Using PVS to Specify and Validate Programmable Logic Controllers

EECS4312-F15 York University
JSO

Contents

- *HYSTERESIS* Block As a Function Table?
- *LIMITS_ALARM* Block from IEC 61131-3
- *HYSTERESIS* & *LIMITS_ALARM*: Verification Roadmap
- Lab

Key Questions

- What are PLCs?
- Why are PLCs important?
- What to specify about PLCs?
- What to verify about PLCs?
- Are function tables useful?
- Why bother PVS?
  - Or, really, why bother mathematics?

A Visual Introduction to PLCs

Note: Many of the PLC and illustration diagrams below are originated from the book *Programmable Logic Controllers* (4th Edition; McGraw-Hill) by Frank D. Petruzella.

Programmable Logic Controllers (PLCs) in a Nutshell

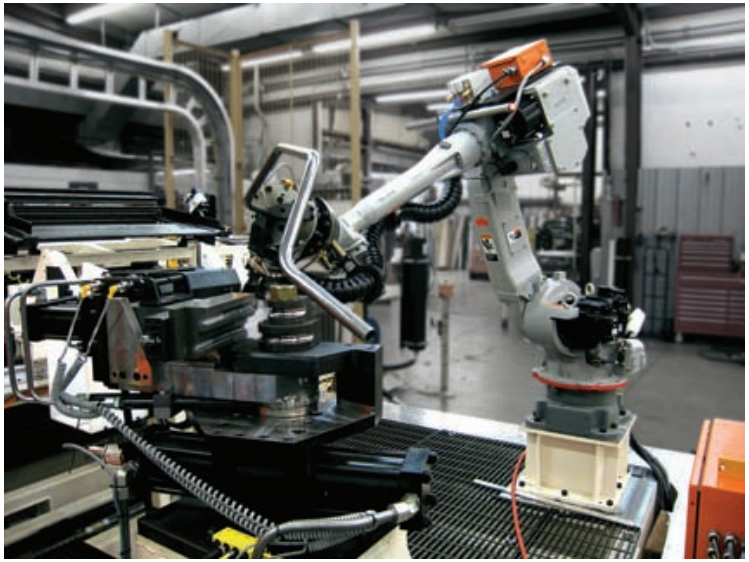
A PLC is a specialized computer that performs control functions.

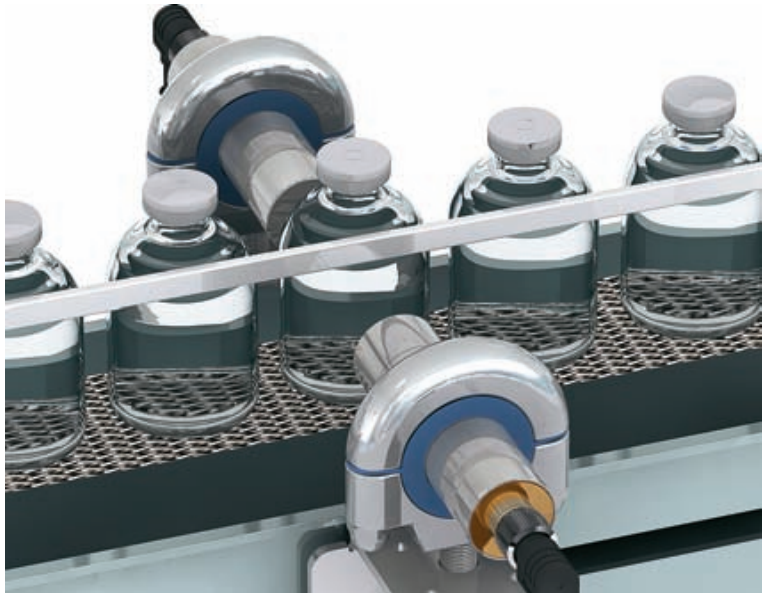
- eliminated the *physical hardwiring* used to implement the control logic for conventional *relay control circuits*
- support more than relay switching tasks: e.g., timing, counting, calculating, comparing, and the processing of analog signals
- multiple *I/O ports* for connections with filed devices ("the real world")

What are the advantages of PLCs?

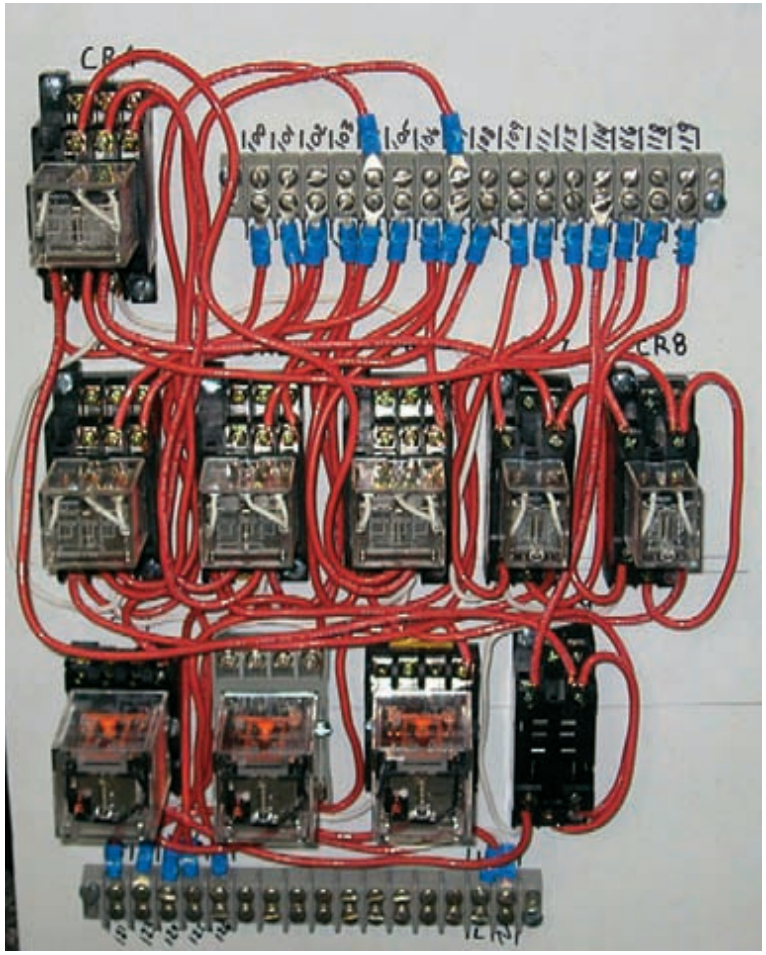
- *flexible*: functions can be programmed and loaded (i.e., no rewiring)
- *reliable*: much less likely to make logical errors
- *responsive*: real-time output updates in response to plant changes
- *maintainable*: program executions can be displayed in a monitor
- *economical*: cheaper to install when ≥ 6 control relays are required

