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 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 interleave
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 \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 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 \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 \textit{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{verbatim}
fun interleaves ::
  ('a ⇒ 'a list ⇒ bool) ⇒ 'a list ⇒ 'a list ⇒ 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
\end{verbatim}
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 \textit{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 \textit{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 $y \neq z$, $P y [z] = True$, $P z [y] = False$, in which case both of the \textit{interleaves} statements $[y, z] ≃ \{ys, zs, P\}$ and $[z, y] ≃ \{ys, zs, P\}$ hold.

\textbf{lemma} \textit{interleaves-length} [rule-format]:
$$xs ≃ \{ys, zs, P\} \rightarrow length xs = length ys + length zs$$
\textbf{proof} (induction $P$ $xs$ $ys$ $zs$ rule: interleaves.induct, simp-all)
\textbf{qed} (rule conjI, (rule-tac [!]) implI+, simp-all)

\textbf{lemma} \textit{interleaves-nil}:
$$[] ≃ \{ys, zs, P\} \rightarrow ys = [] \land zs = []$$
\textbf{by} (rule interleaves.cases [of $(P, [], ys, zs)$], simp-all)

\textbf{lemma} \textit{interleaves-swap}:
$$xs ≃ \{ys, zs, P\} = xs ≃ \{zs, ys, \lambda w ws. \neg P w ws\}$$
\textbf{proof} (induction $P$ $xs$ $ys$ $zs$ rule: interleaves.induct, simp-all)
\textbf{fix} $y' :: 'a$ and $ys' \ zs' P'\n\textbf{show} \neg [] ≃ \{zs', y' \# ys', \lambda w ws. \neg P' w ws\}$ \textbf{by} (cases $zs'$, simp-all)
\textbf{qed}

\textbf{lemma} \textit{interleaves-equal-fst} [rule-format]:
$$xs ≃ \{ys, zs, P\} \rightarrow xs ≃ \{ys', zs, P\} \rightarrow ys = ys'$$
\textbf{proof} (induction $xs$ arbitrary: $ys$ $ys'$ $zs$, (rule-tac [!]) implI+)
\textbf{fix} $ys$ $ys'$ $zs$
\textbf{assume} $[] ≃ \{ys, zs, P\}$

3
hence $ys = [] \land zs = []$ by (rule interleave-nil)
moreover assume $[] \simeq \{ys', zs, P\}$

hence $ys' = [] \land zs = []$ by (rule interleave-nil)
ultimately show $ys = ys'$ by simp

next

fix $x \; xs \; ys \; ys' \; zs$

assume

\begin{align*}
A & : \forall ys \; ys' \; zs. \; xs \simeq \{ys, \; zs, \; P\} \implies xs \simeq \{ys', \; zs, \; P\} \implies ys = ys' \quad \text{and} \\
B & : x \neq xs \simeq \{ys, \; zs, \; P\} \quad \text{and} \\
B' & : x \neq xs \simeq \{ys', \; zs, \; P\}
\end{align*}

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 $xs \simeq \{ws, [], P\} \implies xs \simeq \{ws', [], P\} \implies ws = ws'$

using $A$.

ultimately have $ws = ws'$ by simp

with $Y$ and $Y'$ and $D$ and $D'$ show $\text{thesis}$ 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: interleave-length)

moreover have $\text{length} \; xs = \text{length} \; vs$

using $D$ by (simp add: interleave-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 $xs \simeq ([], vs, P) \implies xs \simeq \{w' \neq ws', vs, P\} \implies$

$[] = w' \neq ws'$

using $A$.

ultimately have $[] = w' \neq ws'$ using $D$ by simp

hence $\text{False}$ by simp

}
ultimately show False ..

next

fix v vs w ws
assume C: zs = v # vs and ys' = []
hence D: xs ≃ {[], vs, P} using B' by simp
assume E: ys = w # ws
have P x xs ∨ ¬ P x xs by simp
moreover {
  assume P x xs
  hence xs ≃ {ws, v # vs, P} using B and C and E by simp
  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 ¬ P x xs
  hence xs ≃ {w # ws, vs, P} using B and C and E by simp
  moreover have xs ≃ {[], vs, P} → xs ≃ {w # ws, vs, P} → [] = 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 ∨ ¬ 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 ≃ {us, 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 ≃ {us', v # vs, P} using B' and C and D' and E by simp
  moreover have xs ≃ {us, v # vs, P} → xs ≃ {us', v # vs, P} →
  ws = ws'
  using A.
  ultimately have ws = ws' by simp
  hence w = w' ∧ ws = ws' using F and F' by simp
}
moreover {
  assume E: ¬ P x xs
  hence xs ≃ {w # ws, vs, P} using B and C and D by simp
  moreover have xs ≃ {w' # ws', vs, P}
  using B' and C and D' and E by simp
  moreover have xs ≃ {w # ws, vs, P} → xs ≃ {w' # ws', vs, P} →
  w # ws = w' # ws'
}
ultimately have \( w \neq w' \neq w' \) by simp
hence \( w = w' \land w = w' \) by simp

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

qed

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

Since \textit{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 \textit{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:
\[ \text{xs} \simeq \{\text{xs}, [], P\} \]
by (induction xs, simp-all)

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

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

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

lemma interleaves-filter [rule-format]:
assumes \( A : \forall x \text{ xs}. P x (\text{filter} Q \text{ xs}) = P x \text{ xs} \)
shows \( \text{xs} \simeq \{\text{ys}, \text{zs}, P\} \implies \text{filter} Q \text{ xs} \simeq \{\text{filter} Q \text{ ys}, \text{filter} Q \text{ zs}, P\} \)
proof (induction \text{xs} arbitrary: \text{ys} \text{zs}, rule-tac [!] impI, simp)
fix \text{ys} \text{zs}
assume \[ \text{[]} \simeq \{\text{ys}, \text{zs}, P\} \]
hence \( \text{ys} = \text{[]} \land \text{zs} = \text{[]} \) by (rule interleaves-nil)
thus \[ \text{[]} \simeq \{\text{filter} Q \text{ ys}, \text{filter} Q \text{ zs}, P\} \] by simp
next
fix \text{x} \text{xs} \text{ys} \text{zs}
assume \[ B : \forall \text{ys}', \text{zs}'. \text{xs} \simeq \{\text{ys}', \text{zs}', P\} \implies \]
\[ \text{filter } Q \, z T \simeq \{ \text{filter } Q \, y_ T', \text{filter } Q \, z T', P \} \text{ and } \]

\[ \text{C: } x \neq z T \simeq \{ y_ T, z T, P \} \]

\[ \text{show } \text{filter } Q \, (x \neq z T) \simeq \{ \text{filter } Q \, y_ T, \text{filter } Q \, z T, P \} \]

