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

; macc.bm
; Circuit is similar to acc, but uses multiplication instead of
; addition, i.e. it’s a multiplying accumulator. It’s expressed in CSXA form,
; which is the form we’ve currently settled on.
; NOTE that it has to be initialized with 1 in order to function right! See
; prod0 for what happens with 0-initialization...
;
;;; DEFINITION OF CIRCUIT:
#|
(setq sysd '(sy-macc (x)
(Ymacc S Times x Ymacc2)
(setq macc ')
;; BM DEFINITIONS and A2 LEMMAS, generated by BMSYSD:
|#
;; comb_times.bm: Times combinational element.
;; U7-DONE

; no character function def since BM already knows about Times..

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

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

;; A2-Begin-S-TIMES

THEOREM: a2-empty-s-times
empty (s-times (x, y)) = empty (x)

THEOREM: a2-e-s-times
(s-times (x, y) = E) = empty (x)

THEOREM: a2-lp-s-times
len (s-times (x, y)) = len (x)

THEOREM: a2-lpe-s-times
eqlen (s-times (x, y), x)

THEOREM: a2-ic-s-times
(len (x) = len (y))
  →  (s-times (i (c_x, x), i (c_y, y)) = i (c_x * c_y, s-times (x, y)))

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

THEOREM: a2-pc-s-times
p (s-times (x, y)) = s-times (p (x), p (y))

THEOREM: a2-hc-s-times
(¬ empty (x)) ∧ (len (x) = len (y))
  →  (h (s-times (x, y)) = (h (x) * h (y)))

2
Theorem: a2-bc-s-times
\[(\text{len}(x) = \text{len}(y)) \rightarrow (b \text{s-times}(x, y)) = \text{s-times}(b \text{x}, b \text{y}))\]

Theorem: a2-bnc-s-times
\[(\text{len}(x) = \text{len}(y)) \rightarrow (\text{bn}(n, \text{s-times}(x, y))) = \text{s-times}(\text{bn}(n, x), \text{bn}(n, y)))\]

;; A2-End-S-TIMES

; eof:comb_times.bm

Definition:
\[
\text{topor-sy-macc}(\text{ln}) = \begin{cases} 
1 & \text{if ln = 'ymacc} \\
0 & \text{else if ln = 'ymacc2} \\
0 & \text{else}
\end{cases}
\]

Definition:
\[
\text{sy-macc}(\text{ln}, x) = \begin{cases} 
\text{s-times}(x, \text{sy-macc('ymacc2, x)}) & \text{if ln = 'ymacc} \\
\text{sfix}(x) & \text{else}
\end{cases}
\]

Theorem: a2-empty-sy-macc
\[
\text{empty}(\text{sy-macc}(\text{ln}, x)) = \text{empty}(x)
\]

Theorem: a2-e-sy-macc
\[
(\text{sy-macc}(\text{ln}, x) = \text{e}) = \text{empty}(x)
\]

Theorem: a2-lp-sy-macc
\[
\text{len}(\text{sy-macc}(\text{ln}, x)) = \text{len}(x)
\]

Theorem: a2-lpe-sy-macc
\[
\text{eqlen}(\text{sy-macc}(\text{ln}, x), x)
\]

Theorem: a2-pc-sy-macc
\[
\text{p}(\text{sy-macc}(\text{ln}, x)) = \text{sy-macc}(\text{ln}, \text{p}(x))
\]

;; A2-End-SY-MACC

;;; SPEC definition:
**Definition:**

numer-macc \(x\)

\[
\begin{align*}
\text{if} & \quad \text{empty} \ (x) \quad \text{then} \quad 1 \\
\text{else} & \quad \text{numer-macc} \ (p \ (x)) \ast 1 \ (x) \quad \text{endif}
\end{align*}
\]

; this is the standard extension from last-char-fun to MLP-string-fun.

**Definition:**

spec-macc \(x\)

\[
\begin{align*}
\text{if} & \quad \text{empty} \ (x) \quad \text{then} \quad E \\
\text{else} & \quad a \ (\text{spec-macc} \ (p \ (x)), \text{numer-macc} \ (x)) \quad \text{endif}
\end{align*}
\]

;;; Circuit CORRECTNESS:

; Macc-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:** macc-correct-ax

\[
\neg \text{empty} \ (x) \rightarrow (l \ (sy-macc \ ('ymacc, x)) = \text{numer-macc} \ (x))
\]

; 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
; We really would like to use it as a one-time rewrite, but it’s a looping
; rule, so we can’t. Instead we have to use it in USE hints, which in case
; of induction, makes things more complex than they should.

**Theorem:** a-p-l-split

\[
\neg \text{empty} \ (x) \\
\rightarrow \ (sy-macc \ ('ymacc, x) \\
\quad = \quad a \ (p \ (sy-macc \ ('ymacc, x)), l \ (sy-macc \ ('ymacc, x))))
\]

**Theorem:** macc-correct

\[
sy-macc \ ('ymacc, x) = \text{spec-macc} \ (x)
\]

; eof: macc.bm
Index

a, 2, 4
a-p-l-split, 4
a2-bc-s-times, 3
a2-bnc-s-times, 3
a2-e-s-times, 2
a2-e-sy-macc, 3
a2-empty-s-times, 2
a2-empty-sy-macc, 3
a2-hc-s-times, 2
a2-ic-s-times, 2
a2-ic-s-times, 2
a2-lc-s-times, 2
a2-lp-s-times, 2
a2-lp-sy-macc, 3
a2-lpe-s-times, 2
a2-lpe-sy-macc, 3
a2-pc-s-times, 2
a2-pc-sy-macc, 3
b, 3
bn, 3
e, 2–4
empty, 2–4
eqlen, 2, 3
h, 2
i, 2, 3
l, 2, 4
len, 2, 3
macc-correct, 4
macc-correct-ax, 4
numer-macc, 4
p, 2–4
s-times, 2, 3
sfix, 3
spec-macc, 4
sy-macc, 3, 4
topor-sy-macc, 3