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

```
; bcdS.bm: a BCD code checker, with serial (bit) input, but built
;          STRUCTURELY.
;    The key lesson is that it helps (and is in fact necessary) to
;    reduce the sysd to a generalized sysd, with just the
;    "important" lines: Paillet's variables d'etats non-eliminables
;     expressed in terms of each other. See SY-B2 below.
;
;    Note: Proving the A2 lemmas about the generalized sysd brings
;    2 issues:
;    - they don't seem to be used at all.
;    - A2-PC loops on its own (no rule!) and is therefore so far
```

```
;      unprovable...
;

;;; CIRCUIT in SUGARED form:

; note: registers are initialized with F instead of 0, since coded
; at logical level.
#|
(setq sysd '(sy-BCDS (x)
(Y01 S not x)
(Y02 S not x)
(Y03 S not YR3)
(Y04 S not YR1)
(Y05 S not YR2)
(Y06 S not YR3)
(Y07 S not YR1)
(Y08 S not YR3)
(Y11 S and3 Y01 YR1 YR3)
(Y12 S and3 Y02 YR2 Y03)
(Y13 S and3 Y04 Y05 Y06)
(Y14 S and3 x Y07 Y08)
(Y15 S and3 x YR1 YR3)
(Y21 S or Y11 Y12)
(Y22 S or Y13 Y14)
(Y23 S or YR2 Y15)
(YR1 R F Y21)
(YR2 R F Y22)
(YR3 R F Y23)
(Y31 S not YR1)
(Y32 S not YR2)
(Y41 S and4 x Y31 Y32 YR3)
(Yout S not Y41)
))

(setq bcdS '(  |#
; BM DEFINITIONS and A2 LEMMAS, generated by BMSYSD:
; comb_and3.bm: Logical And combinational element, with 3 inputs
; U7-DONE
```

DEFINITION: $\text{and3}\,(u1,\, u2,\, u3) = (u1 \wedge u2 \wedge u3)$

```
; Everything below generated by: (bmcomb 'and3 '() '(x1 x2 x3))
```

DEFINITION:
s-and3 $(x1,\ x2,\ x3)$
$=$ **if** empty $(x1)$ **then** E
else a (s-and3 (p $(x1)$, p $(x2)$, p $(x3)$), and3 (l $(x1)$, l $(x2)$, l $(x3)$)) **endif**

```
;; A2-Begin-S-AND3
```

THEOREM: a2-empty-s-and3
empty (s-and3 $(x1,\ x2,\ x3)$) = empty $(x1)$

THEOREM: a2-e-s-and3
(s-and3 $(x1,\ x2,\ x3)$ = E) = empty $(x1)$

THEOREM: a2-lp-s-and3
len (s-and3 $(x1,\ x2,\ x3)$) = len $(x1)$

THEOREM: a2-lpe-s-and3
eqlen (s-and3 $(x1,\ x2,\ x3)$, $x1$)

THEOREM: a2-ic-s-and3
((len $(x1)$ = len $(x2)$) $\land$ (len $(x2)$ = len $(x3)$))
$\rightarrow$ (s-and3 (i $(c\_x1,\ x1)$, i $(c\_x2,\ x2)$, i $(c\_x3,\ x3)$)
$=$ i (and3 $(c\_x1,\ c\_x2,\ c\_x3)$, s-and3 $(x1,\ x2,\ x3)$))

THEOREM: a2-lc-s-and3
($\neg$ empty $(x1)$) $\rightarrow$ (l (s-and3 $(x1,\ x2,\ x3)$) = and3 (l $(x1)$, l $(x2)$, l $(x3)$))

THEOREM: a2-pc-s-and3
p (s-and3 $(x1,\ x2,\ x3)$) = s-and3 (p $(x1)$, p $(x2)$, p $(x3)$)

THEOREM: a2-hc-s-and3
(($\neg$ empty $(x1)$) $\land$ ((len $(x1)$ = len $(x2)$) $\land$ (len $(x2)$ = len $(x3)$)))
$\rightarrow$ (h (s-and3 $(x1,\ x2,\ x3)$) = and3 (h $(x1)$, h $(x2)$, h $(x3)$))