PLCs: Utilized in Automating Industrial Process Control

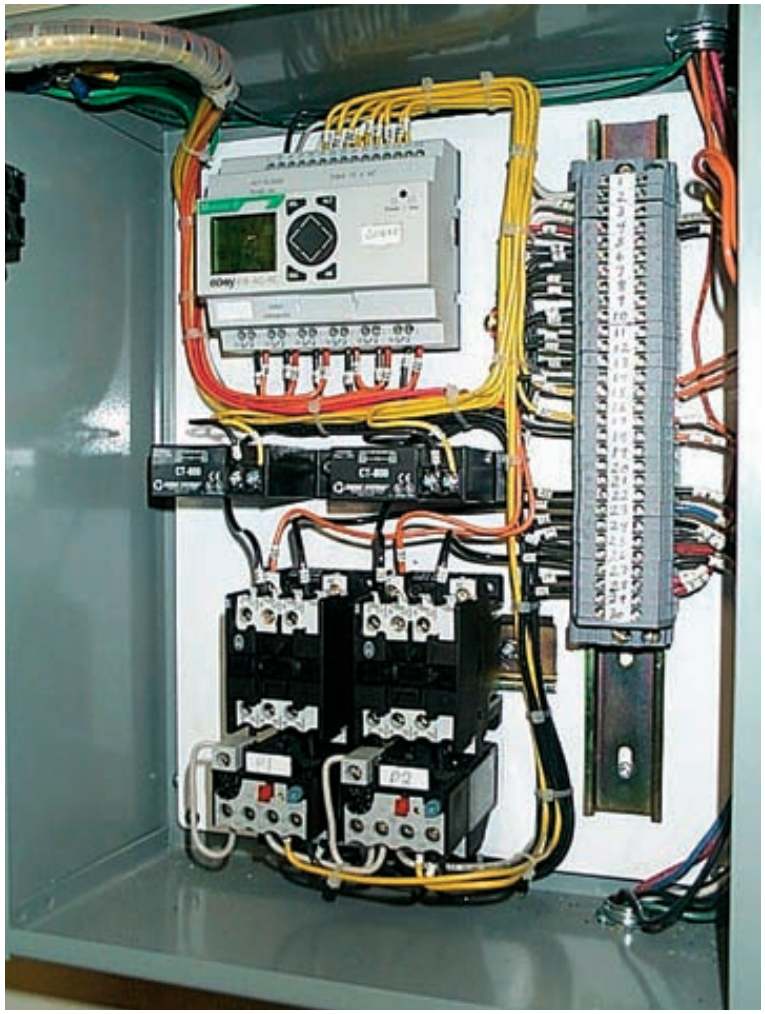




PLCs: Replacement for Relay-based Controllers



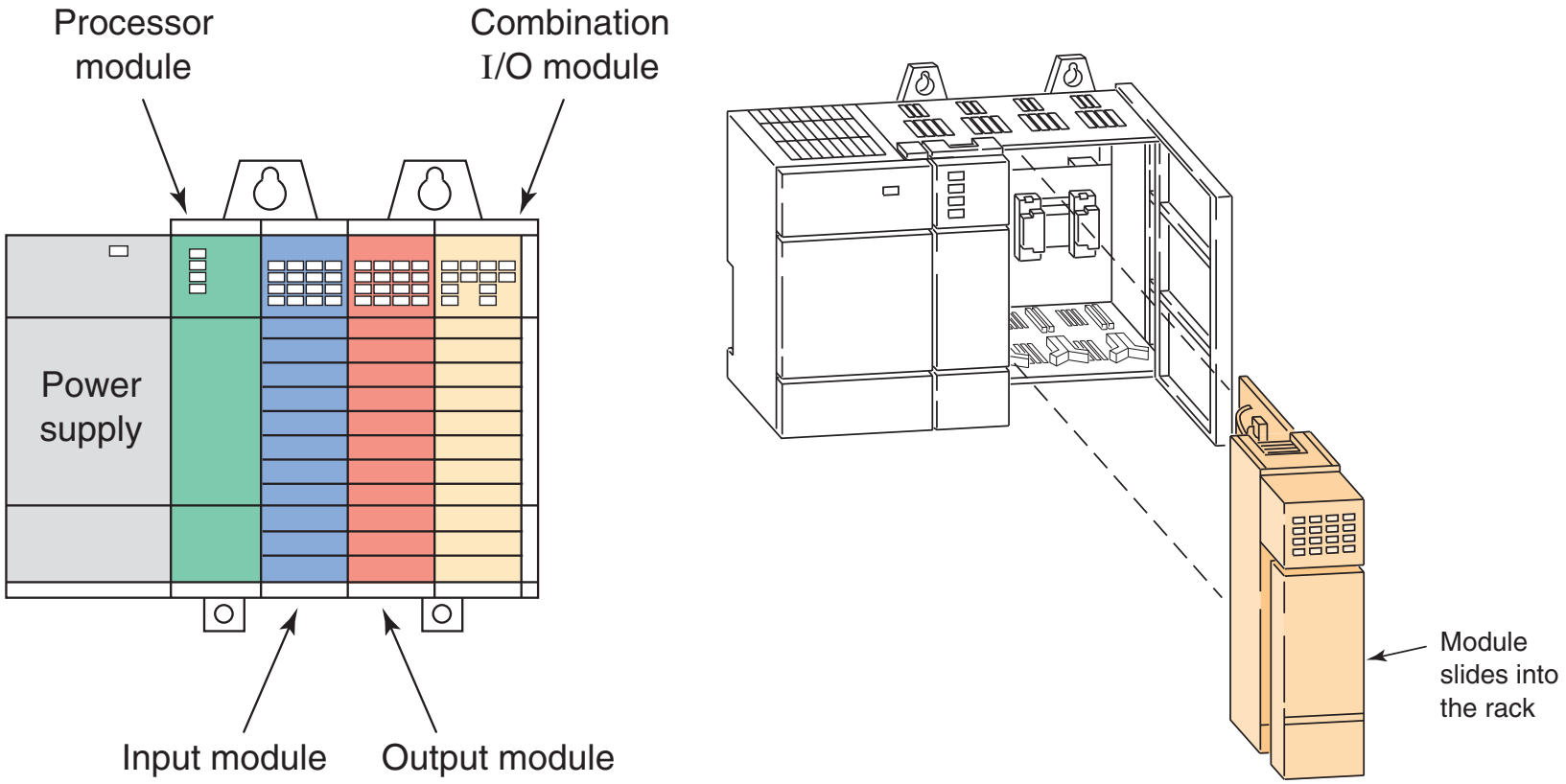
(a) Relay-based Control Panel



(b) PLC-based Control Panel

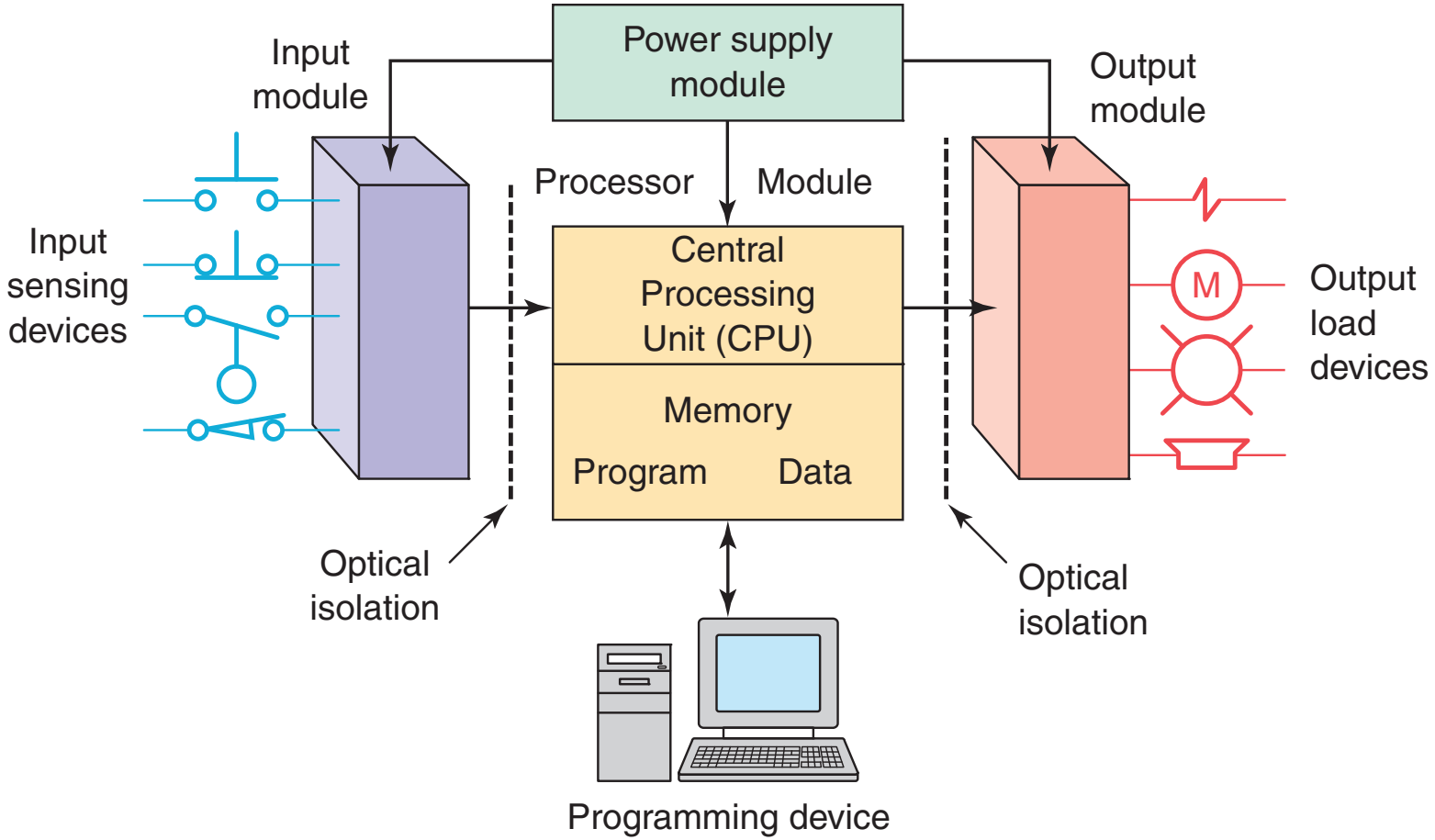
PLCs: Modular Hardware Design

Processor and I/O modules can be plugged into compartments of a rack.

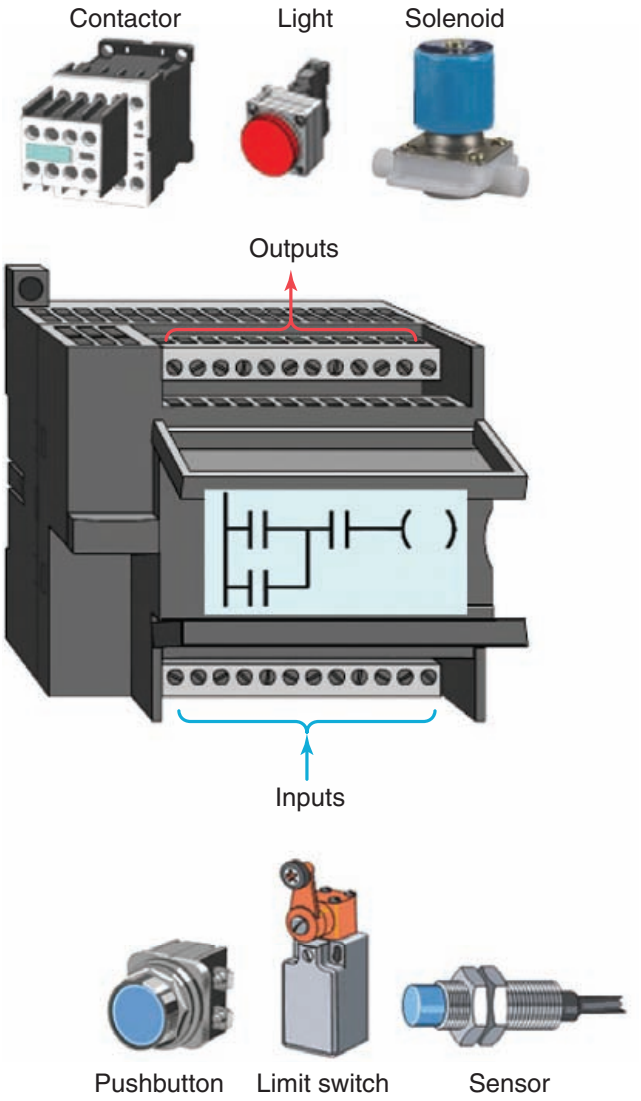


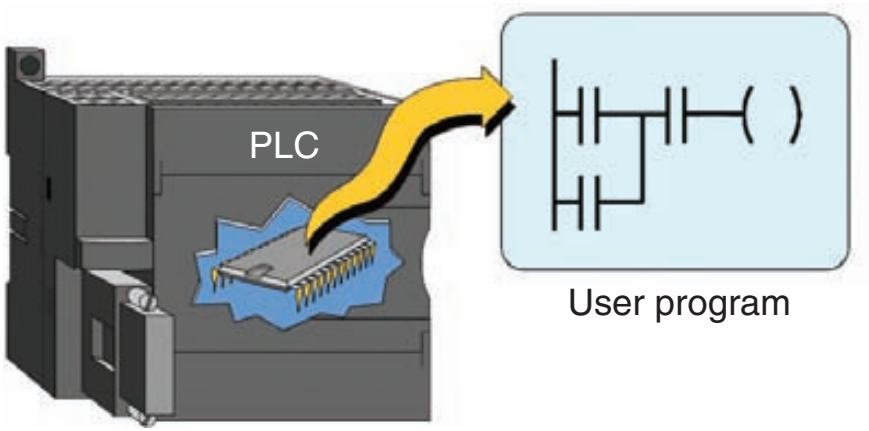
PLCs: Schematic

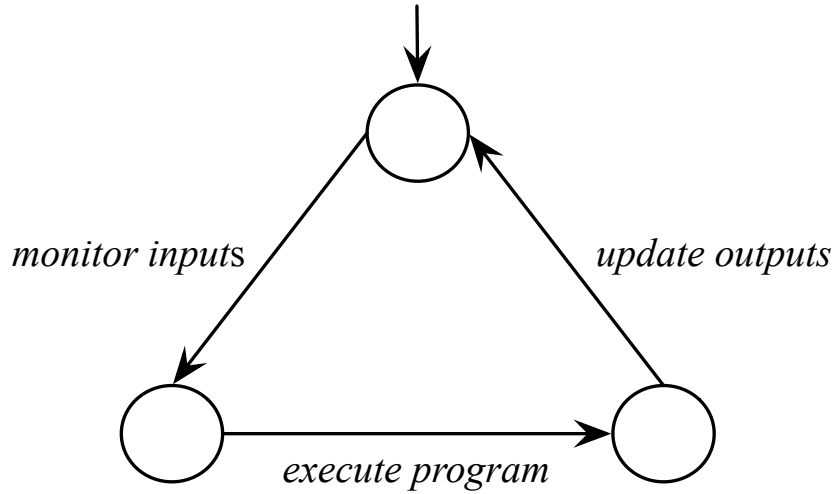
Programs written in, e.g., relay ladder logic, are loaded into the memory.



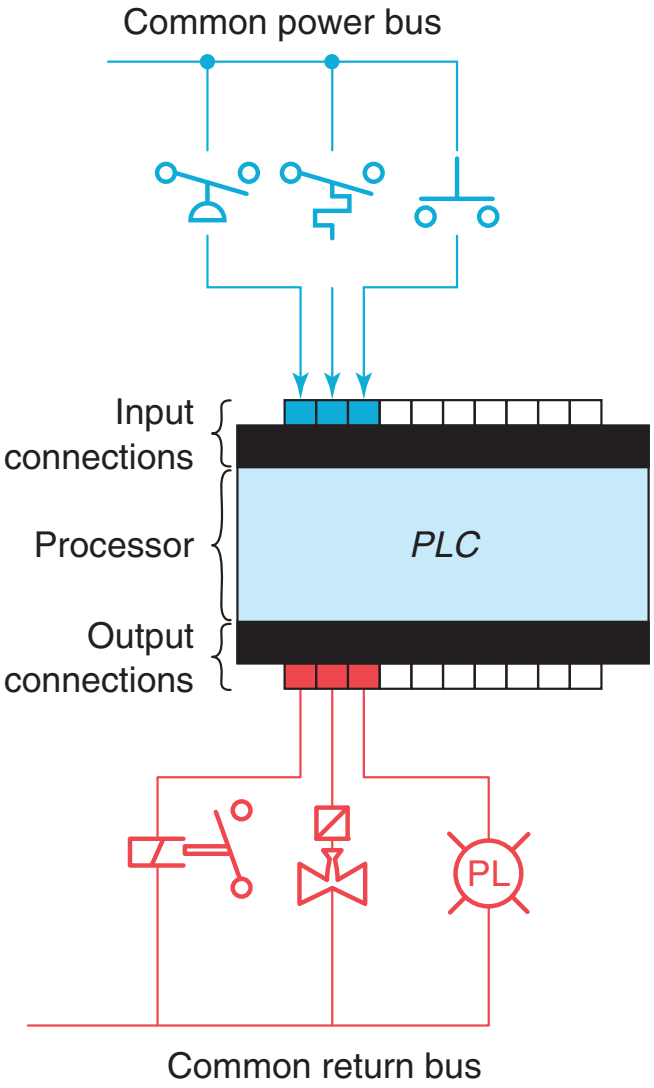
PLCs as Cyclic Executives: Inputs, Outputs, Repeated Scans



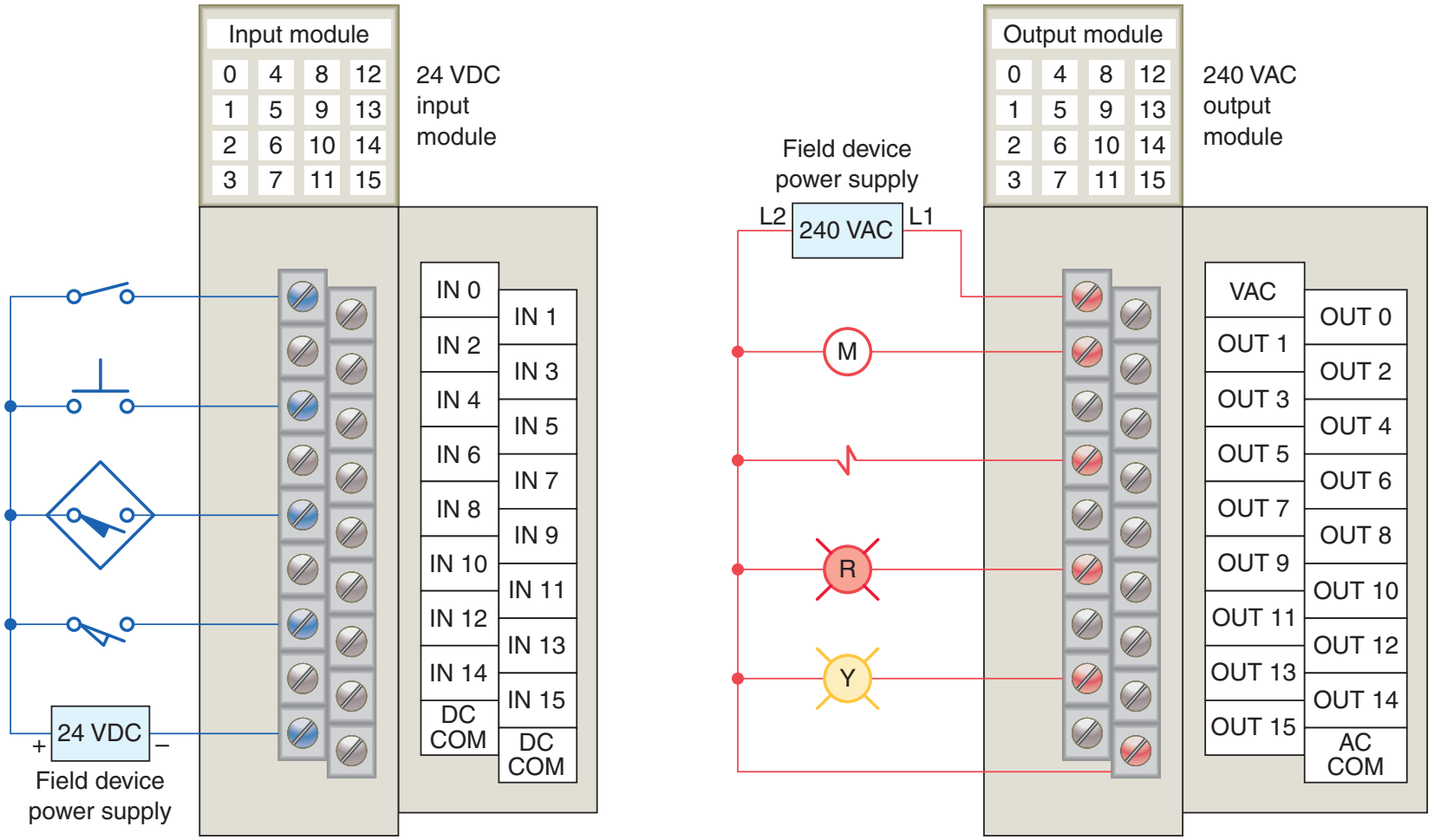




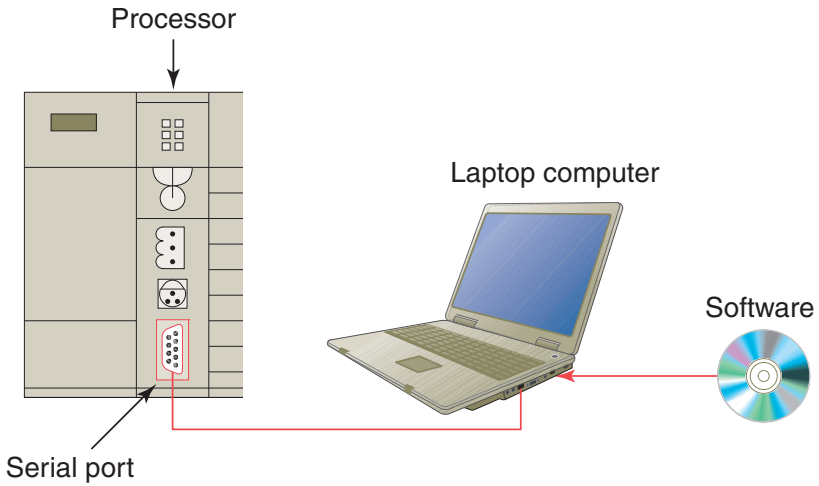
PLCs: Connecting to the "Real World"

