Certification of Safety-Critical, Software-Intensive Systems

EECS4312:
Software Engineering Requirements
Fall 2019

CHEN-WEI WANG

McMaster Centre for Software Certification

- Led a $20M project (MAR.2008 to SEP.2016) of ***ORF-RE*** (Ontario Research Fund for Research Excellence) on the ***Certification of Safety-Critical Software-Intensive Systems***
- Objectives:
  - Certify software through *product*-focused approaches
  - Develop *methods*, *tools*, and a repository of *certified components*
  - Use ***formal methods*** to provide evidence for certification
- Collaborating with U of Waterloo and York U (Toronto)
- Working with industry and regulators to improve software in:
  - Biomedical Devices [IBM]
  - Financial Systems [Legacy Systems International Inc (LSI)]
  - Automotive [General Motors (GM)]
  - *Nuclear* [Candu, OPG, SWI, Radiy/Sunport]
- My contribution: ***verification*** of ***function blocks*** defined in standards for components used in the *nuclear power industry*

Acknowledgement of Collaborators

McSCert, McMaster University, Canada
- Alan Wassyng [faculty, ***P.Eng.***]
- Mark Lawford [faculty, ***P.Eng.***]
- Linna Pang [PhD student]

Software Engineering Laboratory, York University, Canada
- Jonathan Ostroff [faculty, ***P.Eng.***]
- Simon Hudon [PhD student]

Nanyang Technological University, Singapore
- Yang Liu [faculty]

Singapore University of Technology and Design, Singapore
- Jun Sun [faculty]

Developing Safety-Critical Systems

Industrial standards in various domains list *acceptance criteria* for mission- or safety-critical systems that practitioners need to comply with: e.g.,

Aviation Domain: ***RTCA DO-178C*** "*Software Considerations in Airborne Systems and Equipment Certification*"

Nuclear Domain: ***IEEE 7-4.3.2*** "*Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations*"

Two important criteria are:

1. System ***requirements*** are precise and complete
2. System ***implementation*** conforms to the requirements

But how do we accomplish these criteria?

Professional Engineers: Code of Ethics

- *Code of Ethics* is a basic guide for ***professional conduct*** and imposes duties on practitioners, with respect to **society**, **employers**, **clients**, **colleagues** (including employees and subordinates), the **engineering profession** and him or herself.
- It is the duty of a practitioner to act at all times with,
  1. ***fairness*** and ***loyalty*** to the practitioner's associates, employers, clients, subordinates and employees;
  2. ***fidelity*** to public needs;
  3. devotion to ***high ideals*** of personal honour and professional integrity;
  4. ***knowledge*** of developments in the area of professional engineering relevant to any services that are undertaken; and
  5. ***competence*** in the performance of any professional engineering services that are undertaken.
- Consequence of misconduct?
  - ***suspension*** or ***termination*** of professional licenses
  - civil ***law suits***

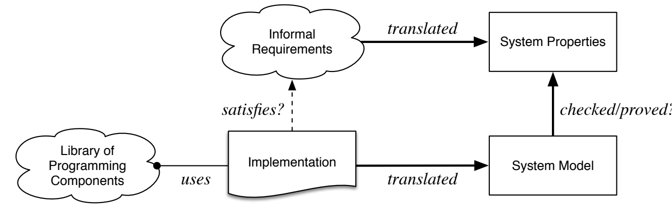
Source: PEO's Code of Ethics

Using Formal Methods to Support the Certification Process

- ***DO-333*** "*Formal methods supplement to DO-178C and DO-278A*" advocates the use of formal methods:

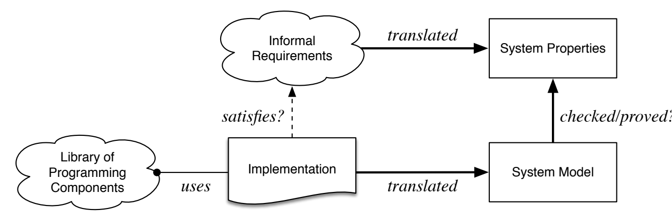
  The use of ***formal methods*** *is motivated by the expectation that, as in other engineering disciplines, performing appropriate* ***mathematical analyses*** *can contribute to establishing the* ***correctness*** *and* ***robustness*** *of a design.*
- FMs, because of their mathematical basis, are capable of:
  - *Unambiguously* describing software system requirements.
  - Enabling *precise* communication between engineers.
  - Providing ***verification evidence*** of:
    - A ***formal*** representation of the system being *healthy*.
    - A ***formal*** representation of the system *satisfying* *safety properties*.