THEOREM: a2-bc-s-and3
((len $(x1)$ = len $(x2)$) $\land$ (len $(x2)$ = len $(x3)$))
$\rightarrow$ (b (s-and3 $(x1,\ x2,\ x3)$) = s-and3 (b $(x1)$, b $(x2)$, b $(x3)$))

THEOREM: a2-bnc-s-and3
((len $(x1)$ = len $(x2)$) $\land$ (len $(x2)$ = len $(x3)$))
$\rightarrow$ (bn (n, s-and3 $(x1,\ x2,\ x3)$) = s-and3 (bn $(n,\ x1)$, bn $(n,\ x2)$, bn $(n,\ x3)$))

```
;; A2-End-S-AND3

; eof:comb_and3.bm

; comb_and4.bm: Logical And combinational element, with 4 inputs
; U7-DONE
```

DEFINITION: $\text{and4}(u1, u2, u3, u4) = (u1 \land u2 \land u3 \land u4)$

```
; Everything below generated by: (bmcomb 'and4 '() '(x1 x2 x3 x4))
```

DEFINITION:
$\text{s-and4}(x1, x2, x3, x4)$
$= \quad$ **if** $\text{empty}(x1)$ **then** E
else $\text{a}(\text{s-and4}(\text{p}(x1), \text{p}(x2), \text{p}(x3), \text{p}(x4)),$
$\text{and4}(\text{l}(x1), \text{l}(x2), \text{l}(x3), \text{l}(x4)))$ **endif**

```
;; A2-Begin-S-AND4
```

THEOREM: a2-empty-s-and4
$\text{empty}(\text{s-and4}(x1, x2, x3, x4)) = \text{empty}(x1)$

THEOREM: a2-e-s-and4
$(\text{s-and4}(x1, x2, x3, x4) = \text{E}) = \text{empty}(x1)$

THEOREM: a2-lp-s-and4
$\text{len}(\text{s-and4}(x1, x2, x3, x4)) = \text{len}(x1)$

THEOREM: a2-lpe-s-and4
$\text{eqlen}(\text{s-and4}(x1, x2, x3, x4), x1)$

THEOREM: a2-ic-s-and4
$((\text{len}(x1) = \text{len}(x2)) \land (\text{len}(x2) = \text{len}(x3)) \land (\text{len}(x3) = \text{len}(x4)))$
$\rightarrow \quad (\text{s-and4}(\text{i}(c\_x1, x1), \text{i}(c\_x2, x2), \text{i}(c\_x3, x3), \text{i}(c\_x4, x4))$
$= \quad \text{i}(\text{and4}(c\_x1, c\_x2, c\_x3, c\_x4), \text{s-and4}(x1, x2, x3, x4)))$

THEOREM: a2-lc-s-and4
$(\neg \text{empty}(x1))$
$\rightarrow \quad (\text{l}(\text{s-and4}(x1, x2, x3, x4)) = \text{and4}(\text{l}(x1), \text{l}(x2), \text{l}(x3), \text{l}(x4)))$

THEOREM: a2-pc-s-and4
$\text{p}(\text{s-and4}(x1, x2, x3, x4)) = \text{s-and4}(\text{p}(x1), \text{p}(x2), \text{p}(x3), \text{p}(x4))$

THEOREM: a2-hc-s-and4
$((\neg \text{empty}(x1))$
$\land \quad ((\text{len}(x1) = \text{len}(x2))$
$\land \quad (\text{len}(x2) = \text{len}(x3))$
$\land \quad (\text{len}(x3) = \text{len}(x4))))$
$\rightarrow \quad (\text{h}(\text{s-and4}(x1, x2, x3, x4)) = \text{and4}(\text{h}(x1), \text{h}(x2), \text{h}(x3), \text{h}(x4)))$

THEOREM: a2-bc-s-and4
$((\text{len}(x1) = \text{len}(x2)) \land (\text{len}(x2) = \text{len}(x3)) \land (\text{len}(x3) = \text{len}(x4)))$
$\rightarrow \quad (\text{b}(\text{s-and4}(x1, x2, x3, x4)) = \text{s-and4}(\text{b}(x1), \text{b}(x2), \text{b}(x3), \text{b}(x4)))$

