

**Procedural Transformation from Formal Software  
Requirement to PLC-based Design**

Junbeom Yoo  
KAIST

Hojung Bang  
KAIST

Sungdeok Cha  
KAIST

CS/TR-2004-198

February 20, 2004

**K A I S T**  
**Department of Computer Science**

# Procedural Transformation from Formal Software Requirement to PLC-based Design

Junbeom Yoo  
KAIST

Hojung Bang  
KAIST

Sungdeok Cha  
KAIST

## Abstract

The software of the nuclear power plant digital control system is a safety-critical system where many techniques must be applied to it in order to preserve safety in the whole system. Formal specifications especially allow the system to be clearly and completely specified in the early requirements specification phase, therefore making it a trusted method for increasing safety. In this paper, we discuss the formal description of the procedure [12], which generates FBD-based PLC design programs from the requirements specification written in NuSCR [11], a formal requirements specification method. Our discussion includes the formal definition of all constructs and transformation algorithm, and makes confirms that systematically generated FBD program has the same behavior with its origin, NuSCR software requirements specification.

## 1 Introduction

In the area of nuclear power plant control systems, the software safety [5] becomes more important with the replacement of existing analog systems, which is based on RLL(Relay Ladder Logic), by digital systems composed of software process controllers [6]. Therefore, formal software requirements specification methods [7] are required to preserve the safety of such systems in the early phase of the software development process. Also software requirements and design specifications which are suitable for the characteristics of nuclear power plants system, are becoming a new research issue by many researchers [10, 9, 4, 1]. NuSCR [11] is a formal specification method specialized for this purpose, and it is being used to formally specify the software requirements specification of DPPS(Digital Plant Protection System) RPS(Reactor Protection System), which is presently being developed at KNICS [4] in Korea. NuSCR has been evaluated as a well qualified and specialized technique for the nuclear domain [13].

In the analysis phase of the software development process for the control software in nuclear power plant systems, the software requirement specification written in natural language, s.t. English or Korean, is prepared. Next, in the design phase, we usually specify the software design requirements with PLC languages, s.t. SFC(Sequential Function Chart), LD(Ladder Diagram), and FBD(Function Block Diagram) [2], which PLC(Programmable Logic Controller) can interpret and compile with. It is because that software in nuclear power plants' control systems is the embedded one that is implemented on PLC.

In this design phase, the developers usually writes the PLC programs manually from the requirements specification. It is due to that the requirements specification is written in a natural language. If we use the formal specification method, i.e. NuSCR, as software requirements specification, we can generate PLC programs from the formal requirements specification, which has a same behavioral aspect with the formal specification, since it has the formal syntax and semantics. This systematic generation of PLC programs can reduce the possible errors occurring in the manual design specification, and also the software development cost and time. From this aspect, we developed the procedure of systematic generation of PLC programs, i.e. FBD, from NuSCR specification [12].

In this paper, we discuss the formal description of the transformation procedure proposed. Our discussion includes the formal definition of all constructs and transformation algorithm, and makes confirms that systematically generated FBD program has the same behavior with its origin, NuSCR software requirements specification. The remainder of the paper is organized as follows: Section 2 introduces the formal syntax and semantics of NuSCR. In Section 3, we introduce the procedure of transformation from NuSCR specification to FBD programs and the formal description of it. We then briefly introduce NuSCR software requirements specification for Reactor Protection System, which is presently developing in Korea, as a real case study, and described the whole transformation procedure from NuSCR specification to FBD programs in Section 4. Conclusion and future work are in Section 5.

## 2 Formal Syntax and Semantics of NuSCR

NuSCR basically uses four constructs, *monitored variable*, *input variable*, *output variable*, and *controlled variable* according to *Parnas' Four-Variable Model*[10]. In addition, NuSCR introduces three other basic constructs, *function variable*, *history variable*, and *timed history variable*. These three constructs can be defined as SDT(Structured Decision Table), FSM(Finite State Machine), and TTS(Timed Transition System) respectively. The relationship of all constructs is represented by FOD, which describes all relationships between all constructs in NuSCR. In this Section, we begin with the basic definition of constructs for NuSCR.

**System Entities** System entities constructing NuSCR software requirements specification are defined as follows:

- $V_{SE}$  is a set of all system entities, defined as  $V_{SE} = V_I \cup V_F \cup V_H \cup V_{TH} \cup V_O$ 
  - $V_I$  : a set of system input variables
  - $V_F$  : a set of function variables
  - $V_H$  : a set of history variables
  - $V_{TH}$  : a set of timed history variables
  - $V_O$  : a set of system output variables
- $D_{SE}$  : a set of all possible valuation domain for every variables in  $V_{SE}$

**System State** To specify the meaning of software system written in NuSCR, we need to define an state valuation  $\sigma$  [8]. If  $S$  is the set of all possible *states* of variables in  $V_{SE}$ ,  $\sigma$  defines a correspondence between every variable and the value that is its current contents. Therefore it is convenient to model  $\sigma$  by function with domain  $V_{SE}$  and co-domain  $D_{SE}$ :

$$\sigma : S = V_{SE} \rightarrow D_{SE}$$

Then for any variable set  $V$  in  $V_{SE}$ ,  $\sigma[V]$  is the contents of  $V$  in current state. The notation  $\sigma[d/V]$  is used to update a state. It means the state  $\sigma'$  such that  $\sigma'[V] = d$  and for all  $V' \neq V$  in  $V_{SE}$ ,  $\sigma'[V'] = \sigma[V']$ . That is,  $\sigma'$  is the same function as  $\sigma$  except at the argument  $V$  which is mapped into  $d$ .

