

#|

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.

```
; countstut.bm: a simple clock counter, and a STUTTERING version of it,
;      to test Stutter formalizaton.
;
; . A2PC blows up on stuttering version, but that's been reduced to a
; a very trivial version (3 lines) which also blows up and on which one
; can see that BM loops without any rule: one of the equality hyp. is used
; in the opposite direction as the definition expansion.
; We use ADD-AXIOM on A2-PCs which blow.
;
;;
;; DEFINITION OF CIRCUITS:
```

```

#|
(setq A '(sy-A (x)
(Yout R 0 Y1)
(Y1 S Inc Yout)
))

(setq B '(sy-B (x)
(Yout R 0 Y1m)
(Y1m S Mux Yst Yout Y1)
(Y1 S Inc Yout)
(Yst R T Yst1)
(Yst1 S Not Yst)
))

(setq C '(sy-C (xst)
(Yout R 0 Y1m)
(Y1m S Mux xst Yout Y1)
(Y1 S Inc Yout)
))

(setq countstut '(|#
; BM DEFINITIONS and A2 LEMMAS, generated by BMSYSD:
; comb_inc.bm: INCrement combinational element
; U7-DONE

```

DEFINITION: $\text{inc}(u) = (1 + u)$

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

DEFINITION:

```

s-inc(x)
= if empty(x) then E
  else a(s-inc(p(x)), inc(l(x))) endif
;;
A2-Begin-S-INC

```

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

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

THEOREM: a2-lp-s-inc
 $\text{len}(\text{s-inc}(x)) = \text{len}(x)$
 THEOREM: a2-lpe-s-inc
 $\text{eqlen}(\text{s-inc}(x), x)$
 THEOREM: a2-ic-s-inc
 $\text{s-inc}(\text{i}(c_x, x)) = \text{i}(\text{inc}(c_x), \text{s-inc}(x))$
 THEOREM: a2-lc-s-inc
 $(\neg \text{empty}(x)) \rightarrow (\text{l}(\text{s-inc}(x)) = \text{inc}(\text{l}(x)))$
 THEOREM: a2-pc-s-inc
 $\text{p}(\text{s-inc}(x)) = \text{s-inc}(\text{p}(x))$
 THEOREM: a2-hc-s-inc
 $(\neg \text{empty}(x)) \rightarrow (\text{h}(\text{s-inc}(x)) = \text{inc}(\text{h}(x)))$
 THEOREM: a2-bc-s-inc
 $\text{b}(\text{s-inc}(x)) = \text{s-inc}(\text{b}(x))$
 THEOREM: a2-bnc-s-inc
 $\text{bn}(n, \text{s-inc}(x)) = \text{s-inc}(\text{bn}(n, x))$
 ;; A2-End-S-INC
 ; eof:comb_inc.bm

DEFINITION:
 $\text{topor-sy-a}(ln)$
 $= \text{if } ln = 'yout \text{ then } 0$
 $\quad \text{elseif } ln = 'y1 \text{ then } 1$
 $\quad \text{else } 0 \text{ endif}$

DEFINITION:
 $\text{sy-a}(ln, x)$
 $= \text{if } ln = 'yout$
 $\quad \text{then if empty}(x) \text{ then E}$
 $\quad \quad \text{else i}(0, \text{sy-a}('y1, p(x))) \text{ endif}$
 $\quad \text{elseif } ln = 'y1 \text{ then s-inc}(\text{sy-a}('yout, x))$
 $\quad \text{else sfix}(x) \text{ endif}$

;; A2-Begin-SY-A

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

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

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

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

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

```
; ; A2-End-SY-A
; 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
 $\text{s-mux}(x1, x2, x3) = \text{s-if}(x1, x2, x3)$

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

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

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

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

;; already loaded in A: (LOAD "Comb/comb_inc.bm")

```

DEFINITION:

```

topor-sy-b(ln)
= if ln = 'yout then 0
  elseif ln = 'y1m then 2
  elseif ln = 'y1 then 1
  elseif ln = 'yst then 0
  elseif ln = 'yst1 then 1
  else 0 endif

```

DEFINITION:

```

sy-b(ln, x)
= if ln = 'yout
  then if empty(x) then E
    else i(0, sy-b('y1m, p(x))) endif
  elseif ln = 'y1m
  then s-mux(sy-b('yst, x), sy-b('yout, x), sy-b('y1, x))
  elseif ln = 'y1 then s-inc(sy-b('yout, x))
  elseif ln = 'yst
  then if empty(x) then E
    else i(t, sy-b('yst1, p(x))) endif
  elseif ln = 'yst1 then s-not(sy-b('yst, x))
  else sf(x) endif

