In Proceedings of the 16th International Conference on Automated Software Engineering, Nov. 2001, pp. 404-407, IEEE.

An Automated Tool for Analyzing Petri Nets using Spin

Gerald C. Gannod[†‡] and Sunil Gupta
Department of Computer Science and Engineering
Arizona State University
Tempe, AZ 85287-5406
E-mail: {gannod, Sunil.Gupta}@asu.edu

Abstract

The Spin model checker is a system that has been used to model and analyze a large number of applications in several domains including the aerospace industry. One of the novelties of Spin is its relatively simple specification language, Promela, as well as the powerful abilities of the model checker. The Petri net notation is a mathematical tool for modeling various classes of systems, especially those that involve concurrency and parallelism. The Honeywell Domain Modeling Environment (DOME) is a tool that supports system design using a wide variety of modeling notations, including UML diagrams and Petri nets. In this paper we describe a tool that supports the use of the Spin model checker to analyze and verify Petri net specifications that have been constructed using the DOME tool. In addition to discussing the translation of Petri nets into Promela, we present several example Petri nets specifications as well as their analysis using Spin.

1 Introduction

The Honeywell Domain Modeling Environment (DOME) [1] is a tool that supports system design using a wide variety of modeling notations, including UML, state transition diagrams, and Petri nets. The main features of DOME include a graphical front-end coupled with a back-end language that facilitates generation of code. As environments such DOME come into more widespread use, the need to provide analysis services that can be integrated with into these environments becomes more necessary.

The Spin model checker [2] is a powerful, but *lightweight* analysis tool that has been used to model and verify both hardware and software systems. Originally developed to model computer and network protocols, the adoption of Spin has found its way into many application domains. One of the novelties of Spin is its relatively simple specification language, *Promela.*

The Petri net [3] is a mathematically rigorous, yet easy to use, graphical notation. Since the introduction of Petri nets in the early 1960's, the notation has been applied to a large number of application domains, has given rise to several different notational variants, and has been supported by a vast array of tools. This paper discusses the translation of Petri net specifications into Promela and the analysis of those specifications using the Spin model checker. Specifically, we describe the use of a simple set of translation rules for converting the DOME Petri net representation into semantically equivalent Promela.

The remainder of this paper is organized as follows. The technique used to translate Petri nets into corresponding Promela specifications is introduced in Section 2. Related work is described in Section 3 while Section 4 draws conclusions and suggests further investigations.

2 Translation Rules

This section describes the rules that are used to translate Petri nets into corresponding Promela specifications for use within Spin.

2.1 Overview

Table 1 provides a summary of Promela, the input language to Spin. In addition to the constructs shown in the table, we make extensive use of the *atomic* construct, which ensures that statements contained within the construct are executed as an atomic unit, and the *len(chanName)* expression, which allows for checking the number of messages contained on a channel. In the process for translating Petri nets into Promela specifications, the focus is a transition-centered one. That is, our approach for translating Petri nets into Promela specifications is based on the semantics of the transition. From this perspective, the mapping of Petri net constructs into Promela constructs proceeds as follows. Petri net transitions are mapped to Promela processes. Petri net places are mapped to Promela channels. Petri net markings and tokens are mapped to messages that are written to channels. Finally, the initial marking is modeled using the Spin `init` process with a token written to each of the appropriate places (channels). One of the advantages of using this basic scheme is that the resulting Promela models retain the same basic structure of the original Petri net. Thus, during analysis activities such as simulation, the behavior of Petri net constructs including transition firings and movement of tokens can be easily observed and related back to the original models.

† Contact Author

‡ This author supported in part by NASA Langley Research Grant NAG 1-2241.

0-7695-1426-X/01 $17.00 © 2001 IEEE

Construct Name	Format
Iteration	`do :: (cond0) -> stmt0; :: (cond1) -> stmt1; ... od`
Alternation	`if :: (cond0) -> stmt0; :: (cond1) -> stmt1; ... fi`
Assignment	`var = expr`
Channel read	`Mychan?var`
Channel write	`Mychan!expr`
Label	`label:`
Process	`active proctype pname() { ... }`

Table 1 Promela Summary

2.1.1 Transition Semantics

In order to properly model a Petri net transition in Promela, two primary issues must be addressed. First, it is necessary to be able to determine when each transition is enabled. Second, it is necessary for the firing of a transition to be performed as an atomic activity.

In the first case, since a transition may depend on several places and a place may be an input to several transitions, tokens must only be consumed from channels when all input places contain tokens. For instance, consider the Petri net shown in Figure 1. The Petri net indicates a situation where place P1 can provide a token to either T0 or T1 but not both. As such, the enabling and subsequent firing of T1 depends entirely upon whether or not P2 eventually receives a token. Figure 2 shows a translation of the transition T1 from Figure 1. In this translation, assuming that there is a token at P1 and none at P2, transition T1 is free to attempt to read from channel (place) P1. Since there is no token at P2, transition T1 blocks and will only proceed if place P2 receives a token. In the meantime, P0 may receive a token, which would have enabled T0. However, since the token at P1 has already been consumed, both transitions T0 and T1 are improperly blocked.