**Condition Statements** Condition statements are the predicates on the value of all entities in  $V_{SE}$ . The condition statements in NuSCR are defined as BNF form as follows:

Let  $r \in V_{SE}$ ,  $v_r \in D_{SE}$ ,  $a, b \in N$ , and  $\otimes \in \{=, \neq, \leq, <, \geq, >\}$ ,  
*simple\_condition* :=  $r \otimes v_r \mid r \otimes r \mid TRUE \mid FALSE$

$$\begin{aligned}
\text{complex\_condition} &:= \text{simple\_condition} \wedge \text{simple\_condition} \\
&\quad | \neg \text{simple\_condition} | \text{simple\_condition} \\
\text{timed\_condition} &:= [a, b] \text{complex\_condition}
\end{aligned}$$

The meaning of *timed\_condition* is according to the semantics of timed transition system [3]. The minimum delay  $a$  in *timed\_condition* means that when the control of timed history node has resided at some location for at least  $a$  time units during which the guard *complex\_condition* has been continuously true, then the transition from this location may occur. The maximum delay  $b$  means that whenever the state of history variable has resided at some location for  $b$  time units during which the guard *complex\_condition* has been continuously true, then the transition from this location has to occur.

**Assignment Statements** Assignment statements mean the valuation of entities in *SE*. The assignment statements in NuSCR are defined as BNF form as follows:

$$\begin{aligned}
\text{Let } r \in V_{SE}, v_r \in D_{SE}, a, b \in N, \text{ and } \oplus \in \{+, -, *, \div\}, \\
\text{assignment} := (r := v_r) | (r := r) | (r := r \oplus r) | (r := r \oplus v_r)
\end{aligned}$$

**Function Variable** Function variable in NuSCR is represented by a function variable node in FOD. It is defined by SDT. Let  $I_{FV}$  be the set of input values from other nodes in FOD into the function variable node itself. Let  $O_{FV}$  be the set of output values from this node. They can be mapped into the set of variables,  $V_{FI}$  and  $V_{FO}$  respectively. Then *comple\_conditions* in SDT are the predicate on  $V_{FI}$ , and actions are the *assignments* on  $V_{FO}$  which is the function variable itself. Therefore, a function variable node can be defined as a function  $f_{FV}$  with input values  $I_{FV}$  to output values  $O_{FV}$  as follows.

$$f_{FV} : I_{FV} \longrightarrow O_{FV}$$

**Definition 1 [SDT: Structured Decision Table]** SDT is defined as a set of a pair  $(p, a)$ , where  $p \in Predicate$  and  $a \in Action$ . *Predicate* is a set of boolean predicates on  $V_{FI}$ , which is the conjunction of *complex\_conditions* in SDT condition statements. *Action* is a set of *assignments* to  $V_{FO}$  which is just the function variable itself.

- $p \in Predicate$  and  $a \in Action$
- if  $p[I_{FV}/V_{FI}]\sigma = TRUE$  then  $a(\sigma) = \sigma[O_{FV}/V_{FO}] = \sigma'[V_{FO}]$       $\lrcorner$

**History Variable** History variable in NuSCR is represented by a history variable node in FOD. It is defined by FSM which is composed of states, transitions between states, and labels on transitions. Let  $I_{HV}$  be the set of input values from other nodes in FOD into the history variable node. Let  $O_{HV}$  be the set of output values from this node. They can be mapped into the set of variables,  $V_{HI}$  and  $V_{HO}$  respectively. Then *complex\_conditions* in FSM's transition labels are the predicate on  $V_{HI}$  and actions are *assignments* on  $V_{HV}$  which is the history variable itself. Therefore, a history variable node can be defined as a function  $f_{HV}$  from input values  $I_{HV}$  to output values  $O_{HV}$  as follows.

$$f_{HV} : I_{HV} \longrightarrow O_{HV}$$

**Definition 2 [FSM: Finite State Machine]** FSM is defined as a relation  $FSM = \langle S_H, s_0, C, A, R \rangle$ , where

- $S_H$  : a set of all states in history variable node
- $s_0$  : initial state in  $S_H$
- $C$  : a set of *complex\_conditions*
- $A$  : a set of *assignments*
- $R$  :
  - a transition relation  $S_H \times C \times A \times S_H$
  - $\exists r (s, c, a, s') \in R$  and  $\exists current\_state \in CS_H$ , such that  
if  $current\_state = s$  and  $c[I_{HV}/V_{HI}]\sigma = TRUE$ ,  
then  $a(\sigma) = \sigma[O_{HV}/V_{HO}] = \sigma'[V_{HO}]$  and  $current\_state' = s'$       $\lrcorner$

$current\_state$  in the definition above means the variable in  $CS_H$ , which indicates the current state of the history node. It will be used in the definition of the overall NuSCR system.

**Timed History Variable** Timed history variable in NuSCR is represented by a timed history variable node in FOD. It is defined by TTS which is a FSM extended with timing constrains  $[a, b]$  in transition labels.  $a$  and  $b$  means the minimum and maximum delay in the transition respectively. Let  $I_{THV}$  be the set of input values from other nodes in FOD into the timed history variable node. Let  $O_{THV}$  be the set of output values from this node. They can be mapped into the set of variables,  $V_{THI}$  and  $V_{THO}$  respectively. Then *timed\_conditions* are the predicate on  $V_{THI}$  and timing constrains  $[a, b]$ , and actions are the *assignment* on  $V_{THV}$  which is the history variable itself. Therefore, a timed history variable node can be defined as a function  $f_{THV}$  from input values  $I_{THV}$  to output values  $O_{THV}$  as follows.

$$f_{THV} : I_{THV} \longrightarrow O_{THV}$$

**Definition 3 [TTS: Timed Transition System]** TTS can be defined as a relation  $TTS = \langle S_{TH}, s_0, C, A, R \rangle$ , where

- $S_{TH}$  : a set of states in timed history variable node  $\times lc$ , where  $lc$  is a local clock in  $LC$
- $s_0$  : initial state in  $S_{TH}$
- $C$  : a set of *complex\_conditions* or *timed\_condition*
- $A$  : a set of *assignments*
- $R$  :
  - a transition relation  $S_{TH} \times C \times A \times S_{TH}$
  - $\exists r (s, c, a, s') \in R$  and  $\exists current\_state \in CS_{TH}$ , such that  
if  $current\_state = s$  and  $c[I_{THV}/V_{THI}]\sigma = TRUE$ ,  
then  $a(\sigma) = \sigma[O_{THV}/V_{THO}] = \sigma'[V_{THO}]$  and  $current\_state' = s'$       $\lrcorner$

$current\_state$  is a variable in  $CS_{TH}$ , which indicates the current state and the current local time. The behavior of transition relations in TTS is a little different from that of FSM because of the timing constraints.