```

;; A2-Begin-SY-B

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

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

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

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

```

; See note at top of file.
;(PROVE-LEMMA A2-PC-SY-B (REWRITE)
;      (EQUAL (P (SY-B LN X)) (SY-B LN (P X)))
;      ((DISABLE S-MUX S-INC S-NOT A2-IC-S-MUX A2-IC-S-INC A2-IC-S-NOT)))

AXIOM: a2-pc-sy-b
p (sy-b (ln, xst)) = sy-b (ln, p (xst))

;; A2-End-SY-B

; BM DEFINITIONS and A2 LEMMAS, generated by BMSYSD:
;; already loaded: (LOAD "Comb/comb_mux.bm")
;; already loaded: (LOAD "Comb/comb_inc.bm")

DEFINITION:
topor-sy-c (ln)
= if ln = 'yout then 0
  elseif ln = 'y1m then 2
  elseif ln = 'y1 then 1
  else 0 endif

DEFINITION:
sy-c (ln, xst)
= if ln = 'yout
  then if empty (xst) then E
    else i (0, sy-c ('y1m, p (xst))) endif
  elseif ln = 'y1m then s-mux (xst, sy-c ('yout, xst), sy-c ('y1, xst))
  elseif ln = 'y1 then s-inc (sy-c ('yout, xst))
  else sfix (xst) endif

;; A2-Begin-SY-C

```

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

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

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

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

```

; blows..
;(PROVE-LEMMA A2-PC-SY-C (REWRITE)
;    (EQUAL (P (SY-C LN XST)) (SY-C LN (P XST)))
;    ((DISABLE S-MUX S-INC A2-IC-S-MUX A2-IC-S-INC)))
; so TEMPORARILY:

AXIOM: a2-pc-sy-c
p (sy-c (ln, xst)) = sy-c (ln, p (xst))

;; A2-End-SY-C

;;; BEGIN: Circuit CORRECTNESS modulo Stuttering.

;; BEGIN: new 2nd order properties for combinationals.

;; END: new 2nd order properties for combinationals.

;;; Get STUTTER theory:
;;;;; TH_STUTTER.BM
;;;
;;; This file contains Stutter theory for BM. It is supposed to be
;;; loaded directly when needed (i.e. not general enough to be
;;; stored in Lib/mlp).
;;;

;;; Our current double P-recursive def. of Stutter:
;;; Originally, it came from THETA-PRF-79 (done while babysitting
;;; for Caroline...) followed by MUCH experimentation and fiddling.

```

DEFINITION:

```

stut-r(x, y)
=  if empty(y) then x
   elseif empty(p(y)) then b(x)
   elseif l(p(y)) then strut-r(x, p(y))
   else b(strut-r(x, p(y))) endif

```

DEFINITION:

```

stut(x, y)
=  if empty(y) then E
   elseif empty(p(y)) then a(E, h(x))
   elseif l(p(y)) then a(strut(p(x), p(y)), l(strut(p(x), p(y))))
   else a(strut(p(x), p(y)), h(strut-r(x, p(y)))) endif

```

```

; Stut-Induct inducts like stut, but without the case disjunction
; on LPx which is useless when we stutter on a line rather than an
; input. The resulting induction is not very different from a
; straight P induction, but it takes care of the empty Px case
; separately, and without bringing an elimination.

```

DEFINITION:

```

stut-induct (x)
=  if empty (x) then 0
   elseif empty (p (x)) then 1
   else stut-induct (p (x)) endif

```

; Properties of Stut:

THEOREM: stut-empty
 $\text{empty}(\text{stut}(x, y)) = \text{empty}(y)$

THEOREM: stut-e
 $(\text{stut}(x, y) = E) = \text{empty}(y)$

THEOREM: stut-p
 $p(\text{stut}(x, y)) = \text{stut}(p(x), p(y))$

; Properties of Stut-R:

```

; Stut-R-E maybe shouldn't be enabled all the time, but when we're
; doing P inductions on Stut-R, this gives the base case. The
; induction step is given by Stut-R-P. Note that we don't have a
; full empty x hyp because Stut-R returns x and not sfix x in case
; y is empty... Maybe we want to fix that at some point.

