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

; funacc.bm
; Circuit is similar to acc, but uses ARBITRARY CHAR-FUN (arity 2) instead of
; addition. It's expressed in CSXA form. Proving it guarantees (at the BM
; level) that the theorems stated in bm_syds_2ndorder.txt are indeed true
; in general, although of course, BM can not instantiate them.
;
;; DEFINITION OF CIRCUIT:
#|
(setq sysd `(sy-funacc (x)
(Yfun S Fun2 x Yreg)
(Yreg R 'finit Yfun)
))

(setq funacc '('
|#
; BM DEFINITIONS and A2 LEMMAS, generated by BMSYSD:
; comb_fun2 bm: Fun2 combinational element
; U7-DONE

; arbitrary Char-Fun of arity 2:
EVENT: Introduce the function symbol fun2 of 2 arguments.

; Everything below generated by: (bmcomb 'Fun2 () '(x y))

DEFINITION:
s-fun2(x, y) = if empty(x) then E
          else a (s-fun2(p(x), p(y)), fun2(l(x), l(y))) endif

;; A2-Begin-S-FUN2

THEOREM: a2-empty-s-fun2
empty(s-fun2(x, y)) = empty(x)

THEOREM: a2-e-s-fun2
(s-fun2(x, y) = E) = empty(x)

THEOREM: a2-lp-s-fun2
len(s-fun2(x, y)) = len(x)

THEOREM: a2-lpe-s-fun2
eqlen(s-fun2(x, y), x)

THEOREM: a2-ic-s-fun2
(len(x) = len(y))
→ (s-fun2(i(c.x, x), i(c.y, y)) = i(fun2(c.x, c.y), s-fun2(x, y)))

THEOREM: a2-lc-s-fun2
(¬ empty(x)) → (1(s-fun2(x, y)) = fun2(1(x), 1(y)))

THEOREM: a2-pc-s-fun2
p(s-fun2(x, y)) = s-fun2(p(x), p(y))

2
THEOREM: a2-hc-s-fun2
\[ ((\neg \text{empty}(x)) \land (\text{len}(x) = \text{len}(y))) \rightarrow (h(s\text{-fun2}(x, y)) = \text{fun2}(h(x), h(y))) \]

THEOREM: a2-bc-s-fun2
\[ (\text{len}(x) = \text{len}(y)) \rightarrow (b(s\text{-fun2}(x, y)) = \text{fun2}(b(x), b(y))) \]

THEOREM: a2-bnc-s-fun2
\[ (\text{len}(x) = \text{len}(y)) \rightarrow (\text{bn}(n, s\text{-fun2}(x, y)) = \text{fun2}(\text{bn}(n, x), \text{bn}(n, y))) \]

; ; A2-End-S-FUN2
;
; eof:comb_fun2.bm

DEFINITION:
\[
\text{topor-sy-funacc}(ln) = \begin{cases} 
1 & \text{if } ln = 'yfun \\
0 & \text{if } ln = 'yreg \\
0 & \text{else}
\end{cases}
\]

DEFINITION:
\[
\text{sy-funacc}(ln, x) = \begin{cases} 
\text{s-fun2}(x, \text{sy-funacc}( 'yreg, x)) & \text{if } ln = 'yfun \\
\text{fn} & \text{if } \text{empty}(x) \\
\text{i}( 'finit, \text{sy-funacc}( 'yfun, p(x))) & \text{else} \\
\text{sfix}(x) & \text{else}
\end{cases}
\]

; ; A2-Begin-SY-FUNACC

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

THEOREM: a2-e-sy-funacc
\[ (\text{sy-funacc}(ln, x) = \text{E}) = \text{empty}(x) \]

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

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

THEOREM: a2-pc-sy-funacc
\[ \text{p}(\text{sy-funacc}(ln, x)) = \text{sy-funacc}(ln, p(x)) \]
;; A2-End-SY-FUNACC

;;; SPEC definition:

**Definition:**
numer-funacc (x) = if empty (x) then 'finit else fun2 (numer-funacc (p (x)), l (x)) endif

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

**Definition:**
spec-funacc (x) = if empty (x) then E else a (spec-funacc (p (x)), numer-funacc (x)) endif

;;; Circuit CORRECTNESS:

; We now declare Fun2 to be COMMUTATIVE because the definition of NUMER-FUNACC
; funs the 2 arguments in the opposite order than the sysd, and hence
; commutativity is required. Alternatively, we could change NUMER-FUNACC’s
; definition. (This has been tested and runs.)

**Axiom:** fun2-comm
fun2 (u, v) = fun2 (v, u)

; Funacc-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.
; NOTE: the simplicity of the proof should imply that disabling the
; char-function in specific instances of accumulator syds should help...

**Theorem:** funacc-correct-ax
(¬ empty (x)) → (l (sy-funacc ('yfun, x)) = numer-funacc (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 . See also THETA.BM .

**Theorem:** a-p-l-split
(¬ empty (x))
→ (sy-funacc ('yfun, x)
 = a (p (sy-funacc ('yfun, x)), l (sy-funacc ('yfun, x))))
THEOREM: funacc-correct
sy-funacc('yfun, x) = spec-funacc(x)

; eof: funacc.bm
;)}
Index

a, 2, 4
a-p-l-split, 4
a2-bc-s-fun2, 3
a2-bnc-s-fun2, 3
a2-e-s-fun2, 2
a2-e-sy-funacc, 3
a2-empty-s-fun2, 2
a2-empty-sy-funacc, 3
a2-hc-s-fun2, 3
a2-ic-s-fun2, 2
a2-ic-s-fun2, 2
a2-lc-s-fun2, 2
a2-lp-s-fun2, 2
a2-lp-sy-funacc, 3
a2-lpe-s-fun2, 2
a2-lpe-sy-funacc, 3
a2-pc-s-fun2, 2
a2-pc-sy-funacc, 3

b, 3
bn, 3

e, 2–4
empty, 2–4
eqlen, 2, 3

fun2, 2–4
fun2-comm, 4
funacc-correct, 5
funacc-correct-ax, 4

h, 3
i, 2, 3
l, 2, 4
len, 2, 3

numer-funacc, 4
p, 2–4
s-fun2, 2, 3

sfix, 3
spec-funacc, 4, 5
sy-funacc, 3–5
topor-sy-funacc, 3