**Function Overview Diagram** FOD in NuSCR describes the relationship between constructs in  $V_{SE}$ . Let  $I_{FOD}$  be the set of input values from out of FOD(i.e. environment or other FODs) into the FOD. Let  $O_{FOD}$  be the set of output values from FOD. They can be mapped into the

set of variables,  $V_{FODI}$  and  $V_{FODO}$  respectively. Also as all nodes in FOD have partial orders according to their execution order and all nodes are defined as functions,  $f_{FOD}$  can be represented as a function composition of all nodes in FOD according to the partial orders on their precedence. Therefore, FOD can be defined as a function  $f_{FOD}$  from input values  $I_{FOD}$  to output values  $O_{FOD}$ .

$$\begin{aligned} f_{FOD} &= f_n \circ \dots \circ f_2 \circ f_1 \\ f_{FOD} &: I_{FOD} \longrightarrow O_{FOD} \end{aligned}$$

**Definition 4 [FOD: Function Overview Diagram]** FOD is defined as a tuple of  $FOD = \langle N, T \rangle$ , where

- $N$ 
  - a set of all nodes in FOD
  - all nodes in  $V_F$  and  $V_H$  and  $V_{TH}$  are defined as functions
  - $V_{FODI}$  in  $V_I$  is a set of input variable nodes in FOD, which mapped as  $\sigma[I_{FOD}/V_{FODI}] = \sigma'[V_{FODI}]$
  - $V_{FODO}$  in  $V_O$  is a set of output variable nodes in FOD, which mapped as  $\sigma[O_{FOD}/V_{FODO}] = \sigma'[V_{FODO}]$
- $T$ 
  - a set of transition  $(n_1, n_2)$  between all nodes  $n_1, n_2$  in  $N$
  - $\forall t = (n_1, n_2) \in T, n_1$  has a precedence on  $n_2$        $\lrcorner$

**Definition 5 [NuSCR Software System]** NuSCR software system is defined as a tuple  $NSS = \langle S, S_0, R, d \rangle$ , in which

- $S$  : a set of system states, which is defined as  $\sigma[V_{SE} \times CS_H \times CS_{TH}]$
- $S_0$  : initial state in  $S$
- $R$  : a set of transition relation  $S \times I \longrightarrow S' \times O$ , where  $I$  and  $O$  are system's input and output values respectively.
- $d$  : system scan cycle time in which the system get the changed valuation function  $\sigma$  periodically       $\lrcorner$

NuSCR software system  $NSS$  uses the definitions of all three basic constructs and FOD.  $NSS$  gets inputs  $I$  from the out of system(i.e. environment), calculates with them, and then emits outputs  $O$  to the outside. In each time that  $NSS$  emits outputs,  $NSS$  changes its internal system states according to the behavior of  $NSS$ . The states of  $NSS$  is defined as  $\sigma[V_{SE} \times CS_H \times CS_{TH}]$ , where  $CS_H$  and  $CS_{TH}$  mean the set of current state of history variable node and timed history variable node respectively. The states of  $NSS$  means the current contents of all variables used in  $NSS$ . The behavior of  $NSS$  is defined based on a function  $f_{FOD}$  defined above. That is, between system states, there exists transition relation s.t.  $R$ , and it corresponds to  $O = f_{FOD}(I)$ .  $NSS$  also operates periodically with system scan cycle time  $d$ . With every time interval  $d$ , it gets the changed valuation function  $\sigma$  for the inputs and outputs of  $NSS$ . This periodic behavior of  $NSS$  is an essential part for digital plant protection system in nuclear power plants, which requires the strict scan cycle time.

### 3 Transformation Procedure from NuSCR to FBD

The overall process that generates FBD program from NuSCR requirements specification [12] has 4 steps. At first we perform the consistency and completeness analysis of the whole individual nodes in FOD, which are defined as SDT, FSM, or TTS. After modifying all nodes to be complete and consistent, we make out 2C-Table for the nodes defined as FSM and TTS. 2C-Table is an intermediate notation that are used to facilitate the FBD generation process for FSM and TTS. In the next step, FBDs are generated from SDTs or 2C-Tables. FBDs generated in this step are the individual and independent FBDs, which are in FOD and defined as SDT, FSM, and TTS. After generating the individual FBDs, we analyze the dependency between all nodes in FOD and decide the total execution order for all nodes in the FOD. This order is also applied to the individual FBDs generated in the same way. PLC executes its application programs sequentially and this execution ordering step is essential.

#### 3.1 Step 1. Completeness and Consistency Checking

For the effective generation of FBD from 3 kinds of nodes in NuSCR, we perform first the analysis about the completeness and consistency of whole individual nodes in FOD. After this first step, we get the complete and consistent three constructs of NuSCR.

SDT, which defines the function variable, is a table consisting of condition and action statements, and duplicated conditions or actions need to be analyzed. The nondeterministic actions resulted from the duplicated conditions also have to be analyzed for the consistent behavior of SDTs.

**Definition 6 [Completeness and Consistency for SDT]** For an SDT  $Sdt = \Sigma(p, a)$ , where the number of  $p$  in  $Sdt$  is  $n$ ,

- **Completeness** :  $\forall (p_1, a_1), \dots, (p_n, a_n) \in Sdt, p_1(\sigma) \vee \dots \vee p_n(\sigma) = TRUE$
- **Consistency** :  $\forall$  pairs  $(p_1, a_1), (p_2, a_2) \in Sdt, p_1(\sigma) \wedge p_2(\sigma) = FALSE \quad \lrcorner$

FSM and TTS, which define the history variable and timed history variable respectively, also have to be analyzed and modified in the same way. Automata, s.t. FSM and TTS, has a feature that if there is no condition satisfied to transit other states in current state, then it maintains the current state. In this case, we need to specify the implicitly omitted transitions explicitly for the completeness. In case of no action statement in the transition label in FSM and TTS, NuSCR regards that the output is set to the same value in the previous scan cycle. We need the explicit addition for such cases. Also as the case of SDT, we have to check whether or not they have the same condition, which have different actions, for consistency.