\[ \text{proof (cases } y_ T, \text{case-tac } []) \, z T, \text{simp-all del: filter.simps, rule ccontr) } \]

\[ \text{assume } y_ T = [] \ \text{and } z T = [] \]

\[ \text{thus } \text{False using } C \text{ by simp} \]

\[ \text{next} \]

\[ \text{fix } z \, y T \]

\[ \text{assume } y_ T = y \neq y_ T' \ \text{and } z T = [] \]

\[ \text{hence } D: x = z \wedge z T \simeq \{ y_ T', [], P \} \text{ using } C \text{ by simp} \]

\[ \text{moreover have } z T \simeq \{ y_ T', [], P \} \rightarrow \]

\[ \text{filter } Q \, z T \simeq \{ \text{filter } Q \, y_ T', \text{filter } Q \, [], P \} \]

\[ \text{using } B \text{.} \]

\[ \text{ultimately have } \text{filter } Q \, z T \simeq \{ [], \text{filter } Q \, z T', P \} \text{ by simp} \]

\[ \text{thus } \text{filter } Q \, (x \neq z T) \simeq \{ [], \text{filter } Q \, (z \neq z T'), P \} \text{ using } D \text{ by simp} \]

\[ \text{next} \]

\[ \text{fix } y \, y_ T' \, z \, z T' \]

\[ \text{assume } y_ T = y \neq y_ T' \ \text{and } z T = z \neq z T' \]

\[ \text{hence } D: x = y \wedge z T \simeq \{ y_ T', [], P \} \text{ using } C \text{ by simp} \]

\[ \text{moreover have } z T \simeq \{ y_ T', [], P \} \rightarrow \]

\[ \text{filter } Q \, z T \simeq \{ \text{filter } Q \, y_ T', \text{filter } Q \, [], P \} \]

\[ \text{using } B \text{.} \]

\[ \text{ultimately have } \text{filter } Q \, z T \simeq \{ [], \text{filter } Q \, y_ T', [], P \} \text{ by simp} \]

\[ \text{thus } \text{filter } Q \, (x \neq z T) \simeq \{ [], \text{filter } Q \, (y \neq y_ T'), [], P \} \text{ using } D \text{ by simp} \]

\[ \text{next} \]

\[ \text{fix } y \, y_ T' \, z \, z T' \]

\[ \text{assume } y_ T = y \neq y_ T' \ \text{and } z T = z \neq z T' \]

\[ \text{hence } D: x = y \wedge z T \simeq \{ y_ T', [], P \} \text{ using } C \text{ by simp} \]

\[ \text{moreover have } z T \simeq \{ y_ T', [], P \} \rightarrow \]

\[ \text{filter } Q \, z T \simeq \{ \text{filter } Q \, y_ T', \text{filter } Q \, [], P \} \]

\[ \text{using } B \text{.} \]

\[ \text{ultimately have } G: \text{filter } Q \, z T \simeq \{ [], \text{filter } Q \, y_ T', \text{filter } Q \, (z \neq z T'), P \} \]

\[ \text{by simp} \]

\[ \text{show } \text{?thesis} \]

\[ \text{proof (cases } Q \, x \text{) } \]

\[ \text{assume } Q \, x \]

\[ \text{hence } \text{filter } Q \, (x \neq z T) \simeq x \neq \text{filter } Q \, z T \text{ by simp} \]

\[ \text{moreover have } \text{filter } Q \, (y \neq y_ T') \simeq x \neq \text{filter } Q \, y_ T' \]

\[ \text{using } (Q \, x) \text{ and } F \text{ by simp} \]

\[ \text{ultimately show } \text{?thesis using } E \text{ and } G \]

\[ \text{by (cases } \text{filter } Q \, (z \neq z T'), \text{simp-all) } \]

\[ \text{next} \]

\[ \text{assume } \neg \, Q \, x \]

\[ \text{hence } \text{filter } Q \, (x \neq z T) \simeq \text{filter } Q \, z T \text{ by simp} \]

\[ 7 \]
moreover have $\text{filter } Q \ (y \neq ys) = \text{filter } Q \ ys'$ using $(\neg Q \ x) \text{ and } F$ by simp
ultimately show $\text{thesis}$ using $E$ and $G$
by (cases $\text{filter } Q \ (z \neq zs')$, simp-all)
qed
next

case False

hence $E: \neg P \ x \ (\text{filter } Q \ xs)$ using $A$ by simp

have $F: x = z \land xs \sim \{y \neq ys', zs', P\}$ using $D$ and $False$ by simp
moreover have $xs \sim \{filter \ Q \ (y \neq ys'), filter \ Q \ zs', P\}$
using $B$.
ultimately have $G: \text{filter } Q \ xs \sim \{\text{filter } Q \ (y \neq ys'), \text{filter } Q \ zs', P\}$
by simp

delimiting

show $\text{thesis}$
proof (cases $Q \ x$)

assume $Q \ x$
hence $\text{filter } Q \ (x \neq xs) = x \neq \text{filter } Q \ xs$ by simp
moreover have $\text{filter } Q \ (z \neq zs') = x \neq \text{filter } Q \ zs'$
using $(\neg Q \ x)$ and $F$ by simp
ultimately show $\text{thesis}$ using $E$ and $G$
by (cases $\text{filter } Q \ (y \neq ys')$, simp-all)

next

assume $\neg Q \ x$
hence $\text{filter } Q \ (x \neq xs) = \text{filter } Q \ xs$ by simp
moreover have $\text{filter } Q \ (z \neq zs') = \text{filter } Q \ zs'$
using $(\neg Q \ x)$ and $F$ by simp
ultimately show $\text{thesis}$ using $E$ and $G$
by (cases $\text{filter } Q \ (z \neq zs')$, simp-all)
qed
qed

lemma interleaves-map [rule-format]:
assumes $A: \text{inj } f$
shows $xs \sim \{ys, zs, P\} \implies
\text{map } f \ xs \sim \{\text{map } f \ ys, \text{map } f \ zs, \lambda w \ ws. \ P \ (\text{inv } f \ w) \ (\text{map } (\text{inv } f) \ ws)\}$
(is $\cdot \implies \sim \sim \{\cdot, \sim, ?P'\}$)

proof (induction $xs$ arbitrary: $ys$ $zs$, rule-tac $[] \ \text{impI, simp-all}$
fix $ys$ $zs$
assume $[] \sim \{ys, zs, P\}$
hence $ys = [] \land zs = []$ by (rule interleaves-nil)
thus $[] \sim \{\text{map } f \ ys, \text{map } f \ zs, ?P'\}$ by simp
next

