall n < \text{length } ws. \ P (ws ! n) (\text{drop } (\text{Suc } n) ws @ xs)$

proof (rule allI, rule impI)

fix $n$

assume $n < \text{length } ws$

16
moreover have \( Suc \ n < Suc \ (length \ ws) \rightarrow \\
\quad P \ ((w \ #\ ws) ! (Suc \ n)) \ (drop \ (Suc \ n) \ ws \ @ xs) \)
using \( B \).
ultimately show \( P \ (ws \ ! \ n) \ (drop \ (Suc \ n) \ ws \ @ xs) \) by simp
qed
moreover have \( ws \ @ xs \simeq \{ws \ @ ys, zs, P\} \).
moreover have \( 0 < Suc \ (length \ ws) \rightarrow P \ ((w \ #\ ws) ! 0) \ (drop 0 ws \ @ xs) \)
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]:
\( ws @ xs \simeq \{ws @ ys, zs, P\} \rightarrow \\
(\forall n < length ws. P (ws ! n) (drop (Suc n) ws @ xs)) \rightarrow \\
xz \simeq \{ys, zs, P\} \)

proof (induction ws, simp-all, (rule impl)+)
fix \( w ws \)
assume \( A:\forall n < Suc \ (length \ ws). P ((w \# ws) ! n) \ (drop n ws @ xs) \)
hence \( 0 < Suc \ (length \ ws) \rightarrow P ((w \# ws) ! 0) \ (drop 0 ws @ xs) \).
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 \( ws @ xs \simeq \{ws @ ys, zs, P\} \rightarrow \\
(\forall n < length ws. P (ws ! n) (drop (Suc n) ws @ xs)) \rightarrow \\
xz \simeq \{ys, zs, P\} \)
ultimately have \( (\forall n < length ws. P (ws ! n) (drop (Suc n) ws @ xs)) \rightarrow \\
xz \simeq \{ys, zs, P\} \)
by simp
moreover have \( \forall n < length ws. P (ws ! n) (drop (Suc n) ws @ xs) \)
proof (rule allI, rule impl)
fix \( n \)
assume \( n < length ws \)
moreover have \( Suc \ n < Suc \ (length \ ws) \rightarrow \\
P ((w \# ws) ! (Suc \ n)) \ (drop (Suc n) ws @ xs) \)
using \( A \).
ultimately show \( P \ (ws ! n) \ (drop \ (Suc n) \ ws @ xs) \) by simp
qed
ultimately show \( xz \simeq \{ys, zs, P\} \).
qed

lemma Interleaves-prefix-fst [rule-format]:
\( \forall n < length ws. P \ (ws ! n) \ (drop \ (Suc n) \ ws @ xs) \rightarrow \\
xz \simeq \{ys, zs, P\} \equiv ws @ xs \simeq \{ws @ ys, zs, P\} \)
proof (rule iffI, erule Interleaves-prefix-fst-1, simp)
qed (erule Interleaves-prefix-fst-2, simp)

lemma Interleaves-prefix-snd [rule-format]:
\( \forall n < length ws. \neg P \ (ws ! n) \ (drop \ (Suc n) \ ws @ xs) \rightarrow \\
\)

17
\[xs \cong \{ys, zs, P\} = ws @ xs \cong \{ys, ws @ zs, P\}\]

**proof** (subst (1 2) Interleaves-swap)

**qed** (rule Interleaves-prefix-fst, simp)

**lemma** Interleaves-all-nil-1 [rule-format]:
\[xs \cong \{xs, [], P\} \rightarrow (\forall n < \text{length } xs. P (xs ! n) (\text{drop} (Suc \ n) \ xs))\]

**proof** (induction \(xs\), simp-all, rule impI, erule conjE, rule allI, rule impI)

fix \(xs \ n\)
assume \(xs \cong \{xs, [], P\} \rightarrow (\forall n < \text{length } xs. P (xs ! n) (\text{drop} (Suc \ n) \ xs))\) and \(xs \cong \{xs, [], P\}\)