**Definition 7 [Completeness and Consistency for FSM, TTS]** For an FSM, TTS  $Aut = \langle S_{Aut}, s_0, C, A, R \rangle$ , where the number of condition  $c \in C$  on the state  $s$  in  $S_{Aut}$  is  $n$ ,

- **Completeness** : for some  $s \in S_{Aut}, \forall c_1$  in  $(s, c_1, a_1, s_1), \dots, c_n$  in  $(s, c_n, a_n, s_n), c_1(\sigma) \vee \dots \vee c_n(\sigma) = TRUE$
- **Consistency** : for some  $s \in S_{Aut}, \forall$  pairs  $r_1 = (s, c_1, a_1, s_1), r_2 = (s, c_2, a_2, s_2) \in R, c_1(\sigma) \wedge c_2(\sigma) = FALSE \quad \lrcorner$

#### 3.2 Step 2. 2C-Table Generation

All nodes in FOD are modified into the consistent and complete ones at the previous step 1. In the second step, among three kinds of variables, the complete and consistent FSM and TTS

are transformed into 2C-Tables. 2C-Table is an intermediate notation to generate FBDs from automata systematically and it is generated in such a way that they have the same behavior with the original automata, s.t. FSM and TTS. 2C-Table has the same shape with SDT, but the different is that it has an additional action part, which is used for indicating the state transitions of automata.

**Definition 8 [2C-Table]** 2C-Table is defined as a set of  $(sc, p, a, sta)$  where,

- $sc$  : state condition, which indicates the current "state" of FSM or TTS. (the current value of *current\_state* variable)
- $p$  : boolean predicate on  $V_{SE}$ , which is the conjunction of *complex\_conditions* in FSM, or *timed\_complex\_conditions* in TTS predicates
- $a$  : *assignment* to the variable, which is the 2C-Table itself
- $sta$  : state transition *assignment* to the state variable *current\_state*, which means the next state of FSM or TTS.       $\lrcorner$

Let  $I_{2C-Table}$  be the set of input values from other nodes in FOD into the history or timed history variable node. Let  $O_{2C-Table}$  be the set of output values from this node. They can be mapped into the set of variables,  $V_{2C-Table-I}$  and  $V_{2C-Table-O}$  respectively.  $V_{2C-Table-I}$  are mapped to  $V_{HI}$  or  $V_{THI}$ , and  $V_{2C-Table-O}$  are mapped to  $V_{HO}$  or  $V_{THO}$ . *timed\_complex\_conditions* or *complex\_conditions*  $p$  are the predicate on  $V_{2C-Table-I}$  and actions  $a$  are the *assignments* on  $V_{2C-Table-O}$ , which is the history variable itself. Then, a (timed) history variable node can be defined as a function  $f_{2C-Table}$  from input values  $I_{2C-Table}$  to output values  $O_{2C-Table}$  as follows:

$$f_{2C-Table} : I_{2C-Table} \longrightarrow O_{2C-Table}$$

The condition statements in the generated 2C-Table have the exclusive feature, because they are generated from the complete and consistent automata. This feature of 2C-Table makes the number of generated function blocks from the 2C-Table compact. It also lets us analyze and estimate the number and kinds of FBs used previously. **Algorithm 1** describes 2C-Table generation procedure from FSM or TTS.

**Algorithm 1 [From FSM, TTS to 2C-Table]** Given FSM or TTS  $Aut = \langle S_{Aut}, s_0, C, A, R \rangle$ , generates a 2C-Table  $2C\_Table = \Sigma(sc, p, a, sta)$ ,

```

0 from_Aut_to_2CTable(Aut) {
1   for all  $s \in S_{Aut}$  {
2     for all possible transition relation  $r = (s, c, a', s')$  {
3        $sc := s$ ;
4        $sta := s'$ ;
5        $p := c$ ;
6        $a := a'$ ;
7     }
8   }
9 }       $\lrcorner$ 

```

**Theorem 1 [Behavioral Preservation for 2C-Table]** 2C-Table generated for FSM or TTS by Algorithm 1 has the same behavior with that of its source, FSM or TTS.



**proof** In the generation Algorithm 1 above, for each possible state transition relation, the corresponding 2C-Table element is produced. As all contents of FSM or TTS are mapped into 2C-Table without omission, the generated 2C-Table and its origin FSM or TTS are behaviorally equivalent.

⌋

### 3.3 Step 3. Basic FBD Generation

In this step, we generate an FBD for each SDT or 2C-Table. The FBD generated corresponds to each node in FOD. FBD is composed of a number of FBs(Function Blocks) and their transition relations. Each arithmetic and logical expression *complex\_conditions* and *assignments* statements in SDT or 2C-Table can be mapped into a corresponding unique FB. Timing expression, s.t.  $[a, b]$ , also has its corresponding FB. (Fig.1) describes the representative FBs, which are used for developing SW controllers in nuclear power plants system, and their formal definitions. We described the case of restricted inputs for the convenience. With the function blocks defined, we can define FBD.

**Definition 9 [FB: Function Block]** Function block FB is defined as a tuple  $\langle Name, IP, OP, BP \rangle$ , where

- *Name* : name of function block
- *IP* : a set of input ports
- *OP* : a set of output ports
- *BP* : behavior description, which is  $\Sigma(p_{FB}, a_{FB})$ , where
  - $p_{FB}$  : predicate on *IP*, which is a conjunction of (*timed*)*complex\_conditions*
  - $a_{FB}$  : *assignment* on *OP*      ⌋

**Theorem 2 [1:1 Correspondence between NuSCR operational operators and FBs]**

*There exists a unique function block for every NuSCR operation operators, s.t. arithmetic, logical, selection, and timing operators.*

**Definition 10 [FBD: Function Block Diagram]** Function block diagram FBD is defined as a tuple  $\langle FBs, V, T \rangle$ , where

- *FBs* : a set of function blocks
- *V* : input and output variables of FBD, s.t.  $V_{FBD-I} \cup V_{FBD-O}$
- *T* : a set of transitions between FBs and V, s.t.
  - $V_{FBD-I} \times FB.IP$
  - $FB.OP \times FB.IP$
  - $FB.OP \times V_{FBD-O}$       ⌋