fix $x$ $xs$ $ys$ $zs$
assume $B: \forall ys. \ xs \sim \{ys, zs, P\} \implies \text{map } f \ xs \sim \{\text{map } f \ ys, \text{map } f \ zs, ?P'\}$ and $C: x \neq xs \sim \{ys, zs, P\}$
show \( f x \# \text{map} f \, xs \simeq \{ \text{map} f \, ys, \text{map} f \, zs, ?P' \} \)

proof (cases \( ys, \text{case-tac } \{ [\], zs \}, \text{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 \wedge zs \simeq [\], zs', P \) using C by simp
moreover have \( xs \simeq [\], zs', P \) \( \rightarrow \) \( \text{map} f \, xs \simeq \{ \text{map} f \, [\], \text{map} f \, zs', ?P' \} \)
  using B .
ultimately show \( f x = f z \wedge \text{map} f \, xs \simeq [\], \text{map} f \, zs', ?P' \) by simp
next
  fix \( y \) ys'
  assume \( ys = y \# ys' \) and \( zs = [\] \)
  hence \( x = y \wedge zs \simeq [ys', [\], P] \) using C by simp
moreover have \( xs \simeq [ys', [\], P] \) \( \rightarrow \) \( \text{map} f \, xs \simeq \{ \text{map} f \, ys', \text{map} f \, [\], ?P' \} \)
  using B .
ultimately show \( f x = f y \wedge \text{map} f \, xs \simeq \{ \text{map} f \, ys', [\], ?P' \} \) 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 \( f x \# \text{map} f \, xs \simeq \{ f y \# \text{map} f \, ys', f z \# \text{map} f \, zs', ?P' \} \)
proof (cases \( P \times xs \))
case True
  hence \( E: ?P' (f x) (\text{map} f \, xs) \) using A by simp
have \( x = y \wedge xs \simeq [ys', z \# zs', P] \) using D and True by simp
moreover have \( xs \simeq [ys', z \# zs', P] \) \( \rightarrow \)
\( \text{map} f \, xs \simeq \{ \text{map} f \, ys', \text{map} f \, (z \# zs'), ?P' \} \)
  using B .
ultimately have \( f x = f y \wedge \text{map} f \, xs \simeq \{ \text{map} f \, ys', \text{map} f \, (z \# zs'), ?P' \} \)
  by simp
thus \( ?\text{thesis} \) using E by simp
next
case False
  hence \( E: \neg ?P' (f x) (\text{map} f \, xs) \) using A by simp
have \( x = z \wedge xs \simeq [y \# ys', zs', P] \) using D and False by simp
moreover have \( xs \simeq [y \# ys', zs', P] \) \( \rightarrow \)
\( \text{map} f \, xs \simeq \{ \text{map} f \, (y \# ys'), \text{map} f \, zs', ?P' \} \)
  using B .
ultimately have \( f x = f z \wedge \text{map} f \, xs \simeq \{ \text{map} f \, (y \# ys'), \text{map} f \, zs', ?P' \} \)
  by simp
thus \( ?\text{thesis} \) using E by simp
qed
qed

lemma interleaves-prefix-fst-1 [rule-format]:
assumes A: \( xs \simeq \{ ys, zs, P \} \)
shows $(\forall n < \text{length } ws. \ P\ (ws \mapsto n) \ (\text{drop } (\text{Suc } n) \ ws @ xs)) \longrightarrow\ 
ws @ xs \simeq \{\ ws @ ys,\ zs,\ P\}$

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

fix $w\ ws$

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

assume $(\forall n < \text{length } ws. \ P\ (ws \mapsto n) \ (\text{drop } (\text{Suc } n) \ ws @ xs)) \longrightarrow\ 
ws @ xs \simeq \{\ ws @ ys,\ zs,\ P\}$

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

proof (rule allI, rule impI)

fix $n$

assume $n < \text{length } ws$

moreover have $\text{Suc } n < \text{Suc } (\text{length } ws) \longrightarrow\ 
P\ ((w \#\ ws) ! (\text{Suc } n)) \ (\text{drop } (\text{Suc } n) \ ws @ xs)$

using $B$ ..

ultimately have $\text{Suc } n < \text{Suc } (\text{length } ws) \longrightarrow\ 
P\ ((w \#\ ws) ! 0) \ (\text{drop } 0\ ws @ xs)$

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]:

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

proof (induction ws, simp-all, (rule impI)+)

fix $w\ ws$

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

hence $0 < \text{Suc } (\text{length } ws) \longrightarrow\ 
P\ ((w \#\ ws) ! 0) \ (\text{drop } 0\ ws @ xs)$ ..

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

ultimately have $(\forall n < \text{length } ws. \ P\ (ws \mapsto n) \ (\text{drop } (\text{Suc } n) \ ws @ xs)) \longrightarrow\ 
xs \simeq \{ys,\ zs,\ P\}$

by simp

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

proof (rule allI, rule impI)

fix $n$

assume $n < \text{length } ws$

moreover have $\text{Suc } n < \text{Suc } (\text{length } ws) \longrightarrow\ 
P\ ((w \#\ ws) ! (\text{Suc } n)) \ (\text{drop } (\text{Suc } n) \ ws @ xs)$

using $A$ ..

ultimately show $P\ (ws \mapsto n) \ (\text{drop } (\text{Suc } n) \ ws @ xs)$ by simp

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

qed

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

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

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

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

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 $P y [x] = \text{True}$ 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 $?zs \simeq \{[], yxs, ?P\}$. Similarly, $[x, y] \simeq \{[x], [y], \lambda w \text{ ws. } P w (\text{ws } @ [x])\}$ 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 $[y] \simeq \{[], [y], \lambda w \text{ ws. } P w (\text{ws } @ [x])\} \implies [x, y] \simeq \{[x], [y], P\}$ 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 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 => 'a list => bool) => 'a list => 'a list => 'a list => bool where
Interleaves P (x # xs) (y # ys) (z # zs) = (if P x xs then x = y \& Interleaves P xs ys (z # zs)
else \( x = z \land \text{Interleaves} \ P \ xs \ (y \ # \ ys) \ zs \) |

\[
\text{Interleaves} \ P \ (x \ # \ xs) \ (y \ # \ ys) \ [] = \\
(P \ xs \land x = y \land \text{Interleaves} \ P \ xs \ ys \ [] ) |
\]

\[
\text{Interleaves} \ P \ (x \ # \ xs) \ [] \ (z \ # \ zs) = \\
(\neg P \ xs \land x = z \land \text{Interleaves} \ P \ xs \ [] \ zs ) |
\]

\[
\text{Interleaves} \cdot \ (\cdot \ # \ -) \ [] \ [] = False |
\]

\[
\text{Interleaves} \cdot \ [] \ (\cdot \ # \ -) = False |
\]

\[
\text{Interleaves} \cdot \ [] \ [] = True
\]

abbreviation Interleaves-syntax ::