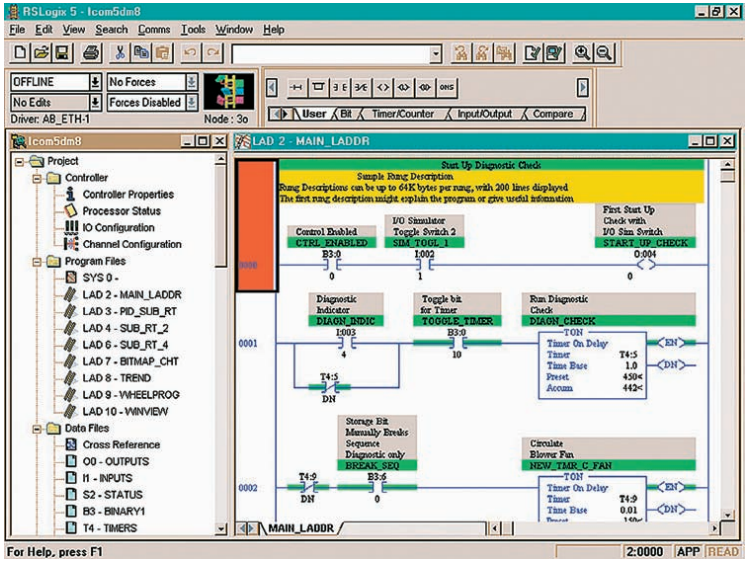


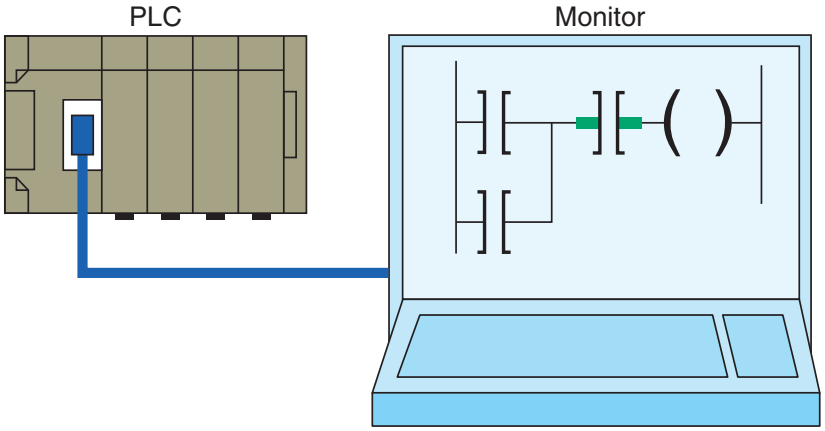
- Field devices are connected through the interface formed by I/O ports
- PLCs *monitor* the plant by receiving input signals from e.g., pushbuttons, limit switches, sensors
- PLCs *respond* to plant changes by sending output signals to e.g., contactor, light, solenoid

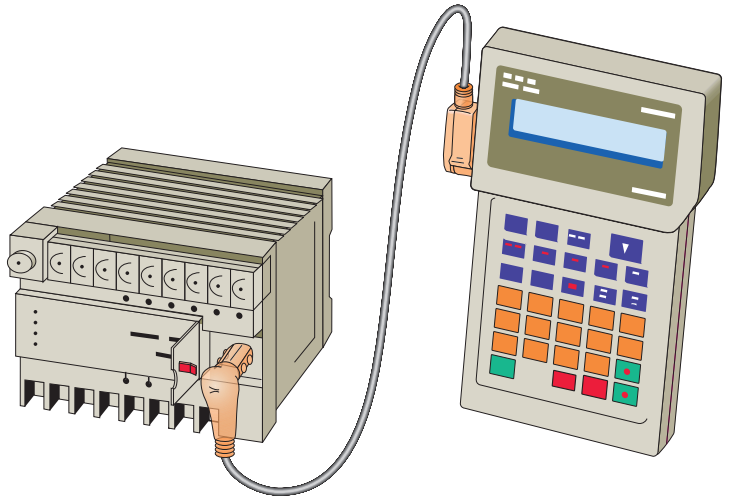

Input module
0 4 8 12
1 5 9 13
2 6 10 14
3 7 11 15
24 VDC input module
IN 0
IN 1
IN 2
IN 3
IN 4
IN 5
IN 6
IN 7
IN 8
IN 9
IN 10
IN 11
IN 12
IN 13
IN 14
IN 15
DC COM
DC COM
24 VDC
+
–
Field device power supply
Output module
0 4 8 12
1 5 9 13
2 6 10 14
3 7 11 15
240 VAC output module
Field device power supply
L2
240 VAC
L1
M
R
Y
VAC
OUT 0
OUT 1
OUT 2
OUT 3
OUT 4
OUT 5
OUT 6
OUT 7
OUT 8
OUT 9
OUT 10
OUT 11
OUT 12
OUT 13
OUT 14
OUT 15
AC COM

PLCs: Programming & Debugging Interfaces

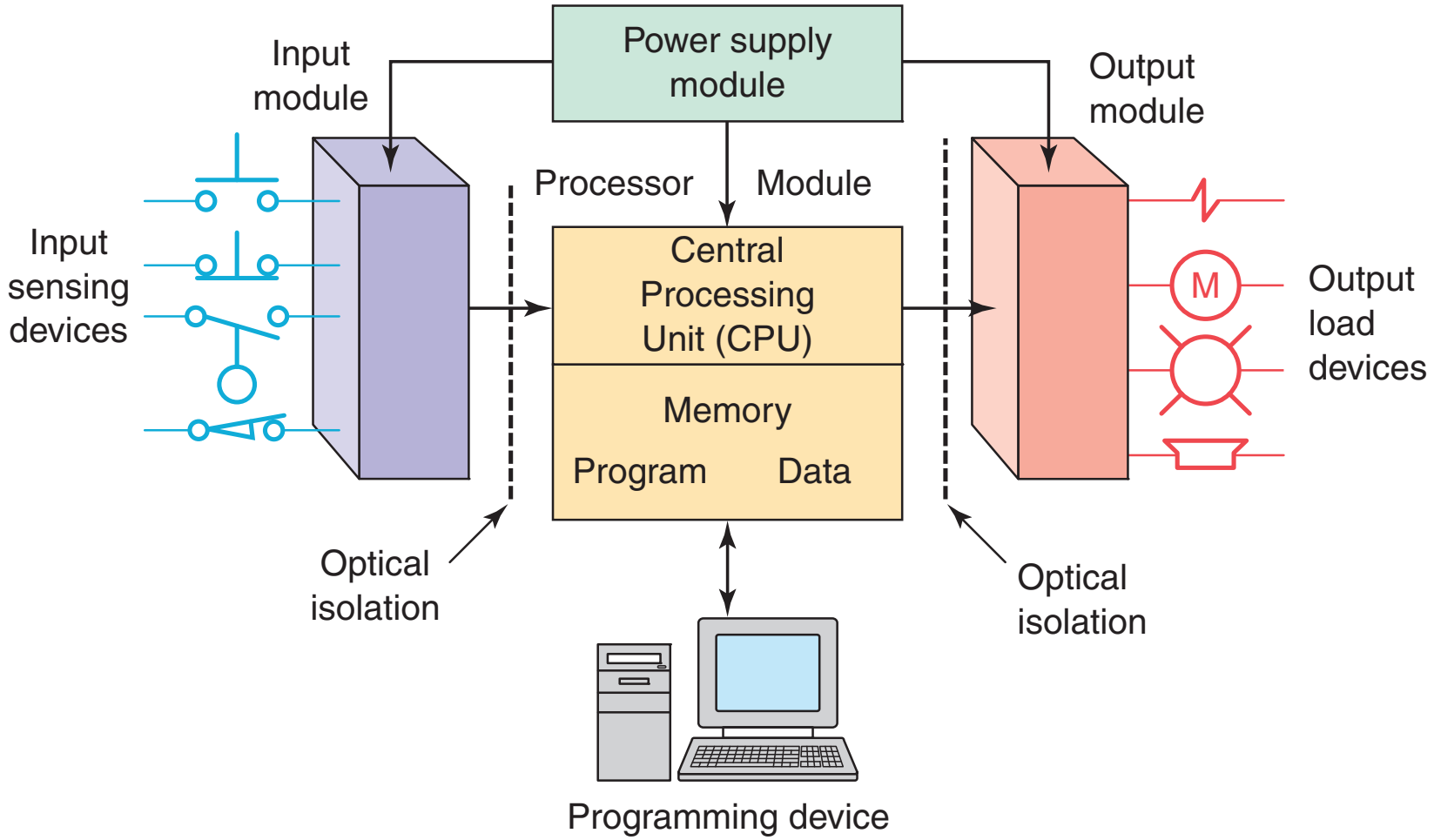








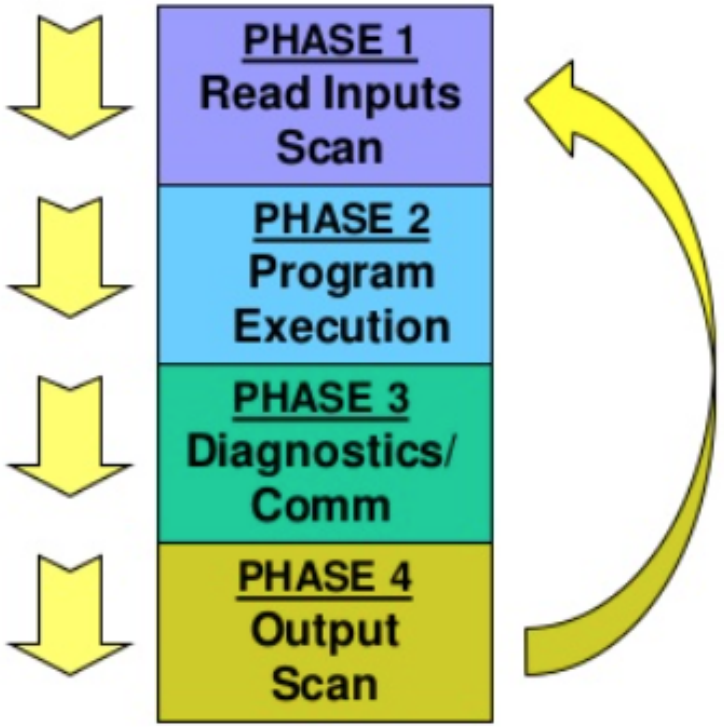
PLC Schematic



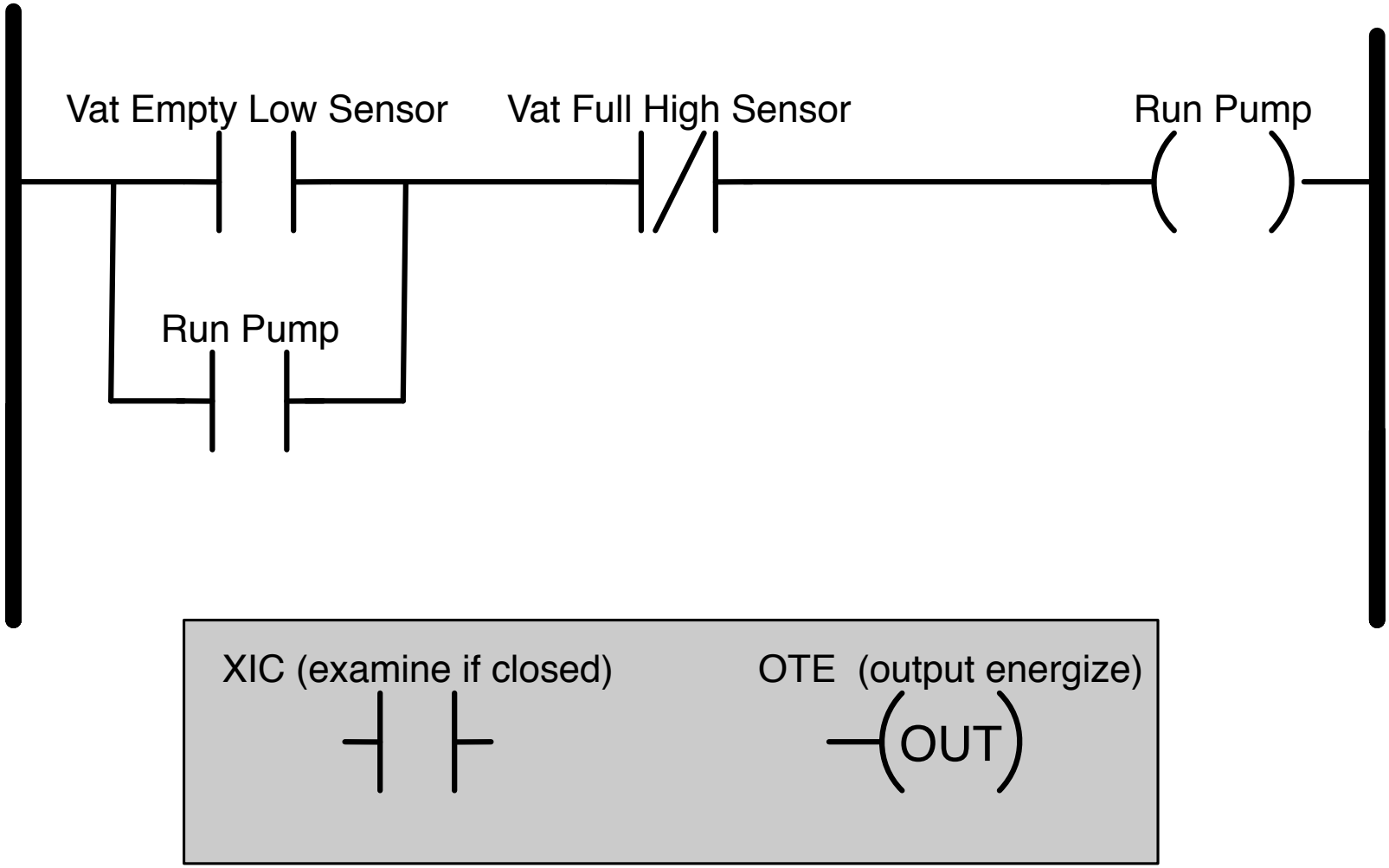
Scan Cycle (5ms to 150ms)

