EVENT: Start with the initial \texttt{nqthm} theory.

\begin{verbatim}
#

Copyright (C) 1994 by John Cowles. All Rights Reserved.

This script is hereby placed in the public domain, and therefore unlimited editing and redistribution is permitted.

NO WARRANTY

John Cowles PROVIDES ABSOLUTELY NO WARRANTY. THE EVENT SCRIPT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE SCRIPT IS WITH YOU. SHOULD THE SCRIPT PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

IN NO EVENT WILL John Cowles BE LIABLE TO YOU FOR ANY DAMAGES, ANY LOST PROFITS, LOST MONIES, OR OTHER SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THIS SCRIPT (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY THIRD PARTIES), EVEN IF YOU HAVE ADVISED US OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY OTHER PARTY.

|#

A NOTE ON SHELLS

by

John Cowles

Department of Computer Science

University of Wyoming

The following is intended to give the reader some insight into SHELLS.

Intuitively a nonempty SEQUENCE is an ordered list, possibly with duplicates, of objects, ( Obj1 Obj2 ... ObjN ).

There are two ways to recursively decompose sequences.

1. A SEQUENCE is either the EMPTY-SEQUENCE or a pair < Obj,Seq >.

2. A SEQUENCE is either the EMPTY-SEQUENCE or a pair [ Seq,Obj ].
\end{verbatim}
Here Obj is an object, Seq is a sequence, and EMPTY-SEQUENCE is the unique sequence which contains no objects. Different pairing brackets, < > and [ ], are used to emphasize which of the decompositions is being used.

Here the Shell Principle is used with decomposition 1 above to add sequences as a "new" data type.

Event: Add the shell cons-seq-first, with bottom object function symbol empty-seq, with recognizer function symbol seq-p, and 2 accessors: first, with type restriction (none-of) and default value empty-seq; final, with type restriction (one-of seq-p) and default value empty-seq.

; default value

; ( CONS-SEQ-FIRST Obj Seq ) returns < Obj,Seq >.
; ( CONS-SEQ-FIRST Obj Non-Seq ) returns < Obj,EMPTY-SEQUENCE >.

; ( EMPTY-SEQ ) returns the EMPTY-SEQUENCE.

; ( SEQ-P Seq ) returns T.
; ( SEQ-P Non-Seq ) returns F.

; ( FIRST < Obj,Seq > ) returns Obj.
; ( FIRST (EMPTY-SEQ) ) returns (EMPTY-SEQ).
; ( FIRST Non-Seq ) returns (EMPTY-SEQ).

; ( FINAL < Obj,Seq > ) returns Seq.
; ( FINAL (EMPTY-SEQ) ) returns (EMPTY-SEQ).
; ( FINAL Non-Seq ) returns (EMPTY-SEQ).

; Definition:
empty-seq-p(s) = ((s = EMPTY-SEQ) ∨ (∼ seq-p(s)))

Definition:
coerce-seq(s) = if seq-p(s) then s
else EMPTY-SEQUENCE endif
The next three functions implement sequence decomposition 2 above.

(CONS-SEQ-LAST Seq Obj) returns [Seq,Obj].
(CONS-SEQ-LAST Non-Seq Obj) returns [(EMPTY-SEQ),Obj].
Here [(EMPTY-SEQ),Obj] is identified with <Obj,(EMPTY-SEQ)>.

(INITIAL [Seq,Obj]) returns Seq.
(INITIAL (EMPTY-SEQ)) returns (EMPTY-SEQ).
(INITIAL Non-Seq) returns (EMPTY-SEQ).

(LAST [Seq,Obj]) returns Obj.
(LAST (EMPTY-SEQ)) returns (EMPTY-SEQ).
(LAST Non-Seq) returns (EMPTY-SEQ).