THEOREM: a2-bnc-s-and4
$((\mathrm{len}(x1) = \mathrm{len}(x2)) \wedge (\mathrm{len}(x2) = \mathrm{len}(x3)) \wedge (\mathrm{len}(x3) = \mathrm{len}(x4)))$
$\rightarrow \quad (\mathrm{bn}(n, \text{s-and4}(x1, x2, x3, x4))$
$\quad = \quad \text{s-and4}(\mathrm{bn}(n, x1), \mathrm{bn}(n, x2), \mathrm{bn}(n, x3), \mathrm{bn}(n, x4)))$

```
;; A2-End-S-AND4

; eof:comb_and4.bm
```

DEFINITION:
topor-sy-bcds (*ln*)
= **if** *ln* = 'y01 **then** 1
elseif *ln* = 'y02 **then** 1
elseif *ln* = 'y03 **then** 1
elseif *ln* = 'y04 **then** 1
elseif *ln* = 'y05 **then** 1
elseif *ln* = 'y06 **then** 1
elseif *ln* = 'y07 **then** 1
elseif *ln* = 'y08 **then** 1
elseif *ln* = 'y11 **then** 2
elseif *ln* = 'y12 **then** 2
elseif *ln* = 'y13 **then** 2
elseif *ln* = 'y14 **then** 2
elseif *ln* = 'y15 **then** 1
elseif *ln* = 'y21 **then** 3
elseif *ln* = 'y22 **then** 3
elseif *ln* = 'y23 **then** 2
elseif *ln* = 'yr1 **then** 0
elseif *ln* = 'yr2 **then** 0
elseif *ln* = 'yr3 **then** 0
elseif *ln* = 'y31 **then** 1
elseif *ln* = 'y32 **then** 1
elseif *ln* = 'y41 **then** 2
elseif *ln* = 'yout **then** 3
else 0 **endif**

DEFINITION:
sy-bcds (*ln*, *x*)
= **if** *ln* = 'y01 **then** s-not (*x*)
elseif *ln* = 'y02 **then** s-not (*x*)
elseif *ln* = 'y03 **then** s-not (sy-bcds ('yr3, *x*))
elseif *ln* = 'y04 **then** s-not (sy-bcds ('yr1, *x*))
elseif *ln* = 'y05 **then** s-not (sy-bcds ('yr2, *x*))

```
elseif ln = 'y06 then s-not (sy-bcds ('yr3, x))
elseif ln = 'y07 then s-not (sy-bcds ('yr1, x))
elseif ln = 'y08 then s-not (sy-bcds ('yr3, x))
elseif ln = 'y11
then s-and3 (sy-bcds ('y01, x), sy-bcds ('yr1, x), sy-bcds ('yr3, x))
elseif ln = 'y12
then s-and3 (sy-bcds ('y02, x), sy-bcds ('yr2, x), sy-bcds ('y03, x))
elseif ln = 'y13
then s-and3 (sy-bcds ('y04, x), sy-bcds ('y05, x), sy-bcds ('y06, x))
elseif ln = 'y14 then s-and3 (x, sy-bcds ('y07, x), sy-bcds ('y08, x))
elseif ln = 'y15 then s-and3 (x, sy-bcds ('yr1, x), sy-bcds ('yr3, x))
elseif ln = 'y21 then s-or (sy-bcds ('y11, x), sy-bcds ('y12, x))
elseif ln = 'y22 then s-or (sy-bcds ('y13, x), sy-bcds ('y14, x))
elseif ln = 'y23 then s-or (sy-bcds ('yr2, x), sy-bcds ('y15, x))
elseif ln = 'yr1
then if empty (x) then E
     else i (f, sy-bcds ('y21, p (x))) endif
elseif ln = 'yr2
then if empty (x) then E
     else i (f, sy-bcds ('y22, p (x))) endif
elseif ln = 'yr3
then if empty (x) then E
     else i (f, sy-bcds ('y23, p (x))) endif
elseif ln = 'y31 then s-not (sy-bcds ('yr1, x))
elseif ln = 'y32 then s-not (sy-bcds ('yr2, x))
elseif ln = 'y41
then s-and4 (x, sy-bcds ('y31, x), sy-bcds ('y32, x), sy-bcds ('yr3, x))
elseif ln = 'yout then s-not (sy-bcds ('y41, x))
else sfix (x) endif
```