While the PLC is running, the scanning process includes the four phases, which are repeated continuously as individual cycles of operation:

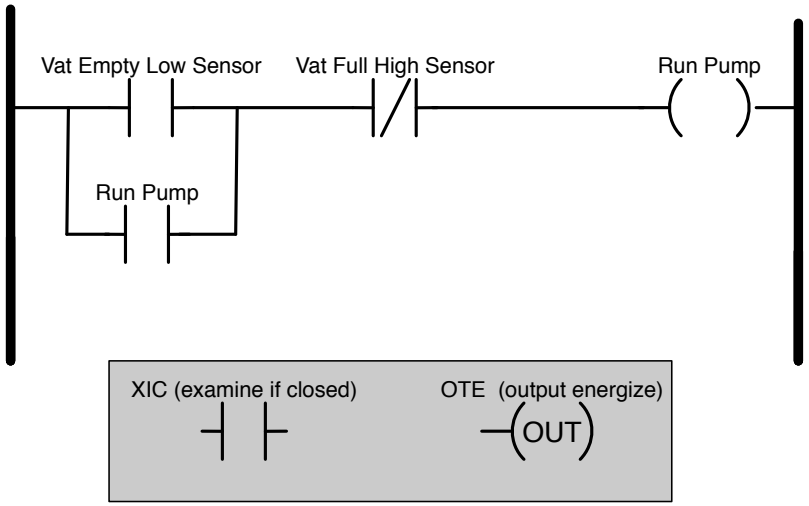
- **Input Scan**- Scan the state of the Inputs
- **Program Execution**- Processes and executes the program logic
- **Housekeeping**- This step includes communications, Internal Diagnostics, etc.
- **Output Scan**- Energize/de-energize the outputs



Ladder Logic to keep a vat of liquid filled: latch-on/latch-off circuit

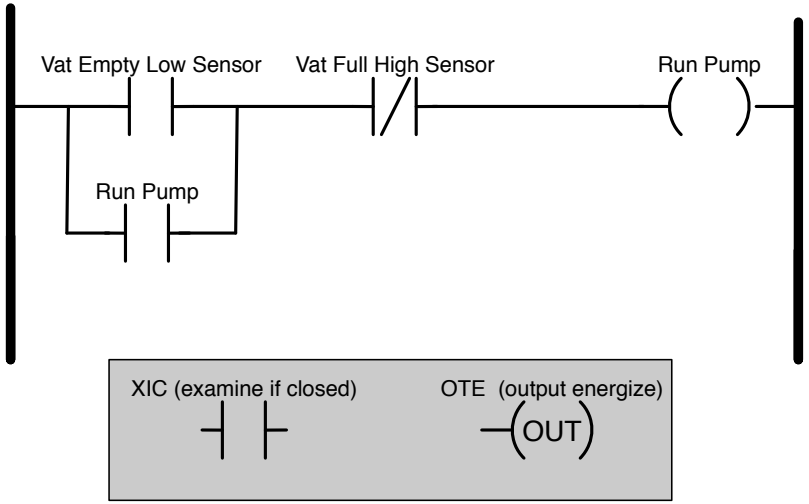


Ladder Logic to keep vat filled: latch-on/latch-off circuit



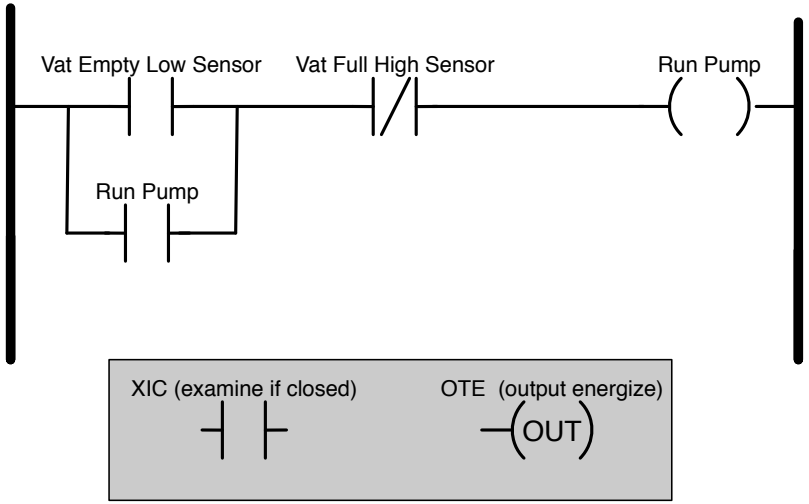
- When the tank is low on slurry a Low-Sensor signal will transition to high and the pump will turn on. When the tank is full a High-Sensor signal will transition to high and the pump will turn off.

Ladder Logic to keep vat filled: latch-on/latch-off circuit



- Suppose we have just the XIC and OTE (grey area) then: (a) The XIC instruction evaluates the data addressed by the associated Boolean tag. The evaluation is true if the data is one and false if the data zero. This is an input instruction and must be followed by an output instruction.
- (b) The OTE instruction writes data to the memory location addressed by the associated boolean tag. The data is set to one when the preceding rung logic is true and set to zero when false.

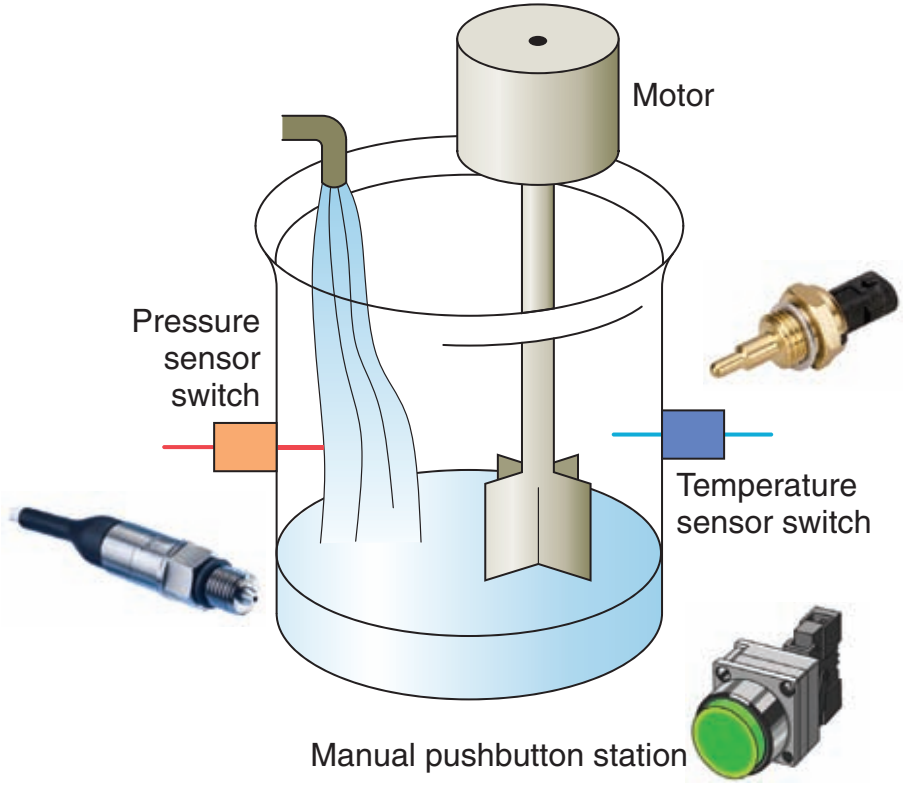
Ladder Logic to keep vat filled: latch-on/latch-off circuit



- OR logic: Need to keep the motor running when the slurry covers the sensor and the Low-Sensor signal transitions from high to low. Once the Pump-Motor is turned on the XIC instruction referencing the Pump-Motor output on the branch circuit will keep the motor running.
- AND logic: The XIO (examine if open) instruction referenced to the High-Sensor signal causes the rung output logic to become false, turning off the Pump-Motor.

Example PLC: A Mixer Process Controller (MPC)

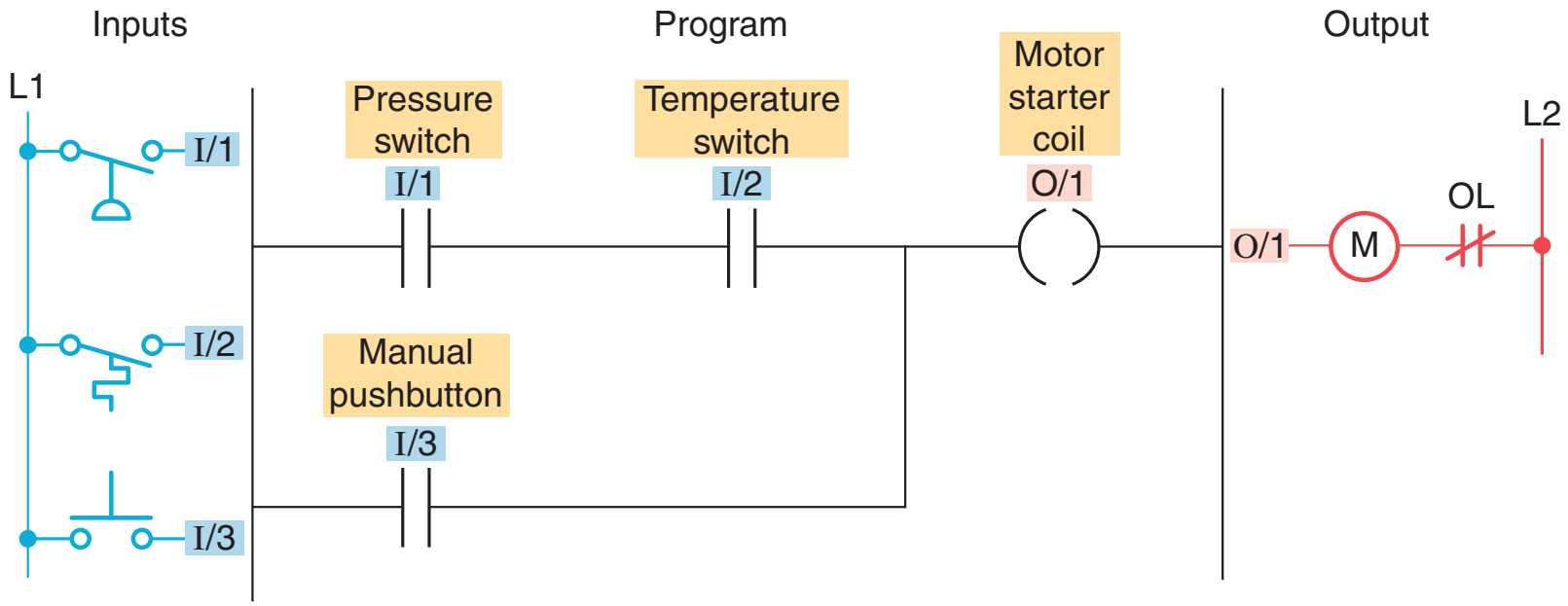
MPC: Problem



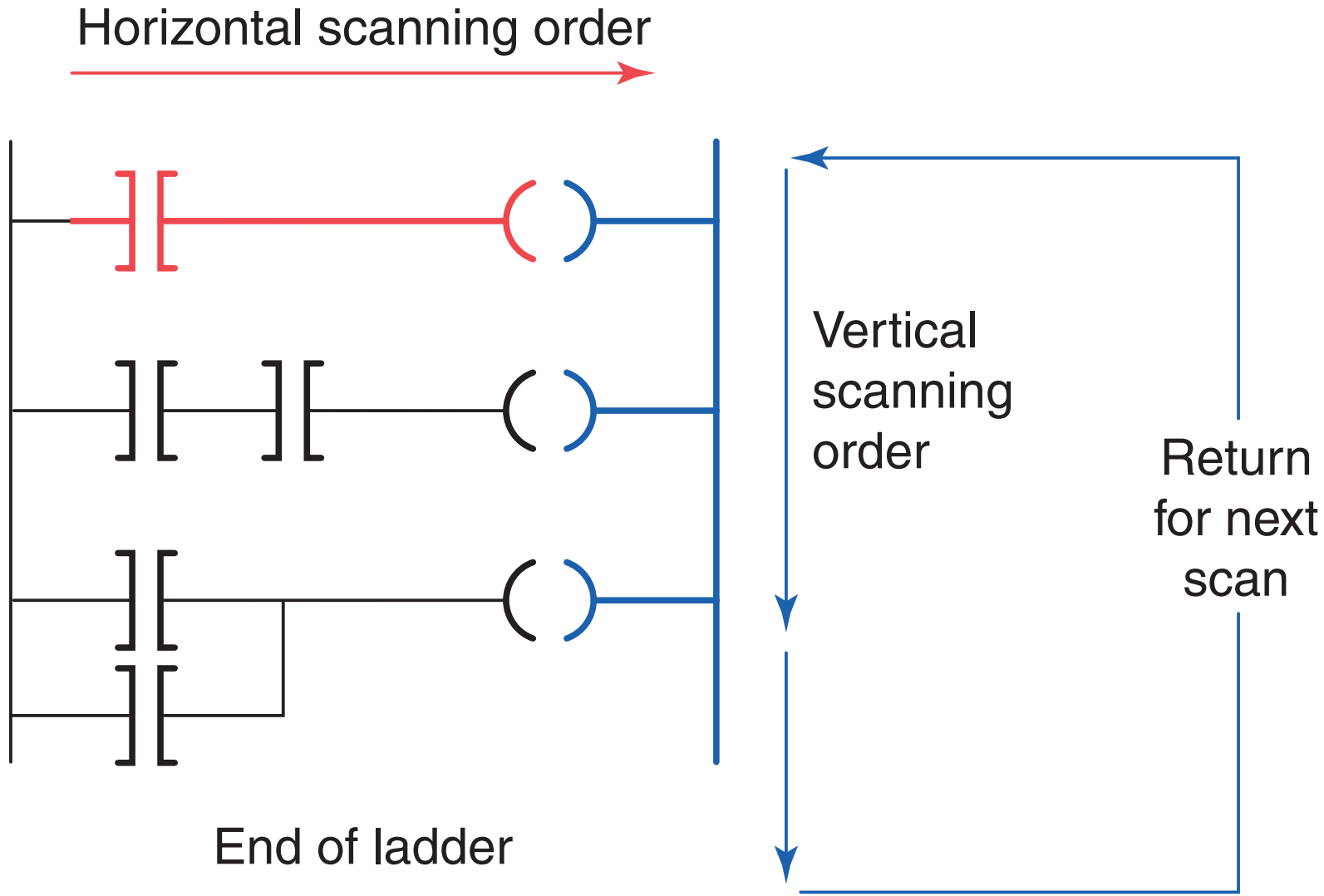
A mixer motor is used to automatically stir the liquid in a vat when the temperature and pressure reach preset values. The operator may start the motor directly by pushing the start button.

