```
#|
```



```
|#
```

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

```
; srccpu.bm is the BIG ONE!
;
; Notes:
; . As in pplfadd, some A2-PCs trigger a BM bug and blow up
;   (here A2-PC-SY-I) so we declare it as axiom (point that out in
;   Personal Confessions section in thesis).
; . Constants:  Initial PC (pci) has to be declared and axiomatized
;   as a numberp, because we do defined arithmetic (add1) on it.
;   Others: initial program and initial register file are quoted
;   constants, because only undefined (dcl) functions act on them.
; . We code "NoAd" (inexistant address) as F, because it makes life
```

```
;   easier by avoiding needless equalities around.  As a general
;   principle (at least in BM), boolean flags should be coded as
;   booleans in the model!  It also means that "small k" and
;   "big k" in Saxe's picture are now the same thing:
;   (k ctl leftinput) = (if ctl F leftinput).


;;; (Sugared) Circuits:
#|
(setq sy-S '(sy-S (x)
(Ypc R (pci) Ypcn)
(Ypcn S mux Ybc Ybt Ypcp1)
(Ypcp1 S inc Ypc)
(Ybc S eql0 Ywd)

(Ypr R 'pro Ypr)
(Ybt S Umb Ypc Ypr)
(Ywa S Umw Ypc Ypr)
(Yra S Umr Ypc Ypr)
(Yf S Umf Ypc Ypr)

(Ywd S Ualu Yf Yrd)
(Yrd S Ulk Yra Yrf)
(Yrf R 'rfi Yrfn)
(Yrfn S Upd Ywa Ywd Yrf)
))

; pipeline elements are indicated by visual offset (and digits):
; Initial values (originally inspired by Lamport):
;
; The branch target (bt) don't matter, they get 0
; The branch control (bc) DO matter, they must start w/ False: F
; The write address (wa) also matter, they must start w/ "NoAd": F

(setq sy-I '(sy-I (x)
(Ypc R (pci) Ypcn)
(Ypcn S mux Ybc0 Ybt0 Ypcp1)
  (Ybt0 R 0 Ybt1)
  (Ybt1 R 0 Ybt2)
  (Ybt2 R 0 Ybt3)
  (Ybt3 R 0 Ybt4)
  (Ybc0 R F Ybc1k)
  (Ybc1k S k Ybc0 Ybc1)
  (Ybc1 R F Ybc2k)
```

```
  (Ybc2k S k Ybc0 Ybc2)
  (Ybc2 R F Ybc3k)
  (Ybc3k S k Ybc0 Ybc3)
  (Ybc3 R F Ybc4k)
  (Ybc4k S k Ybc0 Ybc4)
(Ypcp1 S inc Ypc)
(Ybc4 S eql0 Ywd4)

(Ypr R 'pro Ypr)
(Ybt4 S Umb Ypc Ypr)
(Ywa4 S Umw Ypc Ypr)
(Yra S Umr Ypc Ypr)
(Yf S Umf Ypc Ypr)

(Ywd4 S Ualu Yf Yrd3)
  (Yrd3 S mux Yrb3 Ywd3 Yrd2)
  (Yrd2 S mux Yrb2 Ywd2 Yrd1)
  (Yrd1 S mux Yrb1 Ywd1 Yrd)
  (Yrb3 S equal Yra Ywa3)
  (Yrb2 S equal Yra Ywa2)
  (Yrb1 S equal Yra Ywa1)
  (Ywa3 R F Ywa4k)
  (Ywa2 R F Ywa3k)
  (Ywa1 R F Ywa2k)
  (Ywa4k S k Ybc0 Ywa4)
  (Ywa3k S k Ybc0 Ywa3)
  (Ywa2k S k Ybc0 Ywa2)
  (Ywa1k S k Ybc0 Ywa1)
(Yrd S Ulk Yra Yrf)
(Yrf R 'rfi Yrfn)
(Yrfn S Upd Ywa1k Ywd1 Yrf)
  (Ywd1 R 0 Ywd2)
  (Ywd2 R 0 Ywd3)
  (Ywd3 R 0 Ywd4)

))

(setq srccpu '( |#
; Constants are hand-declared and axiomatized:
```

EVENT: Introduce the function symbol *pci* of 0 arguments.

AXIOM: pci-numberp

PCI $\in \mathbf{N}$

```
; BM DEFINITIONS and A2 LEMMAS, generated by BMSYSD:
; comb_mux.bm: Mux combinational element, i.e. "if".
; U7-DONE
```

DEFINITION:
$\text{mux}\,(u1,\ u2,\ u3)$
$=\ $ **if** $u1$ **then** $u2$
else $u3$ **endif**

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

DEFINITION:
$\text{s-mux}\,(x1,\ x2,\ x3)$
$=\ $ **if** $\text{empty}\,(x1)$ **then** E
else $\text{a}\,(\text{s-mux}\,(\text{p}\,(x1),\ \text{p}\,(x2),\ \text{p}\,(x3)),\ \text{mux}\,(\text{l}\,(x1),\ \text{l}\,(x2),\ \text{l}\,(x3)))$ **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
$(\text{s-mux}\,(x1,\ x2,\ x3) = \text{E}) = \text{empty}\,(x1)$

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

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

THEOREM: a2-ic-s-mux
((len ($x1$) = len ($x2$)) ∧ (len ($x2$) = len ($x3$)))
→ (s-mux (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
(¬ empty ($x1$)) → (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
((¬ empty ($x1$)) ∧ ((len ($x1$) = len ($x2$)) ∧ (len ($x2$) = len ($x3$))))
→ (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 ($x2$)) ∧ (len ($x2$) = len ($x3$)))
→ (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 ($x2$)) ∧ (len ($x2$) = len ($x3$)))
→ (bn (n, s-mux ($x1$, $x2$, $x3$)) = s-mux (bn (n, $x1$), bn (n, $x2$), bn (n, $x3$)))

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

;; A2-End-S-MUX

; eof:comb_mux.bm

; 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
len (s-inc (x)) $=$ len (x)

THEOREM: a2-lpe-s-inc
eqlen (s-inc (x), x)

THEOREM: a2-ic-s-inc
s-inc (i $(c\_x,\ x)$) $=$ i (inc (c_x), s-inc (x))

THEOREM: a2-lc-s-inc
$(\neg$ empty $(x)) \rightarrow$ (l (s-inc (x)) $=$ inc (l (x)))

THEOREM: a2-pc-s-inc
p (s-inc (x)) $=$ s-inc (p (x))

THEOREM: a2-hc-s-inc
$(\neg$ empty $(x)) \rightarrow$ (h (s-inc (x)) $=$ inc (h (x)))

THEOREM: a2-bc-s-inc
b (s-inc (x)) $=$ s-inc (b (x))

THEOREM: a2-bnc-s-inc
bn (n, s-inc (x)) $=$ s-inc (bn $(n,\ x)$)

```
;; A2-End-S-INC

; eof:comb_inc.bm

; Comb_eql0.bm: equal to 0 test, without zerop (in contrast to eq0).
; U7-DONE
```

DEFINITION: eql0 $(u) = (u = 0)$

```
; Everything below generated by: (bmcomb 'eql0 '() '(x))
```

DEFINITION:
s-eql0 (x)
$=$ **if** empty (x) **then** E
else a (s-eql0 (p (x)), eql0 (l (x))) **endif**

```
;; A2-Begin-S-EQL0
```

THEOREM: a2-empty-s-eql0
empty (s-eql0 (x)) $=$ empty (x)

THEOREM: a2-e-s-eql0
(s-eql0 (x) $=$ E) $=$ empty (x)

THEOREM: a2-lp-s-eql0
len (s-eql0 (x)) $=$ len (x)

THEOREM: a2-lpe-s-eql0
eqlen (s-eql0 (x), x)

THEOREM: a2-ic-s-eql0
s-eql0 (i (c_x, x)) $=$ i (eql0 (c_x), s-eql0 (x))

THEOREM: a2-lc-s-eql0
($\neg$ empty (x)) $\rightarrow$ (l (s-eql0 (x)) $=$ eql0 (l (x)))

THEOREM: a2-pc-s-eql0
p (s-eql0 (x)) $=$ s-eql0 (p (x))

THEOREM: a2-hc-s-eql0
($\neg$ empty (x)) $\rightarrow$ (h (s-eql0 (x)) $=$ eql0 (h (x)))

THEOREM: a2-bc-s-eql0
b (s-eql0 (x)) $=$ s-eql0 (b (x))

THEOREM: a2-bnc-s-eql0
bn (n, s-eql0 (x)) $=$ s-eql0 (bn (n, x))

```
;; A2-End-S-EQL0

; eof:comb_eq0.bm

; comb_umb.bm: Umb combinational element (= fun2)
; U7-DONE

; arbitrary Char-Fun of arity 2:
```

EVENT: Introduce the function symbol *umb* of 2 arguments.