'\ a \ list \Rightarrow '\ a \ list \Rightarrow (\ 'a \Rightarrow '\ a \ list \Rightarrow \text{bool} ) \Rightarrow \text{bool}

((\ x\ :: \ 'a) \ [60,60,60]) 51

where \( xs \triangleq \{ys, zs, P\} \equiv \text{Interleaves} \ P \ xs \ ys \ zs \)

In what follows, it is proven that predicate \text{Interleaves} is actually not weaker than, viz. is a sufficient condition for, predicate \text{interleaves}. Moreover, the former predicate is shown to fulfill the same rules as the latter, although sometimes under more stringent assumptions (cf. lemmas \text{Interleaves-all-nil}, \text{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]:

\[
xs \equiv \{ys, zs, P\} \rightarrow xs \simeq \{ys, zs, P\}
\]

**proof** (induction \( P \ xs \ ys \ zs \) rule: interleaves.induct, simp-all)

**qed** (rule conjI, (rule-tac [!] implI)+, simp-all)

**lemma** Interleaves-length:

\[
xs \equiv \{ys, zs, P\} \Rightarrow \text{length} \ xs = \text{length} \ ys + \text{length} \ zs
\]

**by** (drule Interleaves-interleaves, rule interleaves-length)

**lemma** Interleaves-nil:

\[
[] \equiv \{ys, zs, P\} \Rightarrow ys = [] \land zs = []
\]

**by** (drule Interleaves-interleaves, rule interleaves-nil)

**lemma** Interleaves-swap:

\[
xs \equiv \{ys, zs, P\} = xs \equiv \{zs, ys, \lambda w \ ws. \neg P \ w \ ws\}
\]

**proof** (induction \( P \ xs \ ys \ zs \) rule: Interleaves.induct, simp-all)

**fix** \( y' :: \ 'a \ \text{and} \ ys' \ zs' \ P' \)

**show** \( \neg [] \equiv \{zs', y' \ # \ ys', \lambda w \ ws. \neg P' \ w \ ws\} \) **by** (cases zs', simp-all)

**qed**

**lemma** Interleaves-equal-fst:

\[
xs \equiv \{ys, zs, P\} \Rightarrow xs \equiv \{ys', zs, P\} \Rightarrow ys = ys'
\]

**by** ((drule Interleaves-interleaves)+, rule interleaves-equal-fst)

12
lemma Interleaves-equal-snd:
\(\text{xs} \cong \{\text{ys}, \text{zs}, P\} \implies \text{xs} \cong \{\text{ys}, \text{zs'}, P\} \implies \text{zs} = \text{zs'}\)
by ((drule Interleaves-interleaves)+, rule Interleaves-equal-snd)

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

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

lemma Interleaves-filter [rule-format]:
assumes \(A: \forall x \text{xs}. \ P x (\text{filter Q xs}) = P x \text{xs}\)
shows \(\text{xs} \cong \{\text{ys}, \text{zs}, P\} \implies \text{filter Q xs} \cong \{\text{filter Q ys}, \text{filter Q zs}, P\}\)
proof (induction \text{xs} arbitrary: ys zs, rule-tac [\text{[]}] impI, simp)
fix ys zs
assume \(\text{[]} \cong \{\text{ys}, \text{zs}, P\}\)

hence \(\text{ys} = \text{[]} \land \text{zs} = \text{[]}\) by (rule Interleaves-nil)
thus \(\text{[]} \cong \{\text{filter Q ys}, \text{filter Q zs}, P\}\) by simp

next
fix x xs ys zs
assume \(B: \land (\text{ys'}, \text{zs'}, \text{xs} \cong \{\text{ys'}, \text{zs'}, \text{P}\}) \implies \text{filter Q xs} \cong \{\text{filter Q ys'}, \text{filter Q zs'}, P\}\) and
\(C: x \neq \text{xs} \cong \{\text{ys}, \text{zs}, P\}\)
show \(\text{filter Q} (x \neq \text{xs}) \cong \{\text{filter Q ys}, \text{filter Q zs}, P\}\)
proof (cases \text{ys}, case-tac [\text{[]} zs, simp-all del: filter.simps, rule contr)
assume \(\text{ys} = \text{[]} \land \text{zs} = \text{[]}\)
thus False using \(C\) by simp

next
fix z zs'
assume \(\text{ys} = \text{[]} \land \text{zs} = z \neq \text{zs'}\)

hence \(D: \neg P x \text{xs} \land x = z \land \text{xs} \cong \{\text{[]}, z, P\}\) using \(C\) by simp+

moreover have \(\text{xs} \cong \{\text{[]}, P\} \implies \text{filter Q xs} \cong \{\text{filter Q []}, \text{filter Q zs}, P\}\)

using \(B\).
ultimately have \(\text{filter Q xs} \cong \{\text{[]}, \text{filter Q zs'}, P\}\) by simp

moreover have \(\neg P x (\text{filter Q xs})\) using \(A\) and \(D\) by simp+
ultimately show \(\text{filter Q} (x \neq \text{xs}) \cong \{\text{[]}, \text{filter Q} (x \neq z'), P\}\)

using \(D\) by simp

next
fix y ys'
assume \(\text{ys} = y \neq \text{ys'}\) and \(\text{zs} = \text{[]}\)

hence \(D: P x \text{xs} \land x = y \land \text{xs} \cong \{y, P\}\) using \(C\) by simp+

moreover have \(\text{xs} \cong \{y', P\} \implies \text{filter Q xs} \cong \{\text{filter Q y}, \text{filter Q []}, P\}\)

using \(B\).
ultimately have \(\text{filter Q xs} \cong \{\text{filter Q y}, P\}\) by simp
moreover have \( P \, x \) (filter \( Q \, x \)) using \( A \) and \( D \) by simp+
ultimately show \( \text{filter} \, Q \, (x \# \, x) \cong \{\text{filter} \, Q \, (y \# \, y'), [], P\}
using \( D \) by simp

next
fix \( y \, y' \, z \, z' \)
assume \( y \, y' = y' \) and \( z \, z' = z \, z' \)

hence \( D: x \# \, x \cong \{y', z \# \, z', P\} \) using \( C \) by simp

show \( \text{filter} \, Q \, (x \# \, x) \cong \{\text{filter} \, Q \, (y \# \, y'), \text{filter} \, Q \, (z \# \, z'), P\}
proof (cases \( P \, x \, x \))
case True
hence \( E: P \, x \) (filter \( Q \, x \)) using \( A \) by simp
have \( F: x = y \land x \cong \{y', z \# \, z', P\} \) using \( D \) and True by simp

moreover have \( x \cong \{y', z \# \, z', P\} \)

