```
#|

 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.

```
; acc_CSXA00.bm
;   . definition of circuit (only 1 here) [assumes stringadd.bm] :
; - no hint: FAIL
; - w/ TOPOR:  OK!
;   . proof of equivalence w/ Spec(s): OK!
; NOTE: the above comments date back to the hand-generation time, when we
;       were still trying to FIND a way to feed things to BM.  They are kept
;       here for historical purposes only...

;;; DEFINITION OF CIRCUIT:
#|
```

```
(setq sysd '(SY-ACC (x)
(Yacc S Plus x Yacc2)
(Yacc2 R 0 Yacc)
))

(setq acc_CSXA00 '(
-rsb |#
; 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))
$\rightarrow$ (s-plus (i $(c_x,\ x)$, i $(c_y,\ y)$) = i $(c_x + c_y$, s-plus $(x,\ y)$))

THEOREM: a2-lc-s-plus
($\neg$ empty (x)) $\rightarrow$ (l (s-plus $(x,\ y)$) = (l (x) + l (y)))

THEOREM: a2-pc-s-plus
p (s-plus $(x,\ y)$) = s-plus (p (x), p (y))

THEOREM: a2-hc-s-plus
$((\neg \text{ empty}(x)) \wedge (\text{len}(x) = \text{len}(y)))$
$\rightarrow \quad (\text{h}(\text{s-plus}(x, y)) = (\text{h}(x) + \text{h}(y)))$

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

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

```
;; A2-End-S-PLUS

; eof:comb_plus.bm
```

DEFINITION:
topor-sy-acc (ln)
= **if** ln = `'yacc` **then** `1`
elseif ln = `'yacc2` **then** `0`
else `0` **endif**

DEFINITION:
sy-acc (ln, x)
= **if** ln = `'yacc` **then** s-plus (x, sy-acc (`'yacc2`, x))
elseif ln = `'yacc2`
then if empty (x) **then** E
else i (`0`, sy-acc (`'yacc`, p (x))) **endif**
else sfix (x) **endif**

```
;; A2-Begin-SY-ACC
```

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

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

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

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

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

```
;; A2-End-SY-ACC

;;; SPEC definition (hand, of course):
```

DEFINITION:
numer-acc (x)
$=$ **if** empty (x) **then** 0
else numer-acc (p (x)) + l (x) **endif**

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

DEFINITION:
spec-acc (x)
$=$ **if** empty (x) **then** E
else a (spec-acc (p (x)), numer-acc (x)) **endif**

```
;;; Circuit CORRECTNESS:

; Acc-correct-ax-help makes the proof of acc_correct_ax much easier, but is
; not necessary.
;
;(prove-lemma acc-correct-ax-help (rewrite)
;(implies (not (empty x))
;  (equal (L (sy-acc 'yacc x))
; (plus (numer-acc (P x)) (L x))))
;((induct (induct-P x))
; (expand (sy-acc 'yacc x))
;)
;)

; Acc-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.
; BM finds the proof on its own, but still needs quite a few generalizations...
```

THEOREM: acc-correct-ax
$(\neg$ empty $(x)) \rightarrow$ (l (sy-acc ('yacc, x)) $=$ numer-acc (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
$(\neg\ \text{empty}\,(x))$
$\rightarrow\quad (\text{sy-acc}\,(\texttt{'yacc},\ x) = \text{a}\,(\text{p}\,(\text{sy-acc}\,(\texttt{'yacc},\ x)),\ \text{l}\,(\text{sy-acc}\,(\texttt{'yacc},\ x))))$

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

```
; eof: acc_CSXA00.bm
; -rsb ))
```

Index

a, 2, 4, 5
a-p-l-split, 5
a2-bc-s-plus, 3
a2-bnc-s-plus, 3
a2-e-s-plus, 2
a2-e-sy-acc, 3
a2-empty-s-plus, 2
a2-empty-sy-acc, 3
a2-hc-s-plus, 3
a2-ic-s-plus, 2
a2-lc-s-plus, 2
a2-lp-s-plus, 2
a2-lp-sy-acc, 3
a2-lpe-s-plus, 2
a2-lpe-sy-acc, 3
a2-pc-s-plus, 2
a2-pc-sy-acc, 3
acc-correct, 5
acc-correct-ax, 4

b, 3
bn, 3

e, 2–4
empty, 2–5
eqlen, 2, 3

h, 3

i, 2, 3

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

numer-acc, 4

p, 2–5

s-plus, 2, 3
sfix, 3
spec-acc, 4, 5
sy-acc, 3–5

topor-sy-acc, 3