```

THEOREM: stut-r-e
 $\text{stut-r}(E, y) = E$

THEOREM: stut-r-p
 $p(\text{stut-r}(x, y)) = \text{stut-r}(p(x), y)$

THEOREM: stut-r-len
 $\text{len}(x) < (1 + (\text{len}(y) + \text{len}(\text{stut-r}(x, y))))$

THEOREM: stut-r-not-empty
 $(\text{len}(y) < \text{len}(x)) \rightarrow (\neg \text{empty}(\text{stut-r}(x, y)))$

; Stut-Rem removes the trailing Ts of y, but ignores Ly (like R)
; and leaves one T: this weird def, so it works like Stut-R needs!

DEFINITION:

```
stut-rem(y)
= if empty(y) then E
  elseif empty(p(y)) then y
  elseif l(p(y)) then stut-rem(p(y))
  else y endif
```

THEOREM: stut-rem-empty
 $\text{empty}(\text{stut-rem}(x)) = \text{empty}(x)$

THEOREM: stut-rem-len
 $\text{len}(\text{stut-rem}(x)) < (1 + \text{len}(x))$

THEOREM: stut-rem-len2
 $((\neg \text{empty}(p(x))) \wedge l(p(x))) \rightarrow (\text{len}(\text{stut-rem}(x)) < \text{len}(x))$

; Stut-Num counts the number of F in y, ignoring Ly, and starts
; w/ 1, like Stut.

DEFINITION:

```
stut-num(y)
= if empty(y) then 0
  elseif empty(p(y)) then 1
  elseif l(p(y)) then stut-num(p(y))
  else 1 + stut-num(p(y)) endif
```

THEOREM: stut-num-lessp
 $\text{stut-num}(x) < (1 + \text{len}(x))$

THEOREM: stut-num-eq-0
 $(\text{stut-num}(x) = 0) = \text{empty}(x)$

; Requires a small induction.

THEOREM: stut-num-rem-len
 $(\neg \text{empty}(x)) \rightarrow (\text{stut-num}(p(\text{stut-rem}(x))) < \text{len}(x))$

; From Stut-Num and Bn we get a CLOSED FORM for Stut-R !!!

THEOREM: stut-r-closed
 $\text{stut-r}(x, y) = \text{bn}(\text{stut-num}(y), x)$

```
; Stut-inv is the key invariant property during Stuttering:
```

```
THEOREM: stut-inv
```

$$\begin{aligned} & ((\neg \text{empty}(y)) \wedge (\text{len}(x) \not< \text{len}(y))) \\ \rightarrow & \quad (\text{l(stut}(x, y)) = \text{h(stut-r}(x, \text{p(stut-rem}(y)))))) \end{aligned}$$

```
; but we only want to use it during the non-stuttering induction  
; step, and not in general so:
```

```
THEOREM: stut-inv0
```

$$\begin{aligned} & ((\neg \text{empty}(y)) \wedge (\text{len}(x) \not< \text{len}(y)) \wedge (\neg \text{l}(y))) \\ \rightarrow & \quad (\text{l(stut}(x, y)) = \text{h(stut-r}(x, \text{p(stut-rem}(y)))))) \end{aligned}$$

```
EVENT: Disable stut-inv.
```

```
; Now we relate Stut-R for Py and P Rem Py, to get the key to the  
; induction step on main Stut inductions, in the non-stuttering  
; case.  
;  
; It's a BAD rewrite (i.e. expanding, potentially self-applicable),  
; and so are the preliminary lemmas needed to build to it. This  
; is not just an unfortunate construction. It's inherent, because  
; we're essentially giving an alternate definition via a Stut-Rem  
; recursion. And definitions are expanding, self-applicable,  
; rewrites. We get around the problem by lucking out: the  
; hypotheses are sufficient to prevent successful self-applic.
```

```
THEOREM: stut-r-indstep-num
```

$$\begin{aligned} & ((\neg \text{empty}(x)) \wedge \text{l}(x)) \\ \rightarrow & \quad (\text{stut-num}(x) = (1 + \text{stut-num}(\text{p(stut-rem}(x)))))) \end{aligned}$$

```
EVENT: Disable stut-r-indstep-num.
```

```
; OLD induction step prereq: not needed anymore.  
;  
;(prove-lemma Stut-R-indstep-Num-Rem (rewrite)  
(implies (and (not (empty y))  
;           (not (empty (P y)))  
;           (not (L (P y)))  
;           ))  
;   (equal (Stut-Num (P (Stut-Rem (P y))))))
```

```

; (sub1 (Stut-Num (P (Stut-Rem y))))
; ))
;((enable Stut-R-indstep-Num))
;)
;(disable Stut-R-indstep-Num-Rem)

; OLD induction step: not needed anymore.
;
;(prove-lemma Stut-indstep (rewrite)
;(implies (and (not (empty y))
;           (not (empty (P y)))
;           (not (L (P y)))
;           )
;           (equal (Stut-R x (P y))
;           (B (Stut-R x (P (Stut-Rem (P y)))))))
; ))
;((enable Stut-R-indstep-Num-Rem B-Bn-sub1)
; (disable Bn)
; )
;)
;(disable Stut-indstep) ; potentially self-looping...
; ; so we enable explicitly.

; NEW & GENERALIZED induction step hack , note: needs just ONE
; prereq! We're getting cleaner...