**hence** \(A: \forall n < \text{length } xs. P (xs ! n) (\text{drop} (Suc \ n) \ xs)\) ..

assume \(B: P \ x \ xs\ and\)
\(C: n < \text{Suc} (\text{length } xs)\)

**show** \(P ((x \not= xs) ! n) (\text{drop} \ n \ xs)\)

**proof** (cases \(n\), simp-all add: \(B\))

case (Suc \(m\))

**have** \(m < \text{length } xs \rightarrow P (xs ! m) (\text{drop} (Suc \ m) \ xs)\) **using** \(A\) ..

**moreover** **have** \(m < \text{length } xs\) **using** \(C\) **and** \(\text{Suc}\) **by** simp

ultimately **show** \(P (xs ! m) (\text{drop} (Suc \ m) \ xs)\) ..

**qed**

**qed**

**lemma** Interleaves-all-nil-2 [rule-format]:
\[\forall n < \text{length } xs. P (xs ! n) (\text{drop} (Suc \ n) \ xs) \implies xs \cong \{xs, [], P\}\]

**by** (insert Interleaves-prefix-fst [of \(xs\) \(P \ [] \ [\]\)], simp)

**lemma** Interleaves-all-nil:
\[xs \cong \{xs, [], P\} = (\forall n < \text{length } xs. P (xs ! n) (\text{drop} (Suc \ n) \ xs))\]

**proof** (rule iffI, rule allI, rule impI, rule Interleaves-all-nil-1, assumption+)

**qed** (rule Interleaves-all-nil-2, simp)

**lemma** Interleaves-nil-all:
\[xs \cong \{[], zs, P\} = (\forall n < \text{length } xs. \neg P (xs ! n) (\text{drop} (Suc \ n) \ xs))\]

**by** (subst Interleaves-swap, simp add: Interleaves-all-nil)

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

assumes \(A: P \ x \ [\]\)

**shows** \(\neg xs @ [x] \cong \{[], zs, P\}\)

**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** \(w \ zs \ ws\)

**assume** \(B: \forall zs. \neg xs @ [x] \cong \{[], zs, P\}\)

**assume** \(w \not= xs @ [x] \cong \{[], zs, P\}\)

**thus** \(False\) **proof** (cases \(zs\), simp-all, (erule-tac conjE)+)

18
\begin{proof}
\begin{proof}
fix \( z' \)
assume \( xs @ [x] \cong \{ [], z', P \} \)
moreover have \( \neg \; xs @ [x] \cong \{ [], z', P \} \) using \( B \).
ultimately show \textit{False by contradiction} \qed
\end{proof}
\end{proof}

