```
#|

 Copyright (C) 1994 by Alex Bronstein and Carolyn Talcott.  All Rights
 Reserved.

 You may copy and distribute verbatim copies of this Nqthm-1992 event script as
 you receive it, in any medium, including embedding it verbatim in derivative
 works, provided that you conspicuously and appropriately publish on each copy
 a valid copyright notice "Copyright (C) 1994 by Alex Bronstein and Carolyn
 Talcott.  All Rights Reserved."

 NO WARRANTY

 Alex Bronstein and Carolyn Talcott PROVIDE 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 Alex Bronstein or Carolyn Talcott 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.

|#
```

EVENT: Start with the library "mlp" using the compiled version.

```
; sadder.bm: a serial adder (adds the last 2 values of the input)
; This is Paillet example 4, which has a totally fuzzy, and almost
; meaningless SPECIFICATION (verification condition).  Instead, we
; provide our OWN specs, in many different tastes:
; - in terms of stringfuns: correct-1-S
; - in terms of Lastchars (more intuitive):
; - with a "Done" line
; - or predicting "good" times (len x = 0 modulo 2)
;

;;; CIRCUIT in SUGARED form:
```

```
#|
(setq sysd '(sy-SADDER (x)
(Y1 R 'a0 x) ; arbitrary initial value
(Y2 S plus Y1 x)
(Ydone R 1 Y4) ; specific initial value
(Y4 S bnot Ydone)
(Yout S bmux Ydone x Y2)
))

(setq sadder '(  |#
; BM DEFINITIONS and A2 LEMMAS, generated by BMSYSD:
; comb_plus.bm: Plus combinational element.
; U7-DONE

;   no character function definition since BM already knows about Plus..

; Everything below generated by:     (bmcomb 'plus '() '(x y))
```

DEFINITION:
s-plus $(x,\ y)$
$=$ **if** empty (x) **then** E
else a (s-plus (p (x), p (y)), l (x) + l (y)) **endif**

```
;; A2-Begin-S-PLUS
```

THEOREM: a2-empty-s-plus
empty (s-plus $(x,\ y)$) = empty (x)

THEOREM: a2-e-s-plus
(s-plus $(x,\ y)$ = E) = empty (x)

THEOREM: a2-lp-s-plus
len (s-plus $(x,\ y)$) = len (x)

THEOREM: a2-lpe-s-plus
eqlen (s-plus $(x,\ y)$, x)

THEOREM: a2-ic-s-plus
(len (x) = len (y))
$\rightarrow$ (s-plus (i $(c\_x,\ x)$, i $(c\_y,\ y)$) = i $(c\_x + c\_y$, s-plus $(x,\ y))$)

THEOREM: a2-lc-s-plus
($\neg$ empty (x)) $\rightarrow$ (l (s-plus $(x,\ y)$) = (l (x) + l (y)))

THEOREM: a2-pc-s-plus
$\text{p}\,(\text{s-plus}\,(x,\ y)) = \text{s-plus}\,(\text{p}\,(x),\ \text{p}\,(y))$

THEOREM: a2-hc-s-plus
$((\neg\ \text{empty}\,(x)) \wedge (\text{len}\,(x) = \text{len}\,(y)))$
$\rightarrow \quad (\text{h}\,(\text{s-plus}\,(x,\ y)) = (\text{h}\,(x) + \text{h}\,(y)))$

THEOREM: a2-bc-s-plus
$(\text{len}\,(x) = \text{len}\,(y)) \rightarrow (\text{b}\,(\text{s-plus}\,(x,\ y)) = \text{s-plus}\,(\text{b}\,(x),\ \text{b}\,(y)))$

THEOREM: a2-bnc-s-plus
$(\text{len}\,(x) = \text{len}\,(y)) \rightarrow (\text{bn}\,(n,\ \text{s-plus}\,(x,\ y)) = \text{s-plus}\,(\text{bn}\,(n,\ x),\ \text{bn}\,(n,\ y)))$

```
;; A2-End-S-PLUS

; eof:comb_plus.bm

; comb_bnot.bm: Binary Not combinational element
; U7-DONE
```

DEFINITION:
bnot (u)
$=$ **if** $u = \texttt{0}$ **then** `1`
else `0` **endif**

```
; Everything below generated by: (bmcomb 'bnot '() '(x))
```

DEFINITION:
s-bnot (x)
$=$ **if** empty (x) **then** E
else a (s-bnot (p (x)), bnot (l (x))) **endif**

```
;; A2-Begin-S-BNOT
```