To avoid the situation described above, it is necessary to ensure that the contents of each channel are known so that transitions are not prematurely fired. To do so, we used the Promela *len* function to determine that the contents of each of the inputs to a transition do indeed

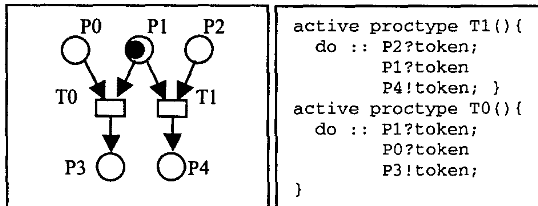

Figure 1 (a) Subset of Petri Net (b) Basic Translation

```
active proctype T1 () {
   do :: ((len(P1) > 0) && (len(P2) > 0)) ->
         P2?token; P1?token
         P4!token;
}
active proctype T0 () {
   do :: ((len(P1) > 0) && (len(P0) > 0)) ->
         P1?token; P0?token
         P3!token;
}
```

Figure 3 Bad Translation - Lack of Atomicity

contain tokens before commencing a transition firing.

The second issue of atomicity is addressed as follows. Once a firing has commenced, reading from input channels (places) and writing to output channels (places) must be completed contiguously. As an example to motivate this requirement, consider again Figure 1 with the modification that all places currently hold tokens. If the translation from Figure 3 is used, then the situation arises whereby transition T1 can begin to fire, but does not complete before channel P1 is read. At this point transition T0 begins and consumes the token at P1. T1, upon continuing, will block for lack of a token on channel P1. Figure 4 shows the corrected translation that addresses both issues and shows the general form of the transitions as translated into Promela.

2.1.2 Properties of Petri Nets

There are several fundamental properties that are interesting with respect to a Petri net including *liveness*, *boundedness*, and *fairness*. For definitions of these concepts, we refer the reader to [6]. Support for identification of the aforementioned properties in Petri nets is achieved by taking advantage of the capabilities of the Spin model checker as well as the built-in features of the Promela language. For instance, in order to determine liveness, the model checker can be invoked to determine the existence of deadlocks, invalid end states and unreachable code. With respect to boundedness and safeness, the explicit definition of channel sizes in Promela to some size (either *k* for *k-bounded* nets or 1 for a safe net) can be coupled with the model checker to verify this property. Finally, by using counters and safety assertions in a Promela specification, the property of fairness can be readily identified.

```
active proctype T1 () {
   do :: atomic{
         ((len(P1) > 0) && (len(P2) > 0)) ->
         P2?token; P1?token; P4!token;
         }
}
active proctype T0 () {
   do :: atomic{
         ((len(P1) > 0) && (len(P0) > 0)) ->
         P1?token; P0?token; P3!token;
         }
}
```

Figure 4 Corrected Translation

Case	ENABLE_TEST	READ_INPUT_TOKENS	WRITE_OUTPUT_TOKENS
One Input, One output	`(len(ip0) > 0)`	`ip0?token;`	`op0!token;`
Many inputs, One output	`((len(ip0) > 0) && … && (len(ipn) > 0))`	`ip0?token; … ipn?token;`	`op0!token;`
One input, Many outputs	`(len(ip0) > 0)`	`ip0?token; … ipn?token;`	`op0!token; … opm!token;`
Many inputs, Many outputs	`((len(ip0) > 0) && … && (len(ipn) > 0))`	`ip0?token; … ipn?token;`	`op0!token; … opm!token;`

Table 2 Translations by case

2.2 Cases

Figure 5 depicts, graphically, each of the different cases that exist for the translation of Petri nets into Promela specifications. Each of the different cases is described in detail and an example sequence of Promela code provided. As per the discussion in Section 2.1, each of these cases ensures that transitions are fired only after enabled and all the necessary actions are performed to complete a firing once it has commenced.

The general form of the body for a Promela specification of a Petri net transition is as follows:

```
do :: atomic{
        ENABLE_TEST -> READ_INPUT_TOKENS;
                        WRITE_OUTPUT_TOKENS; }
od;
```

where ENABLE_TEST is a placeholder for the code necessary to determine if a transition is enabled, READ_INPUT_TOKENS is the placeholder necessary for beginning a transition firing, and WRITE_OUTPUT_TOKENS is the placeholder for the code necessary to complete a transition firing. Table 2 gives the summaries for the translations for each of the given cases shown in Figure 4, where `ip` denotes an input channel, `op` indicates an output channel, and the digit 0 is an enumeration index. It must be noted that since we are discussing each transition type in an isolated context, that this simple enumeration of input and output channels is sufficient. In practice, the index of the channels is based on the global names of the given places and thus would be named according to the topology and labeling of a given Petri net.

2.2.1 Shared Inputs and Shared Outputs

Figures 5(e) and (f) depict two special cases that require special mention. In the first case, the two transitions *conflict* in the sense that they share at least one input. While the translation of this case requires no special handling, it is worth noting that since the actions of checking for an enabled transition, reading tokens from input places, and writing tokens to output places all occur as a single atomic action, a token involved in the conflict will only be consumed by a single transition.

