

#|

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.

```
; counterR.bm: a resetable clock counter, i.e. Paillet example 1.
;
; This is our first circuit w/ 2 inputs, yet to my amazement,
; extra EQLEN hypotheses were NEVER required, neither in the A2
; lemmas, nor in the actual specifications (Paillet, or mine)! Of
; course, when they became required (in Paillet #7) and put in
; Sugar, they were installed here.
;
; NOTE (w/ EMPTY enabled) : Attempts to speed up A2-PC failed
; miserably: disabling the combinationals fails because of the
; uneven cases, and the fact that BM can't derive (empty y) from
```

```

; (empty x) and equal (len x) (len y).
; A problem we have had for a long time, and which will probably
; persist until we solve EQLEN.
; Giving the induction hint ahead of time made things worse, as
; usual. Twiddling with STR-P-I2 improved nothing.

```

```

;;; CIRCUIT in SUGARED form:

```

```

#|
(setq sysd '(sy-COUNT (Xc Xe)
(Ymux S Mux Xc Xe Yinc)
(Yreg R 0 YMux)
(Yinc S Incn Yreg)
))

```

```

(setq counterR '(|#
; BM DEFINITIONS and A2 LEMMAS, generated by BMSYSD:
; comb_mux.bm: Mux combinational element, i.e. "if".
; U7-DONE

```

DEFINITION:

```

mux(u1, u2, u3)
= if u1 then u2
  else u3 endif

```

```

; everything below generated by: (bmcomb 'mux '() '(x1 x2 x3))
; with the EXCEPTIONS/HAND-MODIFICATIONS given below.

```

DEFINITION:

```

s-mux(x1, x2, x3)
= if empty(x1) then E
  else a(s-mux(p(x1), p(x2), p(x3)), mux(l(x1), l(x2), l(x3))) endif

```

```

; SMUX-is-SIF can make things much simpler on occasions:

```

THEOREM: smux-is-sif

```

s-mux(x1, x2, x3) = s-if(x1, x2, x3)

```

EVENT: Disable smux-is-sif.

```

; We take advantage of SMUX-is-SIF for all inductive proofs. To do so we

```

```
; HAND-MODIFY the code generated by Sugar to replace all the hints by
; - A2-EMPTY, A2-PC replace hint with: ((enable smux-is-sif)
; - A2-LP, A2-IC, A2-HC, A2-BC: ((enable smux-is-sif) (disable len))
; - A2-BNC: ((enable smux-is-sif) (disable bn len))
```

```
:: A2-Begin-S-MUX
```

THEOREM: a2-empty-s-mux
 $\text{empty}(\text{s-mux}(x1, x2, x3)) = \text{empty}(x1)$

THEOREM: a2-e-s-mux
 $(\text{s-mux}(x1, x2, x3) = E) = \text{empty}(x1)$

THEOREM: a2-lp-s-mux
 $\text{len}(\text{s-mux}(x1, x2, x3)) = \text{len}(x1)$

THEOREM: a2-lpe-s-mux
 $\text{eqlen}(\text{s-mux}(x1, x2, x3), x1)$

THEOREM: a2-ic-s-mux
 $((\text{len}(x1) = \text{len}(x2)) \wedge (\text{len}(x2) = \text{len}(x3)))$
 $\rightarrow (\text{s-mux}(i(c_x1, x1), i(c_x2, x2), i(c_x3, x3)))$
 $= i(\text{mux}(c_x1, c_x2, c_x3), \text{s-mux}(x1, x2, x3)))$

THEOREM: a2-lc-s-mux
 $(\neg \text{empty}(x1)) \rightarrow (l(\text{s-mux}(x1, x2, x3)) = \text{mux}(l(x1), l(x2), l(x3)))$

THEOREM: a2-pc-s-mux
 $p(\text{s-mux}(x1, x2, x3)) = \text{s-mux}(p(x1), p(x2), p(x3))$