Let  $I_{FBD}$  be the set of input values from other FBDs and  $O_{FBD}$  be the set of output values from this FBD. They can be mapped into the set of variables,  $V_{FBD-I}$  and  $V_{FBD-O}$  respectively.  $V_{2C-Table-I}$  or  $V_{FI}$  in previous step is also mapped to  $V_{FBD-I}$ , and  $V_{2C-Table-O}$  or  $V_{FO}$  is mapped to  $V_{FBD-O}$ . Then we can say that the input of 2C-Table,  $I_{2C-Table}$ , is  $I_{FBD}$  and the output of 2C-Table,  $O_{2C-Table}$ , is  $O_{FBD}$ . Therefore, a 2C-Table, which represents a (timed) history node

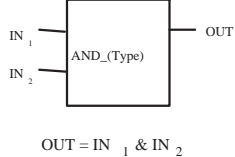
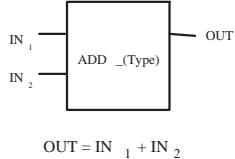
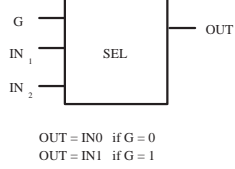
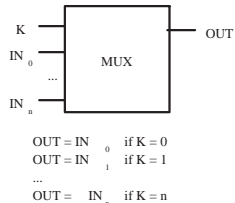
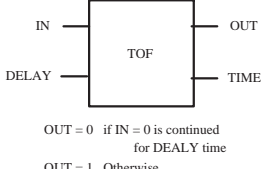
Logical Operation (and, or, >, >=, =, <=, <)	
 <p>OUT = IN<sub>1</sub> &amp; IN<sub>2</sub></p>	$FB_{AND\_Type} = \langle Name, IP, OP, BD \rangle$ <p><i>Name</i> : AND_(Type)  <i>IP</i> : {IN<sub>1</sub>, IN<sub>2</sub>}  <i>OP</i> : {OUT}  <i>BD</i> : {(IN<sub>1</sub> = 1 ; IN<sub>2</sub> = 1, OUT = 1), (IN<sub>1</sub> = 1 ; IN<sub>2</sub> = 0, OUT = 0), (IN<sub>1</sub> = 0 ; IN<sub>2</sub> = 1, OUT = 0), (IN<sub>1</sub> = 0 ; IN<sub>2</sub> = 0, OUT = 0)}</p>
Arithmetic Operation (+, -, *, /)	
 <p>OUT = IN<sub>1</sub> + IN<sub>2</sub></p>	$FB_{ADD\_Type} = \langle Name, IP, OP, BD \rangle$ <p><i>Name</i> : ADD_(Type)  <i>IP</i> : {IN<sub>1</sub>, IN<sub>2</sub>}  <i>OP</i> : {OUT}  <i>BD</i> : {( ; OUT = IN<sub>1</sub> + IN<sub>2</sub>)}</p>
Selection Operation (SEL, MUX)	
 <p>OUT = IN<sub>0</sub> if G = 0  OUT = IN<sub>1</sub> if G = 1</p>	$FB_{SEL} = \langle Name, IP, OP, BD \rangle$ <p><i>Name</i> : SEL  <i>IP</i> : {G, IN<sub>0</sub>, IN<sub>1</sub>}  <i>OP</i> : {OUT}  <i>BD</i> : {(G = 0, OUT = IN<sub>0</sub>), (G = 1, OUT = IN<sub>1</sub>)}</p>
 <p>OUT = IN<sub>0</sub> if K = 0  OUT = IN<sub>1</sub> if K = 1  ...  OUT = IN<sub>n</sub> if K = n</p>	$FB_{MUX} = \langle Name, IP, OP, BD \rangle$ <p><i>Name</i> : MUX  <i>IP</i> : {K, IN<sub>0</sub>, IN<sub>1</sub>, IN<sub>2</sub>}  <i>OP</i> : {OUT}  <i>BD</i> : {(K = 0, OUT = IN<sub>0</sub>), (K = 1, OUT = IN<sub>1</sub>), ..., (K = n, OUT = IN<sub>n</sub>)}</p>
Timer Operation (TOF)	
 <p>OUT = 0 if IN = 0 is continued for DEALY time  OUT = 1 Otherwise</p>	$FB_{TOF} = \langle Name, IP, OP, BD \rangle$ <p><i>Name</i> : TOF  <i>IP</i> : {IN, DELAY}  <i>OP</i> : {OUT, TIME}  <i>BD</i> : {( [DELAY, DELAY] IN = 0, OUT = 0), (IN = 1, OUT = 1)}</p>

Figure 1: Representative function blocks and their formal definitions

or SDT, can be defined as a function  $f_{FBD}$  from input values  $I_{FBD}$  to output values  $O_{FBD}$  as follows:

$$f_{FBD} : I_{FBD} \longrightarrow O_{FBD}$$

**Algorithm 2 [From 2C-Table to FBD]** Given 2C-Table  $2C\_Table = \Sigma(sc, p, a, sta)$ , generates an FBD  $Fbd = \langle FBS, V, T \rangle$ ,

```

0 from_2CTable_to_FBD(2C_Table) {
1   for all conditions and assignments in  $p$  and  $a$  {
2     Construct corresponding partial FBD using FBs;
3   }
4
5   for all  $k$  elements of  $2C\_Table$ , which have the same  $sc$  {
6     Construct partial FBD using  $k - 1$  "SEL" FBs to have one output;
7   }
8
9   for all  $m$  partial FBDs, which have different outputs {
10    Construct an overall FBD using a "MUX" FB to have one final
11    output of FBD and use variable  $status$  as input port  $K$  in "MUX" FB;
12    Construct the other same FBD, which use the variable  $status$ 
13    as the output and input port  $K$  in "MUX" FB;
14  }
15 }

```

**Algorithm 3 [From SDT to FBD]** Given SDT  $Sdt = \Sigma(p, a)$ , generates an FBD  $Fbd = \langle FBS, V, T \rangle$ ,

```

0 from_2CTable_to_FBD(Sdt) {
1   for all conditions and assignments in  $p$  and  $a$  {
2     Construct corresponding partial FBD using FBs;
3   }
4
5   for all  $k$  elements of  $2C\_Table$ , which have the same  $sc$  {
6     Construct an overall FBD using  $k - 1$  "SEL" FBs to have one output;
7   }
8 }

```

(Explanation for Algorithm 2 & 3 is required.)