\textbf{lemma} \textit{Interleaves-suffix-one-fst-2 [rule-format]:}
assumes \( A : \{ \; \} \)
shows \( xs @ [x] \cong \{ ys @ [x], zs, P \} \longrightarrow xs \cong \{ ys, zs, \lambda w. P \; w \; (ws @ [x]) \} \)
(is \(-\) \(-\) \(-\) \(-\) \(-\) \(-\) \(-\))
\begin{proof}
(induction \( xs \) arbitrary; \( ys \; zs \)), \textit{rule I impI, simp-all})
fix \( ys \; zs \)
assume \( [x] \cong \{ ys @ [x], zs, P \} \)
hence \( B : \text{length} \; [x] = \text{length} \; (ys @ [x]) + \text{length} \; zs \)
by \textit{(rule Interleaves-length)}
have \( ys = [] \) by \textit{(cases \( ys \; zs \), simp, insert \( B \), simp)}
moreover from \textit{this} have \( zs = [] \) by \textit{(cases \( zs \; xs \), simp, insert \( B \), simp)}
ultimately show \( [] \cong \{ ys, zs, \; ?P' \} \) by \textit{simp}
\end{proof}
\end{proof}
next
\begin{proof}
fix \( w \; xs \; ys \; zs \)
assume \( B : \bigwedge \; \text{ys} \; \text{zs} \; \text{xs} @ [x] \cong \{ \text{ys} @ [x], \text{zs}, \text{P} \} \longrightarrow xs \cong \{ \text{ys}, \text{zs}, \; ?P' \} \)
assume \( w \; \# \; xs @ [x] \cong \{ \text{ys} @ [x], \text{zs}, \text{P} \} \)
thus \( w \; \# \; xs \cong \{ \text{ys}, \text{zs}, \; ?P' \} \)
\begin{proof}
(cases \( zs \; case-tac [] \) \( ys \; simp-all del: \text{Interleaves.simps(1,3)}, \)
(erule-tac [1-2] \textit{conjE}+))
assume \( zs @ [x] \cong \{ [], [], P \} \)
thus \textit{False by (cases \( zs \; simp-all) \) \textit{next}}
\end{proof}
\end{proof}
\end{proof}
next
\begin{proof}
fix \( y' \)
have \( xs @ [x] \cong \{ y's \@ [x], [], P \} \longrightarrow xs \cong \{ y's', [], ?P' \} \) using \( B \).
moreover assume \( xs @ [x] \cong \{ y's \@ [x], [], P \} \)
ultimately show \( xs \cong \{ y's', [], ?P' \} \).
\end{proof}
\end{proof}
next
\begin{proof}
fix \( z' \; zs' \)
assume \( w \; \# \; xs @ [x] \cong \{ [], z' \; \# \; zs', P \} \)
thus \( w \; \# \; xs \cong \{ [], z' \; \# \; zs', ?P' \} \)
\begin{proof}
(cases \( P \; w \; (xs @ [x]), \; simp-all, erule-tac [] \) \textit{conjE})
assume \( xs @ [x] \cong \{ [], z' \; \# \; zs', P \} \)
moreover have \( \neg \; xs @ [x] \cong \{ [], z' \; \# \; zs', P \} \)
using \( A \) by \textit{(rule Interleaves-suffix-one-aux)}
ultimately show \textit{False by contradiction} \qed
\end{proof}
\end{proof}
\end{proof}
\end{proof}
next
\begin{proof}
have \( xs @ [x] \cong \{ [], zs', P \} \longrightarrow xs \cong \{ [], zs', ?P' \} \) using \( B \) by \textit{simp}
moreover assume \( xs @ [x] \cong \{ [], zs', P \} \)
ultimately show \( xs \cong \{ [], zs', ?P' \} \).
\end{proof}
\end{proof}
next
\begin{proof}
fix \( y' \; y's \; z' \; zs' \)
\end{proof}
\end{proof}
\end{proof}
\end{proof}
\end{proof}
\end{proof}

19
assume \( w \# zs \ocircle [x] \cong \{ y' \# ys' \ocircle [x], z' \# zs', P \} \)

thus \( w \# zs \cong \{ y' \# ys', z' \# zs', ?P' \} \)

proof (cases \( P \) \( w \ocircle [x] \), simp-all, erule-tac \(!] \) conjE)

\begin{align*}
& \text{have} \ x \ocircle [x] \cong \{ y' \# ys' \ocircle [x], z' \# zs', P \} \longrightarrow \ x \cong \{ y' \# ys', z' \# zs', ?P' \} \\
& \quad \text{using} \ B . \\
& \quad \text{moreover assume} \ x \ocircle [x] \cong \{ y' \# ys' \ocircle [x], z' \# zs', P \} \\
& \quad \text{ultimately show} \ x \cong \{ y' \# ys', z' \# zs', ?P' \} ..
\end{align*}

next

\begin{align*}
& \text{have} \ x \ocircle [x] \cong \{ y' \# ys' \ocircle [x], z' \# zs', P \} \longrightarrow \ x \cong \{ y' \# ys', z' \# zs', ?P' \} \\
& \quad \text{using} \ B \text{ by simp} \\
& \quad \text{moreover assume} \ x \ocircle [x] \cong \{ y' \# ys' \ocircle [x], z' \# zs', P \} \\
& \quad \text{ultimately show} \ x \cong \{ y' \# ys', z' \# zs', ?P' \} .. \\
& \text{qed} \\
& \text{qed}
\end{align*}