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

DEFINITION:
s-umb $(x,\ y)$
$=$ **if** empty (x) **then** E
else a (s-umb (p (x), p (y)), umb (l (x), l (y))) **endif**

```
;; A2-Begin-S-UMB
```

THEOREM: a2-empty-s-umb
empty (s-umb $(x,\ y)$) $=$ empty (x)

THEOREM: a2-e-s-umb
(s-umb $(x,\ y) =$ E) $=$ empty (x)

THEOREM: a2-lp-s-umb
len (s-umb $(x,\ y)$) $=$ len (x)

THEOREM: a2-lpe-s-umb
eqlen (s-umb $(x,\ y)$, x)

THEOREM: a2-ic-s-umb
(len $(x) =$ len (y))
$\rightarrow$ (s-umb (i $(c\_x,\ x)$, i $(c\_y,\ y)$) $=$ i (umb $(c\_x,\ c\_y)$, s-umb $(x,\ y)$))

THEOREM: a2-lc-s-umb
($\neg$ empty (x)) $\rightarrow$ (l (s-umb $(x,\ y)$) $=$ umb (l (x), l (y)))

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

THEOREM: a2-hc-s-umb
(($\neg$ empty (x)) $\wedge$ (len $(x) =$ len (y)))
$\rightarrow$ (h (s-umb $(x,\ y)$) $=$ umb (h (x), h (y)))

THEOREM: a2-bc-s-umb
(len $(x) =$ len (y)) $\rightarrow$ (b (s-umb $(x,\ y)$) $=$ s-umb (b (x), b (y)))

THEOREM: a2-bnc-s-umb
(len $(x) =$ len (y)) $\rightarrow$ (bn $(n,$ s-umb $(x,\ y))$ $=$ s-umb (bn $(n,\ x)$, bn $(n,\ y)$))

```
;; A2-End-S-UMB

; eof:comb_umb.bm

; comb_umw.bm: Umw combinational element (= fun2)
; U7-DONE

; arbitrary Char-Fun of arity 2:
```

EVENT: Introduce the function symbol *umw* of 2 arguments.

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

DEFINITION:
s-umw $(x,\ y)$
$=$ **if** empty (x) **then** E
else a (s-umw (p (x), p (y)), umw (l (x), l (y))) **endif**

```
;; A2-Begin-S-UMW
```

THEOREM: a2-empty-s-umw
empty (s-umw $(x,\ y)$) = empty (x)

THEOREM: a2-e-s-umw
(s-umw $(x,\ y)$ = E) = empty (x)

THEOREM: a2-lp-s-umw
len (s-umw $(x,\ y)$) = len (x)

THEOREM: a2-lpe-s-umw
eqlen (s-umw $(x,\ y)$, x)

THEOREM: a2-ic-s-umw
(len (x) = len (y))
$\rightarrow$ (s-umw (i $(c\_x,\ x)$, i $(c\_y,\ y)$) = i (umw $(c\_x,\ c\_y)$, s-umw $(x,\ y)$))

THEOREM: a2-lc-s-umw
($\neg$ empty (x)) $\rightarrow$ (l (s-umw $(x,\ y)$) = umw (l (x), l (y)))

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

THEOREM: a2-hc-s-umw
(($\neg$ empty (x)) $\wedge$ (len (x) = len (y)))
$\rightarrow$ (h (s-umw $(x,\ y)$) = umw (h (x), h (y)))

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

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

```
;; A2-End-S-UMW

; eof:comb_umw.bm

; comb_umr.bm: Umr combinational element (= fun2)
; U7-DONE

; arbitrary Char-Fun of arity 2:
```

EVENT: Introduce the function symbol *umr* of 2 arguments.

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

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

```
;; A2-Begin-S-UMR
```

THEOREM: a2-empty-s-umr
$\text{empty}(\text{s-umr}(x, y)) = \text{empty}(x)$

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

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

THEOREM: a2-lpe-s-umr
$\text{eqlen}(\text{s-umr}(x, y), x)$

THEOREM: a2-ic-s-umr
$(\text{len}(x) = \text{len}(y))$
$\rightarrow (\text{s-umr}(\text{i}(c\_x, x), \text{i}(c\_y, y)) = \text{i}(\text{umr}(c\_x, c\_y), \text{s-umr}(x, y)))$

THEOREM: a2-lc-s-umr
$(\neg \text{empty}(x)) \rightarrow (\text{l}(\text{s-umr}(x, y)) = \text{umr}(\text{l}(x), \text{l}(y)))$

THEOREM: a2-pc-s-umr
$\mathrm{p}(\text{s-umr}(x,\ y)) = \text{s-umr}(\mathrm{p}(x),\ \mathrm{p}(y))$

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

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

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

```
;; A2-End-S-UMR

; eof:comb_umr.bm

; comb_umf.bm: Umf combinational element (= fun2)
; U7-DONE

; arbitrary Char-Fun of arity 2:
```

EVENT: Introduce the function symbol *umf* of 2 arguments.

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

DEFINITION:
$\text{s-umf}(x,\ y)$
$=\quad$ **if** $\text{empty}(x)$ **then** E
else $\mathrm{a}(\text{s-umf}(\mathrm{p}(x),\ \mathrm{p}(y)),\ \text{umf}(\mathrm{l}(x),\ \mathrm{l}(y)))$ **endif**

```
;; A2-Begin-S-UMF
```

THEOREM: a2-empty-s-umf
$\text{empty}(\text{s-umf}(x,\ y)) = \text{empty}(x)$

THEOREM: a2-e-s-umf
$(\text{s-umf}(x,\ y) = \mathrm{E}) = \text{empty}(x)$

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

THEOREM: a2-lpe-s-umf
$\text{eqlen}(\text{s-umf}(x,\ y),\ x)$

THEOREM: a2-ic-s-umf
$(\text{len}(x) = \text{len}(y))$
$\rightarrow \quad (\text{s-umf}(\text{i}(c\_x,\ x),\ \text{i}(c\_y,\ y)) = \text{i}(\text{umf}(c\_x,\ c\_y),\ \text{s-umf}(x,\ y)))$

THEOREM: a2-lc-s-umf
$(\neg\ \text{empty}(x)) \rightarrow (\text{l}(\text{s-umf}(x,\ y)) = \text{umf}(\text{l}(x),\ \text{l}(y)))$

THEOREM: a2-pc-s-umf
$\text{p}(\text{s-umf}(x,\ y)) = \text{s-umf}(\text{p}(x),\ \text{p}(y))$

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

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

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

```
;; A2-End-S-UMF

; eof:comb_umf.bm

; comb_ualu.bm: Ualu combinational element
; U7-DONE

; arbitrary Char-Fun of arity 2:
```

EVENT: Introduce the function symbol *ualu* of 2 arguments.

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

DEFINITION:
$\text{s-ualu}(x,\ y)$
$= \quad$ **if** $\text{empty}(x)$ **then** E
else $\text{a}(\text{s-ualu}(\text{p}(x),\ \text{p}(y)),\ \text{ualu}(\text{l}(x),\ \text{l}(y)))$ **endif**

```
;; A2-Begin-S-UALU
```

THEOREM: a2-empty-s-ualu
$\text{empty}(\text{s-ualu}(x,\ y)) = \text{empty}(x)$

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

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

THEOREM: a2-lpe-s-ualu
$\text{eqlen}(\text{s-ualu}(x,\ y),\ x)$

THEOREM: a2-ic-s-ualu
$(\text{len}(x) = \text{len}(y))$
$\to\quad (\text{s-ualu}(\text{i}(c\_x,\ x),\ \text{i}(c\_y,\ y)) = \text{i}(\text{ualu}(c\_x,\ c\_y),\ \text{s-ualu}(x,\ y)))$

THEOREM: a2-lc-s-ualu
$(\neg\ \text{empty}(x)) \to (\text{l}(\text{s-ualu}(x,\ y)) = \text{ualu}(\text{l}(x),\ \text{l}(y)))$

THEOREM: a2-pc-s-ualu
$\text{p}(\text{s-ualu}(x,\ y)) = \text{s-ualu}(\text{p}(x),\ \text{p}(y))$

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

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

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

```
;; A2-End-S-UALU
```

```
; eof:comb_ualu.bm
```

```
; comb_ulk.bm: Ulk combinational element
; U7-DONE
```

```
; arbitrary Char-Fun of arity 2:
```

EVENT: Introduce the function symbol *ulk* of 2 arguments.

