Proceedings of the Korean Nuclear Society Spring Meeting
Seoul, Korea, May 1998

Mathematical Verification of A Nuclear Power Plant Protection System Function With Combined CPN and PVS

Seo Ryong Koo, Han Seong Son and Poong Hyun Seong

Korea Advanced Institute of Science and Technology
Department of Nuclear Engineering
373-1 Kusong-dong, Yusong-gu
Taejon, Korea 305-701

Abstract

In this work, an automatic software verification method for Nuclear Power Plant (NPP) protection system is developed. This method utilizes Colored Petri Net (CPN) for modeling and Prototype Verification System (PVS) for mathematical verification. In order to help flow-through from modeling by CPN to mathematical proof by PVS, a translator has been developed in this work. The combined method has been applied to a protection system function of Wolsong NPP SDS2(Steam Generator Low Level Trip) and found to be promising for further research and applications.

I. Introduction

In this work, an automatic software verification method for Nuclear Power Plant (NPP) protection system is developed. This method utilizes Colored Petri Net (CPN) for modeling and Prototype Verification System (PVS) for mathematical verification.

For the safety-critical protection systems, complete analysis of the system is needed since an error in the requirements may generate serious faults of software. CPN has been used as an adequate tool of requirement analysis. [1] CPN has some advantages such as rapid prototyping and visualizing of requirements. However, CPN is not proper for the mathematical verification of the system. In this work, therefore, PVS is used for the mathematical verification.

In this work, first, the relevant matters of Steam Generator Low Level Trip (one of the Wolsung NPP SDS2 parameter) are modeled with Design/CPN. This model offers an advantage to the verification process in that the easy communication between users and system developers is possible. Next, PVS specification, which is translated from CPN model, is verified mathematically. The focus in this work is on the flow-through from CPN model to PVS specification and a translator has been developed. The translator, developed in this work, is one of the most important parts in this work.

II. Design/CPN and PVS

Petri Net is a modeling language that has been used in modeling and for analysis of the system. CPN has expressions of concurrency and formal semantics. In addition, Petri Net can visualize the actual system with ease. However, Petri Net is so basic that the ability of expression is limited and CPN has been developed to overcome this limit. In CPN, color refers to the types of data associated with tokens and is comparable to data types in programming languages. Design/CPN is a powerful tool for the Colored Petri Net. The version of CPN used in Design/CPN incorporates variables (representing the binding of identifiers to specific colored tokens), arc inscriptions (expressions), and the code associated with transitions. In Design/CPN, CPN is a graphical programming language with rich specification and simulation possibilities. The programming language through which CPN specifies desired operations in arc expressions and transition codes is ML.

A non-hierarchical CPN is defined as a many-tuple. However, it should be noted that the only purpose of this is to give a mathematically sound and unambiguous definition of CPN and their semantics. It is in principle easy to translate a CPN diagram into a CPN tuple, and vice versa. The tuple form is adequate when we want to formulate general definitions and prove theorems which apply to all of CPN. The graph form is adequate when we want to construct a particular CPN which models a specific system. Also we investigate the relationship between non-hierarchical CPN and Place/Transition Nets(PT-nets), and it turns out that each CPN can be translated into a behaviorally equivalent PT-net, and vice versa. The translation from CPN to PT-nets is unique, while the translation in the other direction can be done in many different ways. The existence of an equivalent PT-net is extremely useful, because it tells us how to generalize the basic concepts and the analysis methods of PT-nets to non-hierarchical CPN. [1]

One example Design/CPN model is shown in Figure II.1. In this Figure, the CPN model represents the Wolsung NPP SDS2 function (Steam Generator Low Level Trip).

PVS is a verification system: an interactive environment for writing formal specifications and performing formal proofs. It builds on nearly 20 years experience at SRI in building verification systems, and on substantial experience with other systems. The distinguished feature of PVS is its synergistic integration of an expressive specification language and powerful theorem-proving capabilities. PVS has been applied successfully to many large and difficult applications in both academic and industrial settings.