MPC: Input and Output Connections

MPC: Controller Program in Ladder Logic



- each ladder diagram (LD) resembles a relay circuit diagram, where
  - vertical lines ("ladders") L1 and L2 represent the power supply
  - horizontal liens ("rungs") represent the *electrical current*
- in a LD, rungs simply represent the *logical path*

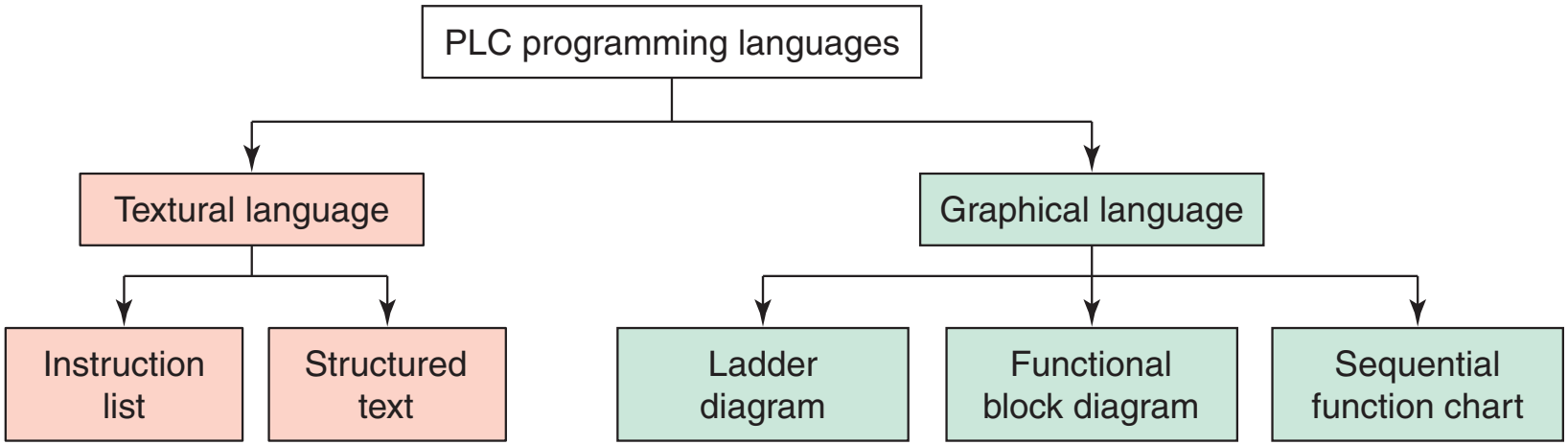

Horizontal scanning order
Vertical scanning order
Return for next scan
End of ladder

IEC 61131-3 Standard for PLCs

- PLCs – widely utilized in real-time and embedded control systems
- Part 3 of IEC 61131 standardizes programming notations for real-time and embedded systems running PLCs
- FBs – abstract units, components *reusable* for programming PLCs
  - basic functions [e.g., +, ≥ 1]
  - atomic, basic FBs [e.g., *hysteresis*]
  - composite FBs [e.g., *limits_alarm*]

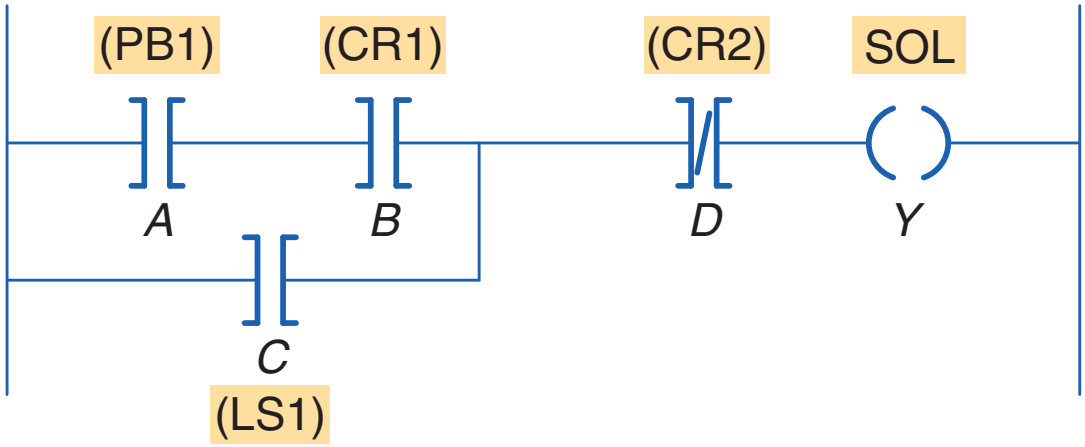
- IEC 61131-3 supplies up to four (HW) implementations for FBs:

  A PLC program written in one notation can be translated into its equivalent in another notation.



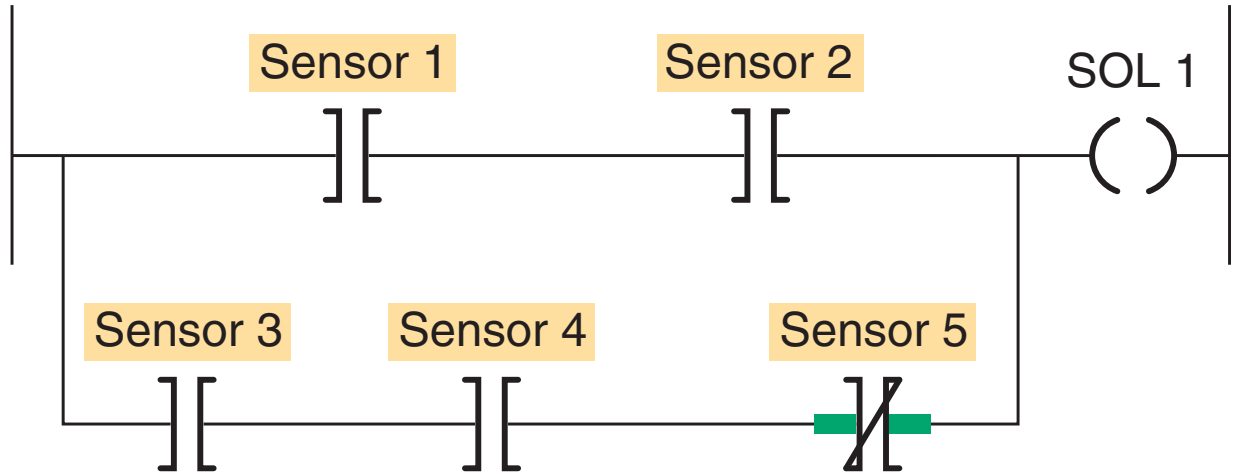
PLC (programmable logic controller); FB (function block)

Programming PLCs: Ladder Diagram vs. Instruction List



START	PB1
AND	CR1
OR	LS1
AND NOT	CR2
OUT	SOL

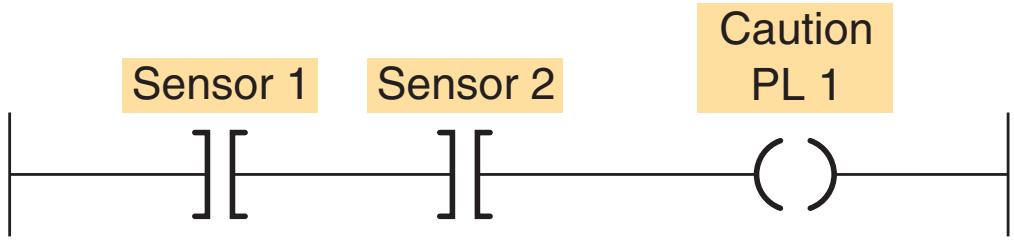
Programming PLCs: Ladder Diagram vs. Structured ST



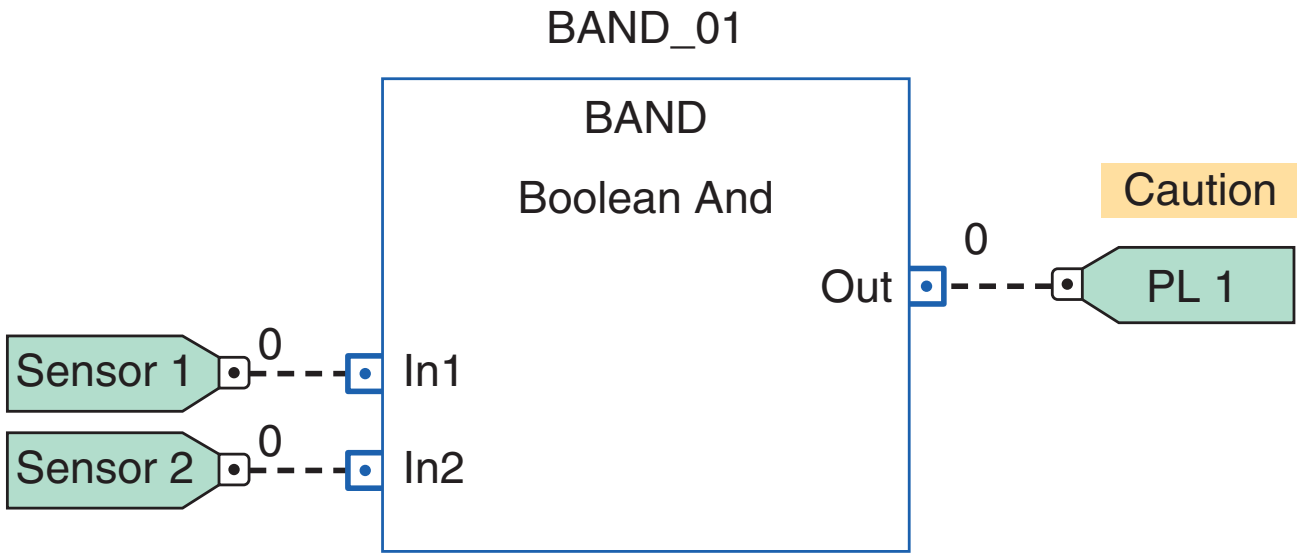
Ladder diagram (LD) program

```
IF Sensor_1 AND Sensor_2 THEN
        SOL_1 := 1;
ELSEIF Sensor_3 AND Sensor_4 AND NOT Sensor_5 THEN
        SOL_1 := 1;
END_IF;
```

Programming PLCs: LD vs. Function Block Diagram



Ladder diagram



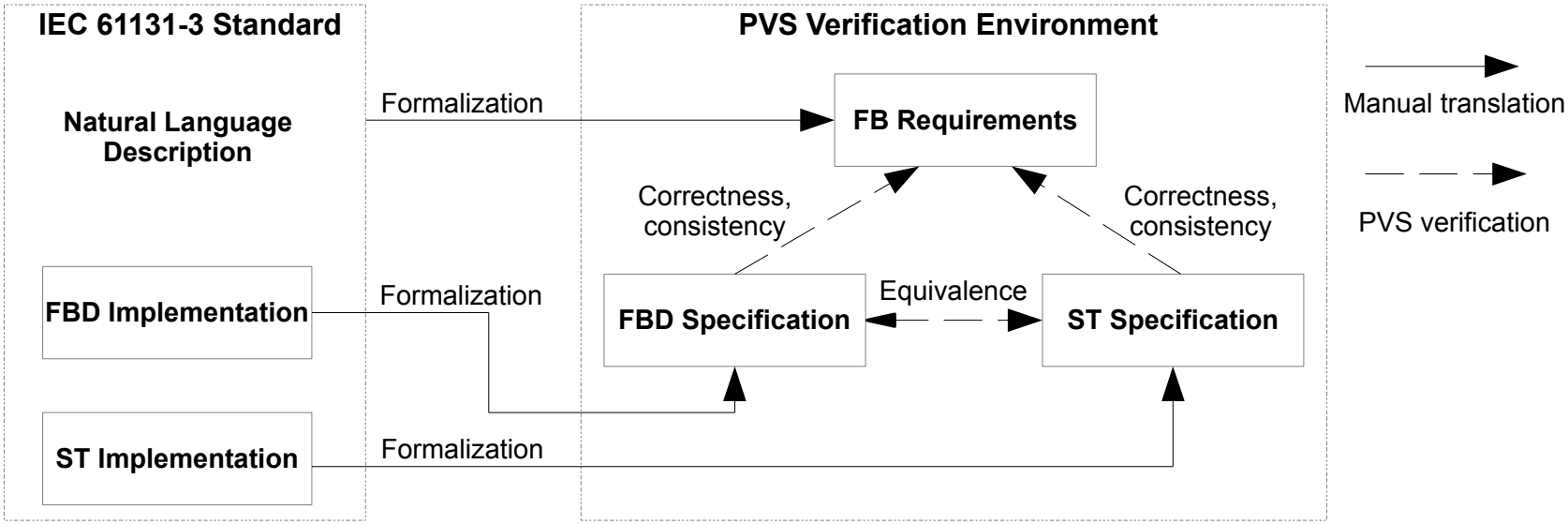
Function block diagram

Motivations, Problems, and Significance

revisit key questions

- **Interests from the Nuclear Industry**
- **Research Questions**
  1. Is system behaviour resulting from FB composition *predictable*?
  2. Does a vendor's FB implementation *conform* to its requirement?
- **A Formal Approach to Certifying FBs** [*tabular expressions* & *PVS*]
  - *shared requirements* between tool vendors and users (e.g., OPG)
  - *verified implementations* w.r.t. the shared requirements
- **Case Studies**
  1. IEC 61131-3 standard of FBs
  2. Subsystems of the SDSs implemented using FBs

Methodology



1. *Requirements* of FB *requirements* are described using function tables
2. *Implementations* of ST and FBD are formalized in PVS
3. *Correctness* is proved in PVS

Two Approaches for Formalizing Function Blocks

	functional approach	*relational* approach
basic blocks	not necessarily bool functions	always bool functions
composite blocks	function applications	logical conjunction
block inputs	as parameters	
block output	as return value	as parameter
behaviour	deterministic	allows non-determinism