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

DEFINITION:
$\text{s-ulk}(x,\ y)$
$= \quad$ **if** $\text{empty}(x)$ **then** E
else $\text{a}(\text{s-ulk}(\text{p}(x),\ \text{p}(y)),\ \text{ulk}(\text{l}(x),\ \text{l}(y)))$ **endif**

```
;; A2-Begin-S-ULK
```

THEOREM: a2-empty-s-ulk
empty (s-ulk $(x,\ y)$) = empty (x)

THEOREM: a2-e-s-ulk
(s-ulk $(x,\ y)$ = E) = empty (x)

THEOREM: a2-lp-s-ulk
len (s-ulk $(x,\ y)$) = len (x)

THEOREM: a2-lpe-s-ulk
eqlen (s-ulk $(x,\ y)$, x)

THEOREM: a2-ic-s-ulk
(len (x) = len (y))
→ (s-ulk (i (c_x, x), i (c_y, y)) = i (ulk (c_x, c_y), s-ulk $(x,\ y)$))

THEOREM: a2-lc-s-ulk
(¬ empty (x)) → (l (s-ulk $(x,\ y)$) = ulk (l (x), l (y)))

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

THEOREM: a2-hc-s-ulk
((¬ empty (x)) ∧ (len (x) = len (y)))
→ (h (s-ulk $(x,\ y)$) = ulk (h (x), h (y)))

THEOREM: a2-bc-s-ulk
(len (x) = len (y)) → (b (s-ulk $(x,\ y)$) = s-ulk (b (x), b (y)))

THEOREM: a2-bnc-s-ulk
(len (x) = len (y)) → (bn (n, s-ulk $(x,\ y)$) = s-ulk (bn (n, x), bn (n, y)))

```
;; A2-End-S-ULK

; eof:comb_ulk.bm

; comb_upd.bm: Upd combinational element
; U7-DONE

; arbitrary Char-Fun of arity 3:
```

EVENT: Introduce the function symbol *upd* of 3 arguments.

```
; AXIOM characterising Upd:  (or at least some): see _s .

; Everything below generated by: (bmcomb 'Upd '() '(x1 x2 x3))
```

DEFINITION:
s-upd $(x1, x2, x3)$
$=$ **if** empty $(x1)$ **then** E
else a (s-upd (p $(x1)$, p $(x2)$, p $(x3)$), upd (l $(x1)$, l $(x2)$, l $(x3)$)) **endif**

```
;; A2-Begin-S-UPD
```

THEOREM: a2-empty-s-upd
empty (s-upd $(x1, x2, x3)$) $=$ empty $(x1)$

THEOREM: a2-e-s-upd
(s-upd $(x1, x2, x3)$ $=$ E) $=$ empty $(x1)$

THEOREM: a2-lp-s-upd
len (s-upd $(x1, x2, x3)$) $=$ len $(x1)$

THEOREM: a2-lpe-s-upd
eqlen (s-upd $(x1, x2, x3)$, $x1$)

THEOREM: a2-ic-s-upd
((len $(x1)$ $=$ len $(x2)$) $\wedge$ (len $(x2)$ $=$ len $(x3)$))
$\rightarrow$ (s-upd (i $(c\_x1, x1)$, i $(c\_x2, x2)$, i $(c\_x3, x3)$)
$=$ i (upd (c_x1, c_x2, c_x3), s-upd $(x1, x2, x3)$))

THEOREM: a2-lc-s-upd
($\neg$ empty $(x1)$) $\rightarrow$ (l (s-upd $(x1, x2, x3)$) $=$ upd (l $(x1)$, l $(x2)$, l $(x3)$))

THEOREM: a2-pc-s-upd
p (s-upd $(x1, x2, x3)$) $=$ s-upd (p $(x1)$, p $(x2)$, p $(x3)$)

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

THEOREM: a2-bc-s-upd
((len $(x1)$ $=$ len $(x2)$) $\wedge$ (len $(x2)$ $=$ len $(x3)$))
$\rightarrow$ (b (s-upd $(x1, x2, x3)$) $=$ s-upd (b $(x1)$, b $(x2)$, b $(x3)$))

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

```
;; A2-End-S-UPD

; eof:comb_upd.bm
```

DEFINITION:

topor-sy-s (ln)

$=$ **if** ln = `'ypc` **then** `0`
elseif ln = `'ypcn` **then** `5`
elseif ln = `'ypcp1` **then** `1`
elseif ln = `'ybc` **then** `4`
elseif ln = `'ypr` **then** `0`
elseif ln = `'ybt` **then** `1`
elseif ln = `'ywa` **then** `1`
elseif ln = `'yra` **then** `1`
elseif ln = `'yf` **then** `1`
elseif ln = `'ywd` **then** `3`
elseif ln = `'yrd` **then** `2`
elseif ln = `'yrf` **then** `0`
elseif ln = `'yrfn` **then** `4`
else `0` **endif**

DEFINITION:

sy-s (ln, x)

$=$ **if** ln = `'ypc`
then if empty (x) **then** E
else i (PCI, sy-s (`'ypcn`, p (x))) **endif**
elseif ln = `'ypcn`
then s-mux (sy-s (`'ybc`, x), sy-s (`'ybt`, x), sy-s (`'ypcp1`, x))
elseif ln = `'ypcp1` **then** s-inc (sy-s (`'ypc`, x))
elseif ln = `'ybc` **then** s-eql0 (sy-s (`'ywd`, x))
elseif ln = `'ypr`
then if empty (x) **then** E
else i (`'pro`, sy-s (`'ypr`, p (x))) **endif**
elseif ln = `'ybt` **then** s-umb (sy-s (`'ypc`, x), sy-s (`'ypr`, x))
elseif ln = `'ywa` **then** s-umw (sy-s (`'ypc`, x), sy-s (`'ypr`, x))
elseif ln = `'yra` **then** s-umr (sy-s (`'ypc`, x), sy-s (`'ypr`, x))
elseif ln = `'yf` **then** s-umf (sy-s (`'ypc`, x), sy-s (`'ypr`, x))
elseif ln = `'ywd` **then** s-ualu (sy-s (`'yf`, x), sy-s (`'yrd`, x))
elseif ln = `'yrd` **then** s-ulk (sy-s (`'yra`, x), sy-s (`'yrf`, x))
elseif ln = `'yrf`
then if empty (x) **then** E
else i (`'rfi`, sy-s (`'yrfn`, p (x))) **endif**

elseif ln = `'yrfn`
then s-upd (sy-s (`'ywa`, x), sy-s (`'ywd`, x), sy-s (`'yrf`, x))
else sfix (x) **endif**

```
;; A2-Begin-SY-S
```

THEOREM: a2-empty-sy-s
empty (sy-s (ln, x)) = empty (x)

THEOREM: a2-e-sy-s
(sy-s (ln, x) = E) = empty (x)

THEOREM: a2-lp-sy-s
len (sy-s (ln, x)) = len (x)

THEOREM: a2-lpe-sy-s
eqlen (sy-s (ln, x), x)

THEOREM: a2-pc-sy-s
p (sy-s (ln, x)) = sy-s (ln, p (x))

```
;; A2-End-SY-S
; BM DEFINITIONS and A2 LEMMAS, generated by BMSYSD:
; comb_k.bm: K combinational element. Semantics by Saxe in rn395 Notes 3.
; Our "k" is his "k" (i.e. little k).
; U7-DONE
```

DEFINITION:
k (u, v)
= **if** u **then** **f**
else v **endif**

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

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

```
;; A2-Begin-S-K
```

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

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

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

THEOREM: a2-lpe-s-k
$\text{eqlen}(\text{s-k}(x,\ y),\ x)$

THEOREM: a2-ic-s-k
$(\text{len}(x) = \text{len}(y)) \rightarrow (\text{s-k}(\text{i}(c\_x,\ x),\ \text{i}(c\_y,\ y)) = \text{i}(\text{k}(c\_x,\ c\_y),\ \text{s-k}(x,\ y)))$

THEOREM: a2-lc-s-k
$(\neg\ \text{empty}(x)) \rightarrow (\text{l}(\text{s-k}(x,\ y)) = \text{k}(\text{l}(x),\ \text{l}(y)))$

THEOREM: a2-pc-s-k
$\text{p}(\text{s-k}(x,\ y)) = \text{s-k}(\text{p}(x),\ \text{p}(y))$

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

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

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

```
;; A2-End-S-K
```

```
; eof:comb_k.bm
```

DEFINITION:
topor-sy-i (ln)
= **if** ln = `'ypc` **then** `0`
elseif ln = `'ypcn` **then** `2`
elseif ln = `'ybt0` **then** `0`
elseif ln = `'ybt1` **then** `0`
elseif ln = `'ybt2` **then** `0`
elseif ln = `'ybt3` **then** `0`
elseif ln = `'ybc0` **then** `0`
elseif ln = `'ybc1k` **then** `1`
elseif ln = `'ybc1` **then** `0`
elseif ln = `'ybc2k` **then** `1`
elseif ln = `'ybc2` **then** `0`

elseif ln = `'ybc3k` **then** `1`
elseif ln = `'ybc3` **then** `0`
elseif ln = `'ybc4k` **then** `8`
elseif ln = `'ypcp1` **then** `1`
elseif ln = `'ybc4` **then** `7`
elseif ln = `'ypr` **then** `0`
elseif ln = `'ybt4` **then** `1`
elseif ln = `'ywa4` **then** `1`
elseif ln = `'yra` **then** `1`
elseif ln = `'yf` **then** `1`
elseif ln = `'ywd4` **then** `6`
elseif ln = `'yrd3` **then** `5`
elseif ln = `'yrd2` **then** `4`
elseif ln = `'yrd1` **then** `3`
elseif ln = `'yrb3` **then** `2`
elseif ln = `'yrb2` **then** `2`
elseif ln = `'yrb1` **then** `2`
elseif ln = `'ywa3` **then** `0`
elseif ln = `'ywa2` **then** `0`
elseif ln = `'ywa1` **then** `0`
elseif ln = `'ywa4k` **then** `2`
elseif ln = `'ywa3k` **then** `1`
elseif ln = `'ywa2k` **then** `1`
elseif ln = `'ywa1k` **then** `1`
elseif ln = `'yrd` **then** `2`
elseif ln = `'yrf` **then** `0`
elseif ln = `'yrfn` **then** `2`
elseif ln = `'ywd1` **then** `0`
elseif ln = `'ywd2` **then** `0`
elseif ln = `'ywd3` **then** `0`
else `0` **endif**