**Theorem 3 [Behavioral Preservation for FBD generatd]** *FBD generated from 2C-Table by Algorithm 2, or generated from SDT by Algorithm 3 have the same behavior with that of its source, 2C-Table or SDT, respectively.*

### 3.4 Step 4. Total FBD Generation

The individual FBD generated from SDT or 2C-Table in step 3 is corresponding to each node in FOD. In the final step, the execution order for all FBDs, which are corresponding to all nodes in an FOD, is analyzed and decided, and then all FBDs are executed sequentially. FOD in NuSCR describes the relationship between all nodes in the FOD. Also as all nodes in FOD have partial orders according to their execution order and all nodes are defined as functions, so an FOD can be represented as a function composition of all nodes in FOD according to the partial orders on their precedence as explained in Section 2. Therefore, the execution order in FOD is used as a

sequential execution order among generated FBDs in step 3, and the total FBD can be represented as a function composition of all individual FBDs according to their sequential execution orders.

**Definition 11 [Total\_FBD]** The total FBD, which is corresponding to an FOD, is defined as a tuple  $\langle FBDs, T \rangle$ , where

- $FBDs$  :
  - a set of all individually generated FBDs.
  - all nodes in  $FBDs$ , which are the individual FBDs, are defined as functions
  - $V_{FBD\_Total\_I}$ , which is a set of input variables in all FBDs, is mapped as  $\sigma[I_{FBD\_Total}/V_{FBD\_Total\_I}] = \sigma'[V_{FBD\_Total\_I}]$
  - $V_{FBD\_Total\_O}$ , which is a set of output variables in all FBDs, is mapped as  $\sigma[O_{FBD\_Total}/V_{FBD\_Total\_O}] = \sigma'[V_{FBD\_Total\_O}]$
- $T$  :
  - a set of transition  $(n_1, n_2)$  between all nodes  $n_1, n_2$  in  $FBDs$
  - $\forall t = (n_1, n_2) \in T, n_1$  has a sequentially execution precedence on  $n_2$  ┘

Let  $I_{FBD\_Total}$  be the set of input values from out of the total FBD into the total FBD, and  $O_{FBD\_Total}$  be the set of output values from the total FBD. They can be mapped into the set of variables,  $V_{FBD\_Total\_I}$  and  $V_{FBD\_Total\_O}$  respectively. The total FBD  $f_{FBD\_Total}$  is defined as:

$$f_{FBD\_Total} = f_{FBD\_n} \circ \dots \circ f_{FBD\_2} \circ f_{FBD\_1}$$

$$f_{FBD\_Total} : I_{FBD\_Total} \longrightarrow O_{FBD\_Total}$$

**Definition 12 [FBD System]** FBD system is defined as a tuple  $FBDS = \langle S, S_0, R, d \rangle$ , where

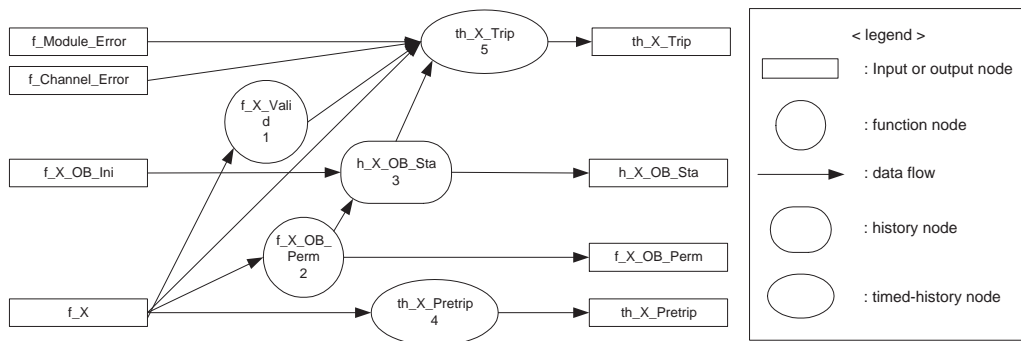
- $S$  : a set of system states, which is defined as  $\sigma[V_{FBD\_Total\_I} \times V_{FBD\_Total\_O}]$
- $S_0$  : initial state in  $S$
- $R$  : a set of transition relation  $S \times I \longrightarrow S' \times O$ , where  $I$  and  $O$  are FBD system's input and output values respectively, where  $O = f_{FBD\_Total}(I)$ .
- $d$  : system scan cycle time in which the system get the changed valuation function  $\sigma$  periodically ┘

**Theorem 4 [Behavioral Preserving Transformation from NSS to FBDS]** *From the definitions and theorems previously proposed, FBD system  $FBDS$ , which is generated from NuSCR specification using the proposed systematic generation method, has the same behavior with NuSCR software system  $NSS$ .*

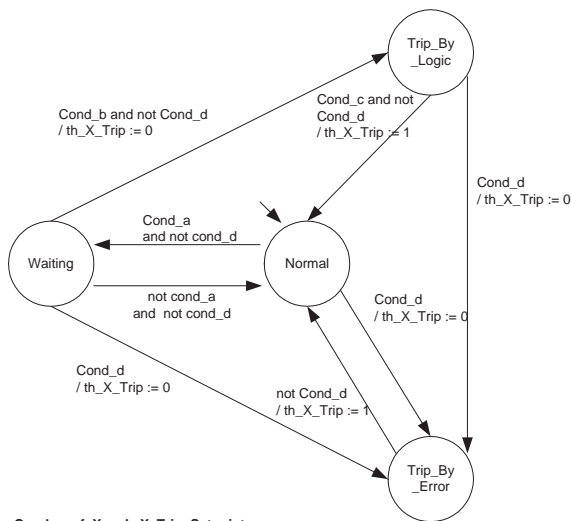
## 4 Case Study: RPS Example

In this section, we introduce a fixed set-point trip with operating bypass example, a trip logic of BP(Bistable Logic) in DPPS(DIgital Plant Protection System) RPS, which is presently at developing in KNICS [4] in Korea. Then, we describe the whole transformation procedure from NuSCR requirements specification to corresponding FBD program with RPS example.