`;; A2-Begin-SY-BCDS`

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

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

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

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

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

```
;; A2-End-SY-BCDS

;;; Circuit CORRECTNESS /Paillet:

; BCD-Lbits defines a correct binary coded decimal, b0 is
; most-significant. It assumes the bits are logical though (in
; contrast to bcd.bm).
```

DEFINITION:
$\text{bcd-lbits}(b0, b1, b2, b3) = ((b0 = \mathbf{f}) \lor ((b1 = \mathbf{f}) \land (b2 = \mathbf{f})))$

```
; CORRECTNESS:

;;; WHAT PAILLET ACTUALLY PROVES:
;;; redone-exactly, to show that we can do it too! And to teach us
;;; how to do some nasty BM control: doing everything backwards!

;; FIRST he starts from the Register-equations, where of course he
;; doesn't mention the initial values... In our case we have to
;; PROVE them:
;; NOTES: 1: even though the expansion hints look gruesome, they
;; are immediate to find out: expand everything once around the
;; register-loop.
;;        2: we push the P's in, because otherwise clearly the eq's
;; will loop, even though of course, they are still provable (done).
;;        3: doing all 3 equations at once is clearer than 1 by 1,
;; since the hint gets given only once, and one doesn't have to
;; disable all other proved equations while doing the next one in
;; order to prevent looping.
;;        4: if we give the most general expression:
;; if empty x e ... then ;
;; we get a self-looping rule since it applies to its rhs, which is
;; still usable, as long as we USE it and DISABLE it simultaneously.
;; But that means we have to specify each use individually, and that
;; could be a pain.
;;   5: Fundamentally, this is an UNFOLDING rule, i.e. rhs is
;; more complex than lhs.  BM can't deal with that, except in the
;; context of a recursive DEFN, which is exactly the same thing,
;; and for which BM has hardwired smarts. Maybe that's the right
;; way to handle UNFOLDING rewrites? Create a DEFN for them and
;; prove equality, and then use the DEFN:
```

DEFINITION:
sy-b2 (ln, x)
= **if** ln = `'yr1`
then if empty (x) **then** E
else i (**f**,
s-or (s-and3 (s-not (p (x)),
sy-b2 (`'yr1`, p (x)),
sy-b2 (`'yr3`, p (x))),
s-and3 (s-not (p (x)),
sy-b2 (`'yr2`, p (x)),
s-not (sy-b2 (`'yr3`, p (x)))))) **endif**
elseif ln = `'yr2`
then if empty (x) **then** E
else i (**f**,
s-or (s-and3 (s-not (sy-b2 (`'yr1`, p (x))),
s-not (sy-b2 (`'yr2`, p (x))),
s-not (sy-b2 (`'yr3`, p (x)))),
s-and3 (p (x),
s-not (sy-b2 (`'yr1`, p (x))),
s-not (sy-b2 (`'yr3`, p (x)))))) **endif**
elseif ln = `'yr3`
then if empty (x) **then** E
else i (**f**,
s-or (sy-b2 (`'yr2`, p (x)),
s-and3 (p (x),
sy-b2 (`'yr1`, p (x)),
sy-b2 (`'yr3`, p (x))))) **endif**
else sfix (x) **endif**

```
; B2 is just a GENERALIZED sysd, and our A2 lemmas should still be
; true. The following were (Sugar) generated by:
; (vp (bma2sysd-aux 'sy-B2 'sy-B2 '(x) '(and3 or and4 not)))
; with A2-PC disabled because last time we tried it looped, and we
; don't need it.

;; A2-Begin-SY-B2
```

THEOREM: a2-empty-sy-b2
empty (sy-b2 (ln, x)) = empty (x)

THEOREM: a2-e-sy-b2
(sy-b2 (ln, x) = E) = empty (x)

THEOREM: a2-lp-sy-b2
len (sy-b2 (ln, x)) = len (x)