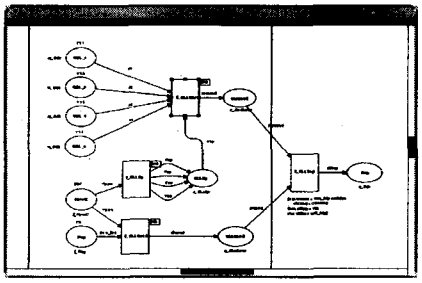

Figure II.1 CPN model of SDS2 Steam Generator Low Level Trip function

III. Verification Technique by Relating CPN with PVS

III-1. CPN modeling

In this work, the target requirement is that for the Wolsung NPP SDS2 function (Steam Generator Low Level Trip). In this requirement, the functional requirements such as f_SLLCond, f_SLLCondA, f_SLLSnr, f_SLLSp, f_SLLSpD, and f_SLLTrip are included. These functional requirements are modeled with Design/CPN.

Figure II.1 shows the top-level CPN model of the function, and other sub-models are shown in Figures III.1 to III.3

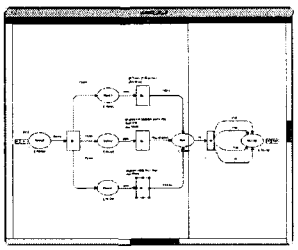

Figure III.1
CPN model of f_SLLSnr function

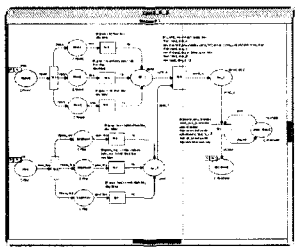

Figure III.2
CPN model of f_SLLSp function

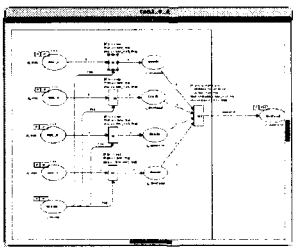

Figure III.3
CPN model of f_SLLCond function

III-2. Conversion to PVS specification language using ML language

It is not meaningless to perform an analysis for each model independently because the analysis of the system is performed in the "divide and conquer" manner. In Design/CPN, it is impossible to access to a data structure of the system directly, but the system information can be extracted by using ML program. That is, Design/CPN has internal functions for asking questions to modeled system. Using this ML functions and programs, we can extract the required system information from the CPN models. The extracted information form is shown as follows. Objects for converting in CPN are Place, Port, Transition, Sub-Transition, Arc, Color, and variable declaration.

```
Place  #  (place id)  #  (place name)  #  (color name)  #  (init mark)  End
Port  #  (port id)  #  (port name)  #  (color name)  #  (init mark)  End
Trans  #  (trans id)  #  (trans name)  #  (guard exp)  #  (code seg)  End
Sub  #  (sub id)  #  (sub name)  #  (guard exp)  #  (code seg)  End
Arc  #  (arc id)  #  (arc exp)  #  (from)  #  (to)  End
Def  #  (declare)  End
```

In PVS, it is hard to verify a system with the above results. Therefore, we need a process that the results from CPN model are converted to PVS inputs (PVS specification language). This process is performed with the translator developed in this work.

As mentioned in section II-1, PVS specification language is composed of simple THEORYs. Therefore, the main page is translated to SLLTrip function THEORY and sub-pages to SLLSnr, SLLSp and SLLCond function THEORYs, respectively. Each THEORY is described in the next section.

```
dictionary   % [ parameters ]
                              : THEORY
  BEGIN
  % ASSUMING
    % assuming declarations
  % ENDASSUMING
  time : TYPE = nat
  m_SGL : TYPE = int
  f_FaveC : TYPE = int
  f_Flog : TYPE = int
  c_SLLSp : TYPE = int
  c_SnrCond : TYPE =   Snr_trip,
Snr_not_trip
  c_SLLCond : TYPE =   CondIn,
CondOut
  c_Trip : TYPE =   trip, not_trip
  Cond_ABC : TYPE =   a_a, a_b,
b_b, a_b_c
  Co : TYPE =   a, b, c

  XT : TYPE = [time -> m_SGL]
  YT : TYPE = [time -> c_SnrCond]
  POWT : TYPE = [time -> f_FaveC]
  POWLOGT : TYPE = [time ->
f_Flog]

  END dictionary
```