DEFINITION:
sy-i (ln, x)
= **if** ln = `'ypc`
then if empty (x) **then** E
else i (PCI, sy-i (`'ypcn`, p (x))) **endif**
elseif ln = `'ypcn`
then s-mux (sy-i (`'ybc0`, x), sy-i (`'ybt0`, x), sy-i (`'ypcp1`, x))
elseif ln = `'ybt0`
then if empty (x) **then** E
else i (`0`, sy-i (`'ybt1`, p (x))) **endif**
elseif ln = `'ybt1`
then if empty (x) **then** E

```
        else i (0, sy-i ('ybt2, p (x))) endif
elseif ln = 'ybt2
then if empty (x)  then E
        else i (0, sy-i ('ybt3, p (x))) endif
elseif ln = 'ybt3
then if empty (x)  then E
        else i (0, sy-i ('ybt4, p (x))) endif
elseif ln = 'ybc0
then if empty (x)  then E
        else i (f, sy-i ('ybc1k, p (x))) endif
elseif ln = 'ybc1k  then s-k (sy-i ('ybc0, x), sy-i ('ybc1, x))
elseif ln = 'ybc1
then if empty (x)  then E
        else i (f, sy-i ('ybc2k, p (x))) endif
elseif ln = 'ybc2k  then s-k (sy-i ('ybc0, x), sy-i ('ybc2, x))
elseif ln = 'ybc2
then if empty (x)  then E
        else i (f, sy-i ('ybc3k, p (x))) endif
elseif ln = 'ybc3k  then s-k (sy-i ('ybc0, x), sy-i ('ybc3, x))
elseif ln = 'ybc3
then if empty (x)  then E
        else i (f, sy-i ('ybc4k, p (x))) endif
elseif ln = 'ybc4k  then s-k (sy-i ('ybc0, x), sy-i ('ybc4, x))
elseif ln = 'ypcp1  then s-inc (sy-i ('ypc, x))
elseif ln = 'ybc4  then s-eql0 (sy-i ('ywd4, x))
elseif ln = 'ypr
then if empty (x)  then E
        else i ('pro, sy-i ('ypr, p (x))) endif
elseif ln = 'ybt4  then s-umb (sy-i ('ypc, x), sy-i ('ypr, x))
elseif ln = 'ywa4  then s-umw (sy-i ('ypc, x), sy-i ('ypr, x))
elseif ln = 'yra  then s-umr (sy-i ('ypc, x), sy-i ('ypr, x))
elseif ln = 'yf  then s-umf (sy-i ('ypc, x), sy-i ('ypr, x))
elseif ln = 'ywd4  then s-ualu (sy-i ('yf, x), sy-i ('yrd3, x))
elseif ln = 'yrd3
then s-mux (sy-i ('yrb3, x), sy-i ('ywd3, x), sy-i ('yrd2, x))
elseif ln = 'yrd2
then s-mux (sy-i ('yrb2, x), sy-i ('ywd2, x), sy-i ('yrd1, x))
elseif ln = 'yrd1
then s-mux (sy-i ('yrb1, x), sy-i ('ywd1, x), sy-i ('yrd, x))
elseif ln = 'yrb3  then s-equal (sy-i ('yra, x), sy-i ('ywa3, x))
elseif ln = 'yrb2  then s-equal (sy-i ('yra, x), sy-i ('ywa2, x))
elseif ln = 'yrb1  then s-equal (sy-i ('yra, x), sy-i ('ywa1, x))
elseif ln = 'ywa3
then if empty (x)  then E
```

else i (**f**, sy-i ('ywa4k, p (x))) **endif**
elseif ln = 'ywa2
then if empty (x) **then** E
else i (**f**, sy-i ('ywa3k, p (x))) **endif**
elseif ln = 'ywa1
then if empty (x) **then** E
else i (**f**, sy-i ('ywa2k, p (x))) **endif**
elseif ln = 'ywa4k **then** s-k (sy-i ('ybc0, x), sy-i ('ywa4, x))
elseif ln = 'ywa3k **then** s-k (sy-i ('ybc0, x), sy-i ('ywa3, x))
elseif ln = 'ywa2k **then** s-k (sy-i ('ybc0, x), sy-i ('ywa2, x))
elseif ln = 'ywa1k **then** s-k (sy-i ('ybc0, x), sy-i ('ywa1, x))
elseif ln = 'yrd **then** s-ulk (sy-i ('yra, x), sy-i ('yrf, x))
elseif ln = 'yrf
then if empty (x) **then** E
else i ('rfi, sy-i ('yrfn, p (x))) **endif**
elseif ln = 'yrfn
then s-upd (sy-i ('ywa1k, x), sy-i ('ywd1, x), sy-i ('yrf, x))
elseif ln = 'ywd1
then if empty (x) **then** E
else i (0, sy-i ('ywd2, p (x))) **endif**
elseif ln = 'ywd2
then if empty (x) **then** E
else i (0, sy-i ('ywd3, p (x))) **endif**
elseif ln = 'ywd3
then if empty (x) **then** E
else i (0, sy-i ('ywd4, p (x))) **endif**
else sfix (x) **endif**

```
;; A2-Begin-SY-I
```

THEOREM: a2-empty-sy-i
empty (sy-i (ln, x)) = empty (x)

THEOREM: a2-e-sy-i
(sy-i (ln, x) = E) = empty (x)

THEOREM: a2-lp-sy-i
len (sy-i (ln, x)) = len (x)

THEOREM: a2-lpe-sy-i
eqlen (sy-i (ln, x), x)

```
; blows up, as usual.
;(PROVE-LEMMA A2-PC-SY-I (REWRITE)
```

```
;    (EQUAL (P (SY-I LN X)) (SY-I LN (P X)))
;    ((DISABLE S-INC S-EQL0 S-UMB S-UMW S-UMR S-UMF S-UALU S-MUX S-EQUAL
;              S-K S-ULK S-UPD A2-IC-S-INC A2-IC-S-EQL0 A2-IC-S-UMB
;              A2-IC-S-UMW A2-IC-S-UMR A2-IC-S-UMF A2-IC-S-UALU
;              A2-IC-S-MUX A2-IC-S-EQUAL A2-IC-S-K A2-IC-S-ULK
;              A2-IC-S-UPD)))
```

AXIOM: a2-pc-sy-i
$\mathrm{p}\,(\text{sy-i}\,(ln,\ x)) = \text{sy-i}\,(ln,\ \mathrm{p}\,(x))$

```
;; A2-End-SY-I


; CORRECTNESS PROOF:

;;; 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 $(\mathrm{p}\,(y))$ **then** $\mathrm{b}\,(x)$
elseif $\mathrm{l}\,(\mathrm{p}\,(y))$ **then** stut-r $(x,\ \mathrm{p}\,(y))$
else $\mathrm{b}\,(\text{stut-r}\,(x,\ \mathrm{p}\,(y)))$ **endif**

DEFINITION:
stut $(x,\ y)$
= **if** empty (y) **then** E
elseif empty $(\mathrm{p}\,(y))$ **then** $\mathrm{a}\,(\mathrm{E},\ \mathrm{h}\,(x))$
elseif $\mathrm{l}\,(\mathrm{p}\,(y))$ **then** $\mathrm{a}\,(\text{stut}\,(\mathrm{p}\,(x),\ \mathrm{p}\,(y)),\ \mathrm{l}\,(\text{stut}\,(\mathrm{p}\,(x),\ \mathrm{p}\,(y))))$
else $\mathrm{a}\,(\text{stut}\,(\mathrm{p}\,(x),\ \mathrm{p}\,(y)),\ \mathrm{h}\,(\text{stut-r}\,(x,\ \mathrm{p}\,(y))))$ **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 $(\mathrm{p}(x))$ **then** `1`
else stut-induct $(\mathrm{p}(x))$ **endif**

```
;; Properties of Stut:
```

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

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

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

```
;; 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
$\text{stut-r}(\mathrm{E},\ y) = \mathrm{E}$

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

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 $(\mathrm{p}(y))$ **then** y
elseif $\mathrm{l}(\mathrm{p}(y))$ **then** stut-rem $(\mathrm{p}(y))$
else y **endif**

THEOREM: stut-rem-empty
empty (stut-rem (x)) $=$ empty (x)