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

assumes \( A : \forall n < \text{length} \ us . \ P (\text{drop} (\text{Suc} n) \ us) \)

shows \( zs \cong \{ ys, zs, \lambda v . \ P v (ys \ocircle \ us) \} \longrightarrow \ x \ocircle \ us \cong \{ ys \ocircle \ us, zs, P \} \)

proof (induction \( zs \) arbitrary: \( ys, zs, \text{rule-tac} [!] \) impI, simp-all)

fix \( ys, zs \)

assume \( [] \cong \{ ys, zs, ?P' \} \)

hence \( ys = [] \land zs = [] \) by (rule Interleaves-nil)

thus \( us \cong \{ ys \ocircle \ us, zs, P \} \) using \( A \) by (simp add: Interleaves-all-nil)

next

fix \( x, xs, ys, zs \)

assume \( \{ ys, zs, ?P' \} \longrightarrow \ x \ocircle \ us \cong \{ ys \ocircle \ us, zs, P \} \)

assume \( x \# xs \cong \{ ys, zs, ?P' \} \)

thus \( x \ocircle \ us \cong \{ ys \ocircle \ us, zs, P \} \)


fix \( P' \ x' \ y' \ y's' \ y's' \)

assume \( B : x' \# xs' \cong \{ y' \# ys', z' \# zs', P' \} \) and

\begin{align*}
& C : ?P' = P' \text{ and} \\
& D : xs = xs' \\
& \text{show} \ x' \# xs' \ocircle \ us \cong \{ y' \# ys' \ocircle \ us, z' \# zs', P \} \\
& \text{proof (cases } P', x' \ocircle \ us') \end{align*}

\begin{align*}
& \text{have} \ xs \cong \{ y', z' \# zs', ?P' \} \longrightarrow \ x \ocircle \ us \cong \{ ys' \ocircle \ us, z' \# zs', P \} \\
& \quad \text{using} \ A . \\
& \quad \text{moreover case } \text{True} \\
& \quad \text{hence} \ xs \cong \{ ys', z' \# zs', ?P' \} \text{ using } B \text{ and } C \text{ and } D \text{ by simp} \\
& \quad \text{ultimately have} \ x \ocircle \ us \cong \{ ys' \ocircle \ us, z' \# zs', P \} . . \\
& \quad \text{moreover have} \ P' (x' \ocircle \ us) \text{ using } C \text{ [symmetric] and True by simp} \\
& \quad \text{moreover have} \ x' = y' \text{ using } B \text{ and True by simp} \\
& \quad \text{ultimately show } \text{?thesis using } D \text{ by simp} \\
& \text{next}
\end{align*}
have \(xs \cong \{y' \neq ys', zs', ?P'\} \rightarrow xs @ ws \cong \{(y' \neq ys') @ ws, zs', P\}\) using \(A\).

moreover case \(False\)

hence \(xs \cong \{y' \neq ys', zs', ?P'\}\) using \(B\) and \(C\) and \(D\) by \textit{simp}

ultimately have \(xs @ ws \cong \{(y' \neq ys') @ ws, zs', P\}\).

moreover have \(\neg P \ x' (xs' @ ws)\) using \(C\) [symmetric] and \(False\) by \textit{simp}

moreover have \(x' = z'\) using \(B\) and \(False\) by \textit{simp}

ultimately show \(?thesis\) using \(D\) by \textit{simp}

qed

\textbf{next}

\textbf{fix} \(P' \ x' \ \overline{xs'} \ y' \ \overline{ys'}\)

