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

; counterR.bm: a resetable clock counter, i.e. Paillet example 1.
;
; This is our first circuit w/ 2 inputs, yet to my amazement,
; extra EQLEN hypotheses were NEVER required, neither in the A2
; lemmas, nor in the actual specifications (Paillet, or mine)! Of
; course, when they became required (in Paillet #7) and put in
; Sugar, they were installed here.
;
; NOTE (w/ EMPTY enabled) : Attempts to speed up A2-PC failed
; miserably: disabling the combinationals fails because of the
; uneven cases, and the fact that BM can’t derive (empty y) from
A problem we have had for a long time, and which will probably persist until we solve EQLEN.

Giving the induction hint ahead of time made things worse, as usual. Twiddling with STR-P-I2 improved nothing.

;;; CIRCUIT in SUGARED form:
#|
(setq sysd '(sy-COUNT (Xc Xe)
(Ymux S Mux Xe Yinc)
(Yreg R 0 YMux)
(Yinc S Incn Yreg)
))

(setq counterR '('|#
; BM DEFINITIONS and A2 LEMMAS, generated by BMSYSD:
; comb_mux.bm: Mux combinational element, i.e. "if".
; U7-DONE

DEFINITION:
mux(u1, u2, u3)
= if u1 then u2
  else u3 endif

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

DEFINITION:
s-mux(x1, x2, x3)
= if empty(x1) then e
  else a(s-mux(p(x1), p(x2), p(x3)), mux(l(x1), l(x2), l(x3))) endif

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

THEOREM: smux-is-sif
s-mux(x1, x2, x3) = s-if(x1, x2, x3)

EVENT: Disable smux-is-sif.

; We take advantage of SMUX-is-SIF for all inductive proofs. To do so we
Theorem: a2-empty-s-mux
empty (s-mux (x1, x2, x3)) = empty (x1)

Theorem: a2-e-s-mux
(s-mux (x1, x2, x3) = e) = empty (x1)

Theorem: a2-lp-s-mux
len (s-mux (x1, x2, x3)) = len (x1)

Theorem: a2-lpe-s-mux
eqlen (s-mux (x1, x2, x3), x1)

Theorem: a2-ic-s-mux
((len (x1) = len (x2)) ∧ (len (x2) = len (x3)))
   → (i (c x1, x1), i (c x2, x2), i (c x3, x3))
   = (mux (c x1, c x2, c x3), s-mux (x1, x2, x3))

Theorem: a2-lc-s-mux
(¬ empty (x1)) → (l (s-mux (x1, x2, x3)) = (l (x1), l (x2), l (x3)))

Theorem: a2-pc-s-mux
p (s-mux (x1, x2, x3)) = s-mux (p (x1), p (x2), p (x3))

Theorem: a2-hc-s-mux
((¬ empty (x1)) ∧ ((len (x1) = len (x2)) ∧ (len (x2) = len (x3))))
   → (h (s-mux (x1, x2, x3)) = (h (x1), h (x2), h (x3)))

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

Theorem: a2-bc-s-mux
((len (x1) = len (x2)) ∧ (len (x2) = len (x3))
   → (b (s-mux (x1, x2, x3)) = s-mux (b (x1), b (x2), b (x3)))

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

Theorem: a2-bnc-s-mux
((len (x1) = len (x2)) ∧ (len (x2) = len (x3))
   → (bn (n, s-mux (x1, x2, x3)) = s-mux (bn (n, x1), bn (n, x2), bn (n, x3)))
EVENT: Introduce the function symbol \( n \) of 0 arguments.

; we may want to add an axiom saying that it’s a number, not needed so far..

**DEFINITION:**
\[
\text{incn}(u) = \begin{cases} 
  0 & \text{if } u = N \\
  1 + u & \text{else}
\end{cases}
\]

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

**DEFINITION:**
\[
\text{s-incn}(x) = \begin{cases} 
  e & \text{if empty}(x) \\
  \text{a}(\text{s-incn}(p(x)), \text{incn}(l(x))) & \text{else}
\end{cases}
\]

;; A2-Begin-S-INCN

**THEOREM:** a2-empty-s-incn
\[
\text{empty}(\text{s-incn}(x)) = \text{empty}(x)
\]

**THEOREM:** a2-e-s-incn
\[
(\text{s-incn}(x) = e) = \text{empty}(x)
\]

**THEOREM:** a2-lp-s-incn
\[
\text{len}(\text{s-incn}(x)) = \text{len}(x)
\]
THEOREM: a2-lpe-s-incn
eqlen (s-incn (x), x)

THEOREM: a2-ic-s-incn
s-incn (i (\text{c}_x, x)) = i (\text{incn} (\text{c}_x), s-incn (x))