(Fig.2) describes a fixed set-point rising trip logic in RPS. It means that the trip set-point is fixed and trip occurs if the input value falls below the set-point. (Fig.2 (a)) is the FOD of this

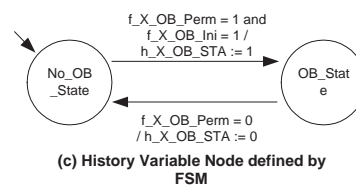


(a) Function Overview Diagram



Cond\_a :  $f\_X \geq k\_X\_Trip\_Setpoint$   
 Cond\_b :  $[k\_Trip\_Delay, k\_Trip\_Delay] (f\_X \geq k\_X\_Trip\_Setpoint \text{ and } h\_X\_OB\_Sta = 0)$   
 Cond\_c :  $f\_X < k\_X\_Trip\_Setpoint - k\_X\_Trip\_Hys$   
 Cond\_d :  $f\_X\_Valid = 1 \text{ or } f\_Module\_Error = 1 \text{ or } f\_Channel\_Error = 1$

(b) Timed History Variable Node defined by TTS



(c) History Variable Node defined by FSM

Conditions		
$k\_X\_MIN \leq f\_X \leq k\_X\_MAX$	T	F
Actions		
$f\_X\_Valid := 0$	X	
$f\_X\_Valid := 1$		X

(d) Function Variable Node defined as SDT

Figure 2: NuSCR specification for a part of RPS

trip logic module. The timed-history variable node  $th\_X\_Trip$ , which is represented as an ellipse in (Fig.2 (a)), is defined in (Fig.2 (b)). The history variable node  $h\_X\_OB\_Sta$ , which is represented as an rounded rectangle, is defined in (Fig.2 (c)). The function variable node  $f\_X\_Valid$ , which is represented as a circle, is also defined in (Fig.2 (d)). We used the timed-history variable node  $th\_X\_Trip$  as the representative to generate the corresponding FBD program as the proposed approach in the previous section.

At the first step, we perform the analysis about the completeness and consistency of whole individual nodes in FOD for the effective generation of FBD. The timed-history variable node  $th\_X\_Trip$  defined as TTS in (Fig.2 (b)) is analyzed, and then the modified TTS is (Fig.3 (a)). We specified explicitly the omitted transitions and performed the consistency checking for transitions as *Definition 7*. In step 2, we generated 2C-Table for the modified TTS in (Fig.3 (a)) as *Algorithm 1*. The generated 2C-Table for  $th\_X\_Trip$  is described in (Fig.3 (b)). As known from *Theorem 1*, this generated 2C-Table has the same behavior with its origin TTS node  $th\_X\_Trip$ .

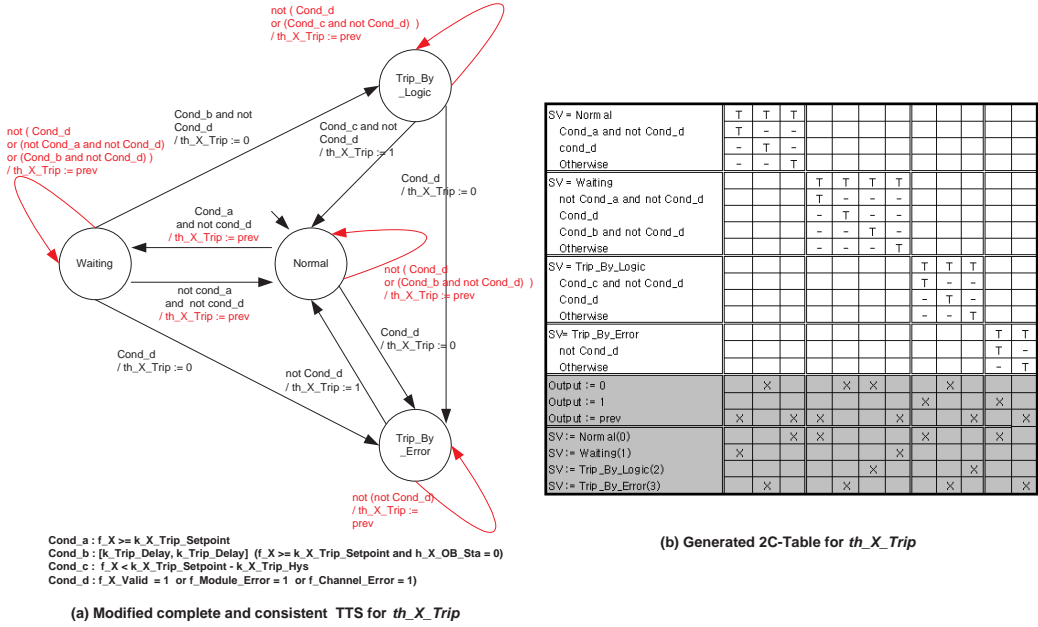


Figure 3: Step 1 & 2 for TTS node  $th\_X\_Trip$

In step 3, we generate an FBD for  $th\_X\_Trip$  node from its 2C-Table as *algorithm 2*. (Fig.4 (a)), which is a preprocessing part, is the generated partial FBD corresponding to line 0 ~ 3 in *Algorithm 2*. The line 5 ~ 7 in *Algorithm 2*, which generates the individual SEL function blocks, corresponds to the four blocks of SEL function blocks whose output ports are the input of MUX\_INT function block in (Fig.4 (b)). The line 9 ~ 11 also corresponds to the MUX\_Int function block (Fig.4 (a)), and line 12 ~ 14 operates as the same way. As known from *Theorem 3*, the generated FBD from 2C-Table as *Algorithm 2* has the same behavior with that of its source.

