Formal Verification of IEC 61131-3 Function Blocks

Linna Pang, Chen-Wei Wang, Mark Lawford, and Alan Wassyng

McMaster Centre for Software Certification, McMaster University

{ pangl, wangcw, lawford, wassyng }@mcmaster.ca

Abstract

- Many industrial control systems use programmable logic controllers (PLCs) since they provide a highly reliable, off-the-shelf hardware platform.
- On the programming side, function blocks (FBs) are reusable components provided by the PLC supplier that can be combined to implement the required system behaviour.
- A higher quality system may be realized if the FBs are pre-certified to be compliant with an international standard such as IEC 61131-3.
- We present an approach:
  - to creating complete and unambiguous FB requirements using tabular expressions
  - to verifying the consistency and correctness of FB implementations in the PVS proof environment
- We apply our approach to the IEC 61131-3 standard, by examining the entire library of FBs and their supplied implementations described in structured text (ST) and function block diagrams (FBDs).
- Our approach identified issues in the standard, including: a) ambiguous behavioural descriptions; b) missing assumptions; c) mismatched types of related variables; and d) erroneous implementations.
- We also provide resolutions to the identified issues.

Background

- *IEC 61131-3* [1] is a standard with over 20 years of use on critical systems running on programmable logic controllers (PLCs).
- *Function blocks (FBs)* are basic design units that implement the behaviour of a PLC, where each FB is a reusable component for building new, more sophisticated components or systems.
- A function block typically has a natural language description of the block behaviour, accompanied by a detailed implementation in the *structured text (ST)* or *function block diagrams (FBD)* description, or in some cases both.
- *Tabular expressions* (a. k. a. function tables or tables) are a promising way for documenting system requirement that has proven to be both practical and effective in industry [5].
- *Prototype Verification System (PVS)* [2] is a general-purpose theorem prover that provides an integrated environment with mechanized support for:
  - writing *specifications* using tabular expressions and (higher-order) predicates
  - conducting (interactively) *proofs* that implementations satisfy the tabular requirements using sequent-style deductions

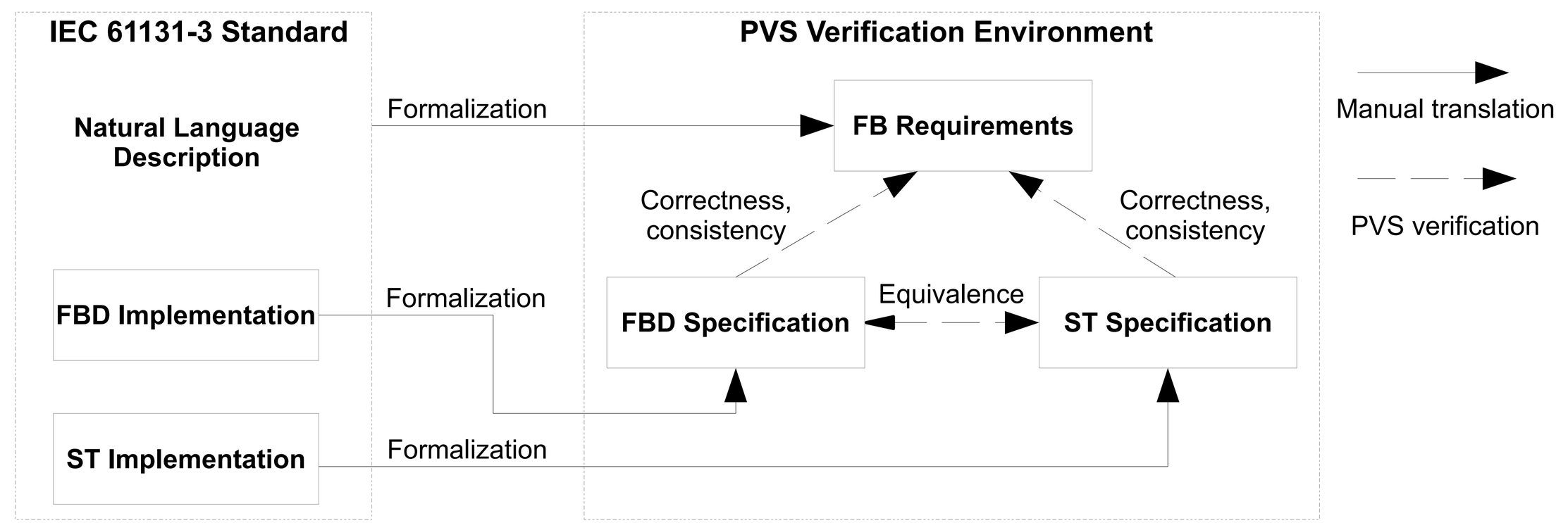
Motivation

- IEC 61131-3 uses FB descriptions that are too close to the level of hardware implementations, making it difficult to argue about the behavioural correctness of FBs.
- The search for higher quality may be realized if the FBs are pre-certified with respect to IEC 61131-3.
- Two acceptance criteria of mission- or safety-critical systems:
  - The system requirements are precise and complete.
  - The system implementation exhibits behaviour that conforms to these requirements.
- Formal descriptions, e.g., tabular expressions, prevent FB tool vendors and users from interpreting the expected behaviours differently.
- Formal descriptions are amenable to mechanized support, e.g., PVS, for verifying the conformance of candidate implementations to the high-level, input-output requirements.