filter \( Q \, x \cong \{\text{filter} \, Q \, y', \text{filter} \, Q \, (z \# \, z'), P\}
using \( B \).
ultimately have \( G: \text{filter} \, Q \, x \cong \{\text{filter} \, Q \, y', \text{filter} \, Q \, (z \# \, z'), P\}
by simp
show \( \text{thesis} \)
proof (cases \( Q \, x \))
assume \( Q \, x \)
hence \( \text{filter} \, Q \, (x \# \, x) = x \# \, \text{filter} \, Q \, x \) by simp

moreover have \( \text{filter} \, Q \, (y \# \, y') = x \# \, \text{filter} \, Q \, y' \)

using (\( Q \, x \)) and \( F \) by simp
ultimately show \( \text{thesis} \) using \( E \) and \( G \)
by (cases \( \text{filter} \, Q \, (z \# \, z') \), simp-all)

next
assume \( \neg \, Q \, x \)
hence \( \text{filter} \, Q \, (x \# \, x) = \text{filter} \, Q \, x \) by simp

moreover have \( \text{filter} \, Q \, (y \# \, y') = \text{filter} \, Q \, y' \)

using (\( \neg \, Q \, x \)) and \( F \) by simp
ultimately show \( \text{thesis} \) using \( E \) and \( G \)
by (cases \( \text{filter} \, Q \, (z \# \, z') \), simp-all)
qed

next
case False
hence \( E: \neg \, P \, x \) (filter \( Q \, x \)) using \( A \) by simp

have \( F: x = z \land x \cong \{y \# \, y', z' \}, P\) using \( D \) and False by simp

moreover have \( x \cong \{y \# \, y', z' \}, P\) \( \rightarrow \)

filter \( Q \, x \cong \{\text{filter} \, Q \, (y \# \, y'), \text{filter} \, Q \, z' \), P\}
using \( B \).
ultimately have \( G: \text{filter} \, Q \, x \cong \{\text{filter} \, Q \, (y \# \, y'), \text{filter} \, Q \, z', P\}
by simp
show \( \text{thesis} \)
proof (cases \( Q \, x \))
assume \( Q \, x \)
hence \( \text{filter} \, Q \, (x \# \, x) = x \# \, \text{filter} \, Q \, x \) by simp

moreover have \( \text{filter} \, Q \, (z \# \, z') = x \# \, \text{filter} \, Q \, z' \)

using (\( Q \, x \)) and \( F \) by simp
ultimately show \( \text{thesis} \) 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' 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 - ⟹ - ≅ {[ - , - , ?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+
morerover 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 ∧ x = y ∧ xs ≅ {ys', [], P} using C by simp+
morerover have xs ≅ {ys', [], P} ⟹ map f xs ≅ {map f ys', map f [], ?P'} using B .
ultimately have map f xs ≅ {map f ys', [], ?P'} by simp
moreover have \(?P'\ (f x) (map f zs)\) using \(A\) and \(D\) by \(\text{simp+}\)
ultimately show \(f x \# \text{map} f \ xs \cong \{ f y \# \text{map} f \ ys', [], \ ?P' \}\)
using \(D\) by \(\text{simp}\)
next
fix \(y\), \(ys\), \(z\), \(zs\)
assume \(ys = y \# ys'\) and \(zs = z \# zs'\)
hence \(D\): \(x \# \text{xs} \cong \{ y \# ys', z \# zs', P \}\) using \(C\) by \(\text{simp}\)
show \(f x \# \text{map} f \ xs \cong \{ f y \# \text{map} f \ ys', f z \# \text{map} f \ zs', \ ?P' \}\)
proof (cases \(P\ x\ xs\))
case \(\text{True}\)
hence \(E\): \(?P'\ (f x) (\text{map} f \ xs)\) using \(A\) by \(\text{simp}\)
have \(x = y \land \text{xs} \cong \{ ys', z \# zs', P \}\) using \(D\) and \(\text{True}\) by \(\text{simp}\)
moreover have \(\text{zs} \cong \{ ys', z \# zs', P \}\) using \(\text{simp}\)
ultimately have \(f x = f y \land \text{map} f \ xs \cong \{ \text{map} f \ ys', \text{map} f (z \# zs'), \ ?P' \}\)
by \(\text{simp}\)
thus \(\text{thesis}\) using \(E\) by \(\text{simp}\)
next
case \(\text{False}\)
hence \(E\): \(\neg \ ?P'\ (f x) (\text{map} f \ xs)\) using \(A\) by \(\text{simp}\)
have \(x = z \land \text{zs} \cong \{ y \# ys', zs', P \}\) using \(D\) and \(\text{False}\) by \(\text{simp}\)
moreover have \(\text{zs} \cong \{ y \# ys', zs', P \}\) using \(\text{simp}\)
ultimately have \(f x = f z \land \text{map} f \ xs \cong \{ \text{map} f (y \# ys'), \text{map} f zs', \ ?P' \}\)
by \(\text{simp}\)
thus \(\text{thesis}\) using \(E\) by \(\text{simp}\)
qed
qed

lemma \(\text{Interleaves-prefix-fst-1}\) [rule-format]:
assumes \(A\): \(\text{xs} \cong \{ y, z, P \}\)
shows \((\forall n < \text{length} \ ws. \ P (\text{ws} \# ! n) (\text{drop} (\text{Suc} \ n) \text{ ws @ xs})) \rightarrow \text{ws @ xs} \cong \{ \text{ws @ y, z, P} \}\)
proof (induction \(ws\), simp-all add: \(A\), rule \(\text{impI}\))
fix \(w\), \(ws\)
assume \(B\): \(\forall n < \text{Suc} (\text{length} \ ws). \ P ((w \# ws) \# ! n) (\text{drop} n \text{ \ws @ xs})\)
assume \((\forall n < \text{length} \ ws. \ P (\text{ws} \# ! n) (\text{drop} (\text{Suc} \ n) \text{ ws @ xs})) \rightarrow \text{ws @ xs} \cong \{ \text{ws @ y, z, P} \}\)
moreover have \(\forall n < \text{length} \ ws. \ P (\text{ws} \# ! n) (\text{drop} \text{Suc} n \text{ ws @ xs})\)
proof (rule \(\text{allI}\), rule \(\text{impI}\))
fix \(n\)
assume \(n < \text{length} \ ws\)
moreover have \(\text{Suc} n < \text{Suc} (\text{length} \ ws) \rightarrow \ P ((w \# ws) \# ! (\text{Suc} \ n)) (\text{drop} (\text{Suc} \ n) \text{ ws @ xs})\)
using \(B\) ..
ultimately show \(P (\text{ws} \# ! n) (\text{drop} \text{Suc} n \text{ ws @ xs})\) by \(\text{simp}\)
qed
ultimately have \( ws @ xs \cong \{ ws @ ys, zs, P \} \) ..
moreover have \( 0 < Suc (\text{length } ws) \rightarrow P ((w \# ws) ! 0) (\text{drop } 0 ws @ xs) \)
using \( B \) ..
hence \( P w (ws @ xs) \) by simp
ultimately show \( w \# ws @ xs \cong \{ w \# ws @ ys, zs, P \} \) by (cases zs, simp-all)
qed

