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 formalization.
;
; . 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: 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
empty(s-inc(x)) = empty(x)

THEOREM: a2-e-s-inc
(s-inc(x) = E) = empty(x)
Theorem: a2-lp-s-inc
\[ \text{len}(s\text{-inc}(x)) = \text{len}(x) \]

Theorem: a2-lpe-s-inc
\[ \text{eqlen}(s\text{-inc}(x), x) \]

Theorem: a2-ic-s-inc
\[ s\text{-inc}(i(c_x, x)) = i(\text{inc}(c_x), s\text{-inc}(x)) \]

Theorem: a2-lc-s-inc
\[ (\neg \text{empty}(x)) \rightarrow (l(s\text{-inc}(x)) = \text{inc}(l(x))) \]

Theorem: a2-ic-s-inc
\[ p(s\text{-inc}(x)) = s\text{-inc}(p(x)) \]

Theorem: a2-ic-s-inc
\[ (\neg \text{empty}(x)) \rightarrow (h(s\text{-inc}(x)) = \text{inc}(h(x))) \]

Theorem: a2-ic-s-inc
\[ b(s\text{-inc}(x)) = s\text{-inc}(b(x)) \]

Theorem: a2-bnc-s-inc
\[ \text{bn}(n, s\text{-inc}(x)) = s\text{-inc}(\text{bn}(n, x)) \]

;; A2-End-S-INC

;; eof:comb_inc.bm

Definition:
\[ \text{topor-sy-a}(ln) = \begin{cases} 0 & \text{if } ln = 'yout \\ 1 & \text{else if } ln = 'y1 \\ 0 & \text{else} \end{cases} \]

Definition:
\[ \text{sy-a}(ln, x) = \begin{cases} \text{E} & \text{if } ln = 'yout \\ \text{E} & \text{else if } \text{empty}(x) \\ i(0, \text{sy-a}( 'y1, p(x))) & \text{endif} \\ \text{E} & \text{elseif } ln = 'y1 \\ \text{E} & \text{else} \end{cases} \]

;; A2-Begin-SY-A
Theorem: a2-empty-sy-a
empty (sy-a (ln, x)) = empty (x)

Theorem: a2-e-sy-a
(sy-a (ln, x) = \epsilon) = empty (x)

Theorem: a2-lp-sy-a
len (sy-a (ln, x)) = len (x)

Theorem: a2-lpe-sy-a
eqlen (sy-a (ln, x), x)

Theorem: a2-pc-sy-a
p (sy-a (ln, x)) = 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 \epsilon 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))
Theorem: a2-empty-s-mux
empty (s-mux (x1, x2, x3)) = empty (x1)

Theorem: a2-e-s-mux
(s-mux (x1, x2, x3) = e) = empty (x1)

Theorem: a2-lp-s-mux
len (s-mux (x1, x2, x3)) = len (x1)

Theorem: a2-lpe-s-mux
eqlen (s-mux (x1, x2, x3), x1)

Theorem: a2-ic-s-mux
((len (x1) = len (x2)) \land (len (x2) = len (x3)))
\rightarrow (i (c_{x1}, x1), i (c_{x2}, x2), i (c_{x3}, x3))
= i (mux (c_{x1}, c_{x2}, c_{x3}), s-mux (x1, x2, x3))

Theorem: a2-lc-s-mux
(\neg empty (x1)) \rightarrow (l (s-mux (x1, x2, x3)) = mux (l (x1), l (x2), l (x3)))

Theorem: a2-pc-s-mux
p (s-mux (x1, x2, x3)) = s-mux (p (x1), p (x2), p (x3))

Theorem: a2-hc-s-mux
((\neg empty (x1)) \land ((len (x1) = len (x2)) \land (len (x2) = len (x3))))
\rightarrow (h (s-mux (x1, x2, x3)) = mux (h (x1), h (x2), h (x3)))

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

Theorem: a2-bc-s-mux
((len (x1) = len (x2)) \land (len (x2) = len (x3)))
\rightarrow (b (s-mux (x1, x2, x3)) = s-mux (b (x1), b (x2), b (x3)))