**Definition:**

cons-seq-last \( (s, c) \)  
\[= \text{if empty-seq-p} (s) \text{ then cons-seq-first} (c, s) \text{ else cons-seq-first} (\text{first} (s), \text{cons-seq-last} (\text{final} (s), c)) \text{ endif} \]

**Definition:**

initial \( (s) \)  
\[= \text{if empty-seq-p} (s) \text{ then EMPTY-SEQ} \text{ elseif final} (s) = \text{EMPTY-SEQ} \text{ then EMPTY-SEQ} \text{ else cons-seq-first} (\text{first} (s), \text{initial} (\text{final} (s))) \text{ endif} \]

**Definition:**

last \( (s) \)  
\[= \text{if empty-seq-p} (s) \text{ then EMPTY-SEQ} \text{ elseif final} (s) = \text{EMPTY-SEQ} \text{ then first} (s) \text{ else last} (\text{final} (s)) \text{ endif} \]

The next 12 rewrite rules and 1 elimination rule would have been explicitly added as axioms to the data base by the shell principle; if sequence decomposition 2 had been used, in place of decomposition 1, as the basis for the shell which added sequences as a new type.

**Theorem:** initial-cons-seq-last

\[\text{initial} (\text{cons-seq-last} (s, c)) = \text{if seq-p} (s) \text{ then } s \text{ else EMPTY-SEQ endif} \]

**Theorem:** initial-nseq-p

\[\neg \text{seq-p} (s) \rightarrow (\text{initial} (s) = \text{EMPTY-SEQ}) \]
THEOREM: initial-type-restriction
\[ (\neg \text{seq-p} (s)) \rightarrow (\text{cons-seq-last} (s, c) = \text{cons-seq-last} (\text{EMPTY-SEQ}, c)) \]

THEOREM: initial-lessp
\[ (\text{seq-p} (s) \land (s \neq \text{EMPTY-SEQ})) \rightarrow (\text{count} (\text{initial} (s)) < \text{count} (s)) \]

THEOREM: initial-lesseqp
\[ \text{count} (s) \not< \text{count} (\text{initial} (s)) \]

THEOREM: last-cons-seq-last
\[ \text{last} (\text{cons-seq-last} (s, c)) = c \]

THEOREM: last-nseq-p
\[ (\neg \text{seq-p} (s)) \rightarrow (\text{last} (s) = \text{EMPTY-SEQ}) \]

THEOREM: last-lessp
\[ (\text{seq-p} (s) \land (s \neq \text{EMPTY-SEQ})) \rightarrow (\text{count} (\text{last} (s)) < \text{count} (s)) \]

THEOREM: last-lesseqp
\[ \text{count} (s) \not< \text{count} (\text{last} (s)) \]

; The next two lemmas are obvious facts used only as ; hints for the proof of the lemma CONS-SEQ-LAST-EQUAL.

THEOREM: initial-apply-equals
\[ (x = y) \rightarrow (\text{initial} (x) = \text{initial} (y)) \]

THEOREM: last-apply-equals
\[ (x = y) \rightarrow (\text{last} (x) = \text{last} (y)) \]

THEOREM: cons-seq-last-equal
\[ (\text{cons-seq-last} (s1, c1) = \text{cons-seq-last} (s2, c2)) \]
\[ = (\text{if seq-p} (s1) \]
\[ \text{then if seq-p} (s2) \text{ then } s1 = s2 \]
\[ \text{else } s1 = \text{EMPTY-SEQ endif} \]
\[ \text{elseif seq-p} (s2) \text{ then } \text{EMPTY-SEQ} = s2 \]
\[ \text{else } t \text{ endif} \]
\[ \land (c1 = c2)) \]

THEOREM: cons-seq-last-initial-last
\[ (\text{cons-seq-last} (\text{initial} (s), \text{last} (s)) \]
\[ = \text{if seq-p} (s) \land (s \neq \text{EMPTY-SEQ}) \text{ then } s \]
\[ \text{else cons-seq-last} (\text{EMPTY-SEQ}, \text{EMPTY-SEQ}) \text{ endif} \]

THEOREM: initial-last-elim
\[ (\text{seq-p} (s) \land (s \neq \text{EMPTY-SEQ})) \]
\[ \rightarrow (\text{cons-seq-last} (\text{initial} (s), \text{last} (s)) = s) \]
Theorem: count-cons-seq-last
\[ \text{count}(\text{cons-seq-last}(s, c)) = (1 + (\text{if seq-p}(s) \text{ then count}(s) \text{ else 0 endif}) + \text{count}(c))) \]

; The next rewrite rule would, in effect, have been implicitly added
; as an axiom to the data base by the shell principle if sequence
; decomposition 2 had been used, in place of decomposition 1, as the
; basis for the shell which added sequences as a new type.