THEOREM: a2-hc-s-mux
 $((\neg \text{empty}(x1)) \wedge ((\text{len}(x1) = \text{len}(x2)) \wedge (\text{len}(x2) = \text{len}(x3))))$
 $\rightarrow (\text{h}(\text{s-mux}(x1, x2, x3)) = \text{mux}(\text{h}(x1), \text{h}(x2), \text{h}(x3)))$

```
;old: ((DISABLE MUX S-MUX) (ENABLE H LEN) (INDUCT (S-MUX X1 X2 X3)))
```

THEOREM: a2-bc-s-mux
 $((\text{len}(x1) = \text{len}(x2)) \wedge (\text{len}(x2) = \text{len}(x3)))$
 $\rightarrow (\text{b}(\text{s-mux}(x1, x2, x3)) = \text{s-mux}(\text{b}(x1), \text{b}(x2), \text{b}(x3)))$

```
;old: ((DISABLE MUX) (ENABLE B LEN) (INDUCT (S-MUX X1 X2 X3)))
```

THEOREM: a2-bnc-s-mux
 $((\text{len}(x1) = \text{len}(x2)) \wedge (\text{len}(x2) = \text{len}(x3)))$
 $\rightarrow (\text{bn}(n, \text{s-mux}(x1, x2, x3)) = \text{s-mux}(\text{bn}(n, x1), \text{bn}(n, x2), \text{bn}(n, x3)))$

```

;old: ((DISABLE MUX S-MUX))

;; A2-End-S-MUX

; eof:comb_mux.bm

; comb_incn.bm: Inc modulo N combinational element, a minor modification
;                of comb_inc which shouldn't cause any difference, unless
;                the loop-around property is used in a critical way, which is
;                rare.
; Note that N is treated as a global constant, but not as an individual
; parameter, so we don't have to carry it around everywhere. This is just
; an experiment, to see what's more convenient.
; U7-DONE

```

EVENT: Introduce the function symbol n of 0 arguments.

; we may want to add an axiom saying that it's a number, not needed so far..

DEFINITION:

```

incn (u)
=  if u = N then 0
   else 1 + u endif

```

; Everything below generated by: (bmcomb 'incn '() '(x))

DEFINITION:

```

s-incn (x)
=  if empty (x) then E
   else a (s-incn (p (x)), incn (l (x))) endif

```

;; A2-Begin-S-INCN

THEOREM: a2-empty-s-incn
 $\text{empty (s-incn (x))} = \text{empty (x)}$

THEOREM: a2-e-s-incn
 $\text{(s-incn (x) = E)} = \text{empty (x)}$

THEOREM: a2-lp-s-incn
 $\text{len (s-incn (x))} = \text{len (x)}$

THEOREM: a2-lpe-s-incn

$$\text{eqlen}(\text{s-incn}(x), x)$$

THEOREM: a2-ic-s-incn

$$\text{s-incn}(i(c.x, x)) = i(\text{incn}(c.x), \text{s-incn}(x))$$

THEOREM: a2-lc-s-incn

$$(\neg \text{empty}(x)) \rightarrow (l(\text{s-incn}(x)) = \text{incn}(l(x)))$$

THEOREM: a2-pc-s-incn

$$p(\text{s-incn}(x)) = \text{s-incn}(p(x))$$

THEOREM: a2-hc-s-incn

$$(\neg \text{empty}(x)) \rightarrow (h(\text{s-incn}(x)) = \text{incn}(h(x)))$$

THEOREM: a2-bc-s-incn

$$b(\text{s-incn}(x)) = \text{s-incn}(b(x))$$

THEOREM: a2-bnc-s-incn

$$\text{bn}(n, \text{s-incn}(x)) = \text{s-incn}(\text{bn}(n, x))$$

;; A2-End-S-INCN

; eof:comb_incncn.bm

DEFINITION:

topor-sy-count(*ln*)

```
= if ln = 'ymux then 2
   elseif ln = 'yreg then 0
   elseif ln = 'yinc then 1
   else 0 endif
```

