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)) → (s-plus (i (c_x, x), i (c_y, y)) = i (c_x + c_y, s-plus (x, y)))

THEOREM: a2-lc-s-plus
(¬ empty (x)) → (l (s-plus (x, y)) = (l (x) + l (y)))

2
THEOREM: a2-pc-s-plus
\[ p \left( s\text{-}plus \left( x, \ y \right) \right) = s\text{-}plus \left( p \left( x \right), \ p \left( y \right) \right) \]

THEOREM: a2-hc-s-plus
\[ \left( \neg \ \text{empty} \left( x \right) \right) \land \left( \text{len} \left( x \right) = \text{len} \left( y \right) \right) \]
\[ \rightarrow \ \left( h \left( s\text{-}plus \left( x, \ y \right) \right) = h \left( x \right) + h \left( y \right) \right) \]

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

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

;; A2-End-S-PLUS

; eof:comb_plus.bm

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

DEFINITION:
bnot \( u \) =
\[ \text{if} \ u = 0 \ \text{then} \ 1 \ \text{else} \ 0 \ \text{endif} \]

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

DEFINITION:
s-bnot \( x \) =
\[ \text{if} \ \text{empty} \left( x \right) \ \text{then} \ \text{E} \ \text{else} \ \text{a} \left( \text{s-bnot} \left( p \left( x \right) \right), \ \text{bnot} \left( l \left( x \right) \right) \right) \ \text{endif} \]

;; A2-Begin-S-BNOT

THEOREM: a2-empty-s-bnot
\[ \text{empty} \left( \text{s-bnot} \left( x \right) \right) = \text{empty} \left( x \right) \]

THEOREM: a2-e-s-bnot
\[ \text{s-bnot} \left( x \right) = \text{E} \right) = \text{empty} \left( x \right) \]

THEOREM: a2-lp-s-bnot
\[ \text{len} \left( \text{s-bnot} \left( x \right) \right) = \text{len} \left( x \right) \]
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
(¬ empty (x)) → (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
b(s-bnot (x)) = s-bnot (b(x))

THEOREM: a2-bc-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
\[ \text{s-bmux}(x_1, x_2, x_3) = \text{s-if}(\text{s-equal}(x_1, \text{s-const}(1, x_1)), x_2, x_3) \]

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
\[ \text{empty}(\text{s-bmux}(x_1, x_2, x_3)) = \text{empty}(x_1) \]

Theorem: a2-e-s-bmux
\[ (\text{s-bmux}(x_1, x_2, x_3) = e) = \text{empty}(x_1) \]

Theorem: a2-lp-s-bmux
\[ \text{len}(\text{s-bmux}(x_1, x_2, x_3)) = \text{len}(x_1) \]

Theorem: a2-lpe-s-bmux
\[ \text{eqlen}(\text{s-bmux}(x_1, x_2, x_3), x_1) \]

Theorem: a2-ic-s-bmux
\[ ((\text{len}(x_1) = \text{len}(x_2)) \land (\text{len}(x_2) = \text{len}(x_3))) \rightarrow (\text{s-bmux}(i(c_{x_1}, x_1), i(c_{x_2}, x_2), i(c_{x_3}, x_3)) = i(\text{bmux}(c_{x_1}, c_{x_2}, c_{x_3}), \text{s-bmux}(x_1, x_2, x_3))) \]

Theorem: a2-lc-s-bmux
\[ (\neg \text{empty}(x_1)) \rightarrow (\text{l}(\text{s-bmux}(x_1, x_2, x_3)) = \text{bmux}(\text{l}(x_1), \text{l}(x_2), \text{l}(x_3))) \]

Theorem: a2-ic-s-bmux
\[ \text{p}(\text{s-bmux}(x_1, x_2, x_3)) = \text{s-bmux}(\text{p}(x_1), \text{p}(x_2), \text{p}(x_3)) \]

Theorem: a2-hc-s-bmux
\[ ((\neg \text{empty}(x_1)) \land ((\text{len}(x_1) = \text{len}(x_2)) \land (\text{len}(x_2) = \text{len}(x_3)))) \rightarrow (\text{h}(\text{s-bmux}(x_1, x_2, x_3)) = \text{bmux}(\text{h}(x_1), \text{h}(x_2), \text{h}(x_3))) \]

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

Theorem: a2-bc-s-bmux
\[ ((\text{len}(x_1) = \text{len}(x_2)) \land (\text{len}(x_2) = \text{len}(x_3))) \rightarrow (\text{b}(\text{s-bmux}(x_1, x_2, x_3)) = \text{s-bmux}(\text{b}(x_1), \text{b}(x_2), \text{b}(x_3))) \]
Theorem: $a_2$-bnc-s-bmux

$((\text{len} \ (x_1) = \text{len} \ (x_2)) \land (\text{len} \ (x_2) = \text{len} \ (x_3)))$

$\rightarrow \ (\text{bn} \ (n, \ s_{-}\text{bmux} \ (x_1, x_2, x_3)) = s_{-}\text{bmux} \ (\text{bn} \ (n, x_1), \ \text{bn} \ (n, x_2), \ \text{bn} \ (n, x_3)))$

Definition: $\text{topor-sy-sadder} \ (ln)$

= if $ln = 'y1$ then 0
else if $ln = 'y2$ then 1
else if $ln = 'ydone$ then 0
else if $ln = 'y4$ then 1
else if $ln = 'yout$ then 2
else 0 endif

