Available online at www.sciencedirect.com
SCIENCE DIRECT®
Reliability Engineering and System Safety 83 (2004) 121–122
RELIABILITY ENGINEERING & SYSTEM SAFETY
www.elsevier.com/locate/ress

Correspondence

Comment on: development of a safety critical software requirements verification method with combined CPN and PVS: a nuclear power plant protection system application

1. Introduction

Son and Seong [1] recently published a paper on how requirements for safety-critical systems can be formally specified and verified using CPN and PVS. They used Wolsung SDS2, currently in operation in Korea, in the case study. While their approach is reasonable and useful, we noticed that their paper is not entirely accurate. We chose to write this correspondence because the clarification of technical errors and/or ambiguities that we found in their paper would benefit research community.

This correspondence is organized as follows. In Section 2, we focus on ambiguities in the published translation algorithm. Section 3 describes errors we found in their model and the corresponding PVS specification.

2. The translation algorithm from CPN model to PVS specification

The translation algorithm, shown in Fig. 7 of the published paper, needs to be updated to properly handle an initial marking and to correctly reflect the firing rule of the CPN. Let us explain each with the example taken from the case study.

The initial marking specified in the CPN must be accurately reflected in the initial assignment of variables in the semantically equivalent PVS specification, and we found out that their algorithm lacks such a step. For example, in the CPN model used in the paper, there is a token whose value is `dtt1`. However, in the translated PVS specification, shown in Fig. 16, there are variables declared and lemmas defined, but no variables are assigned initial values. It is true that the lemmas as defined in the paper do not require explicit assignment on the initial condition. That is, the lemma `Pending` could be evaluated with the value of `t` being zero to reflect the initial marking.[1]

[1] Unfortunately, the lemma definition itself in their paper is technically incorrect.

PVS specification must explicitly specify the initial condition using statements like the one below.

```
PDL_st_Init: LEMMA PDL_st(0) = idle
```

And the translation algorithm, shown in Fig. 7 of their paper, must be changed as follows (steps shown in bold is what we added):

```
begin
read the data structure from the output of extractor;
rewrite initial token assignments at places into LEMMAs;
                                        // a missing step in the paper
do {
   ...
```

Next, the algorithm must be revised to accurately reflect the firing condition of each transition. The following steps are provided:

```
1: ...
2:  if (t has a guard or arc expression) then {
3:    ...
4:    if (there exists input place that is the output place of another transition, ...)
5:      then translate the guard or arc expression into LEMMA;
6:    else ...
7: ...
```

The step shown in line 5 above is incorrect because it does not fully address the firing conditions. That is, a transition may fire ONLY when, in addition to satisfying the guard condition, the input places have sufficient number and type of tokens. For example, in Fig. 14 of the published paper, transition `t1` may fire when the guard condition is true and a token of the type `DT` is present in the place Idle. However, in the translated PVS specification, the definition of the lemma `Pending` does not specify that the previous state must be the Idle. Therefore, the above translation procedure must be changed as follows:

```
1: ...
2:  if (t has a guard or arc expression) then {
3:    ...
4:    if (there exists input place that is the output place of another transition, ...)
5:      then translate the guard or arc expression into LEMMA
6:           as well as the source place information into LEMMA; //modified
7:    else ...
8: ...
```

0951-8320/$ - see front matter
doi:10.1016/j.ress.2003.09.002

Correctly translated PVS specification for the transition *t1* would be:

```
PDL_st_Pending: LEMMA PDL_st(t) = idle
AND (sys2(t + 1) = irr_d AND flinst(t + 1) =
flin_h) => PDL_st(t + 1) = Pending   AND
PDL_st (t + 1) / = idle
```

It states that the system may enter the `Pending` state, at time $t + 1$, when the system is in the `idle` state at time t and the guard condition, `sys2` and `flint`, is satisfied. It also specifies that the system must no longer be in the `idle` state at time $t + 1$ once the transition is taken.

3. Errors in the CPN model and the translated PVS specification

We found several minor errors in the CPN model used in the case study. First, syntax definition of CPN does NOT support specification of timing constraints in terms of duration such as '[4.5, 5.0]' appearing in Fig. 14. In the CPN specification, timing constraint may be specified only, in the current version, as time stamp values associated with tokens. It is possible that the authors used the durational notation given above as 'transition names'—it is legal in the CPN syntax—not as timing constraints associated with the transition. In that case, we believe that the use of such transition names is misleading because it creates an impression that durational specification of the timing constraint is possible. In fact, in the translated PVS specification and verification, we found no information relevant to timing constraints.

Second, we found several incidents where the PVS specification, shown in Fig. 16, is incomplete or incorrect. For example, the t is defined to be a constant value of the type `Time` which, in turn, defined to be of the `real` type. It is not used on the right side of the three `LEMMA` definitions. Therefore, it appears that there is an unnecessary parameter in the definition. Furthermore, if t is meant to reflect the passage of time, t must be declared as a variable, not as a constant. That is, the definition must be changed from t : `Time` to t : `VAR Time`.

Third, the quantifiers for variable n within the three `LEMMA`s should be removed. For example, in the definition of the `LEMMA Pending`, quantifier is used as follows:

```
Pend: LEMMA Pending(t, n)
        = (FORALL (n: Occ_Time):
          IF (Irr_Dly(n) = TRUE…)…)
```

In the above specification, the n on the right side of the =, in `Irr_Dly(n)`, is DIFFERENT from the one on the left side (i.e. `Pending(t, n).`) That is, the above specification must be changed as follows to fix the error.

```
Pend: LEMMA FORALL (n: Occ_Time):

Pending (t, n) = IF (Irr_Dly(n) = TRUE…)
```

The same error is found in the proof obligation (or safety property) `THEOREM DT_Safe.`

Acknowledgements

This work was partially supported by the Korea Science and Engineering Foundation through the Advanced Information Technology Research Center.

References

[1] Son HS, Seong PH. Development of a safety critical software requirements verification method with combined CPN and PVS: a nuclear power plant protection system application. Reliab Engng Syst Saf 2003;80(1):19–32.

Taeho Kim*, Sungdeok Cha
Computer Science Division,
EECS Department and AITRC,
Korea Advanced Institute of Science
and Technology (KAIST),
Daejon 305-701, South Korea
E-mail addresses: thkim@salmosa.kaist.ac.kr,
cha@salmosa.kaist.ac.kr.

* Corresponding author.