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

Theorem: a2-bnc-s-mux
((len (x1) = len (x2)) \land (len (x2) = len (x3)))
\rightarrow (bn (n, s-mux (x1, x2, x3)) = s-mux (bn (n, x1), bn (n, x2), bn (n, x3)))

;old: ((DISABLE MUX) (ENABLE B LEN) (INDUCT (S-MUX X1 X2 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 sfix (x) endif

;; A2-Begin-SY-B

**Theorem:** a2-empty-sy-b
empty (sy-b (ln, x)) = empty (x)

**Theorem:** a2-e-sy-b
(sy-b (ln, x) = E) = empty (x)

**Theorem:** a2-lp-sy-b
len (sy-b (ln, x)) = len (x)

**Theorem:** a2-lpe-sy-b
eqlen (sy-b (ln, x), x)
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
empty(sy-c(ln, xst)) = empty(xst)

Theorem: a2-e-sy-c
(sy-c(ln, xst) = E) = empty(xst)

Theorem: a2-lp-sy-c
len(sy-c(ln, xst)) = len(xst)

Theorem: a2-lpe-sy-c
eqlen(sy-c(ln, xst), xst)
(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:

Definition::

stut-r (x, y)
  = if empty (y) then x
      elseif empty (p(y)) then b (x)
      elseif l (p(y)) then stut-r (x, p(y))
      else b (stut-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 (stut (p(x), p(y)), l (stut (p(x), p(y))))
      else a (stut (p(x), p(y)), h (stut-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:**
\[
\text{stut-induct}(x) = \begin{cases} 
    0 & \text{if } \text{empty}(x) \\
    1 & \text{if } \text{empty}(\text{p}(x)) \\
    \text{stut-induct}(\text{p}(x)) & \text{otherwise}
\end{cases}
\]

;; Properties of Stut:

**Theorem:** stut-empty
\[
\text{empty(stut}(x, y)) = \text{empty}(y)
\]

**Theorem:** stut-e
\[
\text{(stut}(x, y) = \text{E}) = \text{empty}(y)
\]

**Theorem:** stut-p
\[
\text{p(stut}(x, y)) = \text{stut}(\text{p}(x), \text{p}(y))
\]

;; 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
\[
\text{stut-r}(\text{E}, y) = \text{E}
\]

**Theorem:** stut-r-p
\[
\text{p(stut-r}(x, y)) = \text{stut-r}(\text{p}(x), y)
\]

**Theorem:** stut-r-len
\[
\text{len}(x) \leq (1 + (\text{len}(y) + \text{len}(\text{stut-r}(x, y))))
\]

**Theorem:** stut-r-not-empty
\[
(\text{len}(y) < \text{len}(x)) \rightarrow (\neg \text{empty(stut-r}(x, y)))
\]
; 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**

\[ \text{stut-rem}(y) = \begin{cases} 
\text{if} & \text{empty}(y) \text{ then } E \\
\text{elseif} & \text{empty}(p(y)) \text{ then } y \\
\text{elseif} & l(p(y)) \text{ then } \text{stut-rem}(p(y)) \\
\text{else} & y 
\end{cases} \]

**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

\[ (\neg \text{empty}(p(x)) \land 1(p(x))) \rightarrow (\text{len} (\text{stut-rem}(x)) < \text{len}(x)) \]

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

**Definition**

\[ \text{stut-num}(y) = \begin{cases} 
\text{if} & \text{empty}(y) \text{ then } 0 \\
\text{elseif} & \text{empty}(p(y)) \text{ then } 1 \\
\text{elseif} & l(p(y)) \text{ then } \text{stut-num}(p(y)) \\
\text{else} & 1 + \text{stut-num}(p(y)) 
\end{cases} \]

**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

\((\neg \text{empty}(y)) \land (\text{len}(x) \not< \text{len}(y)))\)

\(\rightarrow \quad (l(stut(x, y)) = h(stut-r(x, p(stut-rem(y)))))\)