Example: Requirements of a Sorting Block

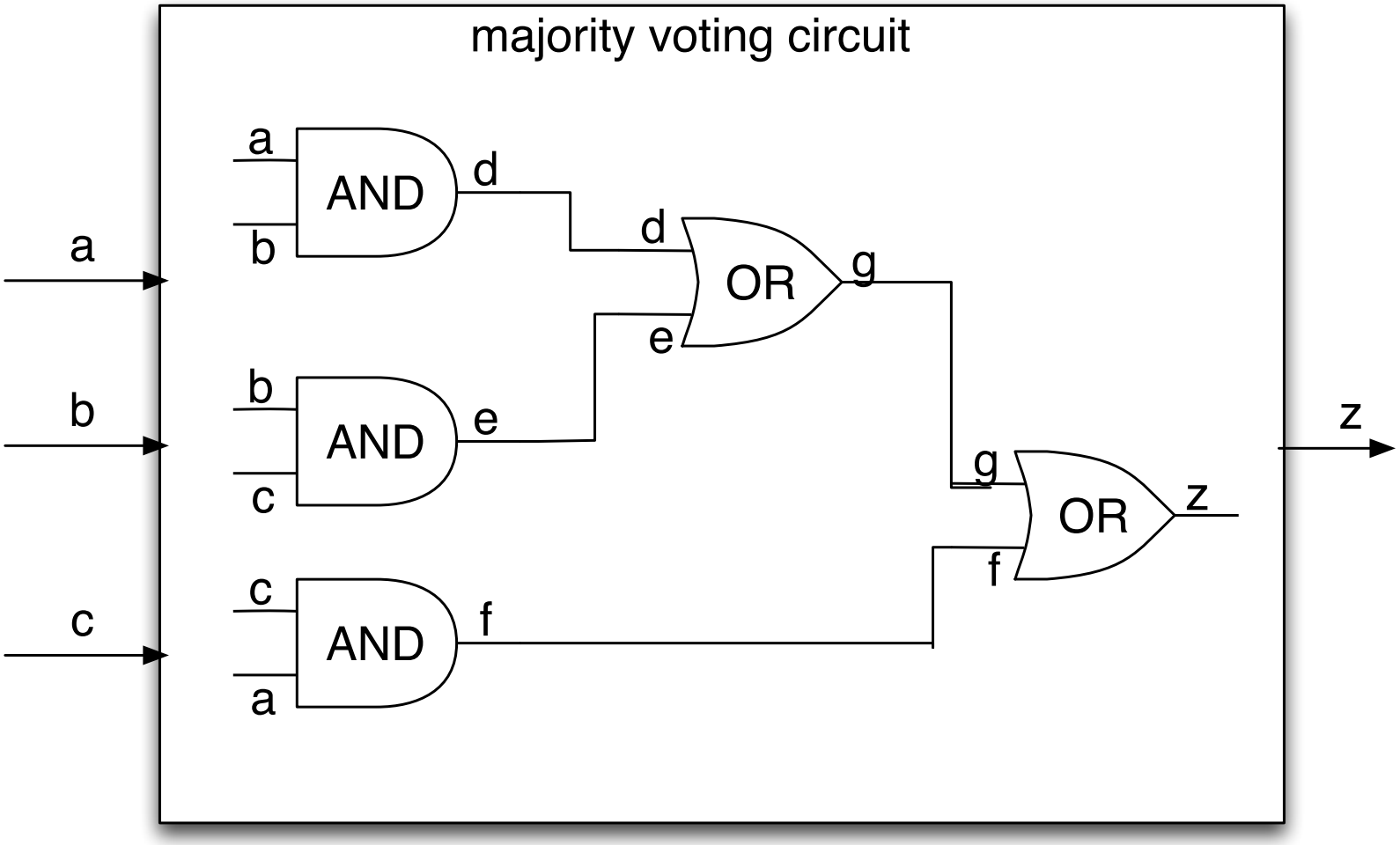
Relational: You may require only that "the list is sorted"

$$
\begin{aligned}
&sort(in\_list,\ out\_list) : bool = \\
&\quad out\_list \in permutations\ (in\_list) \\
&\quad \wedge \\
&\quad (\ \forall i \in 1..(n-1) : out\_list[i-1] \leq out\_list[i]\ )
\end{aligned}
$$

Functional: as opposed to "the list is sorted using a merge sort" function

$$
\begin{aligned}
&sort(in\_list) : LIST = \\
&\quad merge\_sort\ (in\_list) \\
&\quad --recursive function
\end{aligned}
$$

Example: Requirements of the Majority Voting Block



Solution using the functional approach:

$$maj\_vote\_fun\ (a,\ b,\ c : BIT) : BIT =$$
$$OR(\ OR(\ AND(a,\ b),\ AND(b,\ c)\ ),\ AND(c,\ a)\ )$$

Solution using the relational approach:

$$maj\_vote\_rel\ (a,\ b,\ c,\ z : BIT) : bool =$$
$$\exists d, e, f, g \bullet AND\ (a,\ b,\ d) \wedge AND\ (b,\ c,\ e) \wedge AND\ (c,\ a,\ f)$$
$$\wedge\ OR\ (d,\ e,\ g) \wedge OR\ (g,\ f,\ z)$$

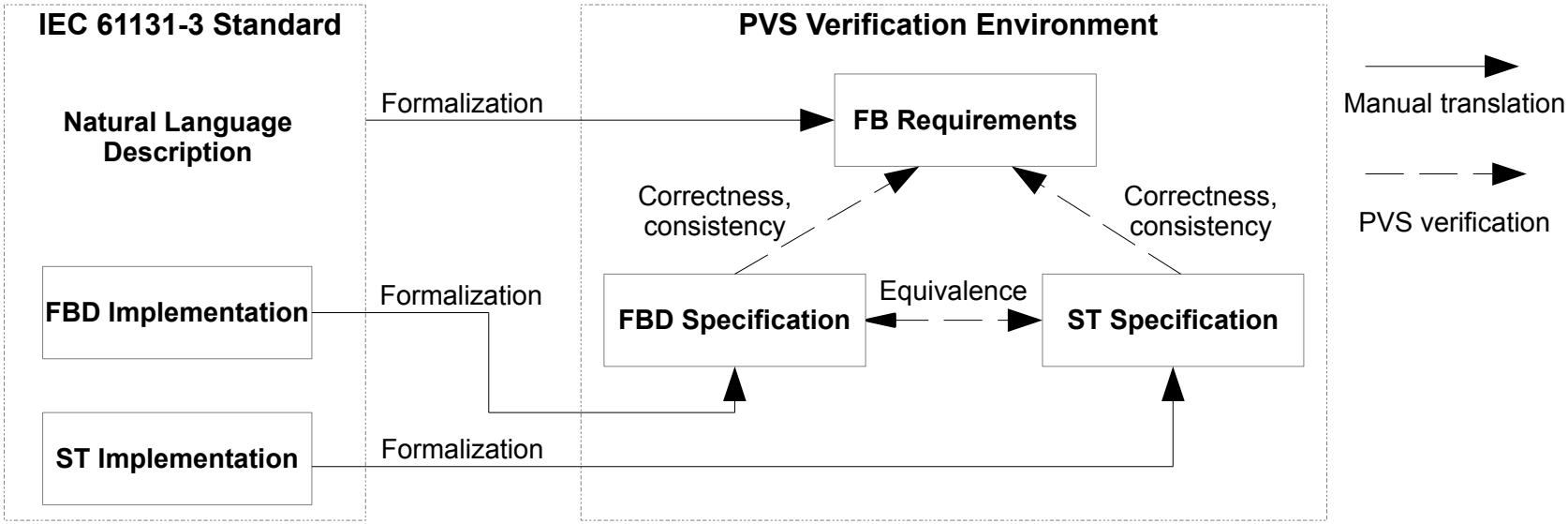
- In both approaches, parameters correspond to the interface variables (excluding intermediate values).
- In the relational approach, an *existential quantifier* is used
  - for each input-output pair, the internal implementation chooses a combination of intermediate values that produce the correct output

IEC 61131-3 Standard for PLCs

- PLCs – widely utilized in real-time and embedded control systems
- Part 3 of IEC 61131 standardizes programming notations for real-time and embedded systems running PLCs
- FBs – abstract units, components reusable for programming PLCs
  - basic functions [e.g., +, ≥ 1]
  - atomic, basic FBs [e.g., *hysteresis*]
  - composite FBs [e.g., *limits_alarm*]

PLC (programmable logic controller); FB (function block)

Methodology



1. Requirements of FB requirements are described using function tables
2. Implementations of ST and FBD are formalized in PVS
3. Correctness is proved in PVS

Principles of Specification

- a relational approach
- each function block is specified as a predicate (i.e., a function returning a Boolean value)
  - parameters represent the list of input and output variables
  - each variable is a timed sequence

    (i.e., a function with domain being *DTIME*)
  - the predicate returns *TRUE* if, across all $i \in DTIME$, the specified relationship among inputs and outputs hold; otherwise, *FALSE*

    For example:

```
x, y, z: VAR [DTIME -> real]
times (x, y, z): bool =
  FORALL (i: DTIME): z(i) = x(i) * y(i)
```

HYSTERESIS Block from IEC 61131-3

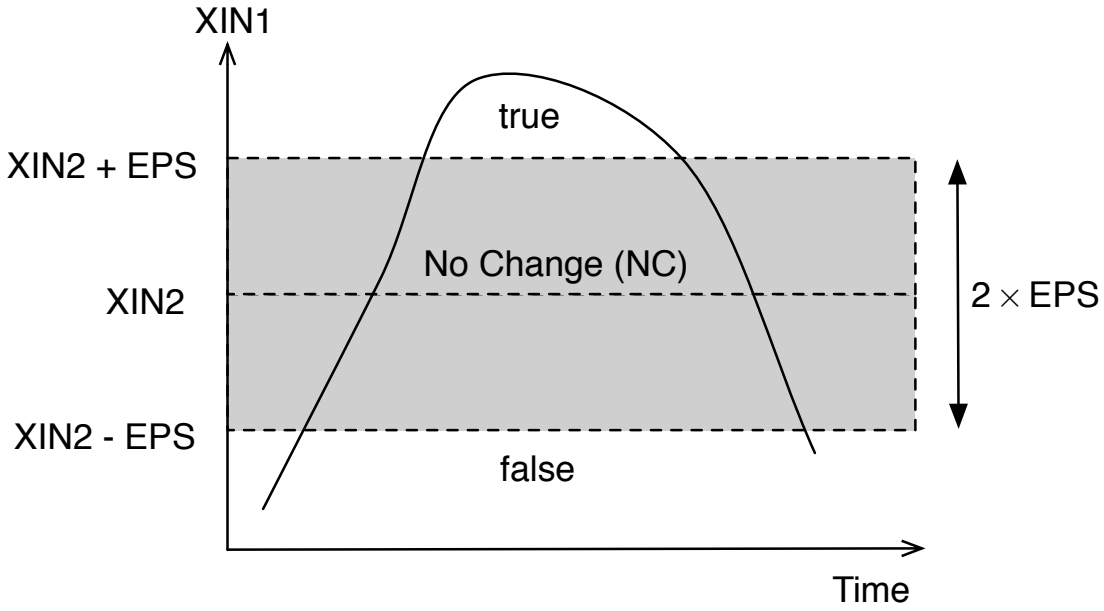
The following declaration and ST are extracted from IEC 61131-3:

```
       +------------+
       | HYSTERESIS |
REAL---|XIN1       Q|---BOOL
REAL---|XIN2        |
REAL---|EPS         |
       +------------+
INPUTS:
  XIN1 : variable value
  XIN2 : set point
  EPS  : hysteresis
OUTPUT:
  Q    : alarm output
```

```
FUNCTION_BLOCK HYSTERESIS
  VAR_INPUT XIN1, XIN2, EPS : REAL; END_VAR
  VAR_OUTPUT Q : BOOL := 0; END_VAR
  IF Q THEN IF XIN1 < (XIN2 - EPS) THEN Q := 0;
  END_IF;
  ELSIF XIN1 > (XIN2 + EPS) THEN Q := 1;
  END_IF ;
END_FUNCTION_BLOCK
```

IEC 61131-3 informal req. of hysteresis *limits_alarm* decl. & FBD

Informal Requirements of *HYSTERESIS*



decl. and ST of hysteresis

Tabular Requirements of *HYSTERESIS*

Draw the function table.

	Result
Condition	Q
$XIN1 < (XIN2 - EPS)$	false
$(XIN2 - EPS) \leq XIN1 \leq (XIN2 + EPS)$	N.C.
$XIN1 > (XIN2 + EPS)$	true

HYSTERESIS in PVS: Interface Variables

```
Hysteresis[delta: posreal] : THEORY
  BEGIN
    importing Time[delta]

    % monitored variables
    XIN1 : VAR [DTIME -> real] % signal value
    XIN2 : VAR [DTIME -> real] % set-point
    EPS  : VAR [DTIME -> real] % hysteresis band size

    % controlled variables
    BOOL : TYPE+ = subrange(0,1)   % BIT type
    Q    : VAR [DTIME -> BOOL]     % alarm
    ...
  END Hysteresis
```

HYSTERESIS in PVS: Tabular Requirements

Translate your function table into PVS. Is it complete and disjoint?

```
hysteresis_req (XIN1, XIN2, EPS, Q): bool =
  FORALL (i: DTIME):
    COND
    i = 0 -> Q(0) = 0,
    i > 0 ->
      COND
      %=============================================
          XIN1(i) >  (XIN2(i) + EPS(i)) -> Q(i) = 1,
      %=============================================
          XIN1(i) <= (XIN2(i) + EPS(i))
      AND XIN1(i) >= (XIN2(i) - EPS(i)) -> Q(i) = Q(i - 1),
      %=============================================
          XIN1(i) <  (XIN2(i) - EPS(i)) -> Q(i) = 0
      %=============================================
      ENDCOND
    ENDCOND
```

HYSTERESIS in PVS: Completeness TCC

```
M-x spt  M-x tccs
```

```
% Coverage TCC generated (at line 71, column 10) for
    % COND XIN1(i) > (XIN2(i) + EPS(i)) -> Q(i) = 1,
    %      XIN1(i) <= (XIN2(i) + EPS(i)) AND XIN1(i) >= (XIN2(i) - EPS(i))
    %        -> Q(i) = Q(i - 1),
    %      XIN1(i) < (XIN2(i) - EPS(i)) -> Q(i) = 0
    % ENDCOND
  % proved - complete
hysteresis_req_TCC2: OBLIGATION
  FORALL (XIN1: [DTIME[delta] -> real], XIN2: [DTIME[delta] -> real],
          EPS: [DTIME[delta] -> real], i: DTIME[delta]):
    i > 0 IMPLIES
     XIN1(i) > (XIN2(i) + EPS(i)) OR
      XIN1(i) <= (XIN2(i) + EPS(i)) AND XIN1(i) >= (XIN2(i) - EPS(i)) OR
       XIN1(i) < (XIN2(i) - EPS(i));
```