```

THEOREM: $\text{stut-r-indstep} \equiv ((\neg \text{empty}(y)) \wedge (\neg \text{l}(y))) \rightarrow (\text{stut-r}(x, y) = \text{b}(\text{stut-r}(x, \text{p}(\text{stut-rem}(y)))))$

EVENT: Disable stut-r-indstep .

; all the internal stuff shouldn't be needed outside:

EVENT: Disable stut-r .

EVENT: Disable stut-num .

EVENT: Disable stut-rem .

```

; Note: by leaving Stut, Stut-inv0, Stut-R-closed enabled, we get
; the effect of an alternate recursive definition of Stut in the
; most convenient form. The remaining uncleanliness is that
; Stut-R-indstep and Stut-R-closed match the same stuff, and need
; to be used at different places in the main proof. So far, we
; survive by extreme cunning: they are in the right order, and
; the hypothesis on Stut-R-indstep prevents wrong occurrences.
; This is neither clear nor robust...

;; eof: th_stutter.bm

;; BEGIN: ACTUAL Circuit CORRECTNESS modulo Stuttering.

; REVERSAL PROPERTY for sy-a:

THEOREM: sy-a-reversal

$$(\neg \text{empty}(\text{bn}(n, \text{sy-a}('yout, p(x)))))$$


$$\rightarrow (\text{h}(\text{b}(\text{bn}(n, \text{sy-a}('yout, x)))) = (1 + \text{h}(\text{bn}(n, \text{sy-a}('yout, p(x))))))$$


THEOREM: count-ac-l

$$\text{l}(\text{stut}(\text{sy-a}('yout, xst), xst)) = \text{l}(\text{sy-c}('yout, xst))$$


; Now extending to strings. For some unknown reason, compared to Funacc, we
; need BOTH splits here... Probably because of some weird non-triggering
; phenomenon in equality hyp usage.

THEOREM: apl-split-cout

$$(\neg \text{empty}(x))$$


$$\rightarrow (\text{sy-c}('yout, x) = \text{a}(\text{p}(\text{sy-c}('yout, x)), \text{l}(\text{sy-c}('yout, x))))$$


THEOREM: apl-split-stuta

$$(\neg \text{empty}(x))$$


$$\rightarrow (\text{stut}(\text{sy-a}('yout, x), x)$$


$$= \text{a}(\text{p}(\text{stut}(\text{sy-a}('yout, x), x)), \text{l}(\text{stut}(\text{sy-a}('yout, x), x))))$$


; and finally:

THEOREM: count-ac-correct

$$\text{stut}(\text{sy-a}('yout, xst), xst) = \text{sy-c}('yout, xst)$$


;; END: ACTUAL Circuit CORRECTNESS modulo Stuttering for A and C.

```

EVENT: Disable count-ac-l.

EVENT: Disable count-ac-correct.

; HCorr properties are the "Hand-Correctness" formulas... They are not
; necessary for the proof of Count-AB-L, but I'm trying to see if they help.

THEOREM: hcorr-ab-t

$$\begin{aligned} & ((\neg \text{empty}(x)) \wedge (\neg \text{empty}(p(x))) \wedge l(\text{sy-b}('yout, p(x)))) \\ \rightarrow & \quad (l(\text{sy-b}('yout, x)) = l(\text{sy-b}('yout, p(x)))) \end{aligned}$$

EVENT: Disable hcorr-ab-t.

; obviously..

THEOREM: hcorr-ab-f

$$\begin{aligned} & ((\neg \text{empty}(x)) \wedge (\neg \text{empty}(p(x))) \wedge (\neg l(\text{sy-b}('yout, p(x))))) \\ \rightarrow & \quad (l(\text{sy-b}('yout, x)) = (1 + l(\text{sy-b}('yout, p(x))))) \end{aligned}$$

EVENT: Disable hcorr-ab-f.