Contributions

1. A practical methodology for formally verifying function blocks compliant with IEC 61131-3.
2. Identification of issues in IEC 61131-3 example function blocks, consisting of:
   (a) ambiguous behavioural descriptions (e.g., *PULSE* timer)
   (b) missing assumptions (e.g., *HYSTERESIS* and *LIMITS_ALARM*)
   (c) mismatched types of related variables (e.g., *PID* and *AVERAGE*)
   (d) erroneous implementations (e.g., *STACK_INT*)
3. Suggested resolutions of all identified issues.

Methodology



1. Create a tabular expression requirements specification in PVS for each FB.
2. Formalize both ST and FBD implementations in PVS as higher-order predicates modelled as timed trajectories.
3. Prove functional equivalence using PVS of the ST and FBD implementations when both are supplied in the standard.
4. Use PVS to prove *consistency* and *correctness* of each implementation with respect to its requirements specification.

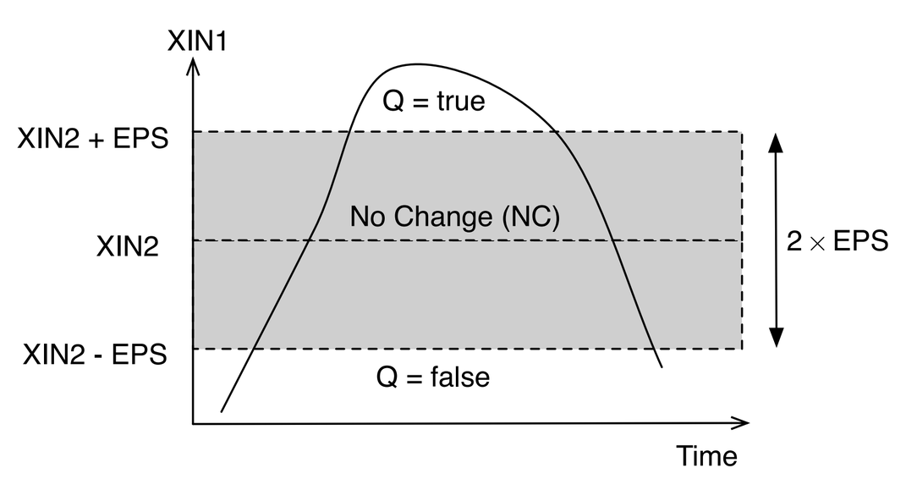
Example of Basic FBs: Hysteresis

Based on the input-output declaration and ST implementation, as suppled by IEC 61131-3 [1]:

```
      (* DECLARATION *)
       +------------+
       | HYSTERESIS |
       |            |
REAL --|XIN1       Q|-- BOOL
REAL --|XIN2        |
REAL --|EPS         |
       |            |
       +------------+
```

```
FUNCTION_BLOCK HYSTERESIS
  (* Boolean hysteresis on difference *)
  (* of REAL inputs, XIN1 - XIN2 *)
  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
```

We derive the tabular requirement for *HYSTERESIS*:



Condition	*Result* Q
$XIN1 > (XIN2 + EPS)$	1
$(XIN2 - EPS) \leq XIN1 \leq (XIN2 + EPS)$	NC
$XIN1 < (XIN2 - EPS)$	0

assume: $EPS \geq 0$

To prove that the ST implementation of *HYSTERESIS*, supplied by IEC 61131-3, satisfies its tabular requirements, we formalize it in PVS:

```
% Inputs
XIN1, XIN2 : VAR [tick -> real]
EPS        : VAR [tick -> posreal] % Assumption: positive deadband size

% Output
Q          : VAR [tick -> bool]

HYSTERESIS_st_impl (XIN1, XIN2, EPS, Q): bool =
  FORALL t:
    Q(t) =
      IF    init(t)                                   THEN False
      ELSIF Q(pre(t)) & XIN1(t) < (XIN2(t) - EPS(t)) THEN False
      ELSIF             XIN1(t) > (XIN2(t) + EPS(t)) THEN True
      ELSE                                                Q(pre(t))
      ENDIF
```