Verification: Building the Product Right?



- ***Implementation*** built via ***reusable programming components***.
- *Goal* : *Implementation Satisfies Intended Requirements*
- To verify this, we ***formalize*** them as a ***system model*** and a set of (real-time) ***properties***, using the specification language of a model checker or a theorem prover.
- Two Verification Issues:
  1. Library components may ***not behave as intended***.
  2. Successful checks/proofs ensure that we *built the product right*, with respect to the informal requirements. But...

Validation: Building the Right Product?

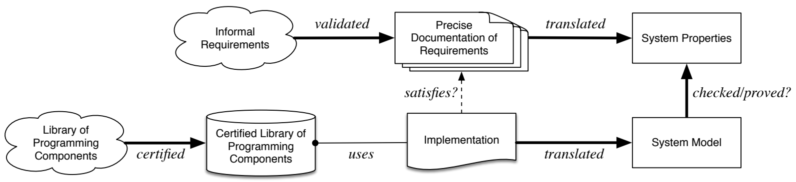


- Successful checks/proofs $\not\Rightarrow$ We *built the right product*.
- The target of our checks/proofs may not be valid:

  The requirements may be ***ambiguous***, ***incomplete***, or ***contradictory***.
- Solution: ***Precise Documentation***

Chen-Wei Wang, Jonathan Ostroff, and Simon Hudon. *Precise Documentation and Validation of Requirements*. In FTSCS. Springer's Communications in Computer and Information Science (CCIS), Volume 419, pp. 262 – 279, 2014.

Building the Right Product Right



- Use ***function tables*** to precisely *document* requirements
- Use the ***PVS theorem prover*** to:
  - *Formulate* library components
  - *Verify* an implementation w.r.t. precise, validated requirements

Cyber-Physical Systems (CPSs)

- Integrations of computation and physical processes
- With ***feedback loops***, embedded computers monitor (via ***sensors***) and control (via ***actuators***) the physical processes.
- The design of CPSs requires the understanding of the *joint dynamics* of computers, software, networks, and physical processes.

Darlington Shutdown Systems (SDSs)

- Two SDSs constitute a safety subsystem.
- Each SDS is a ***watchdog system*** that monitors system parameters of the ***Darlington Nuclear Generating Station*** in Ontario, Canada, and shuts down (i.e., *trips*) the reactor if it observes "bad" behaviour.
- Both SDSs are ***physically isolated*** from the control system.
  - Fully isolated safety systems are *much less complex* than the control systems.
  - This *reduced problem complexity* enables us to design, build, and certify the behaviour of the safety system to a level of quality that would be difficult to achieve for an integrated (and thus more complex) system.
- Both SDSs are completely independent.

The Redesign Project of the Darlington SDSs

- Ontario Hydro (now ***Ontario Power Generation*** Inc. – OPG) developed the original version of the SDS software in late 1980s.
- When seeking for *regulatory approval*, the regulators were not convinced that the software would
  - Perform correctly and reliably
  - Remain correct and reliable under maintenance
- **David Parnas** suggested that a *requirements/design document*, using ***function tables***, be constructed without referencing code.
  - A ***verification*** process conducted after the document ***validated***.
  - The regulators concluded that the software was *safe for use*.

A. Wassyng and M. Lawford. (2003) *Lessons Learned from a Successful Implementation of Formal Methods in an Industrial Project.* FME.

Function Tables

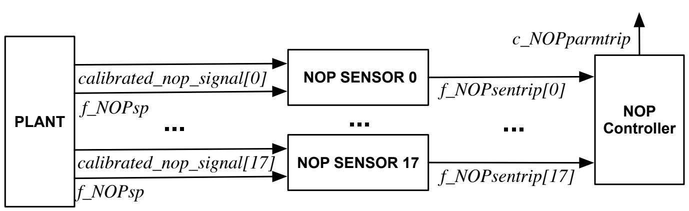
- readable & precise ***documentation*** for complex relations
- suitable for documenting software requirements and design