lemma Interleaves-prefix-fst-2 [rule-format]:
\[
\begin{align*}
ws @ xs & \cong \{ ws @ ys, zs, P \} \\
(\forall n < \text{length } ws. P (ws ! n) (\text{drop } (Suc n) ws @ xs)) & \rightarrow \\
xs & \cong \{ ys, zs, P \}
\end{align*}
\]
proof (induction ws, simp-all, (rule impl)+)
fix \( w ws \)
assume \( A; \forall n < Suc (\text{length } ws). P ((w \# ws) ! n) (\text{drop } n ws @ xs) \)
hence \( 0 < Suc (\text{length } ws) \rightarrow P ((w \# ws) ! 0) (\text{drop } 0 ws @ xs) \) ..
hence \( P w (ws @ xs) \) by simp
moreover assume \( w \# ws @ xs \cong \{ w \# ws @ ys, zs, P \} \)
ultimately have \( ws @ xs \cong \{ ws @ ys, zs, P \} \) by (cases zs, simp-all)
moreover assume \( ws @ xs \cong \{ ws @ ys, zs, P \} \)
ultimately have \( (\forall n < \text{length } ws. P (ws ! n) (\text{drop } (Suc n) ws @ xs)) \rightarrow \\
xs & \cong \{ ys, zs, P \}
by simp
moreover have \( \forall n < \text{length } ws. P (ws ! n) (\text{drop } (Suc n) ws @ xs) \)
proof (rule allI, rule impl)
fix \( n \)
assume \( n < \text{length } ws \)
moreover have \( Suc n < Suc (\text{length } ws) \rightarrow \\
P ((w \# ws) ! (Suc n)) (\text{drop } (Suc n) ws @ xs) \)
using \( A \) ..
ultimately show \( P (ws ! n) (\text{drop } (Suc n) ws @ xs) \) by simp
qed
ultimately show \( xs \cong \{ ys, zs, P \} \) ..
qed

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

lemma Interleaves-prefix-snd [rule-format]:
\[
\forall n < \text{length } ws. \neg P (ws ! n) (\text{drop } (Suc n) ws @ xs) \Rightarrow \\
xz & \cong \{ ys, zs, P \} = ws @ xs \cong \{ ys, ws @ zs, P \}
\]
proof (subst (1 2) Interleaves-swap)
qed (rule Interleaves-prefix-fst, simp)
lemma Interleaves-all-nil-1 [rule-format]:
\(xs \equiv \{xs, [], P\} \rightarrow (\forall n < \text{length} \; xs. \; P \; (xs \; ! \; n) \; (\text{drop} \; (\text{Suc} \; n) \; xs))\)

proof (induction \(xs\), simp-all, rule impI, erule conjE, rule allI, rule impI)
  fix \(x\) \(xs\) \(n\)
  assume \(xs \equiv \{xs, [], P\} \rightarrow (\forall n < \text{length} \; xs. \; P \; (xs \; ! \; n) \; (\text{drop} \; (\text{Suc} \; n) \; xs))\) and \(xs \equiv \{xs, [], P\}\)
  hence \(A: \forall n < \text{length} \; xs. \; P \; (xs \; ! \; n) \; (\text{drop} \; (\text{Suc} \; n) \; xs)\) ..
  assume \(B: P \; x \; xs\) and
  \(C: n < \text{Suc} \; (\text{length} \; xs)\)
  show \(P \; ((x \; \neq \; xs) \; ! \; n) \; (\text{drop} \; n \; xs)\)
  proof (cases \(n\), simp-all add: \(B\))
    case (Suc \(m\))
    have \(m < \text{length} \; xs \rightarrow P \; (xs \; ! \; m) \; (\text{drop} \; (\text{Suc} \; m) \; xs)\) using \(A\) ..
    moreover have \(m < \text{length} \; xs\) using \(C\) and \(\text{Suc}\) by simp
    ultimately show \(P \; (xs \; ! \; m) \; (\text{drop} \; (\text{Suc} \; m) \; xs)\) ..
  qed
qed

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

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

proof (rule iffI, rule allI, rule impI, rule Interleaves-all-nil-1, assumption+)
qed (rule Interleaves-all-nil-2, simp)

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

lemma Interleaves-suffix-one-aux:
  assumes \(A: P \; x \; []\)
  shows \(\neg xs \; @ \; [x] \equiv \{[], zs, P\}\)
proof (induction \(xs\) arbitrary: zs, simp-all, rule-tac [] notI)
  fix \(zs\)
  assume \([x] \equiv \{[], zs, P\}\)
  thus \(False\) by (cases zs, simp-all add: \(A\))
next
  fix \(w\) \(xs\) \(zs\)
  assume \(B: \land_{zs}. \; \neg xs \; @ \; [x] \equiv \{[], zs, P\}\)
  assume \(w \; \neq \; xs \; @ \; [x] \equiv \{[], zs, P\}\)
  thus \(False\) proof (cases zs, simp-all, (erule-tac conjE)+)
    fix zs'
    assume \(zs \; @ \; [x] \equiv \{[], zs', P\}\)
    moreover have \(\neg zs \; @ \; [x] \equiv \{[], zs', P\}\) using \(B\)
    ultimately show \(False\) by contradiction
proof

next

proof

next

next

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

fix ys zs

assume \([x] \cong \{ys \circ [x], zs, P\}\) 

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 \([], \cong \{ys, zs, \neg P\}\) by simp

next

fix w xs ys zs

assume B: \(\{ys, zs, xs \circ [x] \cong \{ys \circ [x], zs, P\}\) 

assumes \(w \# xs \cong \{ys \circ [x], zs, P\}\)

thus \(w \# xs \cong \{ys, zs, \neg P\}\)

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

(erule-tac ![I_2] conjE+)

assume \(xs \circ [x] \cong [[], [], P]\)

thus \(False\) by (cases zs, simp-all)

next

fix \(ys'\)

have \(xs \circ [x] \cong \{ys' \circ [x], [], P\} \rightarrow xs \cong \{ys', [], \neg P\}\)

using B.

moreover assume \(xs \circ [x] \cong \{ys' \circ [x], [], P\}\)

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

next

fix \(z'\) zs'

assume \(w \# xs \circ [x] \cong [[], z' \# zs', P]\)

thus \(w \# xs \cong [[], z' \# zs', \neg P]\)

proof (cases P w (xs \circ [x]), simp-all, erule-tac ![1] conjE)

assume \(xs \circ [x] \cong [[], z' \# zs', P]\)

moreover have \(\neg xs \circ [x] \cong [[], z' \# zs', P]\)