In step 4, finally the execution order for all 5 FBDs in FOD, which are generated as step 3, is analyzed and the decided. The possible execution orders for the 5 nodes composed in FBD in (Fig. 2 (a)) is as follows: At first, there are two partial orders between the 5 nodes in FOD as their input/output relationships. The node numbered 4 has no interaction with other nodes, so it is the independent one.

Partial execution order 1: (1  $\rightarrow$  5)

Partial execution order 2: (2  $\rightarrow$  3  $\rightarrow$  5)

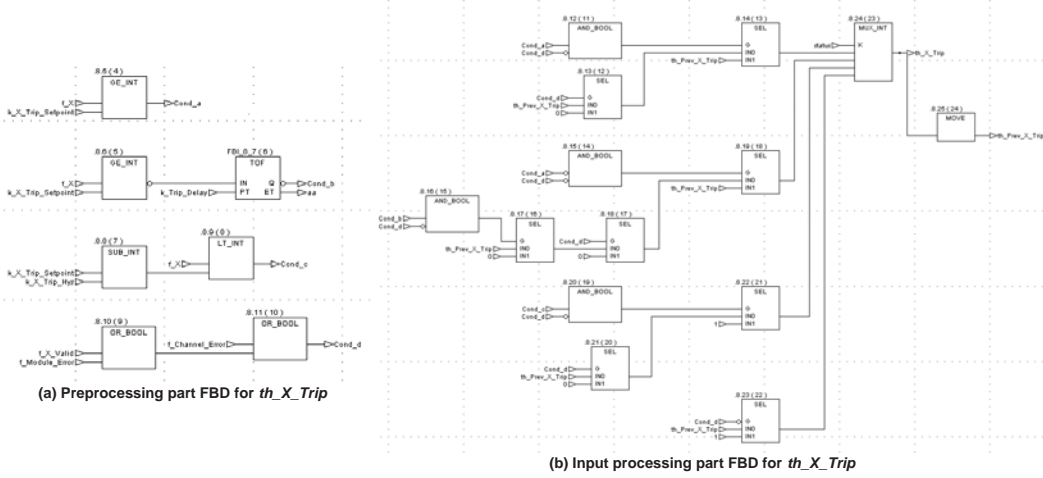


Figure 4: Step 3 for TTS node *th\_X\_Trip*

*Independent execution:* (4)

We can get the whole possible execution order below from these partial ordering and independent relationships. The independent node, numbered 4, can be located at any locations. In this way, we can get the sequentially executable FBD programs generated from NuSCR requirements specification.

*Execution order 1:* (Input)  $\rightarrow$  1  $\rightarrow$  2  $\rightarrow$  3  $\rightarrow$  5  $\rightarrow$  (4)  $\rightarrow$  (Output)

*Execution order 2:* (Input)  $\rightarrow$  2  $\rightarrow$  1  $\rightarrow$  3  $\rightarrow$  5  $\rightarrow$  (4)  $\rightarrow$  (Output)

*Execution order 3:* (Input)  $\rightarrow$  2  $\rightarrow$  3  $\rightarrow$  1  $\rightarrow$  5  $\rightarrow$  (4)  $\rightarrow$  (Output)

## 5 Conclusion and Future Work

In this paper, we describe the formal procedure for the transformation procedure from NuSCR requirements specification to FBD-based design specification. We presented a formal definition of a set of function blocks, which are sufficient to develop software controller in nuclear power plants system. With the formal definition of FBD derived from function blocks and the description of the transformation algorithm in each step, we make confirms that systematically generated FBD design specification has the same behavior with its origin, NuSCR requirements specification.

As we mentioned in [12], the proposed transformation method increases the generated FBD program by 3 times and its execution also takes much time proportionately. To be more appropriate for the characteristics of PLC programs, s.t. periodic operation with a strict time bound, even if 3 times execution time does not result in a problem for the most PLC programs in nuclear power plants domain, the additional modification and optimization manually conducted by expert developers to reduce the size of generated FBD program from NuSCR specification are required. For this FBD optimization, an analysis method to check whether the original generated FBD program and the subsequently modified ones are behaviorally equivalent. We currently focus on the analysis on the behavioral preservation of modifications between subsequently modified FBD design specification.

## References

- [1] Edmund M. Clarke and Jeannette M. Wing. Formal method: State of the art and future direction. *ACM Computing Survey*, 28(4):626–643, December 1996.
- [2] IEC(International Electrotechnical Commission). International standard for programmable controllers: Programming languages, 1993. part 3.
- [3] Thomas A. Henzinger, Zohar Manna, and Amir Pnueli. Timed transition systems. In *REX Workshop*, pages 226–251, 1991.
- [4] KNICS. Korea nuclear instrumentation and control system research and development center. <http://www.knics.re.kr/english/eindex.html>.
- [5] Nancy G. Leveson. *SAFWARE, System safety and Computers*. Addison Wesley, 1995.
- [6] U.S. NRC. *Digital Instrumentation and Control Systems in Nuclear Power Plants: safety and reliability issues*. National Academy Press, 1997.
- [7] Doron A. Peled. *SOFTWARE RELIABILITY METHODS*. Springer, 2001.
- [8] R.D. Tennent. The denotational semantics of programming languages. *Communicatin of the ACM*, 19(8):437–453, 1976.
- [9] A.J. Schouwen Van, D. Parnas, and J. Madey. Documentation of requirements for computer systems. In *RE'93: IEEE International Symposium on Requirements Engineering*, pages 198–207, 1993.
- [10] WolsongNPP2/3/4. Software work practice, procedure for the specification of software requirements for safety critical software. 00-68000-SWP-002, Sept. 1991.
- [11] Junbeom Yoo and Sungdeok Cha. A formal software requirements specification method for digital nuclear plants protection systems. *Journal of Systems and Software accepted*, 2003.
- [12] Junbeom Yoo, Sungdeok Cha, Changhui Kim, and Duck Yong Song. From formal software requirement to plc-based design. *Reliability Engineering and System Safety submitted*, 2003.
- [13] Junbeom Yoo, Sungdeok Cha, Younju Oh, and Changhui Kim. Toward the formal software requirements specification for digital reactor protection systems. *IEEE transaction on Nuclear Science submitted*, 2003.