Condition		Result: f
C_1	$C_{1.1}$	val_1
	$C_{1.2}$	val_2
	...	...
	$C_{1.m}$	val_m
...		...
C_n		val_n

```
IF C1
  IF     C1.1 THEN f = val1
  ELSEIF C1.2 THEN f = val2
  ...
  ELSEIF C1.m THEN f = valm
ELSEIF ...
ELSEIF C_n THEN f = val_n
```

- Two healthiness conditions: [automated in PVS]
  - *completeness* – no missing cases [≥ one row is always true]
  - *disjointness* – deterministic behaviour [rows don't overlap]
- used in Darlington nuclear reactor SDSs [e.g., *f_NOPsentrip*]

Example: Neutron OverPower Unit of Darlington SDS



- ***NOP Controller*** depends on 18 instances of ***Sensor Trip*** units.
- Each sensor *i* monitors two floating-point quantities:
  - *calibrated_nop_signal*[*i*] [a calibrated NOP signal value]
  - *f_NOPsp* [set point value]
- How do we ***formalize*** such informal ***requirements***?
  [function tables!]

NOP Example: Function Tables

Condition	Result: *c_NOPparmtrip*
$\exists i \in 0..17 \bullet f\_NOPsentrip[i] = e\_Trip$	*e_Trip*
$\forall i \in 0..17 \bullet f\_NOPsentrip[i] = e\_NotTrip$	*e_NotTrip*

Table: NOP Controller

Condition	Result: *f_NOPsentrip*[*i*]
$calibrated\_nop\_signal[i] \geq f\_NOPsp$	*e_Trip*
$f\_NOPsp - k\_NOPhys < calibrated\_nop\_signal[i] < f\_NOPsp$	$(f\_NOPsentrip[i])_{-1}$
$calibrated\_nop\_signal[i] \leq f\_NOPsp - k\_NOPhys$	*e_NotTrip*

Table: NOP sensor *i*, $i \in 0..17$ (monitoring *calibrated_nop_signal*[*i*])

Prototype Verification System (PVS)

- interactive environment
  - *specifications* using *higher-order logic* [predicates]
  - *proofs* using sequent-style *deductions* [inference rules]
- direct syntactic support of specifying tabular expressions
  - completeness & disjointness generated as proof obligations
- used for the Darlington SDSs

M. Lawford, P. Froebel, and G. Moum. (2004) *Application of Tabular Methods to the Specification and Verification of a Nuclear Reactor Shutdown System.* Formal Methods in System Design.

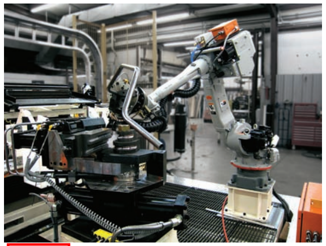
Re-Implementation of the SDSs using PLCs

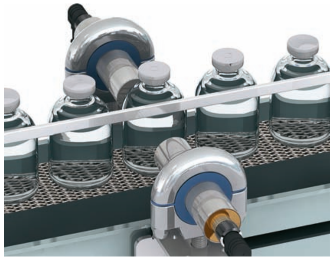
- Input-output behaviour of SDSs has been specified using ***function tables***
- In the refurbishment project, we attempted to verify the re-implementation of SDSs using *Programmable Logic Controllers (PLCs)*

A Visual Introduction to PLCs

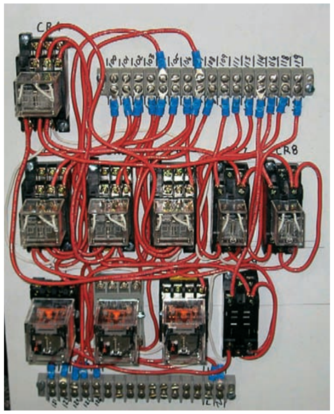
Disclaimer: Many of the PLC and illustration diagrams below are originated from the book *Programmable Logic Controllers* (4th Edition; McGraw-Hill) by Frank D. Petruzella.

