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

```
; prod0_CSXAdd.bm
;   . definition of circuits [assumes stringadd.bm] :
;     - if the circuit has only one line: OK without any hint
;  We MAY want to put the TOPO hint, just for the induction, although
; for one line it probably collapes to the same induction (LEN X).
;     - if the circuit has more than one line:
;    - without hints: FAIL
;          - TOPO0 is not definable, because of loops in the dependency graph!
;    - with TOPOR: OK
; NOTE: the above comments date back to the hand-generation time, when we
;       were still trying to FIND a way to feed things to BM.  They are kept
```

```
;       here for historical purposes only...


;;; DEFINITION OF CIRCUITS:
#|
(setq sysd-prod '(sy-prod (x)
(Yprod S Times x Yprod2)
(Yprod2 R 0 Yprod)
))

(setq sysd-const0 '(sy-const0 (x)
(Yconst0 R 0 Yconst0)
))

(setq prod0_CSXA00 '(
|#

; BM DEFINITIONS and A2 LEMMAS, generated by BMSYSD:
; comb_times.bm: Times combinational element.
; U7-DONE

; no character function def since BM already knows about Times..

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

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

```
;; A2-Begin-S-TIMES
```

Theorem: a2-empty-s-times
empty (s-times $(x,\ y)$) = empty (x)

Theorem: a2-e-s-times
(s-times $(x,\ y)$ = E) = empty (x)

Theorem: a2-lp-s-times
len (s-times $(x,\ y)$) = len (x)

Theorem: a2-lpe-s-times
eqlen (s-times $(x,\ y)$, x)

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

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

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

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

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

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

```
;; A2-End-S-TIMES
```

```
; eof:comb_times.bm
```

DEFINITION:
topor-sy-prod (ln)
= **if** ln = `'yprod` **then** `1`
elseif ln = `'yprod2` **then** `0`
else `0` **endif**

DEFINITION:
sy-prod $(ln,\ x)$
= **if** ln = `'yprod` **then** s-times $(x,$ sy-prod $($`'yprod2`$,\ x))$
elseif ln = `'yprod2`
then if empty (x) **then** E
else i (`0`, sy-prod (`'yprod`, p (x))) **endif**
else sfix (x) **endif**

```
;; A2-Begin-SY-PROD
```

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

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

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

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

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

```
;; A2-End-SY-PROD
; BM DEFINITIONS and A2 LEMMAS, generated by BMSYSD:

; No TOPO def for 1 line sysds because it is not needed and confuses BM
```

DEFINITION:
sy-const0 (ln, x)
= **if** ln = `'yconst0`
then if empty (x) **then** E
else i (`0`, sy-const0 (`'yconst0`, p (x))) **endif**
else sfix (x) **endif**

```
;; A2-Begin-SY-CONST0
```

THEOREM: a2-empty-sy-const0
empty (sy-const0 (ln, x)) = empty (x)

THEOREM: a2-e-sy-const0
(sy-const0 (ln, x) = E) = empty (x)

THEOREM: a2-lp-sy-const0
len (sy-const0 (ln, x)) = len (x)

THEOREM: a2-lpe-sy-const0
eqlen (sy-const0 (ln, x), x)

THEOREM: a2-pc-sy-const0
p (sy-const0 (ln, x)) = sy-const0 (ln, p (x))

```
;; A2-End-SY-CONST0

;;; PROOF OF EQUIVALENCE:

; The key fact about SY-Yconst is that it equals the constant 0 function:
```

THEOREM: sy-const0-is-const
sy-const0 (`'yconst0`, x) = s-const (`0`, x)

```
; The key fact (bug) about prod0 is that both lines also equal const-0 sfun
; CRUCIAL NOTE: we only want the 1st equality, but in order for the induction
; proof to succeed, we need the stronger (global) statement.
```

THEOREM: prod0-is-const
(sy-prod (`'yprod`, x) = s-const (`0`, x))
$\wedge$ (sy-prod (`'yprod2`, x) = s-const (`0`, x))

```
; now the equality is trivial:
```

THEOREM: e_prodconst0
sy-prod (`'yprod`, x) = sy-const0 (`'yconst0`, x)

```
; eof: prod0_CSXA00.bm
;))
```

Index