```
#|

 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.

```
; 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)
```

```
(Ymacc2 R 1 Ymacc)
))

(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))
$\rightarrow$ (s-times (i $(c\_x,\ x)$, i $(c\_y,\ y)$) $=$ i $(c\_x * c\_y$, s-times $(x,\ y)$))

THEOREM: a2-lc-s-times
($\neg$ empty (x)) $\rightarrow$ (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
(($\neg$ empty (x)) $\wedge$ (len $(x) =$ len (y)))
$\rightarrow$ (h (s-times $(x,\ y)$) $=$ (h $(x) *$ h (y)))

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

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:
topor-sy-macc (ln)
= **if** ln = `'ymacc` **then** `1`
elseif ln = `'ymacc2` **then** `0`
else `0` **endif**

DEFINITION:
sy-macc $(ln,\ x)$
= **if** ln = `'ymacc` **then** s-times $(x,$ sy-macc (`'ymacc2`, $x))$
elseif ln = `'ymacc2`
then if empty (x) **then** E
else i (`1`, sy-macc (`'ymacc`, p (x))) **endif**
else sfix (x) **endif**

```
;; A2-Begin-SY-MACC
```

THEOREM: a2-empty-sy-macc
$\text{empty}(\text{sy-macc}(ln,\ x)) = \text{empty}(x)$

THEOREM: a2-e-sy-macc
$(\text{sy-macc}(ln,\ x) = \text{E}) = \text{empty}(x)$

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

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

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

```
;; A2-End-SY-MACC
```

```
;;; SPEC definition:
```

DEFINITION:
numer-macc (x)
$=$ **if** empty (x) **then** `1`
else numer-macc $(\mathrm{p}(x)) * \mathrm{l}(x)$ **endif**

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

DEFINITION:
spec-macc (x)
$=$ **if** empty (x) **then** E
else a (spec-macc $(\mathrm{p}(x))$, numer-macc (x)) **endif**

```
;;; 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
$(\neg\ \text{empty}(x)) \rightarrow (\mathrm{l}(\text{sy-macc}(\texttt{'ymacc}, x)) = \text{numer-macc}(x))$

```
; 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
$(\neg\ \text{empty}(x))$
$\rightarrow$ $(\text{sy-macc}(\texttt{'ymacc}, x)$
$= \mathrm{a}(\mathrm{p}(\text{sy-macc}(\texttt{'ymacc}, x)), \mathrm{l}(\text{sy-macc}(\texttt{'ymacc}, x))))$

THEOREM: macc-correct
$\text{sy-macc}(\texttt{'ymacc}, x) = \text{spec-macc}(x)$

```
; eof: macc.bm
```

Index