THEOREM: a2-empty-s-bnot
$\text{empty}\,(\text{s-bnot}\,(x)) = \text{empty}\,(x)$

THEOREM: a2-e-s-bnot
$(\text{s-bnot}\,(x) = \text{E}) = \text{empty}\,(x)$

THEOREM: a2-lp-s-bnot
$\text{len}\,(\text{s-bnot}\,(x)) = \text{len}\,(x)$

THEOREM: a2-lpe-s-bnot
eqlen (s-bnot (x), x)

THEOREM: a2-ic-s-bnot
s-bnot (i (c_x, x)) = i (bnot (c_x), s-bnot (x))

THEOREM: a2-lc-s-bnot
$(\neg$ empty $(x)) \rightarrow$ (l (s-bnot (x)) = bnot (l (x)))

THEOREM: a2-pc-s-bnot
p (s-bnot (x)) = s-bnot (p (x))

THEOREM: a2-hc-s-bnot
$(\neg$ empty $(x)) \rightarrow$ (h (s-bnot (x)) = bnot (h (x)))

THEOREM: a2-bc-s-bnot
b (s-bnot (x)) = s-bnot (b (x))

THEOREM: a2-bnc-s-bnot
bn $(n$, s-bnot $(x))$ = s-bnot (bn (n, x))

```
;; A2-End-S-BNOT

; eof:comb_bnot.bm

; comb_mux.bm: Mux combinational element, i.e. "if", but with
; U7-DONE
;      control input binary-encoded (i.e. 0=F or 1=T) hardwired, no ref to bobi
```

DEFINITION:
bmux $(u1, u2, u3)$
= **if** $u1$ = `1` **then** $u2$
else $u3$ **endif**

```
; everything below generated by: (bmcomb 'bmux '() '(x1 x2 x3))
; with the EXCEPTIONS/HAND-MODIFICATIONS given below.
```

DEFINITION:
s-bmux $(x1, x2, x3)$
= **if** empty $(x1)$ **then** E
else a (s-bmux (p $(x1)$, p $(x2)$, p $(x3)$), bmux (l $(x1)$, l $(x2)$, l $(x3)$)) **endif**

```
; SBMUX-is-SIF can make things much simpler on occasions:
```

THEOREM: sbmux-is-sif
s-bmux ($x1$, $x2$, $x3$) = s-if (s-equal ($x1$, s-const (1, $x1$)), $x2$, $x3$)

EVENT: Disable sbmux-is-sif.

```
; We take advantage of SBMUX-is-SIF for all inductive proofs. To do so we
; HAND-MODIFY the code generated by Sugar to replace all the hints by:
;   - A2-EMPTY, A2-PC replace hint with: ((enable sbmux-is-sif))
;   - A2-LP, A2-IC, A2-HC, A2-BC: ((enable sbmux-is-sif) (disable len))
;   - A2-BNC: ((enable sbmux-is-sif) (disable bn len))

;; A2-Begin-S-BMUX
```

THEOREM: a2-empty-s-bmux
empty (s-bmux ($x1$, $x2$, $x3$)) = empty ($x1$)

THEOREM: a2-e-s-bmux
(s-bmux ($x1$, $x2$, $x3$) = E) = empty ($x1$)

THEOREM: a2-lp-s-bmux
len (s-bmux ($x1$, $x2$, $x3$)) = len ($x1$)

THEOREM: a2-lpe-s-bmux
eqlen (s-bmux ($x1$, $x2$, $x3$), $x1$)

THEOREM: a2-ic-s-bmux
((len ($x1$) = len ($x2$)) $\land$ (len ($x2$) = len ($x3$)))
$\rightarrow$ (s-bmux (i (c_x1, $x1$), i (c_x2, $x2$), i (c_x3, $x3$))
= i (bmux (c_x1, c_x2, c_x3), s-bmux ($x1$, $x2$, $x3$)))

THEOREM: a2-lc-s-bmux
($\neg$ empty ($x1$)) $\rightarrow$ (l (s-bmux ($x1$, $x2$, $x3$)) = bmux (l ($x1$), l ($x2$), l ($x3$)))

THEOREM: a2-pc-s-bmux
p (s-bmux ($x1$, $x2$, $x3$)) = s-bmux (p ($x1$), p ($x2$), p ($x3$))