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

**Theorem:** stut-inv0

\((\neg \text{empty}(y)) \land (\text{len}(x) \not< \text{len}(y)) \land (\neg l(y)))\)

\(\rightarrow \quad (l(stut(x, y)) = h(stut-r(x, p(stut-rem(y)))))\)

**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

\((\neg \text{empty}(x)) \land l(x))\)

\(\rightarrow \quad (\text{stut-num}(x) = (1 + \text{stut-num}(p(stut-rem(x)))))\)

**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)))))

11
; (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-En-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: stut-r-indstep
((¬ empty (y)) ∧ (¬ l(y)))
→ (stut-r(x, y) = b(stut-r(x, p(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 uncleanness 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...

BEGIN: ACTUAL Circuit CORRECTNESS modulo Stuttering.

REVERSAL PROPERTY for sy-a:

**Theorem**: sy-a-reversal
\[
\neg \text{empty}(\text{bn}(n, \text{sy-a}('yout, p(x)))) \implies (\text{h}(\text{b}(\text{bn}(n, \text{sy-a}('yout, x)))) = (1 + \text{h}(\text{bn}(n, \text{sy-a}('yout, p(x))))))
\]

**Theorem**: count-ac-l
\[1(\text{stut}(\text{sy-a}('yout, xst)), xst)) = 1(\text{sy-c}('yout, xst))\]

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
\[
\neg \text{empty}(x) \implies (\text{sy-c}('yout, x) = \text{a}(\text{p}(\text{sy-c}('yout, x)), 1(\text{sy-c}('yout, x))))
\]

**Theorem**: apl-split-stuta
\[
\neg \text{empty}(x) \implies (\text{stut}(\text{sy-a}('yout, x), x) = \text{a}(\text{p}(\text{stut}(\text{sy-a}('yout, x)), x), 1(\text{stut}(\text{sy-a}('yout, x), x))))
\]

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

\[
((\neg \text{empty}(x)) \land (\neg \text{empty}(p(x))) \land l(\text{sy-b}(\text{\textsc{yst}}, p(x))))
\rightarrow \ (l(\text{sy-b}(\text{\textsc{yout}}, x)) = l(\text{sy-b}(\text{\textsc{yout}}, p(x))))
\]

EVENT: Disable hcorr-ab-t.

; obviously..

**Theorem:** hcorr-ab-f

\[
((\neg \text{empty}(x)) \land (\neg \text{empty}(p(x))) \land (\neg l(\text{sy-b}(\text{\textsc{yst}}, p(x)))))
\rightarrow \ (l(\text{sy-b}(\text{\textsc{yout}}, x)) = (1 + l(\text{sy-b}(\text{\textsc{yout}}, p(x))))
\]

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

\[l(\text{stut}(\text{sy-a}(\text{\textsc{yout}}, x), \text{sy-b}(\text{\textsc{yst}}, x))) = l(\text{sy-b}(\text{\textsc{yout}}, x))\]

**Theorem:** apl-split-bout

\((\neg \text{empty}(x))\)
\rightarrow \ (\text{sy-b}(\text{\textsc{yout}}, x) = a(p(\text{sy-b}(\text{\textsc{yout}}, x)), l(\text{sy-b}(\text{\textsc{yout}}, x))))
\]

**Theorem:** apl-split-stuta2

\((\neg \text{empty}(x))\)
\rightarrow \ (\text{stut}(\text{sy-a}(\text{\textsc{yout}}, x), \text{sy-b}(\text{\textsc{yst}}, x)) = a(p(\text{stut}(\text{sy-a}(\text{\textsc{yout}}, x), \text{sy-b}(\text{\textsc{yst}}, x))), l(\text{stut}(\text{sy-a}(\text{\textsc{yout}}, x), \text{sy-b}(\text{\textsc{yst}}, x))))
\]

**Theorem:** count-ab-correct

\[\text{stut}(\text{sy-a}(\text{\textsc{yout}}, x), \text{sy-b}(\text{\textsc{yst}}, x)) = \text{sy-b}(\text{\textsc{yout}}, x)\]
;;; 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-le-s-inc, 3
a2-le-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

1, 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