THEOREM: stut-rem-len
len (stut-rem (x)) $<$ $(1 +$ len $(x))$

THEOREM: stut-rem-len2
$((\neg$ empty $(\mathrm{p}(x))) \wedge \mathrm{l}(\mathrm{p}(x))) \rightarrow$ (len (stut-rem (x)) $<$ 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 $(\mathrm{p}(y))$ **then** 1
elseif $\mathrm{l}(\mathrm{p}(y))$ **then** stut-num $(\mathrm{p}(y))$
else $1 +$ stut-num $(\mathrm{p}(y))$ **endif**

THEOREM: stut-num-lessp
stut-num (x) $<$ $(1 +$ len $(x))$

THEOREM: stut-num-eq-0
(stut-num $(x) = 0$) $=$ empty (x)

```
; Requires a small induction.
```

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

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

THEOREM: stut-r-closed
stut-r (x, y) $=$ bn (stut-num (y), x)

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

THEOREM: stut-inv
$((\neg \text{ empty}(y)) \wedge (\text{len}(x) \not< \text{len}(y)))$
$\rightarrow \quad (\text{l}(\text{stut}(x, y)) = \text{h}(\text{stut-r}(x, \text{p}(\text{stut-rem}(y)))))$

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

THEOREM: stut-inv0
$((\neg \text{ empty}(y)) \wedge (\text{len}(x) \not< \text{len}(y)) \wedge (\neg \text{ l}(y)))$
$\rightarrow \quad (\text{l}(\text{stut}(x, y)) = \text{h}(\text{stut-r}(x, \text{p}(\text{stut-rem}(y)))))$

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

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))))
```

```
; (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 explicitely.


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

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

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 occurences.
;   This is neither clear nor robust...

;; eof: th_stutter.bm


;;; ALL DOWN TO HERE IN SRCDEF.{LIB,LISP}

;;; The circuit is mostly undefined.  Here are the axioms which
;;; give it its meaning:

; Umb-numberp: all branch targets are numbers; necessary since we
; do Add1 and Sub1 on them, and want that to work out.
```

AXIOM: umb-numberp
$\mathrm{umb}\,(x,\, y) \in \mathbf{N}$

```
; NoAd axiomatization:
; Supplying NoAddress (here: F) as the address to Wsel (here: Upd)
; leaves the RFile unchanged.  NoAd is never issued by the memory
; either as a Write or as a Read:
```

AXIOM: upd-noad
$\mathrm{upd}\,(\mathbf{f},\, x,\, y) = y$

AXIOM: umw-noad
$\mathrm{umw}\,(x,\, y) \neq \mathbf{f}$

AXIOM: umr-noad
$\mathrm{umr}\,(x,\, y) \neq \mathbf{f}$

```
; axiom to characterize Update and Lookup (with any non-F address
; valid):
; Note that if we don't use the hypothesis restricting the address,
; we get an inconsistency in the theory!
```

AXIOM: ulk-upd
$(ad1 \neq \mathbf{f})$
$\rightarrow \quad (\text{ulk}\,(ad1,\ \text{upd}\,(ad2,\ dat,\ rf))$
$\quad = \quad \textbf{if}\ ad1 = ad2\ \ \textbf{then}\ dat$
$\quad\quad \textbf{else}\ \text{ulk}\,(ad1,\ rf)\ \textbf{endif})$

```
;;; Miscellaneous facts, simplifications, etc... about the
;;; circuits:

; Corr-S-PR expresses that the program (Ypr) is constant in S:
```

THEOREM: corr-s-pr
$(\neg\ \text{empty}\,(x)) \rightarrow (\text{l}\,(\text{sy-s}\,(\texttt{'ypr},\ x)) = \texttt{'pro})$

```
; Corr-I-PR expresses that the program (Ypr) is constant in I:
```

THEOREM: corr-i-pr
$(\neg\ \text{empty}\,(x)) \rightarrow (\text{l}\,(\text{sy-i}\,(\texttt{'ypr},\ x)) = \texttt{'pro})$

```
;;; Stuttering Correctness:

; StutCorr-rf-T expresses correctness WHEN stuttering, and only
; Upd-NoAd is needed in order to prove it.
```

THEOREM: stutcorr-rf-t
$((\neg\ \text{empty}\,(x)) \wedge (\neg\ \text{empty}\,(\text{p}\,(x))) \wedge (\neg\ \text{l}\,(\text{sy-i}\,(\texttt{'ywa1k},\ \text{p}\,(x)))))$
$\rightarrow \quad (\text{l}\,(\text{sy-i}\,(\texttt{'yrf},\ x)) = \text{l}\,(\text{sy-i}\,(\texttt{'yrf},\ \text{p}\,(x))))$

EVENT: Disable stutcorr-rf-t.

```
; self-expanding, enable when needed


;; for PC in sy-I though, there is a fundamental difference. It
;; does NOT stutter sy-S PC, instead, there is a "virtual PC" in
;; sy-I which is a true stuttering of sy-S YPC, and which is
;; derivable from sy-I's global view, and with which sy-I RF is
;; in sync:

; comb_sub1.bm: SUB1 combinational element
; U7-DONE
```

```
; no charfun def, already defined in BM.

; Everything below generated by: (bmcomb 'sub1 '() '(x))
```

DEFINITION:
s-sub1 (x)
$=$ **if** empty (x) **then** E
else a (s-sub1 (p (x)), l $(x) - 1$) **endif**

```
;; A2-Begin-S-SUB1
```

THEOREM: a2-empty-s-sub1
empty (s-sub1 (x)) $=$ empty (x)

THEOREM: a2-e-s-sub1
(s-sub1 $(x) =$ E) $=$ empty (x)

THEOREM: a2-lp-s-sub1
len (s-sub1 (x)) $=$ len (x)

THEOREM: a2-lpe-s-sub1
eqlen (s-sub1 (x), x)

THEOREM: a2-ic-s-sub1
s-sub1 (i $(c\_x,\ x)$) $=$ i $(c\_x - 1$, s-sub1 (x))

THEOREM: a2-lc-s-sub1
$(\neg$ empty $(x)) \rightarrow$ (l (s-sub1 (x)) $=$ (l $(x) - 1$))

THEOREM: a2-pc-s-sub1
p (s-sub1 (x)) $=$ s-sub1 (p (x))

THEOREM: a2-hc-s-sub1
$(\neg$ empty $(x)) \rightarrow$ (h (s-sub1 (x)) $=$ (h $(x) - 1$))

THEOREM: a2-bc-s-sub1
b (s-sub1 (x)) $=$ s-sub1 (b (x))

THEOREM: a2-bnc-s-sub1
bn (n, s-sub1 (x)) $=$ s-sub1 (bn $(n,\ x)$)

```
;; A2-End-S-SUB1

; eof:comb_sub1.bm
 ; needed for def of ivPC
```

DEFINITION:
ivpc (x)
$=$ s-if (sy-i (`'ybc0`, x),
  sy-i (`'ybt0`, x),
  s-if (s-not (sy-i (`'ywa3`, x)),
    sy-i (`'ypc`, x),
    s-if (s-not (sy-i (`'ywa2`, x)),
      s-sub1 (sy-i (`'ypc`, x)),
      s-if (s-not (sy-i (`'ywa1`, x)),
        s-sub1 (s-sub1 (sy-i (`'ypc`, x))),
        s-sub1 (s-sub1 (s-sub1 (sy-i (`'ypc`, x))))))))

EVENT: Disable ivpc.

```
; we'll enable when needed
```

```
; sy-I-PC-numberp is a type lemma about I-PC and the branch targets
; needed for future add1-sub1 manipulation.
```

THEOREM: sy-i-pc-numberp
$(\neg$ empty $(x))$
$\rightarrow$ ((l (sy-i (`'ypc`, x)) $\in \mathbf{N}$)
  $\wedge$ (l (sy-i (`'ybt0`, x)) $\in \mathbf{N}$)
  $\wedge$ (l (sy-i (`'ybt1`, x)) $\in \mathbf{N}$)
  $\wedge$ (l (sy-i (`'ybt2`, x)) $\in \mathbf{N}$)
  $\wedge$ (l (sy-i (`'ybt3`, x)) $\in \mathbf{N}$)
  $\wedge$ (l (sy-i (`'ybt4`, x)) $\in \mathbf{N}$))

```
; Useful abbreviations about the semantics of the circuit:
```

DEFINITION:
res $(pc,\ rf) =$ ualu (umf $(pc,$ `'pro`), ulk (umr $(pc,$ `'pro`), rf))

EVENT: Disable res.

```
; n-pc is "Next PC"
```

DEFINITION:
n-pc $(pc,\ rf) =$ mux (eql0 (res $(pc,\ rf)$), umb $(pc,$ `'pro`), $1 + pc$)

EVENT: Disable n-pc.

```
; n-rf is "Next RF"
```

DEFINITION: $\text{n-rf}\,(pc,\ rf) = \text{upd}\,(\text{umw}\,(pc,\ \texttt{'pro}),\ \text{res}\,(pc,\ rf),\ rf)$

EVENT: Disable n-rf.