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

ultimately show \(False\) by contradiction

next

have \(xs \circ [x] \cong [[], zs', P] \rightarrow xs \cong [[], zs', \neg P]\)

using B by simp

moreover assume \(xs \circ [x] \cong [[], zs', P]\)

ultimately show \(xs \cong [[], zs', \neg P]\)

qed

next

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

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

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

proof (cases P w (xs \circ [x]), simp-all, erule-tac ![1] conjE)

have \(xs \circ [x] \cong \{ys' \circ [x], z' \# zs', P\} \rightarrow xs \cong \{ys', z' \# zs', \neg P\}\)
using \( B \).

moreover assume \( xS @ [x] \cong \{ yS' @ [x], z' \neq zS', P \} \)

ultimately show \( xS \cong \{ yS', z' \neq zS', \; \neg P \} \).

next

have \( xS @ [x] \cong \{ y' \neq yS' @ [x], zS', P \} \rightarrow xS \cong \{ y' \neq yS', zS', \; \neg P \} \)

using \( B \) by simp

moreover assume \( xS @ [x] \cong \{ y' \neq yS' @ [x], zS', P \} \)

ultimately show \( xS \cong \{ y' \neq yS', zS', \; \neg P \} \).

qed

qed

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

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

shows \( xS @ [x] \cong \{ yS, zS, \lambda v \; (\text{ws} @ [v]) \rightarrow xS @ [ws] \cong \{ yS @ [ws], zS, P \} \)

(is \( A \cong \{ - , - \; (\neg P) \rightarrow - \} \)

proof (induction \( xS \) arbitrary: \( yS \; zS, \text{rule-tac} \; [!] \; \text{impI}, \; \text{simp-all} \))

fix \( yS \) \( zS \)

assume \( [] \cong \{ yS, zS, \; \neg P \} \)

hence \( yS = [] \wedge zS = [] \) by (rule Interleaves-nil)

thus \( ws \cong \{ yS @ [ws], zS, P \} \) using \( A \) by (simp add: Interleaves-all-nil)

next

fix \( xS \) \( yS \) \( zS \)

assume \( A : \forall yS \; zS \; xS \cong \{ yS, zS, \; \neg P \} \rightarrow xS @ [ws] \cong \{ yS @ [ws], zS, P \} \)

assume \( xS = xS \)

thus \( yS \; zS \)

proof (rule-tac Interleaves.cases [of \( \neg P', x \neq xS, yS, zS \)], simp-all del: Interleaves.simps(1),

(erule-tac conjE)+, (erule-tac \( \neg \neg P \) conjE)+, (erule-tac \( \neg \neg \neg \neg P \) conjE)+)

fix \( P' \) \( x' \; xS' \)

assume \( B : x' \neq xS' \cong \{ y' \neq yS', x' \neq zS', P' \} \) and

\( C : \; \neg P' = P' \); and

\( D : xS = xS' \)

show \( x' \neq xS' @ [ws] \cong \{ y' \neq yS' @ [ws], z' \neq zS', P \} \)

proof (cases \( P' \; x' \; xS' \))

have \( xS \cong \{ yS', z' \neq zS', \; \neg P' \} \rightarrow xS @ [ws] \cong \{ yS' @ [ws], z' \neq zS', P \} \)

using \( A \).

moreover case \( \text{True} \)

hence \( xS \cong \{ yS', z' \neq zS', \; \neg P' \} \) using \( B \) and \( C \) and \( D \) by simp

ultimately have \( xS @ [ws] \cong \{ yS' @ [ws], z' \neq zS', P \} \).

moreover have \( P \; x'(xS' @ [ws]) \) using \( C \) [symmetric] and \( \text{True} \) by simp

moreover have \( z' = y' \) using \( B \) and \( \text{True} \) by simp

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

next

have \( xS \cong \{ y' \neq yS', zS', \; \neg P' \} \rightarrow xS @ [ws] \cong \{ (y' \neq yS') @ [ws], zS', P \} \)

using \( A \).

moreover case \( \text{False} \)

hence \( xS \cong \{ y' \neq yS', zS', \; \neg P' \} \) using \( B \) and \( C \) and \( D \) by simp
ultimately have \( xs \triangleq ys \triangleq \{y \neq ys\} \triangleq ws, zs', P \) ..
moreover have \( \neg x' (xs' @ ws) \) using \( C \) [symmetric] and \( False \) by simp
moreover have \( x' = z' \) using \( B \) and \( False \) by simp
ultimately show \( \neg thesis \) using \( D \) by simp
qed
next
fix \( P' \ x' \ xs' \ y' \ ys' \)
have \( xs \triangleq \{ys', [], \?P'\} \rightarrow xs \triangleq ys' \triangleq \{ys' \triangleq ws, [], P\} \) using \( A \).
moreover assume
\( xs' \triangleq \{ys', [], P'\} \) and
\( B: \?P' = P' \) and
\( C: xs = xs' \)
hence \( xs \triangleq \{ys', [], \?P'\} \) by simp
ultimately have \( xs' \triangleq ws \triangleq \{ys' \triangleq ws, [], P\} \) using \( C \) by simp
moreover assume
\( P' x' xs' \) and
\( x' = y' \)
hence \( P y' (xs' @ ws) \) using \( B \) [symmetric] by simp
ultimately show \( P y' (xs' @ ws) \land xs' @ ws \triangleq \{ys' @ ws, [], P\} \) by simp
next
fix \( P' x' xs' \ y' \ zs' \)
have \( xs \triangleq \{[], zs', \?P'\} \rightarrow xs \triangleq ws \triangleq \{[], \triangleq ws, zs', P\} \) using \( A \).
moreover assume
\( xs' \triangleq \{[], zs', P'\} \) and
\( B: \?P' = P' \) and
\( C: xs = xs' \)
hence \( xs \triangleq \{[], zs', \?P'\} \) by simp
ultimately have \( xs' \triangleq ws \triangleq \{ws, zs', P\} \) using \( C \) by simp
moreover assume
\( \neg P' x' xs' \) and
\( x' = z' \)
hence \( \neg P z' (xs' @ ws) \) using \( B \) [symmetric] by simp
ultimately show \( z' \neq xs' @ ws \triangleq \{ws, z' \neq zs', P\} \) by (cases \( ws, simp-all \))
qed
qed

lemma Interleaves-suffix-one-fst-1 [rule-formal]:
\( P x [\] \rightarrow xs \triangleq \{ys, zs, \lambda w \, ws, P w (ws @ [x])\} \Rightarrow xs @ [x] \triangleq \{ys @ [x], zs, P\} \)
by (rule Interleaves-suffix-fst-1, simp)

lemma Interleaves-suffix-one-fst:
\( P x [\] \rightarrow xs \triangleq \{ys, zs, \lambda w \, ws, P w (ws @ [x])\} = xs @ [x] \triangleq \{ys @ [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:
\( \neg P x [\] \rightarrow \)
lemma Interleaves-suffix-fst-2 [rule-format]:
\( \forall n < \text{length } ws. P (ws \uplus n) (\text{drop } (\text{Suc } n) ws) \rightarrow \neg P x (xs \uplus ws) \)


fix P

assume \( x \neq xs \equiv \{ [], zs, P \} \)

thus \( \neg P x ws \) by (cases zs, simp-all)

next

fix \( w ws P \)

assume

\( A: (\forall n < \text{length } ws. P' (ws \uplus n) (\text{drop } (\text{Suc } n) ws)) \rightarrow \neg P' x (xs \uplus ws) \) and

