```
#|

 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.

```
; 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 $(y)$)
$\rightarrow$ (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
($\neg$ empty $(x)$) $\rightarrow$ (l (s-fun2 $(x,\ y)$) $=$ fun2 (l $(x)$, l $(y)$))

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

THEOREM: a2-hc-s-fun2
$((\neg \text{ empty}(x)) \wedge (\text{len}(x) = \text{len}(y)))$
$\rightarrow \quad (\text{h}(\text{s-fun2}(x,\ y)) = \text{fun2}(\text{h}(x),\ \text{h}(y)))$

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

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

```
;; A2-End-S-FUN2
```

```
; eof:comb_fun2.bm
```

DEFINITION:
topor-sy-funacc $(ln)$
= **if** $ln$ = `'yfun` **then** `1`
**elseif** $ln$ = `'yreg` **then** `0`
**else** `0` **endif**

DEFINITION:
sy-funacc $(ln,\ x)$
= **if** $ln$ = `'yfun` **then** s-fun2 $(x,$ sy-funacc $($`'yreg`$,\ x))$
**elseif** $ln$ = `'yreg`
**then if** empty $(x)$ **then** E
**else** i $($`'finit`, sy-funacc $($`'yfun`, p $(x)))$ **endif**
**else** sfix $(x)$ **endif**

```
;; 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
$\text{p}(\text{sy-funacc}(ln,\ x)) = \text{sy-funacc}(ln,\ \text{p}(x))$

```
;; 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
$(\neg$ empty $(x))$ $\rightarrow$ (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
$(\neg$ empty $(x))$
$\rightarrow$ (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