```
; Now we prove that they are "correct", with respect to our intent.
; Even though we don't use the next 3 lemmas anywhere else, they
; are a useful check - and during the development, help raise my
; confidence level! Moreover, they would be useful if we ever
; wanted to prove properties of the specification circuit (sy-S).
```

THEOREM: corr-n-pc
$((\neg\ \text{empty}\,(x)) \land (\neg\ \text{empty}\,(\text{p}\,(x))))$
$\rightarrow\quad (\text{l}\,(\text{sy-s}\,(\texttt{'ypc},\ x)) = \text{n-pc}\,(\text{l}\,(\text{sy-s}\,(\texttt{'ypc},\ \text{p}\,(x))),\ \text{l}\,(\text{sy-s}\,(\texttt{'yrf},\ \text{p}\,(x)))))$

EVENT: Disable corr-n-pc.

THEOREM: corr-n-rf
$((\neg\ \text{empty}\,(x)) \land (\neg\ \text{empty}\,(\text{p}\,(x))))$
$\rightarrow\quad (\text{l}\,(\text{sy-s}\,(\texttt{'yrf},\ x)) = \text{n-rf}\,(\text{l}\,(\text{sy-s}\,(\texttt{'ypc},\ \text{p}\,(x))),\ \text{l}\,(\text{sy-s}\,(\texttt{'yrf},\ \text{p}\,(x)))))$

EVENT: Disable corr-n-rf.

```
; Corr-S-PR-H and Corr-I-PR-H could be earlier, with the rest of
; the constant treatment for PR, but here is where they were
; needed & proved.
```

THEOREM: corr-s-pr-h
$(\neg\ \text{empty}\,(x)) \rightarrow (\text{h}\,(\text{sy-s}\,(\texttt{'ypr},\ x)) = \texttt{'pro})$

THEOREM: corr-i-pr-h
$(\neg\ \text{empty}\,(x)) \rightarrow (\text{h}\,(\text{sy-i}\,(\texttt{'ypr},\ x)) = \texttt{'pro})$

```
; REVERSAL PROPERTies for sy-S:
;  The SAME EXPAND hint as for Corr-n-pc and Corr-n-rf work.

; sy-S-reversal-pr is FUNDAMENTALLY DIFFERENT because we don't want
; to keep PR as part of the global state, so we deduce an equality
; by induction:
```

THEOREM: sy-s-reversal-pr
$(\neg\ \text{empty}(\text{bn}(n,\ \text{sy-s}(\texttt{'ypr},\ x)))) \rightarrow (\text{h}(\text{bn}(n,\ \text{sy-s}(\texttt{'ypr},\ x))) = \texttt{'pro})$

```
; we need a few simple props of ivPC for the main induction,
; because we have to run with ivPC disabled there:
```

THEOREM: ivpc-e
$\text{empty}(x) \rightarrow (\text{ivpc}(x) = \text{E})$

EVENT: Disable ivpc-e.

```
; enable as needed
```

THEOREM: ivpc-1
$((\neg\ \text{empty}(x)) \land \text{empty}(\text{p}(x))) \rightarrow (\text{l}(\text{ivpc}(x)) = \text{PCI})$

EVENT: Disable ivpc-1.

```
; enable as needed
```

```
; Junction (hack) lemma between the 2 hyps to get the reversals:
```

THEOREM: sy-s-reversal-hyp-hack
$(\neg\ \text{empty}(\text{bn}(n,\ \text{sy-s}(\texttt{'ypc},\ x)))) \rightarrow (\neg\ \text{empty}(\text{bn}(n,\ \text{sy-s}(\texttt{'ypr},\ x))))$

EVENT: Disable sy-s-reversal-hyp-hack.

THEOREM: sy-s-reversal-pc
$$\begin{aligned}&((\neg\ \text{empty}(\text{bn}(n,\ \text{sy-s}(\texttt{'ypc},\ \text{p}(x)))))\\ &\ \land\quad (\neg\ \text{empty}(\text{bn}(n,\ \text{sy-s}(\texttt{'yrf},\ \text{p}(x))))))\\ &\rightarrow\quad (\text{h}(\text{b}(\text{bn}(n,\ \text{sy-s}(\texttt{'ypc},\ x))))\\ &\qquad =\quad \text{n-pc}(\text{h}(\text{bn}(n,\ \text{sy-s}(\texttt{'ypc},\ \text{p}(x)))),\ \text{h}(\text{bn}(n,\ \text{sy-s}(\texttt{'yrf},\ \text{p}(x))))))\end{aligned}$$

THEOREM: sy-s-reversal-rf
$$\begin{aligned}&((\neg\ \text{empty}(\text{bn}(n,\ \text{sy-s}(\texttt{'ypc},\ \text{p}(x)))))\\ &\ \land\quad (\neg\ \text{empty}(\text{bn}(n,\ \text{sy-s}(\texttt{'yrf},\ \text{p}(x))))))\\ &\rightarrow\quad (\text{h}(\text{b}(\text{bn}(n,\ \text{sy-s}(\texttt{'yrf},\ x))))\\ &\qquad =\quad \text{n-rf}(\text{h}(\text{bn}(n,\ \text{sy-s}(\texttt{'ypc},\ \text{p}(x)))),\ \text{h}(\text{bn}(n,\ \text{sy-s}(\texttt{'yrf},\ \text{p}(x))))))\end{aligned}$$

```
;; Now we develop the FUNDAMENTAL INVARIANT for sy-I :

; Level is the only one which needs RES enabled, and it's also
; a bit different from the others, because it's never "reset".


; sy-I-Yra-noad tells BM Yra is always an OK address, in a usable
; way.
```

THEOREM: sy-i-yra-noad
$(\neg\, \text{empty}\,(x))$
$\rightarrow\quad (\text{ulk}\,(\text{l}\,(\text{sy-i}\,(\texttt{'yra},\, x)),\, \text{upd}\,(ad,\, dat,\, rf))$
$\quad = \quad \textbf{if}\ \text{l}\,(\text{sy-i}\,(\texttt{'yra},\, x)) = ad\ \ \textbf{then}\ dat$
$\quad\quad \textbf{else}\ \text{ulk}\,(\text{l}\,(\text{sy-i}\,(\texttt{'yra},\, x)),\, rf)\ \textbf{endif})$

```
; We prove the equalities for YRD lines separately, to minimize
; the mess!
```

THEOREM: sy-i-yrd1
$(\neg\, \text{empty}\,(x))$
$\rightarrow\quad (\text{l}\,(\text{sy-i}\,(\texttt{'yrd1},\, x))$
$\quad = \quad \text{ulk}\,(\text{l}\,(\text{sy-i}\,(\texttt{'yra},\, x)),$
$\quad\quad \text{upd}\,(\text{l}\,(\text{sy-i}\,(\texttt{'ywa1},\, x)),$
$\quad\quad\quad \text{l}\,(\text{sy-i}\,(\texttt{'ywd1},\, x)),$
$\quad\quad\quad \text{l}\,(\text{sy-i}\,(\texttt{'yrf},\, x)))))$

THEOREM: sy-i-yrd2
$(\neg\, \text{empty}\,(x))$
$\rightarrow\quad (\text{l}\,(\text{sy-i}\,(\texttt{'yrd2},\, x))$
$\quad = \quad \text{ulk}\,(\text{l}\,(\text{sy-i}\,(\texttt{'yra},\, x)),$
$\quad\quad \text{upd}\,(\text{l}\,(\text{sy-i}\,(\texttt{'ywa2},\, x)),$
$\quad\quad\quad \text{l}\,(\text{sy-i}\,(\texttt{'ywd2},\, x)),$
$\quad\quad\quad \text{upd}\,(\text{l}\,(\text{sy-i}\,(\texttt{'ywa1},\, x)),$
$\quad\quad\quad\quad \text{l}\,(\text{sy-i}\,(\texttt{'ywd1},\, x)),$
$\quad\quad\quad\quad \text{l}\,(\text{sy-i}\,(\texttt{'yrf},\, x))))))$

THEOREM: sy-i-yrd3
$(\neg\, \text{empty}\,(x))$
$\rightarrow\quad (\text{l}\,(\text{sy-i}\,(\texttt{'yrd3},\, x))$
$\quad = \quad \text{ulk}\,(\text{l}\,(\text{sy-i}\,(\texttt{'yra},\, x)),$
$\quad\quad \text{upd}\,(\text{l}\,(\text{sy-i}\,(\texttt{'ywa3},\, x)),$
$\quad\quad\quad \text{l}\,(\text{sy-i}\,(\texttt{'ywd3},\, x)),$
$\quad\quad\quad \text{upd}\,(\text{l}\,(\text{sy-i}\,(\texttt{'ywa2},\, x)),$
$\quad\quad\quad\quad \text{l}\,(\text{sy-i}\,(\texttt{'ywd2},\, x)),$

```
upd (l (sy-i ('ywa1, x)),
     l (sy-i ('ywd1, x)),
     l (sy-i ('yrf, x)))))))
```