THEOREM: a2-lpe-sy-b2
eqlen (sy-b2 (ln, x), x)

```
;(PROVE-LEMMA A2-PC-SY-B2 (REWRITE)
;    (EQUAL (P (SY-B2 LN X)) (SY-B2 LN (P X)))
;    ((DISABLE S-AND3 S-OR S-AND4 S-NOT A2-IC-S-AND3 A2-IC-S-OR
;              A2-IC-S-AND4 A2-IC-S-NOT)))

;; A2-End-SY-B2

; BCDS-is-B2 is the essence of this simplification.
; Note that replacing the conjunction by:
; (implies (or (equal ln 'YR1) (equal ln 'YR2) (equal ln 'YR3))
;          (equal (sy-bcds ln x) (sy-B2 ln x)))
; makes it UNPROVABLE by BM, because the induction hyp needs to be
; all 3 together, which BM won't do unless there is an explicit AND
; The thm is proved below in 2 immediates cases (on empty x).
```

THEOREM: bcds-is-b2
(sy-bcds (`'yr1`, x) = sy-b2 (`'yr1`, x))
$\land$ (sy-bcds (`'yr2`, x) = sy-b2 (`'yr2`, x))
$\land$ (sy-bcds (`'yr3`, x) = sy-b2 (`'yr3`, x))

```
; at this point we should never need SY-BCDS anymore:
```

EVENT: Disable sy-bcds.

```
; and also he does the expansion for Yout once and for all:
; Note: A-POSTERIORI analysis indicates that this lemma is not
; really useful to BM, which is usual, since it's just a
; non-recursive rewrite, and we might as well give the expand hint
; at the right place.
```

THEOREM: bcds-eq-yout
sy-bcds (`'yout`, x)
= s-not (s-and4 (x,
s-not (sy-b2 (`'yr1`, x)),
s-not (sy-b2 (`'yr2`, x)),
sy-b2 (`'yr3`, x)))

```
;; SECOND, he proves things about his DEROULEMENTS:
; note: all thms below are "one-shot", i.e. disabled and enabled
; explicitely.
```

```
; NOTE: at this point we express everything in terms of B2;
; obviously with BCDS-IS-B2 we can carry everything over.  This
; follows Paillet.
```

THEOREM: bcds-paillet-1
$(\text{len}(x) = \texttt{1})$
$\rightarrow \quad ((\text{l}(\text{sy-b2}(\texttt{'yr1}, x)) = \mathbf{f})$
$\quad \wedge \quad (\text{l}(\text{sy-b2}(\texttt{'yr2}, x)) = \mathbf{f})$
$\quad \wedge \quad (\text{l}(\text{sy-b2}(\texttt{'yr3}, x)) = \mathbf{f}))$

EVENT: Disable bcds-paillet-1.

THEOREM: bcds-paillet-1out
$(\text{len}(x) = \texttt{1}) \rightarrow (\text{l}(\text{sy-bcds}(\texttt{'yout}, x)) = \mathbf{t})$

EVENT: Disable bcds-paillet-1out.

THEOREM: bcds-paillet-2
$(\text{len}(x) = \texttt{2})$
$\rightarrow \quad ((\text{l}(\text{sy-b2}(\texttt{'yr1}, x)) = \mathbf{f})$
$\quad \wedge \quad (\text{l}(\text{sy-b2}(\texttt{'yr2}, x)) = \mathbf{t})$
$\quad \wedge \quad (\text{l}(\text{sy-b2}(\texttt{'yr3}, x)) = \mathbf{f}))$

EVENT: Disable bcds-paillet-2.

THEOREM: bcds-paillet-2out
$(\text{len}(x) = \texttt{2}) \rightarrow (\text{l}(\text{sy-bcds}(\texttt{'yout}, x)) = \mathbf{t})$

EVENT: Disable bcds-paillet-2out.

```
; Note that the "boolp" hyp is not explicit in Paillet...
```

THEOREM: bcds-paillet-3
$((\text{len}(x) = \texttt{3}) \wedge \text{s-boolp}(x))$
$\rightarrow \quad ((\text{l}(\text{sy-b2}(\texttt{'yr1}, x)) = (\neg\, \text{l}(\text{p}(x))))$
$\quad \wedge \quad (\text{l}(\text{sy-b2}(\texttt{'yr2}, x)) = \text{l}(\text{p}(x)))$
$\quad \wedge \quad (\text{l}(\text{sy-b2}(\texttt{'yr3}, x)) = \mathbf{t}))$

EVENT: Disable bcds-paillet-3.

THEOREM: bcds-paillet-3out
$((\text{len}(x) = \texttt{3}) \wedge \text{s-boolp}(x)) \rightarrow (\text{l}(\text{sy-bcds}(\texttt{'yout}, x)) = \mathbf{t})$

EVENT: Disable bcds-paillet-3out.