THEOREM: a2-hc-s-bmux
(($\neg$ empty ($x1$)) $\land$ ((len ($x1$) = len ($x2$)) $\land$ (len ($x2$) = len ($x3$))))
$\rightarrow$ (h (s-bmux ($x1$, $x2$, $x3$)) = bmux (h ($x1$), h ($x2$), h ($x3$)))

```
;old: ((DISABLE BMUX S-BMUX) (ENABLE H LEN) (INDUCT (S-BMUX X1 X2 X3)))
```

THEOREM: a2-bc-s-bmux
((len ($x1$) = len ($x2$)) $\land$ (len ($x2$) = len ($x3$)))
$\rightarrow$ (b (s-bmux ($x1$, $x2$, $x3$)) = s-bmux (b ($x1$), b ($x2$), b ($x3$)))

```
;old: ((DISABLE BMUX) (ENABLE B LEN) (INDUCT (S-BMUX X1 X2 X3)))
```

THEOREM: a2-bnc-s-bmux
$((\text{len}(x1) = \text{len}(x2)) \wedge (\text{len}(x2) = \text{len}(x3)))$
$\rightarrow \quad (\text{bn}(n, \text{s-bmux}(x1, x2, x3)) = \text{s-bmux}(\text{bn}(n, x1), \text{bn}(n, x2), \text{bn}(n, x3)))$

```
;old: ((DISABLE BMUX S-BMUX))

;; A2-End-S-BMUX

; eof:comb_bmux.bm
```

DEFINITION:
topor-sy-sadder (ln)
= **if** ln = `'y1` **then** `0`
elseif ln = `'y2` **then** `1`
elseif ln = `'ydone` **then** `0`
elseif ln = `'y4` **then** `1`
elseif ln = `'yout` **then** `2`
else `0` **endif**

DEFINITION:
sy-sadder (ln, x)
= **if** ln = `'y1`
then if empty (x) **then** E
else i (`'a0`, p (x)) **endif**
elseif ln = `'y2` **then** s-plus (sy-sadder (`'y1`, x), x)
elseif ln = `'ydone`
then if empty (x) **then** E
else i (`1`, sy-sadder (`'y4`, p (x))) **endif**
elseif ln = `'y4` **then** s-bnot (sy-sadder (`'ydone`, x))
elseif ln = `'yout`
then s-bmux (sy-sadder (`'ydone`, x), x, sy-sadder (`'y2`, x))
else sfix (x) **endif**

```
;; A2-Begin-SY-SADDER
```

THEOREM: a2-empty-sy-sadder
empty (sy-sadder (ln, x)) = empty (x)

THEOREM: a2-e-sy-sadder
(sy-sadder (ln, x) = E) = empty (x)

THEOREM: a2-lp-sy-sadder
$\mathrm{len}\,(\text{sy-sadder}\,(ln,\ x)) = \mathrm{len}\,(x)$

THEOREM: a2-lpe-sy-sadder
$\mathrm{eqlen}\,(\text{sy-sadder}\,(ln,\ x),\ x)$

THEOREM: a2-pc-sy-sadder
$\mathrm{p}\,(\text{sy-sadder}\,(ln,\ x)) = \text{sy-sadder}\,(ln,\ \mathrm{p}\,(x))$

```
;; A2-End-SY-SADDER

;;; Circuit CORRECTNESS /Paillet:

;; we can do the actual Deroulement, which is a very WEAK spec
;; indeed..
```

THEOREM: sadder-paillet
$(\mathrm{len}\,(x) = \mathtt{2}) \rightarrow (\mathrm{l}\,(\text{sy-sadder}\,(\mathtt{'yout},\ x)) = (\mathrm{l}\,(\mathrm{p}\,(x)) + \mathrm{l}\,(x)))$

```
;; or we can prove the natural spec:
```

THEOREM: sadder-correct-1
$$
\begin{array}{l}
(\neg\ \mathrm{empty}\,(\mathrm{p}\,(x)))\\
\rightarrow \quad (\mathrm{l}\,(\text{sy-sadder}\,(\mathtt{'yout},\ x))\\
\qquad = \quad \textbf{if}\ \mathrm{l}\,(\text{sy-sadder}\,(\mathtt{'ydone},\ x)) = \mathtt{1}\ \ \textbf{then}\ \mathrm{l}(x)\\
\qquad\qquad \textbf{else}\ \mathrm{l}\,(\mathrm{p}\,(x)) + \mathrm{l}\,(x)\ \textbf{endif})
\end{array}
$$

THEOREM: sadder-but-wiser
`'horning-said-that`

```
;; There are 2 lessons from the above lemma:
;; 1) if you're reading this in the thesis, congratulations
;;    for reading so far.
;; 2) if someone claims he proved some formula mechanically,
;;    make sure the formula means something...

;; we could look at it in terms of STRING-FUNS:
```