Definition: $\text{sy-sadder} \ (ln, x)$

= if $ln = 'y1$
then if empty $(x)$ then e
else i('a0, p(x)) endif
else if $ln = 'y2$ then s-plus (sy-sadder ('y1, x), x)
else if $ln = 'ydone$
then if empty $(x)$ then e
else i(1, sy-sadder ('y4, p(x))) endif
else if $ln = 'y4$ then s-bnot (sy-sadder ('ydone, x))
else if $ln = 'yout$
then s-bmux (sy-sadder ('ydone, x), x, sy-sadder ('y2, x))
else sfix $(x)$ endif

Theorem: $a_2$-empty-sy-sadder

empty (sy-sadder $(ln, x)$) = empty $(x)$

Theorem: $a_2$-e-sy-sadder

(sy-sadder $(ln, x) = E) = empty \ (x)$
**Theorem**: a2-lp-sy-sadder  
len(sy-sadder(ln, x)) = len(x)

**Theorem**: a2-lpe-sy-sadder  
eqlen(sy-sadder(ln, x), x)

**Theorem**: a2-pc-sy-sadder  
p(sy-sadder(ln, x)) = sy-sadder(ln, 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  
(len(x) = 2) → (l(sy-sadder('yout, x)) = (l(p(x)) + l(x)))

;; or we can prove the natural spec:

**Theorem**: sadder-correct-1  
(¬empty(p(x)))  
→ (l(sy-sadder('yout, x))  
    = if l(sy-sadder('ydone, x)) = 1 then l(x)  
    else l(p(x)) + l(x) endif)

**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**:  
sadder-spec-s(x)  
= s-if(s-equal(sy-sadder('ydone, x), s-const(1, sy-sadder('ydone, x))), x,  
s-plus(p(x), x))
; one "trick" which makes this weird (non-MLP looking) spec work:

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

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

**Theorem:** saddler-correct-1-s-key
\[
\begin{align*}
\text{s-if}(\text{s-equal}(\text{sy-sadder('ydone, x)}, \text{s-const}(1, \text{sy-sadder('ydone, x)))), \\
\quad x, \\
\quad i(u, \text{s-plus}(p(x), x))) \\
= \text{s-if}(\text{s-equal}(\text{sy-sadder('ydone, x)}, \text{s-const}(1, \text{sy-sadder('ydone, x)))), \\
\quad x, \\
\quad \text{s-plus}(p(x), x))
\end{align*}
\]

**Theorem:** saddler-correct-1-s
\[
s\text{-sadder('yout, x)} = \text{sadder-spec-s}(x)
\]

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

**Theorem:** saddler-correct-2
\[
((\neg \text{empty}(x)) \land (\neg \text{empty}(p(x))) \land (l(\text{sy-sadder('ydone, x)}) = 0)) \\
\rightarrow (l(\text{sy-sadder('yout, x)}) = (l(p(x)) + l(x)))
\]

; Note: saddler-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 EM-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-ymone  
(\neg \text{empty}(x)) 
\rightarrow (l(\text{sy-sadder}('ymone, x))) 

= \text{if } (\text{len}(x) \mod 2) = 0 \text{ then } 0 
\quad \text{else } 1 \text{ endif} 

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

THEOREM: sadder-correct-3  
((\neg \text{empty}(x)) \land ((\text{len}(x) \mod 2) = 0)) 
\rightarrow (l(\text{sy-sadder}('ymout, x)) = (l(\text{p}(x)) + 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

a, 2–4
a2-bc-s-bmux, 5
a2-bc-s-bnot, 4
a2-bc-s-plus, 3
a2-bnc-s-bmux, 6
a2-bnc-s-bnot, 4
a2-bnc-s-plus, 3
a2-e-s-bmux, 5
a2-e-s-bnot, 3
a2-e-s-plus, 2
a2-e-sy-sadder, 6
a2-empty-s-bmux, 5
a2-empty-s-bnot, 3
a2-empty-s-plus, 2
a2-empty-sy-sadder, 6
a2-hc-s-bmux, 5
a2-hc-s-bnot, 4
a2-hc-s-plus, 3
a2-ic-s-bmux, 5
a2-ic-s-bnot, 4
a2-ic-s-plus, 2
a2-le-s-bmux, 5
a2-le-s-bnot, 4
a2-le-s-plus, 2
a2-lp-s-bmux, 5
a2-lp-s-bnot, 3
a2-lp-s-plus, 2
a2-lp-sy-sadder, 7
a2-lpe-s-bmux, 5
a2-lpe-s-bnot, 4
a2-lpe-s-plus, 2
a2-lpe-sy-sadder, 7
a2-pc-s-bmux, 5
a2-pc-s-bnot, 4
a2-pc-s-plus, 3
a2-pc-sy-sadder, 7

b, 3–5
bmux, 4, 5
bn, 3, 4, 6
bnot, 3, 4
e, 2–6
empty, 2–9
eqlen, 2, 4, 5, 7
h, 3–5, 8
i, 2, 4–6, 8
1, 2–5, 7–9
len, 2, 3, 5–7, 9
p, 2–9
s-bmux, 4–6
s-bnot, 3, 4, 6
s-const, 5, 7, 8
s-equal, 5, 7, 8
s-if, 5, 7, 8
s-plus, 2, 3, 6–8
s-plus-one-off, 8
sadder-but-wiser, 7
sadder-correct-1, 7
sadder-correct-1-s, 8
sadder-correct-1-s-key, 8
sadder-correct-2, 8
sadder-correct-3, 9
sadder-correct-3-ydone, 9
sadder-paillet, 7
sadder-spec-s, 7, 8
sbmux-is-sif, 5
sfix, 6
sy-sadder, 6–9
topor-sy-sadder, 6