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:  and3(u1, u2, u3) = (u1 \land u2 \land u3)

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

\[
s-\text{and3}(x_1, x_2, x_3) = \begin{cases} 
\text{if empty}(x_1) & \text{then } E \\
\text{else } a(s-\text{and3}(p(x_1), p(x_2), p(x_3)), \text{and3}(l(x_1), l(x_2), l(x_3))) & \text{endif}
\end{cases}
\]

**Theorem:** a2-empty-s-and3

\[
\text{empty}(s-\text{and3}(x_1, x_2, x_3)) = \text{empty}(x_1)
\]

**Theorem:** a2-e-s-and3

\[
(s-\text{and3}(x_1, x_2, x_3) = E) = \text{empty}(x_1)
\]

**Theorem:** a2-lp-s-and3

\[
\text{len}(s-\text{and3}(x_1, x_2, x_3)) = \text{len}(x_1)
\]

**Theorem:** a2-lpe-s-and3

\[
\text{eqlen}(s-\text{and3}(x_1, x_2, x_3), x_1)
\]

**Theorem:** a2-ic-s-and3

\[
((\text{len}(x_1) = \text{len}(x_2)) \land (\text{len}(x_2) = \text{len}(x_3))) \\
\rightarrow (s-\text{and3}(i(c_{x_1}, x_1), i(c_{x_2}, x_2), i(c_{x_3}, x_3))) \\
\quad = i(\text{and3}(c_{x_1}, c_{x_2}, c_{x_3}), s-\text{and3}(x_1, x_2, x_3)))
\]

**Theorem:** a2-lc-s-and3

\[
(\neg \text{empty}(x_1)) \rightarrow (l(s-\text{and3}(x_1, x_2, x_3)) = \text{and3}(l(x_1), l(x_2), l(x_3)))
\]

**Theorem:** a2-pc-s-and3

\[
p(s-\text{and3}(x_1, x_2, x_3)) = s-\text{and3}(p(x_1), p(x_2), p(x_3))
\]

**Theorem:** a2-hc-s-and3

\[
((\neg \text{empty}(x_1)) \land ((\text{len}(x_1) = \text{len}(x_2)) \land (\text{len}(x_2) = \text{len}(x_3)))) \\
\rightarrow (h(s-\text{and3}(x_1, x_2, x_3)) = \text{and3}(h(x_1), h(x_2), h(x_3)))
\]

**Theorem:** a2-bc-s-and3

\[
((\text{len}(x_1) = \text{len}(x_2)) \land (\text{len}(x_2) = \text{len}(x_3))) \\
\rightarrow (b(s-\text{and3}(x_1, x_2, x_3)) = s-\text{and3}(b(x_1), b(x_2), b(x_3)))
\]

**Theorem:** a2-bnc-s-and3

\[
((\text{len}(x_1) = \text{len}(x_2)) \land (\text{len}(x_2) = \text{len}(x_3))) \\
\rightarrow (bn(n, s-\text{and3}(x_1, x_2, x_3)) = s-\text{and3}(bn(n, x_1), bn(n, x_2), bn(n, x_3)))
\]

;; 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:
\[s\text{-and4}(x1, x2, x3, x4) = \begin{cases} 
\text{if empty}(x1) \text{ then } e \\
\text{else } a(s\text{-and4}(p(x1), p(x2), p(x3), p(x4)), \\
\text{and4}(l(x1), l(x2), l(x3), l(x4)))) \end{cases} \]

;; A2-Begin-S-AND4

Theorem: a2-empty-s-and4
\[\text{empty}(s\text{-and4}(x1, x2, x3, x4)) = \text{empty}(x1)\]

Theorem: a2-e-s-and4
\[(s\text{-and4}(x1, x2, x3, x4) = e) = \text{empty}(x1)\]

Theorem: a2-lp-s-and4
\[\text{len}(s\text{-and4}(x1, x2, x3, x4)) = \text{len}(x1)\]