We also translate the tabular requirement of *HYSTERESIS* into PVS:

```
HYSTERESIS_tab_req (XIN1, XIN2, EPS, Q): bool =
  FORALL t:
    Q(t) =
      IF init(t) THEN False
      ELSE LET prev = Q(pre(t)) IN
       TABLE
        | XIN1(t) < (XIN2(t) - EPS(t))                                 | False ||
        | (XIN2(t) - EPS(t)) <= XIN1(t) & XIN1(t) <= (XIN2(t) + EPS(t)) | prev  ||
        | (XIN2(t) + EPS(t)) < XIN1(t)                                 | True  ||
       ENDTABLE
      ENDIF
```

Finally, we prove that the ST implementation is:

- *correct* (i.e.., satisfies its requirement)
- *consistent* (i.e., each input vector has a corresponding output)

```
Hysteresis_correctness: THEOREM
  FORALL XIN1, XIN2, EPS, Q:
    HYSTERESIS_st_impl(XIN1, XIN2, EPS, Q) IMPLIES
      HYSTERESIS_tab_req(XIN1, XIN2, EPS, Q)

Hysteresis_consistency: THEOREM
  FORALL XIN1, XIN2, EPS:
    EXISTS Q:
      HYSTERESIS_st_impl(XIN1, XIN2, EPS, Q)
```

Remark. Upon certifying the *HYSTERESIS* block, we may reuse its requirements predicate *HYSTERESIS_tab_req* to certify others (e.g., the *LIMITS_ALARM* block) that use it as a component.

