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

; counter.bm: a simple clock counter, x is the clock.
;   similar to acc when we take our output from the combinational elt.
;
; RESULTS:
; if we look at the combinational output (Ycomb)
;   with initial value of reg: 0 , OK can prove Ycomb = len . [log: counter]
; with initial value of reg: 1 , fails (expectedly) in COUNT-CORRECT-AX
;   in case p x = e, [log: counter2]
; if we look at Yreg (i.e. register output) then:
; with initial value of reg: 0 , fails (expectedly) in COUNT-CORRECT-AX
;   in case p x = e, [log: counter4]
with initial value of reg: 1, OK can prove Yreg = len. [log: counter3]

ANALYSIS:

The issue is a confusion of the intent of the SPEC: if we are counting the number of "pulses" then indeed we can look at the Reg-output, and initialize it with zero, because pulse = len(clk) - 1 = len(any input string) - 1.

In other words, the issue is with NUMER-COUNT x = LEN x. If we want to count the number of pulses, then we should have number-count x = len x - 1, and then initialize w/ 0 and look at Reg-out. Check: OK!

CONCLUSION:

The model (essentially Mealy) is fine even for looking at Reg-outs, but be careful of translating specs involving number of "clock ticks". My tics are really "periods", and the engineer's tick are "pulses" of which there are always one less when based on an operational semantics looking at things "at the end of clock periods".

COMPARISON w/ PAILLET, and PAILLET inferred spec:

;;; DEFINITION OF CIRCUIT:

(setq sysd '(sy-count (x)
(Ycomb S Inc Yreg)
(Yreg R 0 Ycomb)
))

(setq counter '( |#
; BM DEFINITIONS and A2 LEMMAS, generated by BMSYSD:
; comb_inc.bm: INCrement combinational element
; U7-DONE

DEFINITION: inc(u) = (1 + u)

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

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

;; A2-Begin-S-INC

THEOREM: a2-empty-s-inc
empty(s-inc(x)) = empty(x)
Theorem: a2-e-s-inc
\( (s\text{-inc}(x) = E) = \text{empty}(x) \)

Theorem: a2-lp-s-inc
\( \text{len}(s\text{-inc}(x)) = \text{len}(x) \)

Theorem: a2-lpe-s-inc
\( \text{eqlen}(s\text{-inc}(x), x) \)

Theorem: a2-ic-s-inc
\( s\text{-inc}(i(c_x, x)) = i(\text{inc}(c_x), s\text{-inc}(x)) \)

Theorem: a2-lc-s-inc
\( (\neg \text{empty}(x)) \rightarrow (l(s\text{-inc}(x)) = \text{inc}(l(x))) \)

Theorem: a2-pc-s-inc
\( p(s\text{-inc}(x)) = s\text{-inc}(p(x)) \)

Theorem: a2-hc-s-inc
\( (\neg \text{empty}(x)) \rightarrow (h(s\text{-inc}(x)) = \text{inc}(h(x))) \)

Theorem: a2-bc-s-inc
\( b(s\text{-inc}(x)) = s\text{-inc}(b(x)) \)

Theorem: a2-bnc-s-inc
\( \text{bn}(n, s\text{-inc}(x)) = s\text{-inc}(\text{bn}(n, x)) \)

;; A2-End-S-INC

; eof:comb_inc.bm

Definition:
\( \text{topor-sy-count}(ln) = \)
\( = \) if \( ln = 'ycomb \) then 1
\( \) elseif \( ln = 'yreg \) then 0
\( \) else 0 endif

Definition:
\( \text{sy-count}(ln, x) = \)
\( = \) if \( ln = 'ycomb \) then \( s\text{-inc}(\text{sy-count}('yreg, x)) \)
\( \) elseif \( ln = 'yreg \)
\( \) then if \( \text{empty}(x) \) then E
\( \) else \( i(0, \text{sy-count}('ycomb, p(x))) \) endif
\( \) else \( \text{sfix}(x) \) endif

3
THEOREM: a2-empty-sy-count
empty (sy-count (ln, x)) = empty (x)

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

THEOREM: a2-lp-sy-count
len (sy-count (ln, x)) = len (x)