Theorem: a2-lpe-s-and4
\[\text{eqlen}(s\text{-and4}(x1, x2, x3, x4), x1)\]

Theorem: a2-ic-s-and4
\[\left(\text{len}(x1) = \text{len}(x2) \land \text{len}(x2) = \text{len}(x3) \land \text{len}(x3) = \text{len}(x4)\right) \rightarrow \left(h(s\text{-and4}(x1, x2, x3, x4)) = \text{and4}(h(x1), h(x2), h(x3), h(x4))\right)\]

Theorem: a2-bc-s-and4
\[\left((\text{len}(x1) = \text{len}(x2)) \land (\text{len}(x2) = \text{len}(x3)) \land (\text{len}(x3) = \text{len}(x4))\right) \rightarrow \left(b(s\text{-and4}(x1, x2, x3, x4)) = s\text{-and4}(b(x1), b(x2), b(x3), b(x4))\right)\]
**Theorem:** a2-bnc-s-and4

$((\text{len}(x1) = \text{len}(x2)) \land (\text{len}(x2) = \text{len}(x3)) \land (\text{len}(x3) = \text{len}(x4))) 
\rightarrow (\text{bn}(n, \text{and4}(x1, x2, x3, x4)) 
= \text{and4}(\text{bn}(n, x1), \text{bn}(n, x2), \text{bn}(n, x3), \text{bn}(n, x4)))$

;; 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))
  else 0 endif
elseif ln = 'y06 then s-not (sy-bcdfs ('yr3, x))
elseif ln = 'y07 then s-not (sy-bcdfs ('yr1, x))
elseif ln = 'y08 then s-not (sy-bcdfs ('yr3, x))
elseif ln = 'y11 then s-and3 (sy-bcdfs ('y01, x), sy-bcdfs ('yr1, x), sy-bcdfs ('yr3, x))
elseif ln = 'y12 then s-and3 (sy-bcdfs ('y02, x), sy-bcdfs ('yr2, x), sy-bcdfs ('y03, x))
elseif ln = 'y13 then s-and3 (sy-bcdfs ('y04, x), sy-bcdfs ('y05, x), sy-bcdfs ('y06, x))
elseif ln = 'y14 then s-and3 (x, sy-bcdfs ('y07, x), sy-bcdfs ('y08, x))
elseif ln = 'y15 then s-and3 (x, sy-bcdfs ('yr1, x), sy-bcdfs ('yr3, x))
elseif ln = 'y21 then s-or (sy-bcdfs ('y11, x), sy-bcdfs ('y12, x))
elseif ln = 'y22 then s-or (sy-bcdfs ('y13, x), sy-bcdfs ('y14, x))
elseif ln = 'y23 then s-or (sy-bcdfs ('yr2, x), sy-bcdfs ('y15, x))
elseif ln = 'yr1 then if empty (x) then E
else i(f, sy-bcdfs ('y21, p(x))) endif
elseif ln = 'yr2 then if empty (x) then E
else i(f, sy-bcdfs ('y22, p(x))) endif
elseif ln = 'yr3 then if empty (x) then E
else i(f, sy-bcdfs ('y23, p(x))) endif
elseif ln = 'y31 then s-not (sy-bcdfs ('yr1, x))
elseif ln = 'y32 then s-not (sy-bcdfs ('yr2, x))
elseif ln = 'y41 then s-and4 (x, sy-bcdfs ('y31, x), sy-bcdfs ('y32, x), sy-bcdfs ('yr3, x))
elseif ln = 'yout then s-not (sy-bcdfs ('y41, x))
else sfix (x) endif

;; A2-Begin-SY-BCDS

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

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

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

THEOREM: a2-lpe-sy-bcdfs
eqlen (sy-bcdfs (ln, x), x)
THEOREM: a2-pc-sy-bcds
\[ p(\text{sy-bcds}(ln, x)) = \text{sy-bcds}(ln, p(x)) \]

;; 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:
bcd-lbits(b0, b1, b2, b3) = ((b0 = f) \lor ((b1 = f) \land (b2 = 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:

7
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 immediate cases (on empty x).