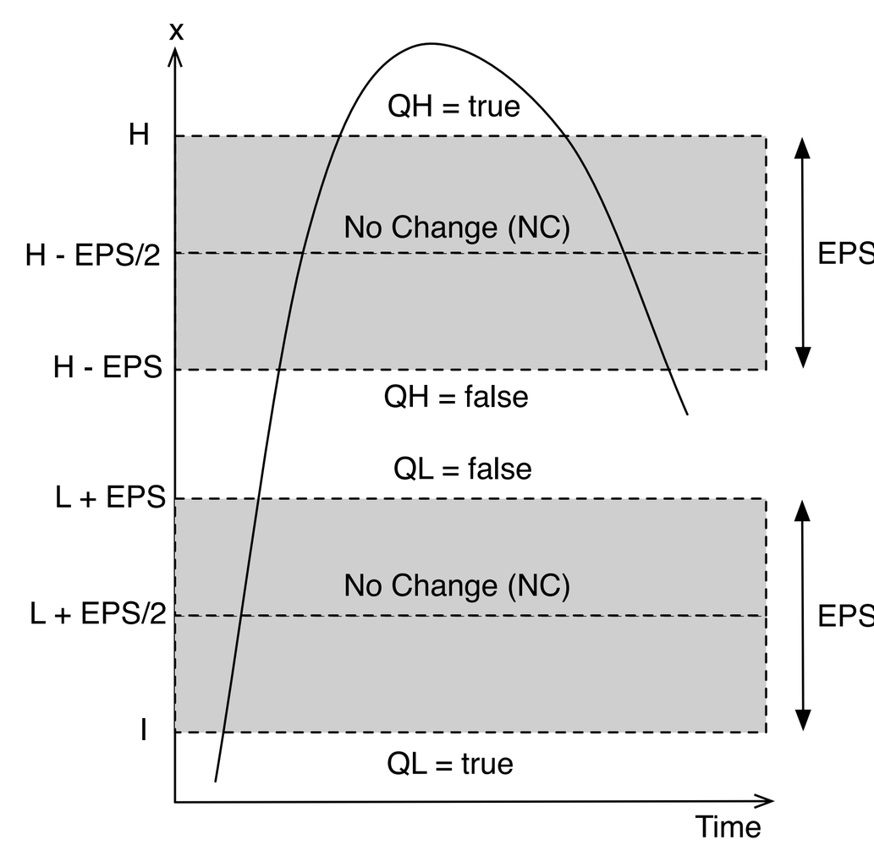
Example of Composite FBs: Limits Alarm

Based on the input-output declaration and FBD implementation, as suppled by IEC 61131-3 [1]:

```
  (* DECLARATION *)
     +---------+
     | LIMITS_ |
     | ALARM   |
REAL--|H      QH|--BOOL
REAL--|X       Q|--BOOL
REAL--|L      QL|--BOOL
REAL--|EPS      |
     +---------+
FUNCTION_BLOCK LIMITS_ALARM
  VAR_INPUT
    H   : REAL; (* High limit     *)
    X   : REAL; (* Variable value *)
    L   : REAL; (* Lower limit    *)
    EPS : REAL; (* Hysteresis     *)
  END_VAR
  VAR_OUTPUT
    QH : BOOL; (* High flag    *)
    Q  : BOOL; (* Alarm output *)
    QL : BOOL; (* Low flag     *)
  END_VAR
END_FUNCTION_BLOCK
```

```
            (* Function block body in FBD language *)
                                   HIGH_ALARM
                                 +------------+
                                 | HYSTERESIS |
X------------------------+--|XIN1        Q|--+----------QH
                  +---+ w2|  |            |  |
H---------------| - |------|XIN2          |  |
              +---|   |   |  |            |  |
              |   +---+   |  |            |  |
              +--------------|EPS         |  |  +-----+
        +---+w1|           |  +------------+  +--| >=1 |
EPS --| / |--|           |                    |     |--Q
2.0 --|   |  |           |   LOW_ALARM      +--|     |
        +---+  |           |  +------------+  |  +-----+
              |   +---+ w3|  | HYSTERESIS |  |
L ---------------| + |------|XIN1        Q|--+-----------QL
              |   |   |   |  |            |
              +---|   |   +--|XIN2        |
              |   +---+      |            |
              +--------------|EPS         |
                             +------------+
```

We derive the tabular requirement for *LIMITS_ALARM*:



Condition	*Result* Q
$QH \vee QL$	1
$\neg(QH \vee QL)$	0

assume: $L + EPS < H - EPS$

Condition	*Result* QH
$X > H$	1
$H - EPS \leq X \leq H$	NC
$X < H - EPS$	0

assume: $EPS > 0$

Condition	*Result* QL
$X < L$	1
$L \leq X \leq L + EPS$	NC
$X > L + EPS$	0

assume: $EPS > 0$

We formalize the FBD implementation of *LIMITS_ALARM* and its tabular requirements in PVS:

```
% Assumption: low and high alarm hysteresis regions do not overlap
HIGH_LIMIT: TYPE =
  [ l: [tick -> real], eps: [tick -> posreal] ->
      { h: [tick -> real] | FORALL (t: tick): h(t) - eps(t) > l(t) + eps(t) } ]

% Inputs
X, L       : VAR [tick -> real]
EPS        : VAR [tick -> posreal]
H          : VAR HIGH_LIMIT

% Outputs
QH, Q, QL : VAR [tick -> bool]

LIMITS_ALARM_fbd_impl (X, H, L, EPS, QH, Q, QL): bool =
  EXISTS (w1: [tick -> posreal]), (w2, w3: [tick -> real]):
      DIV(EPS, (LAMBDA (t: tick): 2.0), w1)
    & SUB(H(L, EPS), w1, w2)
    & ADD(L, w1, w3)
    & HYSTERESIS_tab_req(X, w2, w1, QH)
    & HYSTERESIS_tab_req(w3, x, w1, QL)
    & DISJ(QH, QL, Q)
```

Finally, we prove that the FBD implementation of *LIMITS_ALARM* is *correct* and *consistent*.

```
LIMITS_ALARM_correctness: THEOREM
  FORALL X, H, L, EPS, QH, Q, QL:
    LIMITS_ALARM_fbd_impl(H, X, L, EPS, QH, Q, QL) IMPLIES
      LIMITS_ALARM_tab_req(H, X, L, EPS, QH, Q, QL)

LIMITS_ALARM_consistency: THEOREM
  FORALL H, X, L, EPS:
    EXISTS QH, Q, QL:
      LIMITS_ALARM_fbd_impl(H, X, L, EPS, QH, Q, QL)
```

Forthcoming Research

Verification of the TCDD (Trip Computer Design Description) of one of the shutdown systems (SDSs), owned by Ontario Power Generation (OPG), using the pre-verified FBs in IEC 61131-3

References

[1] IEC. *61131-3 Ed. 2.0 en:2003: Programmable Controllers — Part 3: Programming Languages*. International Electrotechnical Commission, 2003.

[2] Sam Owre, John M. Rushby, and Natarajan Shankar. PVS: A Prototype Verification System. In *CADE*, volume 607 of *LNCS*, pages 748–752, 1992.

[3] Linna Pang, Chen-Wei Wang, Mark Lawford, and Alan Wassyng. "Formalizing and Verifying Function Blocks using Tabular Expressions and PVS. In *FTSCS*, volume 419 of *Communications in Computer and Information Science*, pages 163–178. Spring, 2013.

[4] Linna Pang, Chen-Wei Wang, Mark Lawford, and Alan Wassyng. Formal verification of IEC 61131-3 function blocks using tabular expressions. *Science of Computer Programming*, 2014. Invited for Submission. Submission ID: SCICO-D-14-00102.

[5] Alan Wassyng and Mark Lawford. Lessons learned from a successful implementation of formal methods in an industrial project. In *FME*, pages 133–153, 2003.