HYSTERESIS in PVS: Disjointness TCC

```
M-x spt  M-x tccs
```

```
% Disjointness TCC generated (at line 71, column 10) for
    % COND XIN1(i) > (XIN2(i) + EPS(i)) -> Q(i) = 1,
    %      XIN1(i) <= (XIN2(i) + EPS(i)) AND XIN1(i) >= (XIN2(i) - EPS(i))
    %        -> Q(i) = Q(i - 1),
    %      XIN1(i) < (XIN2(i) - EPS(i)) -> Q(i) = 0
    % ENDCOND
  % unfinished
hysteresis_req_TCC1: OBLIGATION
  FORALL (XIN1: [DTIME[delta] -> real], XIN2: [DTIME[delta] -> real],
          EPS: [DTIME[delta] -> real], i: DTIME[delta]):
    i > 0 IMPLIES
     NOT (XIN1(i) > (XIN2(i) + EPS(i)) AND
           XIN1(i) <= (XIN2(i) + EPS(i)) AND XIN1(i) >= (XIN2(i) - EPS(i)))
      AND
      NOT (XIN1(i) > (XIN2(i) + EPS(i)) AND XIN1(i) < (XIN2(i) - EPS(i))) AND
       NOT ((XIN1(i) <= (XIN2(i) + EPS(i)) AND XIN1(i) >= (XIN2(i) - EPS(i)))
             AND XIN1(i) < (XIN2(i) - EPS(i)));
```

HYSTERESIS in PVS: Proving Disjointness TCC

```
hysteresis_req_TCC1 :

  |-------
{1}   FORALL (XIN1: [DTIME[delta] -> real], XIN2: [DTIME[delta] -> real],
              EPS: [DTIME[delta] -> real], i: DTIME[delta]):
        i > 0 IMPLIES
         NOT (XIN1(i) > (XIN2(i) + EPS(i)) AND
               XIN1(i) <= (XIN2(i) + EPS(i)) AND
                XIN1(i) >= (XIN2(i) - EPS(i)))
          AND
          NOT (XIN1(i) > (XIN2(i) + EPS(i)) AND
                XIN1(i) < (XIN2(i) - EPS(i)))
           AND
           NOT ((XIN1(i) <= (XIN2(i) + EPS(i)) AND
                  XIN1(i) >= (XIN2(i) - EPS(i)))
                 AND XIN1(i) < (XIN2(i) - EPS(i)))

Rule? (grind)
Trying repeated skolemization, instantiation, and if-lifting,
this simplifies to:
hysteresis_req_TCC1 :

{-1}  i!1 >= 0
{-2}  i!1 > 0
{-3}  XIN1!1(i!1) > EPS!1(i!1) + XIN2!1(i!1)
{-4}  XIN1!1(i!1) < (XIN2!1(i!1) - EPS!1(i!1))
  |-------

Rule?
```

HYSTERESIS: cannot prove disjointness TCC

```
Proof summary for theory Hysteresis_tcc
    BOOL_TCC1.............................proved - complete
    hysteresis_st_TCC1....................proved - complete
    hysteresis_st_TCC2....................proved - complete
    hysteresis_st_TCC3....................proved - complete
    hysteresis_req_TCC1...................unfinished
    hysteresis_req_TCC2...................proved - complete
    correct_hysteresis_st.................unfinished   ...
```

```
% hysteresis_req_TCC1: OBLIGATION
{-1}  i > 0
{-2}  XIN1(i) > EPS(i) + XIN2(i)
{-3}  XIN1(i) < (XIN2(i) - EPS(i))
|-------
```

Suppose $\{X2 \rightarrow 1, EPS \rightarrow -1\}$. Then any value for X1 in the range $0 <$ X1 AND X1 < 2 will trigger the first and third conditions in the specification function table.

HYSTERESIS: cannot prove disjointness TCC

Condition	*Result* Q
XIN1 < (XIN2 − EPS)	false
(XIN2 − EPS) ≤ XIN1 ≤ (XIN2 + EPS)	N.C.
XIN1 > (XIN2 + EPS)	true

```
% hysteresis_req_TCC1: OBLIGATION
{-1}  i > 0
{-2}  XIN1(i) > EPS(i) + XIN2(i)
{-3}  XIN1(i) < (XIN2(i) - EPS(i))
|-------
```

Suppose $\{X2 \rightarrow 1, EPS \rightarrow -1\}$. Then any value for X1 in the range 0 < X1 AND X1 < 2 will trigger the first and third conditions in the specification function table.

Cannot prove disjointness TCC – What to do?

$\{X2 \rightarrow 1, EPS \rightarrow -1\}$. Then any value for X1 in the range
0 < X1 AND X1 < 2 will satisfy the first and third conditions in the specification function table.

Q is set to FALSE (0) and TRUE (1) simultaneously!

	Result
Condition	Q
XIN1 < (XIN2 − EPS)	false
(XIN2 − EPS) ≤ XIN1 ≤ (XIN2 + EPS)	N.C.
XIN1 > (XIN2 + EPS)	true

So what to do?

Open Your Mind – By Closing Your Laptop!

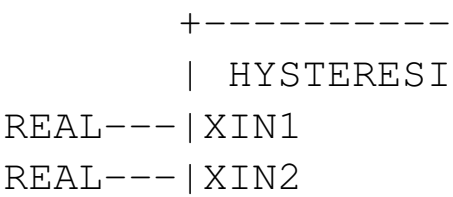
HYSTERESIS in PVS: Tabular Requirements Revisited

We impose an environment assumption (on the monitored variables).

```
% environment assumption
env (XIN1, XIN2, EPS): bool =
  FORALL (i: DTIME): EPS(i) > 0

hysteresis_req (XIN1, XIN2, EPS, Q): bool =
  env (XIN1, XIN2, EPS) % necessary for disjointness TCC
  IMPLIES
  FORALL (i: DTIME):
    COND
    i = 0 -> ...
    i > 0 ->
      COND
      ...
      ENDCOND
    ENDCOND
```

HYSTERESIS Block As a Function Table?




```
INPUTS:
  XIN1 : variable value
  XIN2 : set point
  EPS  : hysteresis
OUTPUT:
  Q    : alarm output
```

```
FUNCTION_BLOCK HYSTERESIS
  VAR_INPUT XIN1, XIN2, EPS : REAL; END_VAR
  VAR_OUTPUT Q : BOOL := 0; END_VAR
  IF Q THEN IF XIN1 < (XIN2 - EPS) THEN Q := 0;
  END_IF;
  ELSIF XIN1 > (XIN2 + EPS) THEN Q := 1;
  END_IF ;
END_FUNCTION_BLOCK
```

IEC 61131-3 informal req. of hysteresis *limits_alarm* decl. & FBD

HYSTERESIS in PVS: ST Implementation

Formalize the ST implementation of *HYSTERESIS* in PVS.

```
hysteresis_st (XIN1, XIN2, EPS, Q): bool =
  FORALL (i: DTIME):
    COND i = 0 -> Q(0) = 0,
         i > 0 ->
           IF Q(i - 1) = 1 THEN
             IF XIN1(i) < (XIN2(i) - EPS(i)) THEN
               Q(i) = 0
             ELSE % implicit from the ST
               Q(i) = Q(i - 1)
             ENDIF
           ELSIF XIN1(i) > (XIN2(i) + EPS(i)) THEN
             Q(i) = 1
           ELSE   % implicit from the ST
             Q(i) = Q(i - 1)
           ENDIF
    ENDCOND
```

HYSTERESIS in PVS: Proof of Implementation Correctness

Across all time instants, executing the ST implementation of *HYSTERESIS*, under the environment assumption (that $EPS > 0$), entails its black-box requirements.

State this precisely as a PVS theorem.

```
correct_hysteresis_st : THEOREM
  ( env(XIN1, XIN2, EPS)
    AND
    hysteresis_st(XIN1, XIN2, EPS, Q) )
  IMPLIES
  hysteresis_req(XIN1, XIN2, EPS, Q)
```

Try `(grind)`!

Do we need $EPS \geq 0$ or $EPS > 0$?

$EPS \geq 0$?

```
% environment assumption
env (XIN1, XIN2, EPS): bool =
  FORALL (i: DTIME): EPS(i) > 0

correct_hysteresis_st : CONJECTURE
     (env(XIN1, XIN2, EPS)
   AND
     hysteresis_st(XIN1, XIN2, EPS, Q) )
   IMPLIES
     hysteresis_req(XIN1, XIN2, EPS, Q)
```

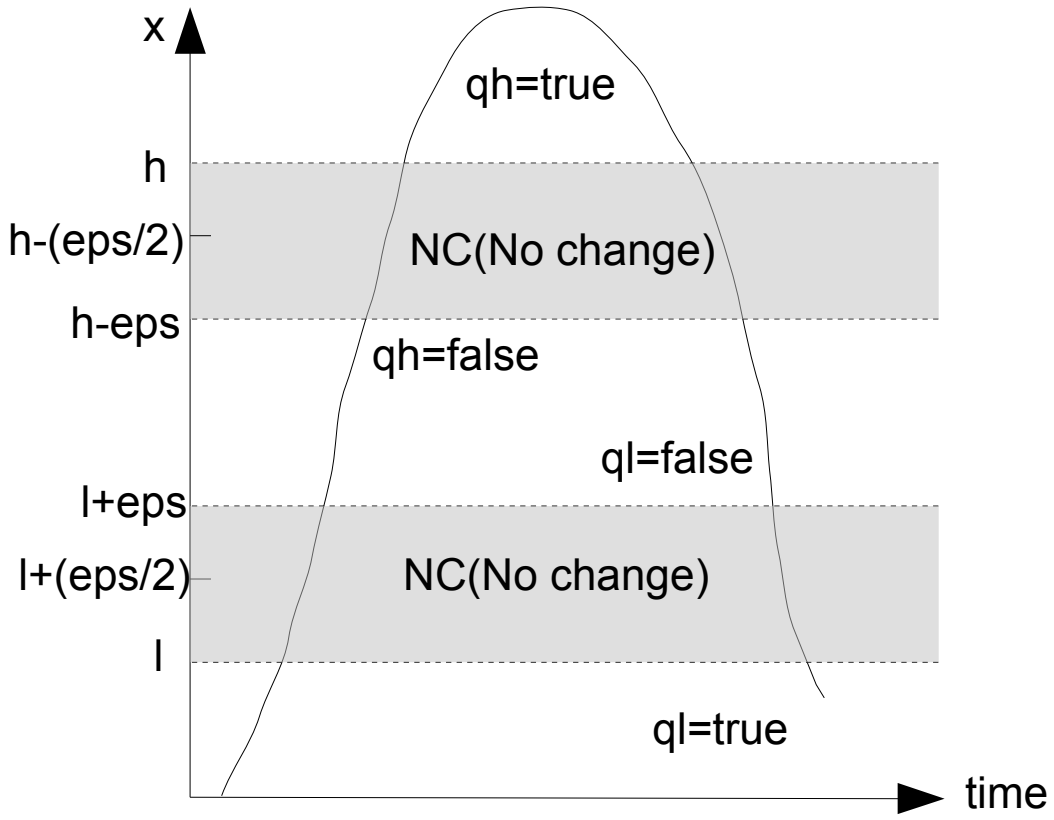
$EPS = 0$ is complete and disjoint but unrealistic (no deadband).

HYSTERESIS in PVS: Proof Summary

```
 Proof summary for theory Hysteresis
    BOOL_TCC1.............................proved - complete    [shostak](0.00 s)
    hysteresis_st_TCC1....................proved - complete    [shostak](0.01 s)
    hysteresis_st_TCC2....................proved - complete    [shostak](0.01 s)
    hysteresis_st_TCC3....................proved - complete    [shostak](0.00 s)
    hysteresis_req_TCC1...................proved - complete    [shostak](0.06 s)
    hysteresis_req_TCC2...................proved - complete    [shostak](0.04 s)
    hysteresis_req_TCC3...................proved - complete    [shostak](0.02 s)
    hysteresis_req_TCC4...................proved - complete    [shostak](0.02 s)
    correct_hysteresis_st.................proved - complete    [shostak](0.17 s)
    Theory totals: 9 formulas, 9 attempted, 9 succeeded (0.34 s)

Grand Totals: 9 proofs, 9 attempted, 9 succeeded (0.34 s)
```

Informal Requirements of *limits_alarm*



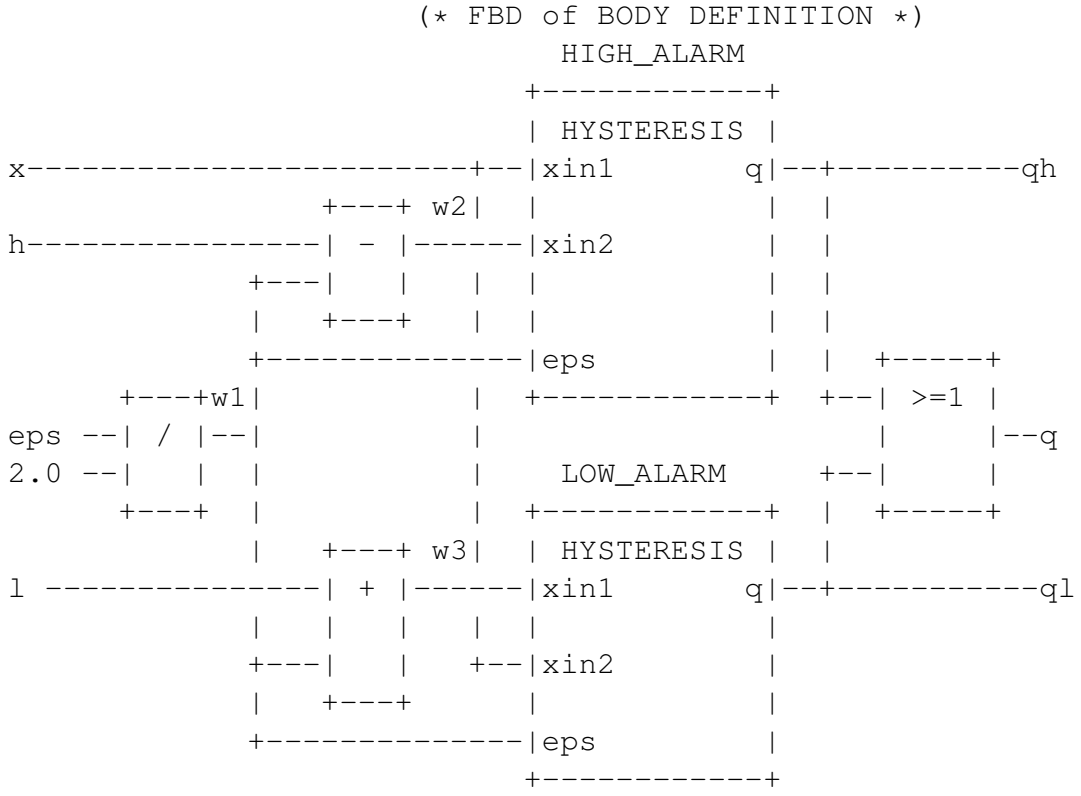
decl. & FBD of *limits_alarm*

LIMITS_ALARM Block from IEC 61131-3

```
   (* DECLARATION *)
       +--------+
       |LIMITS_ |
       |  ALARM |
REAL --|h     qh|-- BOOL
REAL --|x      q|-- BOOL
REAL --|l     ql|-- BOOL
REAL --|eps     |
       +--------+

INPUTS:
h   : high limit
x   : variable value
l   : low limit
eps: hysteresis

OUTPUTS:
qh : high flag
q   : alarm output
ql : low flag
```