Theorem: bcds-is-b2
(sy-bcds ('yr1, x) = sy-b2 ('yr1, x))
\wedge (sy-bcds ('yr2, x) = sy-b2 ('yr2, x))
\wedge (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-PPOSTERIORI 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
\[
(len(x) = 1) \rightarrow ((l(sy-b2('yr1, x)) = f) \\
\land (l(sy-b2('yr2, x)) = f) \\
\land (l(sy-b2('yr3, x)) = f))
\]

**Event:** Disable bcds-paillet-1.

**Theorem:** bcds-paillet-1out
\[
(len(x) = 1) \rightarrow (l(sy-bcds('yout, x)) = t)
\]

**Event:** Disable bcds-paillet-1out.

**Theorem:** bcds-paillet-2
\[
(len(x) = 2) \rightarrow ((l(sy-b2('yr1, x)) = f) \\
\land (l(sy-b2('yr2, x)) = t) \\
\land (l(sy-b2('yr3, x)) = f))
\]

**Event:** Disable bcds-paillet-2.

**Theorem:** bcds-paillet-2out
\[
(len(x) = 2) \rightarrow (l(sy-bcds('yout, x)) = t)
\]

**Event:** Disable bcds-paillet-2out.

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

**Theorem:** bcds-paillet-3
\[
((len(x) = 3) \land s-boolp(x)) \rightarrow ((l(sy-b2('yr1, x)) = (\neg l(p(x)))) \\
\land (l(sy-b2('yr2, x)) = l(p(x))) \\
\land (l(sy-b2('yr3, x)) = t))
\]

**Event:** Disable bcds-paillet-3.

**Theorem:** bcds-paillet-3out
\[
((len(x) = 3) \land s-boolp(x)) \rightarrow (l(sy-bcds('yout, x)) = t)
\]
Event: Disable bcds-paillet-3out.

Theorem: bcds-paillet-4
\[ ((\text{len}(x) = 4) \land \text{s-boolp}(x)) \rightarrow \left( (l(sy-b2('yr1, x)) = \neg l(p(x)) \land \neg l(p(p(x)))) \land (l(sy-b2('yr2, x)) = f) \land (l(sy-b2('yr3, x)) = \neg (l(p(p(x))) \lor (l(p(x)) \land \neg l(p(p(x))))) \right) \]

Event: Disable bcds-paillet-4.

; and his conclusion:

Theorem: bcds-paillet-4out
\[ ((\text{len}(x) = 4) \land \text{s-boolp}(x)) \rightarrow (l(sy-bcds('yout, x)) = \neg l(x) \lor (\neg l(p(x)) \land \neg l(p(p(x)))) \land (l(sy-b2('yr1, x)) = f) \land (l(sy-b2('yr2, x)) = f) \land (l(sy-b2('yr3, x)) = f)) \]

Event: Disable bcds-paillet-4out.

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

Theorem: bcds-paillet-4out-correct
\[ ((\text{len}(x) = 4) \land \text{s-boolp}(x)) \rightarrow (l(sy-bcds('yout, x)) = \text{bcd-lbits}(l(x), l(p(x)), l(p(p(x))), l(p(p(p(x))))) \land (l(sy-b2('yr1, x)) = f) \land (l(sy-b2('yr2, x)) = f) \land (l(sy-b2('yr3, x)) = f)) \]

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

; and the last "re-initialization" condition:

Theorem: bcds-paillet-5
\[ ((\text{len}(x) = 5) \land \text{s-boolp}(x)) \rightarrow (l(sy-b2('yr1, x)) = f) \land (l(sy-b2('yr2, x)) = f) \land (l(sy-b2('yr3, x)) = f)) \]