; Count-AB-L succeeds with either HCorrs enabled, or the expansion hint
; therein. The costs (time/clarity) seem equal. In the future, if the
; effort involved in getting HCorrs is greater, the dichotomy may be useful.

THEOREM: count-ab-l

$$l(\text{stut}(\text{sy-a}('yout, x), \text{sy-b}('yout, x))) = l(\text{sy-b}('yout, x))$$

THEOREM: apl-split-bout

$$\begin{aligned} & (\neg \text{empty}(x)) \\ \rightarrow & \quad (\text{sy-b}('yout, x) = a(p(\text{sy-b}('yout, x)), l(\text{sy-b}('yout, x)))) \end{aligned}$$

THEOREM: apl-split-stuta2

$$\begin{aligned} & (\neg \text{empty}(x)) \\ \rightarrow & \quad (\text{stut}(\text{sy-a}('yout, x), \text{sy-b}('yout, x))) \\ = & \quad a(p(\text{stut}(\text{sy-a}('yout, x), \text{sy-b}('yout, x))), \\ & \quad l(\text{stut}(\text{sy-a}('yout, x), \text{sy-b}('yout, x)))) \end{aligned}$$

THEOREM: count-ab-correct

$$\text{stut}(\text{sy-a}('yout, x), \text{sy-b}('yout, x)) = \text{sy-b}('yout, x)$$

```
; ; END: ACTUAL Circuit CORRECTNESS modulo Stuttering for A and B.  
; eof: countstut.bm  
; ))
```

Index

- a, 2, 4, 8, 13, 14
- a2-bc-s-inc, 3
- a2-bc-s-mux, 5
- a2-bnc-s-inc, 3
- a2-bnc-s-mux, 5
- a2-e-s-inc, 2
- a2-e-s-mux, 5
- a2-e-sy-a, 4
- a2-e-sy-b, 6
- a2-e-sy-c, 7
- a2-empty-s-inc, 2
- a2-empty-s-mux, 5
- a2-empty-sy-a, 4
- a2-empty-sy-b, 6
- a2-empty-sy-c, 7
- a2-hc-s-inc, 3
- a2-hc-s-mux, 5
- a2-ic-s-inc, 3
- a2-ic-s-mux, 5
- a2-lc-s-inc, 3
- a2-lc-s-mux, 5
- a2-lp-s-inc, 3
- a2-lp-s-mux, 5
- a2-lp-sy-a, 4
- a2-lp-sy-b, 6
- a2-lp-sy-c, 7
- a2-lpe-s-inc, 3
- a2-lpe-s-mux, 5
- a2-lpe-sy-a, 4
- a2-lpe-sy-b, 6
- a2-lpe-sy-c, 7
- a2-pc-s-inc, 3
- a2-pc-s-mux, 5
- a2-pc-sy-a, 4
- a2-pc-sy-b, 7
- a2-pc-sy-c, 8
- apl-split-bout, 14
- apl-split-cout, 13
- apl-split-stuta, 13
- apl-split-stuta2, 14
- b, 3, 5, 8, 12, 13
- bn, 3, 5, 10, 13
- count-ab-correct, 14
- count-ab-l, 14
- count-ac-correct, 13
- count-ac-l, 13
- e, 2–10
- empty, 2–14
- eqlen, 3–7
- h, 3, 5, 8, 11, 13
- hcorr-ab-f, 14
- hcorr-ab-t, 14
- i, 3, 5–7
- inc, 2, 3
- l, 2–5, 8, 10–14
- len, 3–7, 9–11
- mux, 4, 5
- p, 2–14
- s-if, 4
- s-inc, 2, 3, 6, 7
- s-mux, 4–7
- s-not, 6
- sfix, 3, 6, 7
- smux-is-sif, 4
- stut, 8, 9, 11, 13, 14
- stut-e, 9
- stut-empty, 9
- stut-induct, 9
- stut-inv, 11
- stut-inv0, 11
- stut-num, 10, 11
- stut-num-eq-0, 10
- stut-num-lessp, 10
- stut-num-rem-len, 10

stut-p, 9
stut-r, 8–12
stut-r-closed, 10
stut-r-e, 9
stut-r-indstep, 12
stut-r-indstep-num, 11
stut-r-len, 9
stut-r-not-empty, 9
stut-r-p, 9
stut-rem, 10–12
stut-rem-empty, 10
stut-rem-len, 10
stut-rem-len2, 10
sy-a, 3, 4, 13, 14
sy-a-reversal, 13
sy-b, 6, 7, 14
sy-c, 7, 8, 13

topor-sy-a, 3
topor-sy-b, 6
topor-sy-c, 7