Theorem: cons-seq-last-not-empty-seq
\[ \text{cons-seq-last}(s, c) \neq \text{empty-seq} \]

;====================================================================
; The next two functions give different versions of REVERSE.

Definition:
\[ \text{reverse1}(s) = \text{if empty-seq-p}(s) \text{ then } s \text{ else cons-seq-last(reverse1(final)(s)), first(s)) endif} \]

Definition:
\[ \text{reverse2}(s) = \text{if empty-seq-p}(s) \text{ then } s \text{ else cons-seq-first(last(s), reverse2(initial(s))) endif} \]

; Use the Theorem Prover to verify the following proposed theorems.
;
; 1. For i=1 and i=2, (EQUAL (REVERSEi (REVERSEi S)) S )
;
; 2. (EQUAL (REVERSE1 S) (REVERSE2 S) )
;====================================================================

; The next four functions give different versions of CONCATENATION.

Definition:
\[ \text{concat1}(s1, s2) = \text{if empty-seq-p}(s1) \text{ then coerse-seq}(s2) \text{ else cons-seq-first(first(s1), concat1(final(s1), s2)) endif} \]
**Definition:**
\[
\text{concat2} (s1, s2) = \begin{cases} 
\text{coerce-seq} (s1) & \text{if empty-seq-p} (s2) \\
\text{concat2} (\text{cons-seq-last} (s1, \text{first} (s2)), \text{final} (s2)) & \text{else}
\end{cases}
\]

**Definition:**
\[
\text{concat3} (s1, s2) = \begin{cases} 
\text{coerce-seq} (s1) & \text{if empty-seq-p} (s2) \\
\text{cons-seq-last} (\text{concat3} (s1, \text{initial} (s2)), \text{last} (s2)) & \text{else}
\end{cases}
\]

**Definition:**
\[
\text{concat4} (s1, s2) = \begin{cases} 
\text{coerce-seq} (s2) & \text{if empty-seq-p} (s1) \\
\text{concat4} (\text{initial} (s1), \text{cons-seq-first} (\text{last} (s1), s2)) & \text{else}
\end{cases}
\]

; Use the Theorem Prover to verify the following proposed theorems.

; 1. For i=1, i=2, i=3, and i=4, CONCATi is associative.

; 2. For i=1 and i=2; and for j=1, j=2, j=3, and j=4;
; ( EQUAL (REVERSEi (CONCATj S1 S2))
; (CONCATj (REVERSEi S2) (REVERSEi S1)) )

; 3. For i and j such that 1 <= i < j <= 4,
; ( EQUAL (CONCATi S1 S2) (CONCATj S1 S2) )
Index

cocerce-seq, 2, 5, 6
concat1, 5
concat2, 6
concat3, 6
concat4, 6
cons-seq-first, 2, 3, 5, 6
cons-seq-last, 3–6
cons-seq-last-equal, 4
cons-seq-last-initial-last, 4
cons-seq-last-not-empty-seq, 5
count-cons-seq-last, 5

empty-seq, 2–5
empty-seq-p, 2, 3, 5, 6

final, 3, 5, 6
first, 3, 5, 6

initial, 3–6
initial-apply-equals, 4
initial-cons-seq-last, 3
initial-last-elem, 4
initial-leqseq, 4
initial-lessp, 4
initial-nseq-p, 3
initial-type-restriction, 4

last, 3–6
last-apply-equals, 4
last-cons-seq-last, 4
last-leqseq, 4
last-lessp, 4
last-nseq-p, 4

reverse1, 5
reverse2, 5

seq-p, 2–5