DEFINITION:

sy-count(*ln*, *xc*, *xe*)

```
= if ln = 'ymux then s-mux(xc, xe, sy-count('yinc, xc, xe))
   elseif ln = 'yreg
   then if empty(xc) then E
        else i(0, sy-count('ymux, p(xc), p(xe))) endif
   elseif ln = 'yinc then s-incn(sy-count('yreg, xc, xe))
   else sfix(xc) endif
```

;; A2-Begin-SY-COUNT

```

THEOREM: a2-empty-sy-count
(len(xc) = len(xe)) → (empty(sy-count(ln, xc, xe)) = empty(xc))

THEOREM: a2-e-sy-count
(len(xc) = len(xe)) → ((sy-count(ln, xc, xe) = E) = empty(xe))

THEOREM: a2-lp-sy-count
(len(xc) = len(xe)) → (len(sy-count(ln, xc, xe)) = len(xc))

THEOREM: a2-lpe-sy-count
(len(xc) = len(xe)) → eqlen(sy-count(ln, xc, xe), xc)

THEOREM: a2-pc-sy-count
(len(xc) = len(xe))
→ (p(sy-count(ln, xc, xe)) = sy-count(ln, p(xc), p(xe)))

;; A2-End-SY-COUNT

;;; Circuit CORRECTNESS /Paillet:

; Note that as originally stated in Paillet, with the P outside of
; sy-count makes for a looping (unfolding) which would have to be
; proved kludgeily, and would be useless. The following rule can
; be used as a rewrite.

THEOREM: count-paillet-correct
((¬ empty(xc)) ∧ (¬ empty(xe)))
→ (sy-count('yreg, xc, xe)
    = i(0, s-if(p(xc), p(xe), s-incn(sy-count('yreg, p(xc), p(xe))))))

; The "last-char" reading of the spec yields:
; NOTE: we can prove it by repeating the same hint and disabling
; CORRECT, i.e. independently. Trying to use CORRECT fails
; miserably because it also triggers on:
; (sy-count 'Yreg (P Xc) (P Xe)). Note also that we need the
; EQ-LEN hyp because we need A2-EMPTY-SY-COUNT.

THEOREM: count-paillet-correct-l
((¬ empty(p(xc))) ∧ (¬ empty(p(xe))) ∧ (len(xc) = len(xe)))
→ (l(sy-count('yreg, xc, xe))
    = if l(p(xc)) then l(p(xe))
      else incn(l(sy-count('yreg, p(xc), p(xe)))) endif)

; eof: counterR.bm
;))

```

Index

a, 2, 4
a2-bc-s-incn, 5
a2-bc-s-mux, 3
a2-bnc-s-incn, 5
a2-bnc-s-mux, 3
a2-e-s-incn, 4
a2-e-s-mux, 3
a2-e-sy-count, 6
a2-empty-s-incn, 4
a2-empty-s-mux, 3
a2-empty-sy-count, 6
a2-hc-s-incn, 5
a2-hc-s-mux, 3
a2-ic-s-incn, 5
a2-ic-s-mux, 3
a2-lc-s-incn, 5
a2-lc-s-mux, 3
a2-lp-s-incn, 4
a2-lp-s-mux, 3
a2-lp-sy-count, 6
a2-lpe-s-incn, 5
a2-lpe-s-mux, 3
a2-lpe-sy-count, 6
a2-pc-s-incn, 5
a2-pc-s-mux, 3
a2-pc-sy-count, 6

b, 3, 5
bn, 3, 5

count-paillet-correct, 6
count-paillet-correct-l, 6

e, 2–6
empty, 2–6
eqlen, 3, 5, 6

h, 3, 5

i, 3, 5, 6
incn, 4–6

l, 2–6
len, 3, 4, 6
mux, 2, 3

n, 4

p, 2–6

s-if, 2, 6
s-incn, 4–6
s-mux, 2, 3, 5
sfix, 5
smux-is-sif, 2
sy-count, 5, 6

topor-sy-count, 5