\textbf{have} \(xs \cong \{ys', [], ?P'\} \rightarrow xs @ ws \cong \{ys' @ ws, [], P\}\) using \(A\).

moreover assume

\(xs' \cong \{ys', [], P'\}\) and

\(B; ?P' = P'\) and

\(C; xs = xs'\)

\textbf{hence} \(xs \cong \{ys', [], ?P'\}\) by \textit{simp}

ultimately have \(xs' @ ws \cong \{ys' @ ws, [], P\}\) using \(C\) by \textit{simp}

moreover assume

\(P' \ x' \overline{xs'}\) and

\(x' = y'\)

\textbf{hence} \(P \ y'(xs' @ ws)\) using \(B\) [symmetric] by \textit{simp}

ultimately show \(P \ y'(xs' @ ws) \land xs' @ ws \cong \{ys' @ ws, [], P\}\) by \textit{simp}

\textbf{next}

\textbf{fix} \(P' \ x' \overline{xs'} \ z' \overline{zs'}\)

\textbf{have} \(xs \cong \{[], zs', ?P'\} \rightarrow xs @ ws \cong \{[] @ ws, zs', P\}\) using \(A\).

moreover assume

\(xs' \cong \{[], zs', P'\}\) and

\(B; ?P' = P'\) and

\(C; xs = xs'\)

\textbf{hence} \(xs \cong \{[], zs', ?P'\}\) by \textit{simp}

ultimately have \(xs' @ ws \cong \{ws, zs', P\}\) using \(C\) by \textit{simp}

moreover assume

\(\neg P' \ x' \overline{xs'}\) and

\(x' = z'\)

\textbf{hence} \(\neg P \ z'(xs' @ ws)\) using \(B\) [symmetric] by \textit{simp}

ultimately show \(z' \neq xs' @ ws \cong \{ws, z' \neq zs', P\}\) by \((cases\ ws, \text{simp-all})\)

\textbf{qed}

\textbf{lemma} \textit{Interleaves-suffix-one-fst-1} [rule-format]:

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

by \((\text{rule}\ \text{Interleaves-suffix-fst-1}, \text{simp})\)

\textbf{lemma} \textit{Interleaves-suffix-one-fst}:

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

\textbf{proof} \((\text{rule}\ \text{iffI}, \text{rule}\ \text{Interleaves-suffix-one-fst-1}, \text{assumption+})\)

21
**Lemma** Interleaves-suffix-one-snd:
\[ \neg P \ x \ \text{implies}\ \forall x \in \text{xs, zs}, \lambda w \ \text{ws}. \ P \ w \ (\text{ws} \ @ \ [x]) = \text{xs} \ @ \ [x] \cong \{\text{gs}, \text{zs} @ [x], P\} \]
by (subst (1 2) Interleaves-swap, rule Interleaves-suffix-one-fst).

**Lemma** Interleaves-suffix-aux [rule-format]:
\[ (\forall n < \text{length ws}. \ P \ (\text{ws} ! n) \ (\text{drop} \ (\text{Suc n} \ \text{ws}))) \implies \]
\[ x \ # \ \text{xs} \ @ \ \text{ws} \cong \{\text{ws}, \text{zs}, P\} \implies \neg P \ x \ (\text{xs} \ @ \ \text{us}) \]


fix P

assume x # xs \cong \{[], zs, P\}
thus \neg P \ x \ xs \ by (cases zs, simp-all)

next

fix w ws P

assume

A: \forall P'. (\forall n < \text{length ws}. \ P' \ (\text{ws} ! n) \ (\text{drop} \ (\text{Suc n} \ \text{ws}))) \implies \\
x \ # \ \text{xs} @ \ \text{ws} \cong \{\text{ws}, \text{zs}, P\} \implies \neg P' \ x \ (\text{xs} @ \ \text{us}) \text{ and} \\
B: \forall n < \text{Suc} \ (\text{length ws}). \ P \ ((\text{us} @ [w]) ! n) \\
(\text{drop} \ (\text{Suc n} \ \text{ws} @ \ (\text{Suc n} - \text{length ws}) [w]))