```
; now we do the invariants, per level:
```

THEOREM: sy-i-inv-4

```
 (¬ empty (x))
 →   ((l (sy-i ('ybc4, x)) = eql0 (l (sy-i ('ywd4, x))))
      ∧   (l (sy-i ('ybt4, x)) = umb (l (sy-i ('ypc, x)), 'pro))
      ∧   (l (sy-i ('ywa4, x)) = umw (l (sy-i ('ypc, x)), 'pro))
      ∧   (l (sy-i ('ywd4, x))
            =   res (l (sy-i ('ypc, x)),
                     upd (l (sy-i ('ywa3, x)),
                          l (sy-i ('ywd3, x)),
                          upd (l (sy-i ('ywa2, x)),
                               l (sy-i ('ywd2, x)),
                               upd (l (sy-i ('ywa1, x)),
                                    l (sy-i ('ywd1, x)),
                                    l (sy-i ('yrf, x)))))))
      ∧   (l (sy-i ('yrd3, x))
            =   ulk (l (sy-i ('yra, x)),
                     upd (l (sy-i ('ywa3, x)),
                          l (sy-i ('ywd3, x)),
                          upd (l (sy-i ('ywa2, x)),
                               l (sy-i ('ywd2, x)),
                               upd (l (sy-i ('ywa1, x)),
                                    l (sy-i ('ywd1, x)),
                                    l (sy-i ('yrf, x))))))))
```

EVENT: Disable sy-i-yrd1.

EVENT: Disable sy-i-yrd2.

EVENT: Disable sy-i-yrd3.

```
; Note that we repeat sy-I-Yrdn for each n, and this may be costly
; later. On the other hand, it's nice to have one property which
; expresses the "complete" correctness of a stage.
```

THEOREM: sy-i-inv-3

$$
\begin{array}{l}
(\neg\ \mathrm{empty}\,(x)) \\
\rightarrow\quad \textbf{if}\ \neg\ \mathrm{l}\,(\text{sy-i}\,(\texttt{'ywa3},\ x)) \\
\qquad \textbf{then}\ (\neg\ \mathrm{l}\,(\text{sy-i}\,(\texttt{'ybc3},\ x))) \wedge (\neg\ \mathrm{l}\,(\text{sy-i}\,(\texttt{'ywa2},\ x))) \\
\qquad \textbf{else}\ (\mathrm{l}\,(\text{sy-i}\,(\texttt{'ybc3},\ x)) = \mathrm{eql0}\,(\mathrm{l}\,(\text{sy-i}\,(\texttt{'ywd3},\ x)))) \\
\qquad\qquad \wedge\quad (\mathrm{l}\,(\text{sy-i}\,(\texttt{'ybt3},\ x)) \\
\qquad\qquad\qquad =\quad \mathrm{umb}\,(\mathrm{l}\,(\text{sy-i}\,(\texttt{'ypc},\ x)) - 1,\ \texttt{'pro})) \\
\qquad\qquad \wedge\quad (\mathrm{l}\,(\text{sy-i}\,(\texttt{'ywa3},\ x)) \\
\qquad\qquad\qquad =\quad \mathrm{umw}\,(\mathrm{l}\,(\text{sy-i}\,(\texttt{'ypc},\ x)) - 1,\ \texttt{'pro})) \\
\qquad\qquad \wedge\quad (\mathrm{l}\,(\text{sy-i}\,(\texttt{'ywd3},\ x)) \\
\qquad\qquad\qquad =\quad \mathrm{res}\,(\mathrm{l}\,(\text{sy-i}\,(\texttt{'ypc},\ x)) - 1, \\
\qquad\qquad\qquad\qquad\qquad \mathrm{upd}\,(\mathrm{l}\,(\text{sy-i}\,(\texttt{'ywa2},\ x)), \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathrm{l}\,(\text{sy-i}\,(\texttt{'ywd2},\ x)), \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathrm{upd}\,(\mathrm{l}\,(\text{sy-i}\,(\texttt{'ywa1},\ x)), \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathrm{l}\,(\text{sy-i}\,(\texttt{'ywd1},\ x)), \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathrm{l}\,(\text{sy-i}\,(\texttt{'yrf},\ x)))))) \\
\qquad\qquad \wedge\quad (\mathrm{l}\,(\text{sy-i}\,(\texttt{'yrd2},\ x)) \\
\qquad\qquad\qquad =\quad \mathrm{ulk}\,(\mathrm{l}\,(\text{sy-i}\,(\texttt{'yra},\ x)), \\
\qquad\qquad\qquad\qquad\qquad \mathrm{upd}\,(\mathrm{l}\,(\text{sy-i}\,(\texttt{'ywa2},\ x)), \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathrm{l}\,(\text{sy-i}\,(\texttt{'ywd2},\ x)), \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathrm{upd}\,(\mathrm{l}\,(\text{sy-i}\,(\texttt{'ywa1},\ x)), \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathrm{l}\,(\text{sy-i}\,(\texttt{'ywd1},\ x)), \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathrm{l}\,(\text{sy-i}\,(\texttt{'yrf},\ x)))))) \ \textbf{endif}
\end{array}
$$

THEOREM: sy-i-inv-2

$$
\begin{array}{l}
(\neg\ \mathrm{empty}\,(x)) \\
\rightarrow\quad \textbf{if}\ \neg\ \mathrm{l}\,(\text{sy-i}\,(\texttt{'ywa2},\ x)) \\
\qquad \textbf{then}\ (\neg\ \mathrm{l}\,(\text{sy-i}\,(\texttt{'ybc2},\ x))) \wedge (\neg\ \mathrm{l}\,(\text{sy-i}\,(\texttt{'ywa1},\ x))) \\
\qquad \textbf{else}\ (\mathrm{l}\,(\text{sy-i}\,(\texttt{'ybc2},\ x)) = \mathrm{eql0}\,(\mathrm{l}\,(\text{sy-i}\,(\texttt{'ywd2},\ x)))) \\
\qquad\qquad \wedge\quad (\mathrm{l}\,(\text{sy-i}\,(\texttt{'ybt2},\ x)) \\
\qquad\qquad\qquad =\quad \mathrm{umb}\,((\mathrm{l}\,(\text{sy-i}\,(\texttt{'ypc},\ x)) - 1) - 1,\ \texttt{'pro})) \\
\qquad\qquad \wedge\quad (\mathrm{l}\,(\text{sy-i}\,(\texttt{'ywa2},\ x)) \\
\qquad\qquad\qquad =\quad \mathrm{umw}\,((\mathrm{l}\,(\text{sy-i}\,(\texttt{'ypc},\ x)) - 1) - 1,\ \texttt{'pro})) \\
\qquad\qquad \wedge\quad (\mathrm{l}\,(\text{sy-i}\,(\texttt{'ywd2},\ x)) \\
\qquad\qquad\qquad =\quad \mathrm{res}\,((\mathrm{l}\,(\text{sy-i}\,(\texttt{'ypc},\ x)) - 1) - 1, \\
\qquad\qquad\qquad\qquad\qquad \mathrm{upd}\,(\mathrm{l}\,(\text{sy-i}\,(\texttt{'ywa1},\ x)), \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathrm{l}\,(\text{sy-i}\,(\texttt{'ywd1},\ x)), \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathrm{l}\,(\text{sy-i}\,(\texttt{'yrf},\ x))))) \\
\qquad\qquad \wedge\quad (\mathrm{l}\,(\text{sy-i}\,(\texttt{'yrd1},\ x)) \\
\qquad\qquad\qquad =\quad \mathrm{ulk}\,(\mathrm{l}\,(\text{sy-i}\,(\texttt{'yra},\ x)), \\
\qquad\qquad\qquad\qquad\qquad \mathrm{upd}\,(\mathrm{l}\,(\text{sy-i}\,(\texttt{'ywa1},\ x)), \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathrm{l}\,(\text{sy-i}\,(\texttt{'ywd1},\ x)), \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathrm{l}\,(\text{sy-i}\,(\texttt{'yrf},\ x))))) \ \textbf{endif}
\end{array}
$$

THEOREM: sy-i-inv-1

$(\neg\, \text{empty}\,(x))$
$$\begin{array}{ll}\rightarrow & \textbf{if } \neg\, \text{l}\,(\text{sy-i}\,(\texttt{'ywa1},\ x))\ \ \textbf{then } \neg\, \text{l}\,(\text{sy-i}\,(\texttt{'ybc1},\ x)) \\ & \textbf{else } (\text{l}\,(\text{sy-i}\,(\texttt{'ybc1},\ x)) = \text{eql0}\,(\text{l}\,(\text{sy-i}\,(\texttt{'ywd1},\ x)))) \\ & \qquad \wedge\ \ (\text{l}\,(\text{sy-i}\,(\texttt{'ybt1},\ x)) \\ & \qquad\qquad = \ \ \text{umb}\,(((\text{l}\,(\text{sy-i}\,(\texttt{'ypc},\ x)) - 1) - 1) - 1,\ \texttt{'pro})) \\ & \qquad \wedge\ \ (\text{l}\,(\text{sy-i}\,(\texttt{'ywa1},\ x)) \\ & \qquad\qquad = \ \ \text{umw}\,(((\text{l}\,(\text{sy-i}\,(\texttt{'ypc},\ x)) - 1) - 1) - 1,\ \texttt{'pro})) \\ & \qquad \wedge\ \ (\text{l}\,(\text{sy-i}\,(\texttt{'ywd1},\ x)) \\ & \qquad\qquad = \ \ \text{res}\,(((\text{l}\,(\text{sy-i}\,(\texttt{'ypc},\ x)) - 1) - 1) - 1, \\ & \qquad\qquad\qquad \text{l}\,(\text{sy-i}\,(\texttt{'yrf},\ x)))) \\ & \qquad \wedge\ \ (\text{l}\,(\text{sy-i}\,(\texttt{'yrd},\ x)) \\ & \qquad\qquad = \ \ \text{ulk}\,(\text{l}\,(\text{sy-i}\,(\texttt{'yra},\ x)),\ \text{l}\,(\text{sy-i}\,(\texttt{'yrf},\ x)))) \ \textbf{endif}\end{array}$$