\( B: \forall n < \text{Suc } (\text{length } ws). P (ws @ [w]) \uplus n \)

\((\text{drop } (\text{Suc } n) ws @ drop (\text{Suc } n \text{ length } ws) [w])\)

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

let \( ?P' = \lambda v. P v (ws @ [w]) \)

have \((\forall n < \text{length } ws. ?P' (ws \uplus n) (\text{drop } (\text{Suc } n) ws)) \rightarrow \neg ?P' x (xs @ ws) \)

using A.

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

proof (rule allI, rule impl)

fix \( n \)

assume D: \( n < \text{length } ws \)

moreover have \( n < \text{Suc } (\text{length } ws) \rightarrow P (ws @ [w] ! n) \)

(\( \text{drop } (\text{Suc } n) ws @ drop (\text{Suc } n - \text{length } ws) [w] \))

using B.. .

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

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

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

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

qed

ultimately have \( x \neq xs @ ws \equiv \{ ws, zs, ?P' \} \rightarrow \neg ?P' x (xs @ ws) \) .

moreover have \( \text{length } ws < \text{Suc } (\text{length } ws) \rightarrow P ((ws @ [w] ! \text{length } ws) \)

(\( \text{drop } (\text{Suc } (\text{length } ws)) ws @ drop (\text{Suc } (\text{length } ws) - \text{length } ws) [w] \))

using B.. .

hence \( P w [] \) by simp

hence \( x \neq xs @ us \equiv \{ ws, zs, ?P' \} \)

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

ultimately have \( \neg ?P' x (xs @ ws) \) ..

thus \( \neg P x (xs @ ws @ [w]) \) by simp

qed

lemma Interleaves-suffix-fst-2 [rule-format]:
assumes $A$: $\forall n < \text{length} \, ws. \, P \, (\text{drop} (\text{Suc} \, n) \, ws)$

shows $xz @ ws \equiv \{ys \, @ \, ws, \, zs, \, P\} \rightarrow xz \equiv \{ys, \, zs, \, \lambda e \, ws. \, P \, v \, (ws \, @ \, ws)\}$

(is $\rightarrow \rightarrow - \equiv \{\cdot, \cdot, \cdot, P\}\})

proof (induction $xz$ arbitrary: $ys \, zs$, rule-tac $[\!]$ impI, simp-all)

fix $ys \, zs$

assume $ws \equiv \{ys \, @ \, ws, \, zs, \, P\}$

hence $B$: $\text{length} \, ws = \text{length} \, (ys \, @ \, ws) + \text{length} \, zs$

by (rule Interleaves-length)

have $ys: \text{length} \, ys = [\!]$ by (cases $ys$, simp, insert $B$, simp)

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

with $ys$ show $[\!] \equiv \{ys, \, zs, \, ?P\}$ by simp

next

fix $xz \, ys \, zs$

assume $B$: $\cap ys \, zs. \, xz \, @ \, ws \equiv \{ys \, @ \, ws, \, zs, \, P\} \rightarrow xz \equiv \{ys, \, zs, \, ?P\}$

assume $x \# xz \, @ \, ws \equiv \{ys \, @ \, ws, \, zs, \, P\}$

thus $x \# xz \equiv \{ys, \, zs, \, ?P\}$

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

(erule-tac $[2]$ conjE $+$)

assume $C$: $x \# xz \, @ \, ws \equiv \{ws, \, [], \, P\}$

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

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

thus False by simp

next

fix $ys'$

have $xz \, @ \, ws \equiv \{ys' \, @ \, ws, \, [], \, P\} \rightarrow xz \equiv \{ys', \, [], \, ?P\}$ using $B$.

moreover assume $xz \, @ \, ws \equiv \{ys' \, @ \, ws, \, [], \, P\}$

ultimately show $xz \equiv \{ys', \, [], \, ?P\}$ .

next

fix $z' \, zs'$

assume $x \# xz \, @ \, ws \equiv \{ws, \, z' \# zs', \, P\}$

thus $x \# xz \equiv \{[], \, z' \# zs', \, ?P\}$

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

case True

moreover assume $x \# xz \, @ \, ws \equiv \{ws, \, z' \# zs', \, P\}$

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

by (rule Interleaves-suffix-aux)

ultimately show False by contradiction

next

case False

moreover assume $x \# xz \, @ \, ws \equiv \{ws, \, z' \# zs', \, P\}$

ultimately have $x = z' \wedge xz \, @ \, ws \equiv \{ws, \, zs', \, P\}$ by (cases $ws$, simp-all)

moreover have $xz \, @ \, ws \equiv \{[], \, @ \, ws, \, zs', \, P\} \rightarrow xz \equiv \{[], \, zs', \, ?P\}$

using $B$.

ultimately have $x = z' \wedge xz \equiv \{[], \, zs', \, ?P\}$ by simp

qed

next

fix $y' \, ys' \, z' \, zs'$

assume $x \# xz \, @ \, ws \equiv \{y' \# ys' \, @ \, ws, \, z' \# zs', \, P\}$

thus $x \# xz \equiv \{y' \# ys', \, z' \# zs', \, ?P\}$

23
proof (cases P x (xs @ ws), simp-all, erule-tac ![ conjE )
have xs @ ws \equiv \{ys' @ ws, z' \# zs', P\} \rightarrow xs \equiv \{ys', z' \# zs', ?P'\}
  using B .
moreover assume xs @ ws \equiv \{ys' @ ws, z' \# zs', P\}
ultimately show xs \equiv \{ys', z' \# zs', ?P'\} ..
next
have xs @ ws \equiv \{y' \# ys' @ ws, zs', P\} \rightarrow xs \equiv \{y' \# ys', zs', ?P'\}
  using B by simp
moreover assume xs @ ws \equiv \{y' \# ys' @ ws, zs', P\}
ultimately show xs \equiv \{y' \# ys', zs', ?P'\} ..
qed
qed

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

end

References