PLCs: Utilized in Automating Industrial Process Control

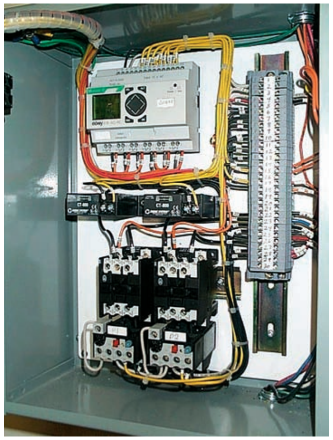




PLCs: Replacing Relay-based Controllers

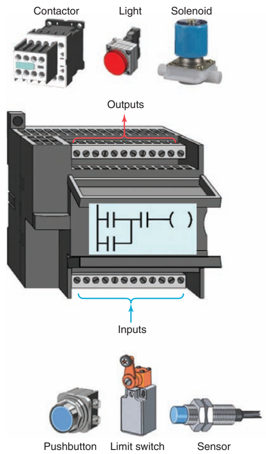


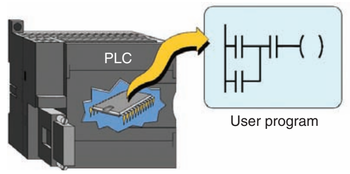
(a) Relay-based Control Panel

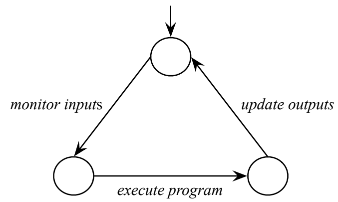


(b) PLC-based Control Panel

PLCs as Cyclic Executives: Inputs, Outputs, Repeated Scans

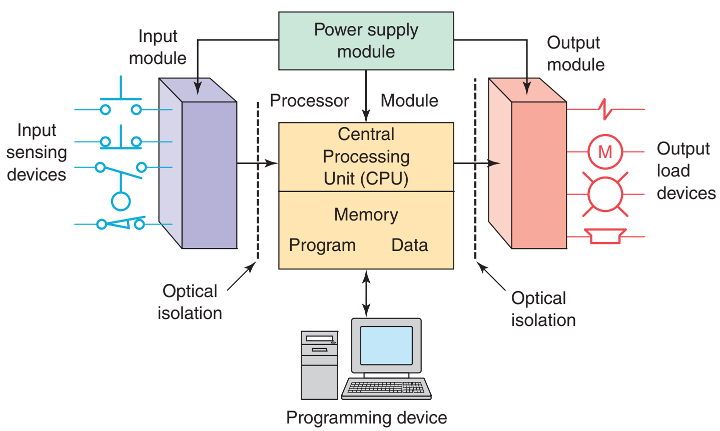




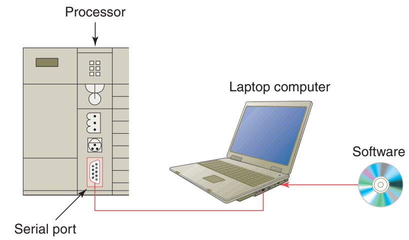


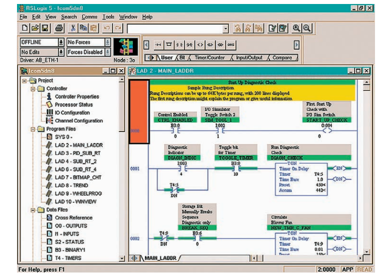
PLCs: Schematic

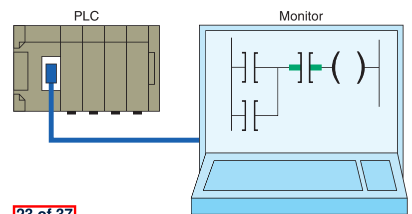
Programs in, e.g., ladder logic, are loaded into memory.

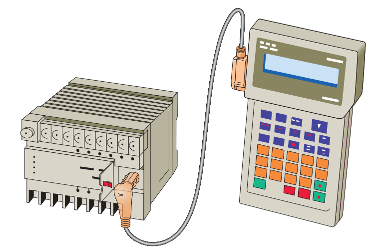


PLCs: Programming & Debugging Interfaces









Using Theorem Proving to Certify Components

- IEC 61131 Standard of PLCs
- Annex F of IEC 61131-3
- A formal approach to certifying the FB library
- Example Issues