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

hence C: (x # xs @ ws) @ [w] \cong \{ws @ [w], zs, P\} by simp

let \( ?P' = \lambda v \ (\text{us} @ [w]) \)

have (\forall n < \text{length ws}. \ ?P' \ (\text{ws} ! n) \ (\text{drop} \ (\text{Suc n} \ \text{ws}))) \implies \\
x \ # \ \text{xs} @ \ \text{ws} \cong \{\text{ws}, \text{zs}, ?P'\} \implies \neg ?P' \ x \ (\text{xs} @ \ \text{us})

using A .

moreover have \forall n < \text{length ws}. \ ?P' \ (\text{ws} ! n) \ (\text{drop} \ (\text{Suc n} \ \text{ws}))

**Proof** (rule allI, rule impl)

fix n

assume D: n < \text{length ws}

moreover have n < \text{Suc} \ (\text{length ws}) \implies \ P \ ((\text{us} @ [w]) ! n) \\
(\text{drop} \ (\text{Suc n} \ \text{ws} @ \ (\text{Suc n} - \text{length ws}) [w]))

using B ..

ultimately have P \ ((\text{us} @ [w]) ! n) \ (\text{drop} \ (\text{Suc n} \ \text{ws} @ [w])) by simp

moreover have n < \text{length} \ (\text{butlast} \ (\text{us} @ [w])) using D by simp

hence \text{butlast} \ (\text{us} @ [w]) ! n = \ (\text{us} @ [w]) ! n by (rule nth-butlast)

ultimately show P \ (\text{ws} ! n) \ (\text{drop} \ (\text{Suc n} \ \text{ws} @ [w])) by simp

**Qed**

ultimately have x # xs @ ws \cong \{ws, zs, ?P'\} \implies \neg ?P' \ x \ (\text{xs} @ \ \text{us}) ..

moreover have length ws < \text{Suc} \ (\text{length ws}) \implies \ P \ ((\text{us} @ [w]) ! \text{length ws}) \\
(\text{drop} \ (\text{Suc} \ (\text{length ws}) \ \text{ws} @ \ (\text{Suc} \ (\text{length ws}) - \text{length ws}) [w]))

using B ..

hence P \ w \ [] by simp

hence x # xs @ ws \cong \{ws, zs, ?P'\}

using C by (rule Interleaves-suffix-one-fst-2)

ultimately have \neg ?P' \ x \ (\text{xs} @ \ \text{us}) ..
thus \( \neg P \ x \ (xs @ ws @ [w]) \) by simp

qed

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

assumes \( A: \forall n < \text{length } ws. \ P (ws ! n) \ (\text{drop } (\text{Suc } n) \ ws) \)

shows \( xs @ ws \cong \{ ys @ ws, zs, P \} \longrightarrow xs \cong \{ ys, zs, \lambda v. \ v \ (ws @ ws) \} \)

\( (\text{is } \longrightarrow \cong \cong \ (\cdot, \cdot, \ ?P') \) \)

proof (induction \( zs \) arbitrary: \( ys \ zs \), rule-tac [!] \( \text{impI} \), simp-all)

fix \( ys \ zs \)
assume \( ws \cong \{ ys @ ws, zs, P \} \)

hence \( ys = [] \) by (cases \( ys \), simp, insert \( B \), simp)

moreover from this have \( zs = [] \) by (cases \( zs \), simp, insert \( B \), simp)

ultimately show \( [] \cong \{ ys, zs, \ ?P' \} \) by simp

next

fix \( x \ xs \ ys \ zs \)
assume \( B: \{ \forall zs. \ xs @ ws \cong \{ ys @ ws, zs, P \} \longrightarrow xs \cong \{ ys, zs, \ ?P' \} \)