THEOREM: bcds-paillet-4
$((\mathrm{len}(x) = \mathtt{4}) \land \mathrm{s\text{-}boolp}(x))$
$\rightarrow\ ((\mathrm{l}(\mathrm{sy\text{-}b2}(\mathtt{'yr1}, x)) = ((\neg\, \mathrm{l}(\mathrm{p}(x))) \land (\neg\, \mathrm{l}(\mathrm{p}(\mathrm{p}(x))))))$
$\quad \land\ (\mathrm{l}(\mathrm{sy\text{-}b2}(\mathtt{'yr2}, x)) = \mathbf{f})$
$\quad \land\ (\mathrm{l}(\mathrm{sy\text{-}b2}(\mathtt{'yr3}, x))$
$\quad\quad = \ (\mathrm{l}(\mathrm{p}(\mathrm{p}(x))) \lor (\mathrm{l}(\mathrm{p}(x)) \land (\neg\, \mathrm{l}(\mathrm{p}(\mathrm{p}(x))))))))$

EVENT: Disable bcds-paillet-4.

```
; and his conclusion:
```

THEOREM: bcds-paillet-4out
$((\mathrm{len}(x) = \mathtt{4}) \land \mathrm{s\text{-}boolp}(x))$
$\rightarrow\ (\mathrm{l}(\mathrm{sy\text{-}bcds}(\mathtt{'yout}, x))$
$\quad = \ ((\neg\, \mathrm{l}(x)) \lor ((\neg\, \mathrm{l}(\mathrm{p}(x))) \land (\neg\, \mathrm{l}(\mathrm{p}(\mathrm{p}(x)))))))$

EVENT: Disable bcds-paillet-4out.

```
; from which he leaves to the reader the real conclusion:
```

THEOREM: bcds-paillet-4out-correct
$((\mathrm{len}(x) = \mathtt{4}) \land \mathrm{s\text{-}boolp}(x))$
$\rightarrow\ (\mathrm{l}(\mathrm{sy\text{-}bcds}(\mathtt{'yout}, x))$
$\quad = \ \mathrm{bcd\text{-}lbits}(\mathrm{l}(x), \mathrm{l}(\mathrm{p}(x)), \mathrm{l}(\mathrm{p}(\mathrm{p}(x))), \mathrm{l}(\mathrm{p}(\mathrm{p}(\mathrm{p}(x))))))$

EVENT: Disable bcds-paillet-4out-correct.

```
; and the last "re-initialization" condition:
```

THEOREM: bcds-paillet-5
$((\mathrm{len}(x) = \mathtt{5}) \land \mathrm{s\text{-}boolp}(x))$
$\rightarrow\ ((\mathrm{l}(\mathrm{sy\text{-}b2}(\mathtt{'yr1}, x)) = \mathbf{f})$
$\quad \land\ (\mathrm{l}(\mathrm{sy\text{-}b2}(\mathtt{'yr2}, x)) = \mathbf{f})$
$\quad \land\ (\mathrm{l}(\mathrm{sy\text{-}b2}(\mathtt{'yr3}, x)) = \mathbf{f}))$

EVENT: Disable bcds-paillet-5.

```
;;; WHAT I CAN PROVE! :


; The following lemma was thought to help, but in fact it just
; slows things:
;
;(prove-lemma STR-remainder-len (rewrite)
;(implies (and (not (zerop p)) (not (empty x)))
;  (equal (remainder (len x) p)
; (if (equal (remainder (len (P x)) p) (sub1 p))
;     0
;     (add1 (remainder (len (P x)) p)))))
;((disable remainder) ; so we go directly to ARI-remainder-add1
; (enable LEN) ; explicit
; )
;)
```