Figure III.4

Specification language

of dictionary.pvs

```
variables   % [ parameters ]
                              : THEORY
  BEGIN
  % ASSUMING
    % assuming declarations
  % ENDASSUMING
  IMPORTING dictionary
  % variable declare

  t : time
  s1 : m_SGL
  s2 : m_SGL
  s3 : m_SGL
  s4 : m_SGL
  pow : POWT
  pow_log : POWLOGT
  sp : c_SLLSp
  snrcond : c_SnrCond
  sllcond : c_SLLCond
  prev : c_SLLCond
  slltrip : c_Trip
  x1 : XT
  x2 : XT
  x3 : XT
  x4 : XT
  y1 : YT
  y2 : YT
  y3 : YT
  y4 : YT
  cond_a : Co
  cond_l : Co
  cond_al : Cond_ABC

  END variables
```

Figure III.5

Specification language

of variables.pvs

```
SLLTrip   % [ parameters ]
                              : THEORY
  BEGIN
  % ASSUMING
    % assuming declarations
  % ENDASSUMING

  IMPORTING dictionary
  IMPORTING variables
  IMPORTING SLLSnr
  IMPORTING SLLCond
  IMPORTING SLLSp

  f_SLLTrip   :   THEOREM   (IF
snrcond=Snr_trip AND sllcond=CondIn
THEN slltrip=trip ELSE slltrip=not_trip
ENDIF)

  END SLLTrip
```

Figure III.6

Specification language

of SLLTrip.pvs

III-3. Verification and Validation using PVS

This section describes PVS specification converted from SDS2 system models by the translator. The PVS specification is composed of SLLTrip.pvs(Figure III.6), SLLSnr.pvs(Figure III.7), SLLSp.pvs(Figure III.8), and SLLCond.pvs(Figure III.9) files, named after CPN model pages. The declaration part is translated to dictionary.pvs(Figure III.4) and variables.pvs(Figure III.5) files which are imported to the THEORYs using IMPORTING command.

In this work, the method of verification is the mathematical verification for the converted PVS specification language. PVS specification language for each page is represented with timed states. Thus, we have considered that the signal and the power of the system are functions of time. Then, it is proved mathematically using the PVS proof system.

```
SLLSnr   % [ parameters ]
                    : THEORY
  BEGIN
  % ASSUMING
   % assuming declarations
  % ENDASSUMING
  IMPORTING dictionary
  IMPORTING variables

  t1 : AXIOM (IF x1(t) <= sp THEN
y1(t)=Snr_trip ELSE y1(t)=Snr_not_trip
ENDIF)
  t2 : AXIOM (IF x2(t) <= sp THEN
y2(t)=Snr_trip ELSE y2(t)=Snr_not_trip
ENDIF)
  t3 : AXIOM (IF x3(t) <= sp THEN
y3(t)=Snr_trip ELSE y3(t)=Snr_not_trip
ENDIF)
  t4 : AXIOM (IF x4(t) <= sp THEN
y4(t)=Snr_trip ELSE y4(t)=Snr_not_trip
ENDIF)
  t10 : THEOREM
        (FORALL(t:time):(IF
        y1(t)=y2(t) AND y2(t)=y3(t)
        AND y3(t)=y4(t) AND
        y4(t)=Snr_not_trip THEN
        snrcond=Snr_not_trip ELSE
        snrcond = Snr_trip ENDIF))

  END SLLSnr
```