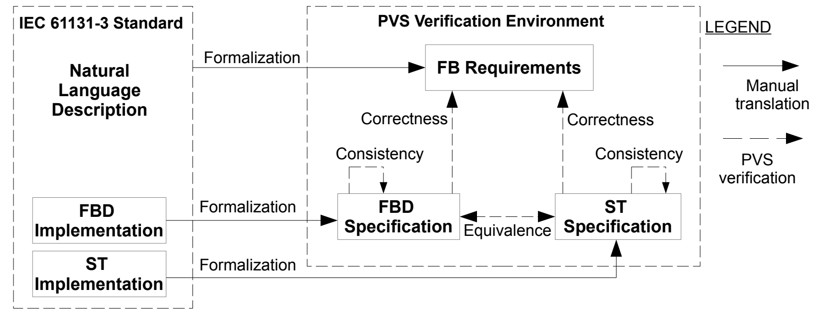
IEC 61131-3 (ed 2.0, 2003): A Standard of PLCs

- *Function Blocks* (FBs): reusable components for programming PLCs.
- First published in 1993, IEC 61131-3 attempts to standardize the programming notations of PLCs using FBs:
  - IL (Instruction List)
  - ***ST*** (***Structured Text***)
  - LD (Ladder Diagram)
  - ***FBD*** (***Function Block Diagram***)
- There are three categories of FBs:
  - basic, stateless functions [e.g., +, ≥ 1, *bcd2int*]
  - ***basic*** FBs [e.g., *hysteresis*]
  - ***composite*** FBs [e.g., *limits_alarm*]

Annex F of IEC 61131-3: A Function Block Library

- ***IEC 61131-3 Annex F*** lists a library of commonly-used FBs.
- PLC manufactures often provide a ***"IEC 61131-3 compliant"*** FB library with their product.
- For the purpose of the *re-implementation of SDS1 using FBs*, we formally certify Annex F using:
  - ***function tables*** [requirements specification]
  - ***PVS theorem prover*** [verification]
- Examined ***29 FBs*** in the library, with a focus on implementations specified in ST and FBD:
  - ***10 issues found*** [ambiguities, missing assumptions, errors]
  - *Lack of precise, black-box requirements* has led to these issues ***unnoticed for ≥ 20 years***!

Formal Verification of the FB Library: How?



1. Formalize FB ***requirements*** as function tables
2. Formalize ST and FBD ***implementations***
3. ***Prove*** correctness and consistency of individual FBs
4. Identify *issues* in ***IEC 61131-3 Annex F*** & Propose solutions

Verification Results from Theorem Proving

Found issues in Annex F of IEC 61131-3:

1. ***Ambiguous behaviour***
   - Incomplete timing diagrams: *pulse* timer
   - Implicit delay unit: *sr* block
2. ***Missing assumptions***
   - input limits: *ctud* block, *hysteresis_alarm*, **limits_alarm** block
   - possible misusage: *delay* block
   - possible division-by-zero: *average*, *pid*
   - possible invalid array indexing: *diffeq*
3. ***Erroneous implementation***
   - inconsistent implementations: *stack_int*

For each issues, we propose *a* solution.

Example 1: Inconsistent Implementations for *STACK_INT*

```
PUSH_STK:
  +------+              +------+        +------+
  |  :=  |              |  +   |        |  =   |
1--|EN ENO|-----------|EN ENO|-------|EN ENO|--
0--|      |--EMPTY 1--|      |--PTR--|G     |--+--OFLO
  +------+         +--|      |  NI---|      |  |
PTR----------------+  +------+       +------+  |
    +------------------------------+-----------+
    |    +------+                  |   +-----+
    |    |  :=  |                  |   | SEL |
    +---O|EN ENO|                  +---|G    |-----OUT
IN--+-----|      |--STK[PTR]  +-------|IN0  |
   |     +------+             |   0---|IN1  |
   +--------------------------+       +-----+
```

```
ELSIF PUSH & NOT OFLO THEN
   EMPTY := 0; PTR := PTR+1; OFLO := (PTR = NI);
   IF NOT OFLO THEN OUT := IN ; STK[PTR] := IN;
   ELSE OUT := 0;
   END_IF ;
END_IF ;
```

- The ***two alternative implementations are inconsistent*** as to when to push an item onto a LIFO stack:

  FBD version specifies that the push operation is performed when the stack is already overflowed!
- We proposed to add a negation gate between *OFLO* to *EN*.
- Does it make sense to fix the ST implementation instead?