THEOREM: bcds-paillet-r-correct

$$
\begin{array}{l}
((\neg\, \mathrm{empty}\,(x)) \wedge \text{s-boolp}\,(x)) \\
\rightarrow \quad ((\mathrm{l}\,(\text{sy-b2}\,(\texttt{'yr1},\, x)) \\
\qquad = \quad \textbf{if}\ (\mathrm{len}\,(x)\ \textbf{mod}\ 4) = \texttt{1}\ \ \textbf{then f} \\
\qquad\qquad \textbf{elseif}\ (\mathrm{len}\,(x)\ \textbf{mod}\ 4) = \texttt{2}\ \ \textbf{then f} \\
\qquad\qquad \textbf{elseif}\ (\mathrm{len}\,(x)\ \textbf{mod}\ 4) = \texttt{3}\ \ \textbf{then}\ \neg\, \mathrm{l}\,(\mathrm{p}\,(x)) \\
\qquad\qquad \textbf{else}\ (\neg\, \mathrm{l}\,(\mathrm{p}\,(x))) \wedge (\neg\, \mathrm{l}\,(\mathrm{p}\,(\mathrm{p}\,(x))))\ \textbf{endif}) \\
\qquad \wedge \quad (\mathrm{l}\,(\text{sy-b2}\,(\texttt{'yr2},\, x)) \\
\qquad\quad = \quad \textbf{if}\ (\mathrm{len}\,(x)\ \textbf{mod}\ 4) = \texttt{1}\ \ \textbf{then f} \\
\qquad\qquad\quad \textbf{elseif}\ (\mathrm{len}\,(x)\ \textbf{mod}\ 4) = \texttt{2}\ \ \textbf{then t} \\
\qquad\qquad\quad \textbf{elseif}\ (\mathrm{len}\,(x)\ \textbf{mod}\ 4) = \texttt{3}\ \ \textbf{then}\ \mathrm{l}\,(\mathrm{p}\,(x)) \\
\qquad\qquad\quad \textbf{else f endif}) \\
\qquad \wedge \quad (\mathrm{l}\,(\text{sy-b2}\,(\texttt{'yr3},\, x)) \\
\qquad\quad = \quad \textbf{if}\ (\mathrm{len}\,(x)\ \textbf{mod}\ 4) = \texttt{1}\ \ \textbf{then f} \\
\qquad\qquad\quad \textbf{elseif}\ (\mathrm{len}\,(x)\ \textbf{mod}\ 4) = \texttt{2}\ \ \textbf{then f} \\
\qquad\qquad\quad \textbf{elseif}\ (\mathrm{len}\,(x)\ \textbf{mod}\ 4) = \texttt{3}\ \ \textbf{then t} \\
\qquad\qquad\quad \textbf{else}\ \mathrm{l}\,(\mathrm{p}\,(\mathrm{p}\,(x))) \\
\qquad\qquad\qquad \vee \quad (\mathrm{l}\,(\mathrm{p}\,(x)) \wedge (\neg\, \mathrm{l}\,(\mathrm{p}\,(\mathrm{p}\,(x)))))\ \textbf{endif}))
\end{array}
$$

```
; and finally, the true, general correctness of Paillet#5 :
```

THEOREM: bcds-paillet-yout-correct

$$
\begin{array}{l}
((\neg\, \mathrm{empty}\,(x)) \wedge \text{s-boolp}\,(x)) \\
\rightarrow \quad (\mathrm{l}\,(\text{sy-bcds}\,(\texttt{'yout},\, x)) \\
\qquad = \quad \textbf{if}\ (\mathrm{len}\,(x)\ \textbf{mod}\ 4) = \texttt{0} \\
\qquad\qquad \textbf{then}\ \text{bcd-lbits}\,(\mathrm{l}\,(x), \mathrm{l}\,(\mathrm{p}\,(x)), \mathrm{l}\,(\mathrm{p}\,(\mathrm{p}\,(x))), \mathrm{l}\,(\mathrm{p}\,(\mathrm{p}\,(\mathrm{p}\,(x)))))
\end{array}
$$

else t endif)

```
; we also check the claim we make in the Ccube89 report, that we
; need only one "build-up" lemma (R-correct). In works fine
; (same # cases, time) with the (obvious) expand hint we used
; in BCDS-eq-Yout:
; (expand (SY-BCDS 'Yout x)
;  (SY-BCDS 'Y41 x)
;  (SY-BCDS 'Y31 x)
;  (SY-BCDS 'Y32 x)
;  )
; once again confirming that reifying rewrite lemmas which do not
; involve induction does not really benefit Boyer-Moore, even
; though it benefit humans!

; eof: bcdS.bm
;))
```

Index