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
((¬ empty (x)) ∧ s-boolp (x))
→ ((l (sy-b2 (‘yr1, x))
  =  if (len (x) mod 4) = 1 then f
      elseif (len (x) mod 4) = 2 then f
      elseif (len (x) mod 4) = 3 then ¬1(p(x))
      else ¬1(p(x)) ∧ ¬1(p(p(x))) endif)
∧ (l (sy-b2 (‘yr2, x))
  =  if (len (x) mod 4) = 1 then f
      elseif (len (x) mod 4) = 2 then t
      elseif (len (x) mod 4) = 3 then 1(p(x))
      else f endif)
∧ (l (sy-b2 (‘yr3, x))
  =  if (len (x) mod 4) = 1 then f
      elseif (len (x) mod 4) = 2 then f
      elseif (len (x) mod 4) = 3 then t
      else 1(p(p(x)))
      ∨ (1(p(x)) ∧ ¬1(p(p(x)))) endif))

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

Theorem: bcds-paillet-yout-correct
((¬ empty (x)) ∧ s-boolp (x))
→ (l (sy-bcds (‘yout, x))
  =  if (len (x) mod 4) = 0
      then bcd-lbits (l(x), 1(p(x)), 1(p(p(x))), 1(p(p(p(x)))))
      ∧ (l (sy-b2 (‘yr2, x))
  =  if (len (x) mod 4) = 1 then f
      elseif (len (x) mod 4) = 2 then f
      elseif (len (x) mod 4) = 3 then t
      else f endif)
∧ (l (sy-b2 (‘yr3, x))
  =  if (len (x) mod 4) = 1 then f
      elseif (len (x) mod 4) = 2 then f
      elseif (len (x) mod 4) = 3 then t
      else 1(p(p(x)))
      ∨ (1(p(x)) ∧ ¬1(p(p(x)))) endif))

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

a, 3, 4
a2-bc-s-and3, 3
a2-bc-s-and4, 4
a2-bnc-s-and3, 3
a2-bnc-s-and4, 5
a2-e-s-and3, 3
a2-e-s-and4, 4
a2-e-sy-b2, 8
a2-e-sy-bcds, 6
a2-empty-s-and3, 3
a2-empty-s-and4, 4
a2-empty-sy-b2, 8
a2-empty-sy-bcds, 6
a2-hc-s-and3, 3
a2-hc-s-and4, 4
a2-ic-s-and3, 3
a2-ic-s-and4, 4
a2-ic-sy-bcds, 4
a2-lc-s-and3, 3
a2-lc-s-and4, 4
a2-lc-sy-bcds, 4
a2-lp-s-and3, 3
a2-lp-s-and4, 4
a2-lp-sy-b2, 8
a2-lp-sy-bcds, 6
a2-lpe-s-and3, 3
a2-lpe-s-and4, 4
a2-lpe-sy-b2, 9
a2-lpe-sy-bcds, 6
a2-pc-s-and3, 3
a2-pc-s-and4, 4
a2-pc-sy-bcds, 7
and3, 2, 3
and4, 4

bcd-paillet, 3, 6
bcds-paillet-3out, 10
bcds-paillet-4, 11
bcds-paillet-4out, 11
bcds-paillet-4out-correct, 11
bcds-paillet-5, 11
bcds-paillet-r-correct, 12
bcds-paillet-yout-correct, 12
bn, 3, 5
e, 3, 4, 6, 8
empty, 3, 4, 6, 8, 12
eqen, 3, 4, 6, 9
h, 3, 4
i, 3, 4, 6, 8
l, 3, 4, 10–12
len, 3–6, 8, 10–12
p, 3, 4, 6–8, 10–12
s-and3, 3, 6, 8
s-and4, 4–6, 9
s-boolp, 10–12
s-not, 5, 6, 8, 9
s-or, 6, 8
sfix, 6, 8
sy-b2, 8–12
sy-bcds, 5–7, 9–12
topor-sy-bcds, 5

b, 3, 4
bcd-lbits, 7, 11, 12
bcds-eq-yout, 9
bcds-is-b2, 9
bcds-paillet-1, 10
bcds-paillet-1out, 10
bcds-paillet-2, 10
bcds-paillet-2out, 10