THEOREM: a2-lpe-sy-count
eqlen (sy-count (ln, x), x)

THEOREM: a2-pc-sy-count
p (sy-count (ln, x)) = sy-count (ln, p (x))

;; SPEC definition:
;; FIRST (misunderstood) spec:
;; note: written when looking at the combinational output..
;
;;(defn numer-count (x)
;;  (if (empty x)
;;    0
;;    (inc (numer-count (p x)) )))
;;
;; intent verification: should have numer-count = len; prove but don’t use.
;;(prove-lemma numer-count-len ()
;;(equal (numer-count x) (len x))
;;)

DEFINITION:
numer-count (x) = if empty (x) then 0 else len (x) – 1 endif

;a cheap way: (len (p x))

; this is the standard extension from last-char-fun to MLP-string-fun,
; see theta.bm .
**Definition:**

\[
\text{spec-count}(x) = \begin{cases} 
\text{if empty}(x) & \text{then } E \\
\text{else } \text{a}(\text{spec-count}(p(x)), \text{numer-count}(x)) \end{cases}
\]

; Paillet’s spec do not define an additional function, just a relation
; that must be verified by the circuit output. See below.

;; Circuit CORRECTNESS:
; THIS is where it matters which line we take as output!

; Count-correct-ax is a "predicative correctness statement", i.e. what we would
; do if we didn’t have functional equality as a specification method, but
; instead used a purely axiomatic approach.

**Theorem:** count-correct-ax
\[
(\neg \text{empty}(x)) \rightarrow (l(\text{sy-count}(\text{yreg}, x)) = \text{numer-count}(x))
\]

; to go to a functional equality once we have the "last" (ax) statement is
; a trivial induction, if we start out with an P-L split which is unnatural
; for BM, so we force it w/ a USE hint of A-p-l-split

**Theorem:** a-p-l-split
\[
(\neg \text{empty}(x)) \rightarrow (\text{sy-count}(\text{yreg}, x) = \text{a}(\text{p}(\text{sy-count}(\text{yreg}, x)), l(\text{sy-count}(\text{yreg}, x))))
\]

**Theorem:** count-correct
\[
\text{sy-count}(\text{yreg}, x) = \text{spec-count}(x)
\]

; PAILLET CORRECTNESS:

**Event:** Disable count-correct.

**Event:** Disable count-correct-ax.

; count-paillet-correct obtained trivially (without recursion) as expected.

**Theorem:** count-paillet-correct
\[
(\neg \text{empty}(x)) \rightarrow (\text{sy-count}(\text{yreg}, x) = i(0, \text{s-inc}(\text{p}(\text{sy-count}(\text{yreg}, x)))))
\]

; my (more intuitive) version of "simple" (no recursion req’d) correctness:
THEOREM: count-correct-simple
((¬ empty (x)) ∧ (¬ empty (p (x))))
→ (l (sy-count ('yreg, x)) = inc (l (p (sy-count ('yreg, x)))))

; eof: counter.bm
;)}
Index

a, 2, 5
a-p-l-split, 5
a2-bc-s-inc, 3
a2-bnc-s-inc, 3
a2-e-s-inc, 3
a2-e-sy-count, 4
a2-empty-s-inc, 2
a2-empty-sy-count, 4
a2-hc-s-inc, 3
a2-ic-s-inc, 3
a2-ic-s-inc, 3
a2-lc-s-inc, 3
a2-lp-s-inc, 3
a2-lp-sy-count, 4
a2-lpe-s-inc, 3
a2-lpe-sy-count, 4
a2-pc-s-inc, 3
a2-pc-sy-count, 4

b, 3
bn, 3

count-correct, 5
count-correct-ax, 5
count-correct-simple, 6
count-paillet-correct, 5

e, 2–5
empty, 2–6
eqlen, 3, 4

h, 3

i, 3, 5
inc, 2, 3, 6

l, 2, 3, 5, 6
len, 3, 4

numer-count, 4, 5

p, 2–6

s-inc, 2, 3, 5

sfix, 3
spec-count, 5
sy-count, 3–6
topor-sy-count, 3