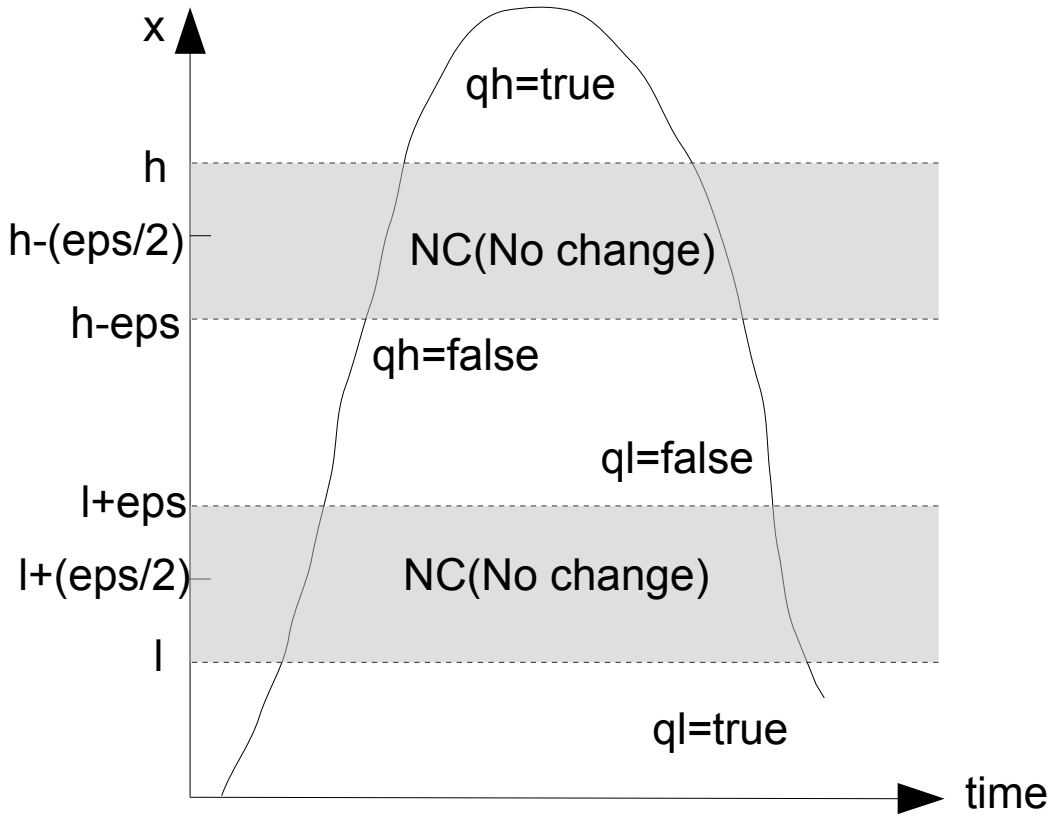


$$qh = HYSTERESIS(x,\ h - (eps / 2.0)\ ,\ (eps / 2.0))$$

$$ql = HYSTERESIS(\ l + (eps / 2.0)\ ,\ x,\ (eps / 2.0))$$

IEC 61131-3 informal requirement of *limits_alarm* hysteresis decl. & ST

So what is the Requirements Function Table?



decl. & FBD of *limits_alarm*

Tabular Requirements of *LIMITS_ALARM*

Draw the function table. Make a separate table for each controlled variable.

Condition	*Result* Q
QH ∨ QL	True
¬(QH ∨ QL)	False

Condition	*Result* QH
X > H	True
H - EPS ≤ X ≤ H	NC
X < H − EPS	False

assume: $EPS > 0$

Condition	*Result* QL
X < L	True
L ≤ X ≤ L + EPS	NC
X > L + EPS	False

assume: $EPS > 0$

LIMITS_ALARM in PVS: Interface Variables

```
Limits_Alarm[delta: posreal] : THEORY
  BEGIN
    importing Time[delta]
    importing Hysteresis[delta]

    % Monitored Variables
    H   : VAR [DTIME -> real]   % high limit
    X   : VAR [DTIME -> real]   % signal value
    L   : VAR [DTIME -> real]   % low limit
    EPS : VAR [DTIME -> real]   % hysteresis band size
    % controlled variables
    QH  : VAR [DTIME -> BOOL]   % high alarm
    Q   : VAR [DTIME -> BOOL]   % any alarm
    QL  : VAR [DTIME -> BOOL]   % low alarm
    ...
  END Limits_Alarm
```

LIMITS_ALARM in PVS: Basic FBs for FBD Implementation

```
r1, r2, r3 : VAR [DTIME -> real]
nzr        : VAR [DTIME -> {r: real | r /= 0}]
b1, b2, b3 : VAR [DTIME -> BOOL]

% basic functions for the FBD implementation
minus   (r1, r2, r3) : bool =
  FORALL (i: DTIME): r3(i) = r1(i) - r2(i)
plus    (r1, r2, r3) : bool =
  FORALL (i: DTIME): r3(i) = r1(i) + r2(i)
divide  (r1, nzr, r3): bool =
  FORALL (i: DTIME): r3(i) = r1(i) / nzr(i)
gte_one (b1, b2, b3) : bool =
  FORALL (i: DTIME):
    b3(i) = IF b1(i) + b2(i) >= 1 THEN 1 ELSE 0 ENDIF
```

LIMITS_ALARM in PVS: FBD Implementation

Formalize the FBD implementation of *LIMITS_ALARM* in PVS.

Hint 1: Hide intermediate values using ∃.

Hint 2: Use either *hysteresis_req* or *hysteresis_req*.

```
% FBD implementation of the LIMITS_ALARM block
limits_alarm_fbd (H, X, L, EPS, QH, Q, QL): bool =
 EXISTS (w1, w2, w3 : [DTIME -> real]):
     divide          (EPS, (LAMBDA (i: DTIME): 2.0), w1)
 AND minus           (H   , w1                     , w2)
 AND plus            (L   , w1                     , w3)
 AND hysteresis_req  (X   , w2                     , w1, QH)
 AND hysteresis_req  (w3  , X                      , w1, QL)
 AND gte_one         (QH  , QL                     , Q)
```

Why not use the predicate *hysteresis_st*?

Hint. By monotonicity of ∧ and weakening/strengthening.

LIMITS_ALARM in PVS: Reusing Req. for Component FBs

Known: $hysteresis\_st \Rightarrow hysteresis\_req$ [correct ST implementation]

Prove[a]**:** $(hysteresis\_req \wedge P \Rightarrow Q) \Rightarrow (hysteresis\_st \wedge P \Rightarrow Q)$

Proof:

$$
\begin{aligned}
& (hys\_req \wedge P) \Rightarrow Q \\
\Rightarrow & \{ \textit{monotonicity of } \wedge \} \\
& (hys\_req \wedge P) \wedge \underline{hys\_st} \Rightarrow Q \wedge \underline{hys\_st} \\
\equiv & \{ hys\_st \Rightarrow hys\_req,\ (p \Rightarrow q) \equiv (p \wedge q \equiv p) \} \\
& (\boxed{hys\_st} \wedge P) \Rightarrow Q \wedge hys\_st \\
\Rightarrow & \{ p \wedge q \Rightarrow q,\ \textit{transitivity of } \Rightarrow \} \\
& (hys\_st \wedge P) \Rightarrow Q
\end{aligned}
$$

[a] P denotes other component FBs; Q denotes the desired property.

LIMITS_ALARM in PVS: Tabular Requirements

Translate your function tables into PVS. Is it complete and disjoint?

```
env_1 (H, X, L, EPS): bool = FORALL (i: DTIME): EPS(i) > 0

high_alarm_req (H, X, L, EPS, QH): bool =
  env_1 (H, X, L, EPS) IMPLIES ... % see hysteresis_req

low_alarm_req (H, X, L, EPS, QL): bool =
  env_1 (H, X, L, EPS) IMPLIES ... % see hysteresis_req

limits_alarm_req (H, X, L, EPS, QH, Q, QL): bool =
      high_alarm_req (H, X, L, EPS, QH)
  AND low_alarm_req  (H, X, L, EPS, QL)
  AND gte_one        (QH, QL, Q)
```

LIMITS_ALARM in PVS: Implementation Correctness

Across all time instants, executing the FBD implementation of *LIMITS_ALARM*, under the environment assumption (that $EPS > 0$), entails its black-box requirements.

```
correct_limits_alarm_fbd : THEOREM
      (env_1              (H, X, L, EPS)
       AND
       limits_alarm_fbd (H, X, L, EPS, QH, Q, QL))
      IMPLIES
      limits_alarm_req (H, X, L, EPS, QH, Q, QL)
```

Unfortunately, `(grind)` won't work! Show that you're smarter than it

There is also an additional environmental assumption that will be needed.

LIMITS_ALARM in PVS: Proof of Implementation Correctness

See the separate in-class handout.

LIMITS_ALARM in PVS: Validation of Requirements

The high alarm and low alarm are never tripped simultaneously.

State this precisely as a PVS theorem.

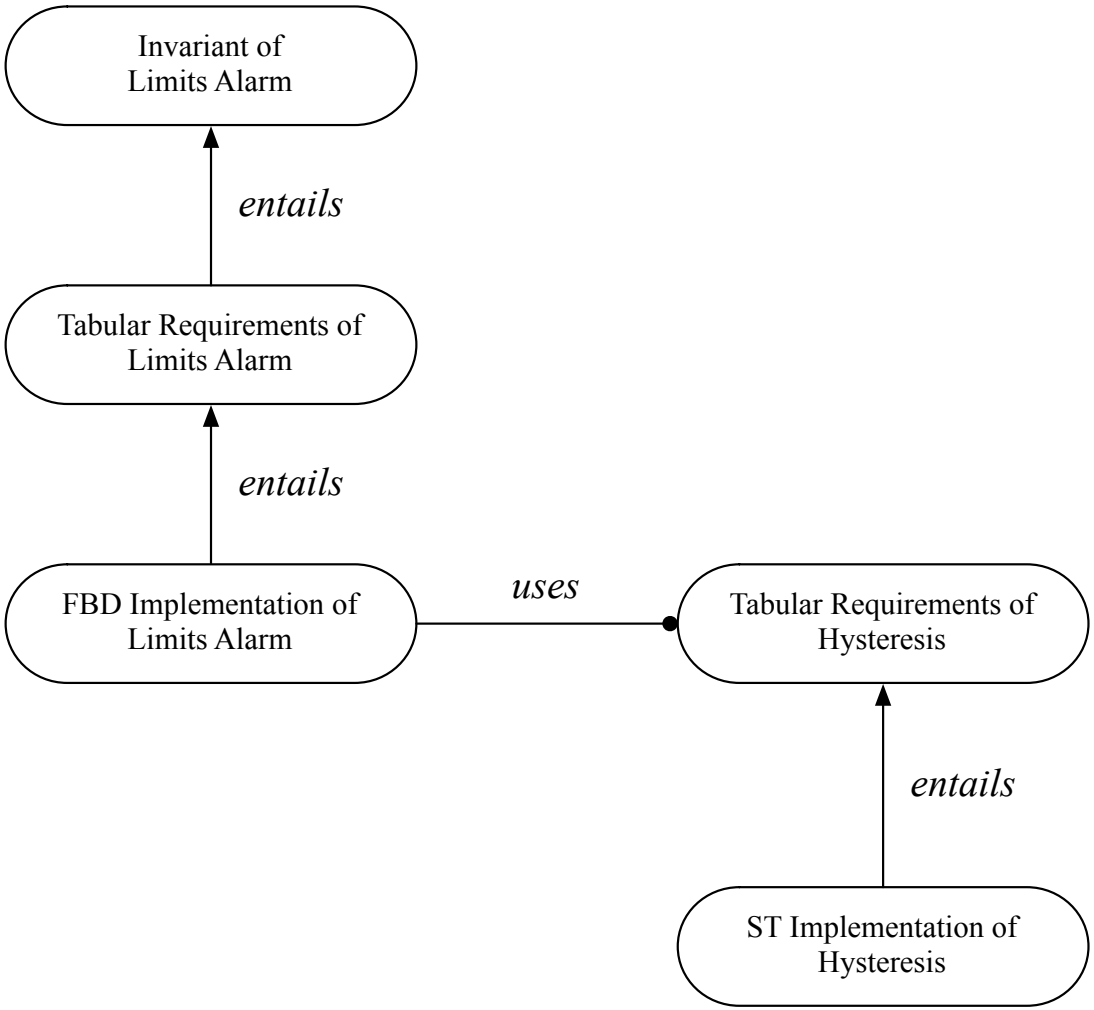
```
% 2nd environment assumption. What does it mean?
env_2 (H, X, L, EPS): bool =
  ???

no_simultaneous_alarms (QH, Q, QL): bool =
  FORALL (i: DTIME): NOT( QH(i) = 1 AND QL(i) = 1 )

req_entails_inv : THEOREM
  ( env_1 (H, X, L, EPS) AND env_2 (H, X, L, EPS)
   AND limits_alarm_req (H, X, L, EPS, QH, Q, QL) )
  IMPLIES no_simultaneous_alarms (QH, Q, QL)
```

Why not use *limits_alarm_fbd*? **Hint.** By the transitivity of $\Rightarrow$.

HYSTERESIS & LIMITS_ALARM: Verification Roadmap



Lab

In order to save your opportunity of learning, we will NOT make the PVS sources available!

1. Create two theories `Hysteresis` (see earlier Lab) and `Limits_Alarm`
2. Reproduce the various functions and proofs in PVS.
3. In particular, when validating `Limits_Alarm`'s requirements, convince yourself as to why a second environmental assumption `env_2` is necessary.

Index