DEFINITION:
$$
\begin{array}{l}
\text{sadder-spec-s}\,(x)\\
= \quad \text{s-if}\,(\text{s-equal}\,(\text{sy-sadder}\,(\mathtt{'ydone},\ x),\ \text{s-const}\,(\mathtt{1},\ \text{sy-sadder}\,(\mathtt{'ydone},\ x))),\\
\qquad\qquad x,\\
\qquad\qquad \text{s-plus}\,(\mathrm{p}\,(x),\ x))
\end{array}
$$

```
; one "trick" which makes this weird (non-MLP looking) spec work:
```

THEOREM: s-plus-one-off
$(\neg\ \text{empty}\,(x)) \rightarrow (\text{s-plus}\,(\text{i}\,(u,\ \text{p}\,(x)),\ x) = \text{i}\,(u + \text{h}\,(x),\ \text{s-plus}\,(\text{p}\,(x),\ x)))$

```
; and the second "trick" is that the off-by-one sum, and the
; toggling contribute to working things out:
```

THEOREM: sadder-correct-1-s-key

$$
\begin{aligned}
&\text{s-if}\,(\text{s-equal}\,(\text{sy-sadder}\,(\texttt{'ydone},\ x),\ \text{s-const}\,(\texttt{1},\ \text{sy-sadder}\,(\texttt{'ydone},\ x))),\\
&\quad x,\\
&\quad \text{i}\,(u,\ \text{s-plus}\,(\text{p}\,(x),\ x)))\\
= \quad &\text{s-if}\,(\text{s-equal}\,(\text{sy-sadder}\,(\texttt{'ydone},\ x),\ \text{s-const}\,(\texttt{1},\ \text{sy-sadder}\,(\texttt{'ydone},\ x))),\\
&\quad x,\\
&\quad \text{s-plus}\,(\text{p}\,(x),\ x))
\end{aligned}
$$

THEOREM: sadder-correct-1-s
$\text{sy-sadder}\,(\texttt{'yout},\ x) = \text{sadder-spec-s}\,(x)$

```
;; Going back to looking in terms of Last-chars, we would most
;; likely only specify (more partially):
```

THEOREM: sadder-correct-2

$$
\begin{aligned}
&((\neg\ \text{empty}\,(x)) \land (\neg\ \text{empty}\,(\text{p}\,(x))) \land (\text{l}\,(\text{sy-sadder}\,(\texttt{'ydone},\ x)) = \texttt{0}))\\
\rightarrow \quad &(\text{l}\,(\text{sy-sadder}\,(\texttt{'yout},\ x)) = (\text{l}\,(\text{p}\,(x)) + \text{l}\,(x)))
\end{aligned}
$$

```
; Note: sadder-correct-2 is proved immediately from correct-1 or
; correct-1-S, or can be obtained from scratch with the same hints
; as correct-1.
; The fact that such a partial spec is BM-provable probably depends
; on the fact that it requires NO induction.  In cases with
; induction, too weak a spec would probably not go through.  In
; other words, we might have to specify a second property which
; also specifies what happens in the "previous" or "uninteresting"
; (Done not raised) cases.

; Another way to look at the spec is to PREDICT the timing of the
; output, as opposed to just looking at the Done (Ydone) line.
; This of course will not always be possible. The key to the
; prediction is:
```

THEOREM: sadder-correct-3-ydone
$(\neg \text{ empty}(x))$
$\rightarrow \quad (\text{l}(\text{sy-sadder}(\texttt{'ydone}, x))$
$\quad = \quad \textbf{if } (\text{len}(x) \textbf{ mod } 2) = \texttt{0} \textbf{ then } \texttt{0}$
$\quad\quad \textbf{else } \texttt{1} \textbf{ endif})$

```
; or with a bit more work we can prove the 2-tick rhythm:
```

THEOREM: sadder-correct-3
$((\neg \text{ empty}(x)) \wedge ((\text{len}(x) \textbf{ mod } 2) = \texttt{0}))$
$\rightarrow \quad (\text{l}(\text{sy-sadder}(\texttt{'yout}, x)) = (\text{l}(\text{p}(x)) + \text{l}(x)))$

```
; Note: removing the hyp: (not (empty (P x))) is not harmful,
; but it just forces BM into a couple of cases and 1 elimination
; before it gets to the real meat.

; eof: sadder.bm
;))
```

Index