Figure III.7
Specification language
of SLLSnr.pvs

```
SLLSp   % [ parameters ]
                    : THEORY
  BEGIN
  % ASSUMING
   % assuming declarations
  % ENDASSUMING
  IMPORTING dictionary
  IMPORTING variables

  t8 : THEOREM
(FORALL(t:time):( IF pow(t) < 0 THEN
sp=923 ELSE (IF pow(t)>=0 AND
pow(t)<90 THEN sp=(28*pow(t)+923)
ELSE sp=3438 ENDIF) ENDIF))

  END SLLSp
```

Figure III.8
Specification language
of SLLSp.pvs

```
SLLCond   % [ parameters ]
                    : THEORY
  BEGIN
  % ASSUMING
   % assuming declarations
  % ENDASSUMING
  IMPORTING dictionary
  IMPORTING variables

  t11 : THEOREM
(FORALL(t:time):(IF pow(t)<8 THEN
cond_a=a ELSE (IF pow(t)>=8 AND
pow(t)<10 THEN cond_a=b ELSE
cond_a=c ENDIF) ENDIF))

  t12 : THEOREM
(FORALL(t:time):(IF pow_log(t)<3299
THEN cond_l=a ELSE (IF
pow_log(t)>=3299 AND
pow_log(t)<3349 THEN cond_l=b
ELSE cond_l=c ENDIF) ENDIF))

  t19 : THEOREM (IF cond_a=a AND
cond_l=a THEN cond_al=a_a ELSE (IF
(cond_a=a AND cond_l=b) OR
(cond_a=b AND cond_l=a) THEN
cond_al=a_b ELSE (IF cond_a=b AND
cond_l=b THEN cond_al=b_b ELSE
cond_al=a_b_c ENDIF) ENDIF)
ENDIF)


  t20 : THEOREM (IF (cond_al=a_b
OR cond_al=b_b) AND prev=CondIn
THEN sllcond=CondIn ELSE (IF
cond_al=a_b_c THEN sllcond=CondIn
ELSE sllcond=CondOut ENDIF)
ENDIF)

  END SLLCond
```

Figure III.9
Specification language
of SLLCond.pvs

IV. Conclusion and Further Study

In this study, we could visualize the system requirements easily by using CPN model and verify the system mathematically by using PVS. The integration of CPN and PVS, which realizes rapid prototyping and mathematical verification, can enhance the software reliability. CPN and PVS can cover disadvantage of each other. In order to help flow-through from modeling by CPN to mathematical proof by PVS, a translator has been developed and used in this work. Our software verification method is demonstrated to be useful with a simple example application.

In the future, we are planning to use this software verification method for safety critical system in other areas.

[References]

[1] Kurt Jensen, "Coloured Petri Nets (Basic Concepts, Analysis Methods and Practical Use Volume 1), Second Edition", Springer-Verlag Berlin Heidelberg, 1997.

[2] Tae-ho Kim, "Verification of Safety-Critical System Requirements using PVS", Master Thesis, Department of Computer Science, KAIST, 1997

[3] Jeffrey D. Ullman, "Elements of ML Programming", Prenice-Hall, Inc, 1994.

[4] Judy Crow, Sam Owre, John Rushby, Natarajan Shankar, Mandayam Srivas, "A Tutorial Introduction to PVS", Computer Science Laboratory, SRI International, Updated June 1995.

[5] S. Owre, N. Shankar and J. M. Rushby, "The PVS Specification Language(Beta Release), Computer Science Laboratory, SRI International, April 12, 1993.

[6] N. Shankar, S. Owre and J. M. Rushby, "The PVS Proof Checker: A Reference Manual(Beta Release), Computer Science Laboratory, SRI International, March 31, 1993.

[7] Sam Owre and John Rushby, "FME '96 Tutorial: An Introduction to Some Advanced Capabilities of PVS", Computer Science Laboratory, SRI International, 1996.