THEOREM: a2-lc-s-incn
(\neg \text{empty} (x)) \to (l (s-incn (x)) = \text{incn} (l (x)))

THEOREM: a2-pe-s-incn
p (s-incn (x)) = s-incn (p (x))

THEOREM: a2-hc-s-incn
(\neg \text{empty} (x)) \to (h (s-incn (x)) = \text{incn} (h (x)))

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

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

;; A2-End-S-INCN

; eof:comb_incn.bm

DEFINITION:
topor-sy-count (ln) =
  if \text{ln} = \text{ymux} then 2
  elseif \text{ln} = \text{yreg} then 0
  elseif \text{ln} = \text{yinc} then 1
  else 0 endif

DEFINITION:
sy-count (ln, xc, xe) =
  if \text{ln} = \text{ymux} then \text{s-mux} (xc, xe, sy-count (\text{yinc}, xc, xe))
  elseif \text{ln} = \text{yreg} then
    if \text{empty} (xe) then E
    else i (0, sy-count (\text{ymux}, p (xc), p (xe))) endif
  elseif \text{ln} = \text{yinc} then s-incn (sy-count (\text{yreg}, xc, xe))
  else \text{sfix} (xc) endif

;; A2-Begin-SY-COUNT
Theorem: a2-empty-sy-count
(len (xc) = len (xe)) → (empty (sy-count (ln, xc, xe)) = empty (xc))

Theorem: a2-e-sy-count
(len (xc) = len (xe)) → ((sy-count (ln, xc, xe) = e) = empty (xc))

Theorem: a2-lp-sy-count
(len (xc) = len (xe)) → (len (sy-count (ln, xc, xe)) = len (xc))

Theorem: a2-lpe-sy-count
(len (xc) = len (xe)) → eqlen (sy-count (ln, xc, xe), xc)

Theorem: a2-pc-sy-count
(len (xc) = len (xe)) → (p (sy-count (ln, xc, xe))) = sy-count (ln, p (xc), p (xe)))

;; A2-End-SY-COUNT

;; Circuit CORRECTNESS /Paillet:

; Note that as originally stated in Paillet, with the P outside of
; sy-count makes for a looping (unfolding) which would have to be
; proved kludgeily, and would be useless. The following rule can
; be used as a rewrite.

Theorem: count-paillet-correct
((¬ empty (xc)) ∧ (∧ empty (xe)))
→ (sy-count (ʼyreg, xc, xe)
    = i (0, s-if (p (xc), p (xe), s-incn (sy-count (ʼyreg, p (xc), p (xe))))))

; The "last-char" reading of the spec yields:
; NOTE: we can prove it by repeating the same hint and disabling
; CORRECT, i.e. independently. Trying to use CORRECT fails
; miserably because it also triggers on:
; (sy-count ʼYreg (P Xc) (P Xe)). Note also that we need the
; EQ-LEN hyp because we need A2-EMPTY-SY-COUNT.

Theorem: count-paillet-correct-l
((¬ empty (p (xc))) ∧ (∧ empty (p (xe))) ∧ len (xc) = len (xe))
→ (l (sy-count (ʼyreg, xc, xe))
    = if l (p (xc)) then l (p (xe))
       else incn l (sy-count (ʼyreg, p (xc), p (xe)))) endif)

; eof: counterR.bm
;))
Index

a, 2, 4
a2-bc-s-incn, 5
a2-bc-s-mux, 3
a2-bnc-s-incn, 5
a2-bnc-s-mux, 3
a2-e-s-incn, 4
a2-e-s-mux, 3
a2-e-sy-count, 6
a2-empty-s-incn, 4
a2-empty-s-mux, 3
a2-empty-sy-count, 6
a2-hc-s-incn, 5
a2-hc-s-mux, 3
a2-ic-s-incn, 5
a2-ic-s-mux, 3
a2-le-s-incn, 5
a2-le-s-mux, 3
a2-lp-s-incn, 4
a2-lp-s-mux, 3
a2-lp-sy-count, 6
a2-lpe-s-incn, 5
a2-lpe-s-mux, 3
a2-lpe-sy-count, 6
a2-pc-s-incn, 5
a2-pc-s-mux, 3
a2-pc-sy-count, 6

b, 3, 5
bn, 3, 5

count-paillet-correct, 6
count-paillet-correct-l, 6

e, 2–6
eempty, 2–6
eqlen, 3, 5, 6

h, 3, 5

i, 3, 5, 6
incn, 4–6

1, 2–6
len, 3, 4, 6

mux, 2, 3

n, 4

p, 2–6

s-if, 2, 6
s-incn, 4–6
s-mux, 2, 3, 5
sfix, 5
smux-is-sif, 2
sy-count, 5, 6

topor-sy-count, 5