```
; NOTES:
; .  Having the invariants written as above entail the cost of
; carrying lots of irrelevant information in each proof, and
; potentially crushing BM.  So instead we'll phrase each needed
; property later as a good rewrite, and prove it by instantiating
; the right invariant(s).
; .  What we did is a hand-induction on the DEPTH of the pipeline.
; Notice that NO explicit induction was necessary, simply proving
; the thms in the right order (4-3-2-1)..


; StutCorr-pc-T expresses correctness of ivPC WHEN stuttering.

; We rewrite just the portions of the invariants we need for
; StutCorr-pc-T so that BM will get that proof in finite time...
```

Theorem: sc-pc-t-1
$(\neg\, \text{l}\,(\text{sy-i}\,(\texttt{'ywa1},\ x))) \rightarrow (\neg\, \text{l}\,(\text{sy-i}\,(\texttt{'ybc1},\ x)))$

Event: Disable sc-pc-t-1.

Theorem: sc-pc-t-2
$(\neg\, \text{l}\,(\text{sy-i}\,(\texttt{'ywa2},\ x))) \rightarrow (\neg\, \text{l}\,(\text{sy-i}\,(\texttt{'ybc2},\ x)))$

Event: Disable sc-pc-t-2.

Theorem: sc-pc-t-3
$(\neg\, \text{l}\,(\text{sy-i}\,(\texttt{'ywa3},\ x))) \rightarrow (\neg\, \text{l}\,(\text{sy-i}\,(\texttt{'ybc3},\ x)))$

Event: Disable sc-pc-t-3.

THEOREM: stutcorr-pc-t
$((\neg\ \text{empty}\,(x)) \wedge (\neg\ \text{empty}\,(\text{p}\,(x))) \wedge (\neg\ \text{l}\,(\text{sy-i}\,(\texttt{'ywa1k},\ \text{p}\,(x)))))$
$\rightarrow\quad (\text{l}\,(\text{ivpc}\,(x)) = \text{l}\,(\text{ivpc}\,(\text{p}\,(x))))$

EVENT: Disable stutcorr-pc-t.

```
; self-expanding, enable when needed
```

```
; Now we express the "on the right also" invariants in a useful
; manner...
```

THEOREM: sc-pc-f-3
$(\neg\ \text{l}\,(\text{sy-i}\,(\texttt{'ywa3},\ x))) \rightarrow (\neg\ \text{l}\,(\text{sy-i}\,(\texttt{'ywa2},\ x)))$

EVENT: Disable sc-pc-f-3.

THEOREM: sc-pc-f-2
$(\neg\ \text{l}\,(\text{sy-i}\,(\texttt{'ywa2},\ x))) \rightarrow (\neg\ \text{l}\,(\text{sy-i}\,(\texttt{'ywa1},\ x)))$

EVENT: Disable sc-pc-f-2.

```
; In order to get the arithmetic to work out, we have to prove that
; we never subtract from zero...
```

THEOREM: sc-pcn0-0
$((\neg\ \text{empty}\,(x)) \wedge (\neg\ \text{l}\,(\text{sy-i}\,(\texttt{'ywa2},\ x))) \wedge \text{l}\,(\text{sy-i}\,(\texttt{'ywa3},\ x)))$
$\rightarrow\quad (\text{l}\,(\text{sy-i}\,(\texttt{'ypc},\ x)) \neq \texttt{0})$

THEOREM: sc-pcn0-1
$((\neg\ \text{empty}\,(x))$
$\wedge\quad (\neg\ \text{l}\,(\text{sy-i}\,(\texttt{'ywa1},\ x)))$
$\wedge\quad \text{l}\,(\text{sy-i}\,(\texttt{'ywa2},\ x))$
$\wedge\quad \text{l}\,(\text{sy-i}\,(\texttt{'ywa3},\ x)))$
$\rightarrow\quad ((\text{l}\,(\text{sy-i}\,(\texttt{'ypc},\ x)) - 1) \neq \texttt{0})$

THEOREM: sc-pcn0-2
$((\neg\ \text{empty}\,(x))$
$\wedge\quad \text{l}\,(\text{sy-i}\,(\texttt{'ywa1},\ x))$
$\wedge\quad \text{l}\,(\text{sy-i}\,(\texttt{'ywa3},\ x))$
$\wedge\quad \text{l}\,(\text{sy-i}\,(\texttt{'ywa2},\ x)))$
$\rightarrow\quad (((\text{l}\,(\text{sy-i}\,(\texttt{'ypc},\ x)) - 1) - 1) \neq \texttt{0})$

```
; Now it goes through:
```

THEOREM: stutcorr-pc-f
$((\neg \text{ empty}\,(x)) \wedge (\neg \text{ empty}\,(\text{p}\,(x))) \wedge \text{l}\,(\text{sy-i}\,(\texttt{'ywa1k}, \text{p}\,(x))))$
$\rightarrow \quad (\text{l}\,(\text{ivpc}\,(x)) = \text{n-pc}\,(\text{l}\,(\text{ivpc}\,(\text{p}\,(x))), \text{l}\,(\text{sy-i}\,(\texttt{'yrf}, \text{p}\,(x)))))$

EVENT: Disable stutcorr-pc-f.

THEOREM: stutcorr-rf-f
$((\neg \text{ empty}\,(x)) \wedge (\neg \text{ empty}\,(\text{p}\,(x))) \wedge \text{l}\,(\text{sy-i}\,(\texttt{'ywa1k}, \text{p}\,(x))))$
$\rightarrow \quad (\text{l}\,(\text{sy-i}\,(\texttt{'yrf}, x)) = \text{n-rf}\,(\text{l}\,(\text{ivpc}\,(\text{p}\,(x))), \text{l}\,(\text{sy-i}\,(\texttt{'yrf}, \text{p}\,(x)))))$

EVENT: Disable stutcorr-rf-f.

```
; We have to simultaneously prove stuttering of RF and ivPC:
```

THEOREM: stutcorr-l
$(\text{l}\,(\text{stut}\,(\text{sy-s}\,(\texttt{'ypc}, x), \text{s-not}\,(\text{sy-i}\,(\texttt{'ywa1k}, x)))) = \text{l}\,(\text{ivpc}\,(x)))$
$\wedge \quad (\text{l}\,(\text{stut}\,(\text{sy-s}\,(\texttt{'yrf}, x), \text{s-not}\,(\text{sy-i}\,(\texttt{'ywa1k}, x)))) = \text{l}\,(\text{sy-i}\,(\texttt{'yrf}, x)))$

```
; Now standard passage to strings:
```

THEOREM: apl-split-irf
$(\neg \text{ empty}\,(x))$
$\rightarrow \quad (\text{sy-i}\,(\texttt{'yrf}, x) = \text{a}\,(\text{p}\,(\text{sy-i}\,(\texttt{'yrf}, x)), \text{l}\,(\text{sy-i}\,(\texttt{'yrf}, x))))$

THEOREM: apl-split-stut
$(\neg \text{ empty}\,(x))$
$\rightarrow \quad (\text{stut}\,(\text{sy-s}\,(\texttt{'yrf}, x), \text{s-not}\,(\text{sy-i}\,(\texttt{'ywa1k}, x)))$
$\quad = \quad \text{a}\,(\text{p}\,(\text{stut}\,(\text{sy-s}\,(\texttt{'yrf}, x), \text{s-not}\,(\text{sy-i}\,(\texttt{'ywa1k}, x)))),$
$\quad\quad \text{l}\,(\text{stut}\,(\text{sy-s}\,(\texttt{'yrf}, x), \text{s-not}\,(\text{sy-i}\,(\texttt{'ywa1k}, x))))))$

```
; YEEEEAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH!
```

THEOREM: src-cpu-correct
$\text{stut}\,(\text{sy-s}\,(\texttt{'yrf}, x), \text{s-not}\,(\text{sy-i}\,(\texttt{'ywa1k}, x))) = \text{sy-i}\,(\texttt{'yrf}, x)$

```
; eof: srccpu.bm
;))
```

Index