assume \( x \# xs @ ws \cong \{ ys @ ws, zs, P \} \)

thus \( x \# xs \cong \{ ys, zs, \ ?P' \} \)

proof (cases \( zs \), case-tac [!] \( ys \), simp-all del: Interleaves.simps(1,3),
(cerule-tac [2] \text{conjE}+)\)

assume \( C: x \# xs @ ws \cong \{ ws, [], P \} \)

have \( \text{length } (x \# xs @ ws) = \text{length } ws + \text{length } [] \)

by (insert Interleaves-length [OF \( C \)], simp)

thus False by simp

next

fix \( ys' \)

have \( xs @ ws \cong \{ ys' @ ws, [], P \} \longrightarrow xs \cong \{ ys', [], ?P' \} \) using \( B \).

moreover assume \( xs @ ws \cong \{ ys' @ ws, [], P \} \)

ultimately show \( xs \cong \{ ys', [], ?P' \} \).

next

fix \( z' zs' \)

assume \( x \# xs @ ws \cong \{ ws, z' \# zs', P \} \)

thus \( x \# xs \cong \{ [], z' \# zs', \ ?P' \} \)

proof (cases \( P \ x \ (zs @ ws) \), simp-all)

case True

moreover assume \( x \# xs @ ws \cong \{ ws, z' \# zs', P \} \)

with \( A \) [rule-format] have \( \neg P \ x \ (xs @ ws) \)

by (rule Interleaves-suffix-aux)

ultimately show False by contradiction

next
case False

moreover assume \( x \# xs @ ws \cong \{ ws, z' \# zs', P \} \)

ultimately have \( x = z' \land xs @ ws \cong \{ ws, zs', P \} \) by (cases \( ws \), simp-all)

moreover have \( xs @ ws \cong \{ [], zs', P \} \longrightarrow xs \cong \{ [], zs', ?P' \} \)

using \( B \).

ultimately show \( x = z' \land xs \cong \{ [], zs', ?P' \} \) by simp

qed

23
next
  fix y' ys' z' zs'
  assume x # xs @ ws ≡ \{ y' # y's' @ ws, z' # zs', P \}
  thus x # xs ≡ \{ y' # y's', z' # zs', \#P' \}
  proof (cases P x (xs @ ws), simp-all, erule tac [!] conjE)
    have xs @ ws ≡ \{ y's' @ ws, z' # zs', P \} —→ xs ≡ \{ y's', z' # zs', \#P' \}
      using B.
    moreover assume xs @ ws ≡ \{ y's' @ ws, z' # zs', P \}
    ultimately show xs ≡ \{ y's', z' # zs', \#P' \} ..
  next
    have xs @ ws ≡ \{ y' # y's' @ ws, zs', P \} —→ xs ≡ \{ y' # y's', zs', \#P' \}
      using B by simp
    moreover assume xs @ ws ≡ \{ y' # y's' @ ws, zs', P \}
    ultimately show xs ≡ \{ y' # y's', zs', \#P' \} ..
  qed
  qed

lemma Interleaves-suffix-fst \[ rule-format \]:
  \forall n < \text{length ws}. P (\text{ws} ! n) (\text{drop} (\text{Suc} n) \text{ws}) —→
  xs ≡ \{ y's, zs, \lambda v \text{ vs}. P v (\text{vs} @ \text{ws}) \} = xs @ ws ≡ \{ y's @ 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 < \text{length ws}. \neg P (\text{ws} ! n) (\text{drop} (\text{Suc} n) \text{ws}) —→
  xs ≡ \{ y's, zs, \lambda v \text{ vs}. P v (\text{vs} @ \text{ws}) \} = xs @ ws ≡ \{ y's @ ws, zs, P \}
  by (subst (1 2) Interleaves-swap, rule Interleaves-suffix-fst, simp)
end

References