Example 2: Up and Down Counters

- An **up-down counter** (*CTUD*) consists of an ***up counter*** (*CTU*) and a ***down counter*** (*CTD*).
- The output counter value ***CV*** is:
  - **Incremented** (using the up counter) if a ***rising edge*** is detected on an input condition ***CU***
  - **Decremented** (using the down counter) if a ***rising edge*** is detected on the input ***CD***.

    Actions of increment and decrement are subject to a high limit ***PVmax*** and a low limit ***PVmin***.
- The initial value of ***CV*** is:
  - **Loaded** to a preset value ***PV*** if a load flag ***LD*** is *TRUE*
  - **Defaulted** to ***0*** if a reset condition ***R*** is enabled
- Two Boolean outputs are produced to reflect the change on *CV*:
  - $\boldsymbol{QU} \equiv (CV > PV)$
  - $\boldsymbol{QD} \equiv (CV <= 0)$

Example 2: Informal Requirements

```
        +--------+
        |  CUTD  |
        |        |
BOOL -->CU    QU|-- BOOL
BOOL -->CD    QD|-- BOOL
BOOL --|R        |
BOOL --|LD       |
 INT --|PV    CV|-- INT
        |        |
        +--------+
```

```
FUNCTION_BLOCK CTUD
  VAR_INPUT
    CU, CD : BOOL R_EDGE; (* Value to be counted up/down *)
    R      : BOOL         (* Reset *)
    LD     : BOOL         (* Load value flag *)
    PV     : INT          (* Preset value *)
  END_VAR
  VAR_OUTPUT
    QU : BOOL (* Compare CV with PV for up counter *)
    QD : BOOL (* Compare CV with 0 for down counter *)
    CV : INT  (* Current counted value *)
  END_VAR
  IF R THEN CV := 0 ;
  ELSIF LD THEN CV := PV ;
  ELSE
    IF NOT (CU AND CD) THEN
      IF CU AND (CV < PVmax)
      THEN CV := CV + 1 ;
      ELSIF CD AND (CV > PVmin)
      THEN CV := CV - 1 ;
      END IF ;
    END IF ;
  END IF ;
  QU := (CV >= PV) ;
  QD := (CV <= 0) ;
END_FUNCTION_BLOCK
```


Example 2: Issues?

- What if $\boxed{PVmax < PVmin}$?

  ⇒ The enabling condition of counter:

  $$PVmin < CV < PVmax \equiv \textbf{\textit{false}}$$

- What if $\boxed{LD \wedge PV \leq PVmin}$ (CV loaded with PV)?

  In the next cycle, if CD is ***true***, then the enabling condition of $\boxed{\text{decrement}}$:

  $$\begin{aligned} & CD \wedge (CV > PVmin) \\ \equiv\ & \{\ CV \texttt{ was preset to } PV \leq PVmin\ \} \\ & CD \wedge (PV > PVmin) \\ \equiv\ & \{\ \texttt{contriction}\ \} \\ & \textbf{\textit{false}} \end{aligned}$$

- What if $\boxed{LD \wedge PV \geq PVmax}$?

Example 2: Resolution?

Function Table!

<table>
<tr><th colspan="4">Condition</th><th>Result
CV</th></tr>
<tr><td colspan="4">R</td><td>0</td></tr>
<tr><td rowspan="7">¬R</td><td colspan="3">LD</td><td>PV</td></tr>
<tr><td rowspan="6">¬LD</td><td colspan="2">CU ∧ CD</td><td>NC</td></tr>
<tr><td rowspan="2">CU∧¬CD</td><td>$CV_{-1} < PVmax$</td><td>$CV_{-1}+1$</td></tr>
<tr><td>$CV_{-1} \geq PVmax$</td><td>NC</td></tr>
<tr><td rowspan="2">¬CU∧CD</td><td>$CV_{-1} > PVmin$</td><td>$CV_{-1}-1$</td></tr>
<tr><td>$CV_{-1} \leq PVmin$</td><td>NC</td></tr>
<tr><td colspan="2">¬CU ∧ ¬CD</td><td>NC</td></tr>
</table>

assume: $PVmin < PV < PVmax$

Beyond this lecture . . .

Linna Pang, **Chen-Wei Wang**, Mark Lawford, and Alan Wassyng. ***Formal Verification of Function Blocks Applied to IEC 61131-3***. In Science of Computer Programming (SCP), Volume 113, December 2015, pp. 149 – 190.

Index (1)

Index (2)

Index (3)