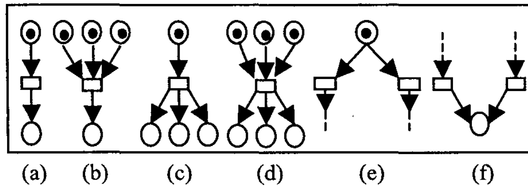

Figure 5 Translation Cases

Similarly the case depicted in Figure 5(f) represents a special case where a single place can potentially receive tokens from several transitions. Again, while no special treatment is necessary to handle this case it is interesting to note that if the number of tokens in the target place reaches a designated size, the boundedness of the model can potentially come into question, thus providing valuable information regarding the Petri net.

2.3 Examples and Validation

Figure 6(a)-(g) shows several Petri nets whose boundedness and liveness properties are well-known [4]. Figure 6(h) shows the Petri net for Peterson's mutual exclusion algorithm. Of these eight examples, four are bounded, four and unbounded, four are live and four are non-live. Table 3 gives the summary of the analysis of each of the Petri nets. Of the four models in the test suite that were found to be non-live, Spin was able to identify three of them correctly via the detection of unreachable code (by way of assertion) while the last non-live model was identified through an invalid end state. None of the remaining cases had unreachable code or invalid end states and thus were deemed to be live.

With respect to boundedness, those models that were bounded were also found to be safe. The unbounded models were correctly identified via assertions that checked the size of each channel in the models.

3 Related Work

Several Petri net tools have been suggested including systems such as Design/CPN [5] and PEP [6]. The primary difference between these tools and the work presented in this paper is that while the main goal of the aforementioned tools are to explicitly support modeling

Petri net	Bound	Live	Liveness Method
(a)	Y	Y	NA
(b)	N	N	Assertion
(c)	N	Y	NA
(d)	Y	N	Assertion
(e)	Y	N	End-state label
(f)	N	Y	NA
(g)	N	N	Assertion
(h)	Y	Y	NA

Table 3 Summary of Spin Analysis

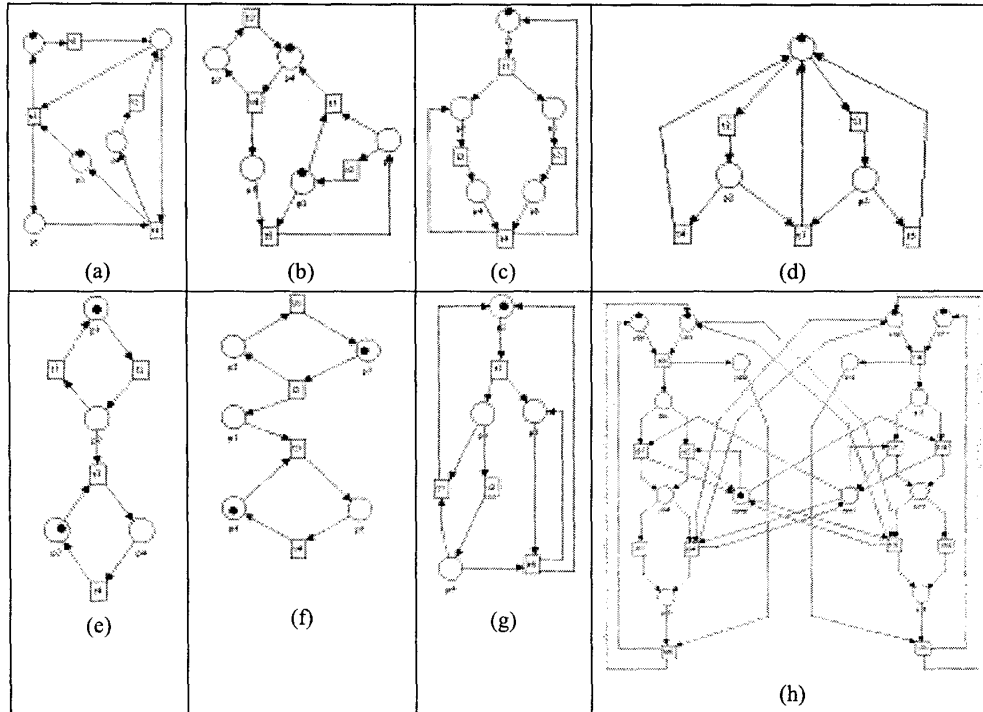

Figure 6 Eight Petri Nets with known Liveness and Boundedness Properties, B – Bounded, $\underline{B}$ – Unbounded, L – Live, $\underline{L}$ – Non-live [4]. (a)BL. (b) $\underline{B}\underline{L}$. (c) $\underline{B}$L. (d) B$\underline{L}$. (e) B$\underline{L}$. (f) $\underline{B}$L. (g) $\underline{B}\underline{L}$. (h) BL.

and analysis of Petri nets in a seamless environment, our work is intended to provide a medium for integrating a modeling environment (DOME) with an analysis environment (Spin) while demonstrating a lightweight a means for gaining access to many services seen in most Petri net tools.

4 Conclusions and Future Investigations

One of the advantages of the Spin model checking system is that it supports the use of a lightweight method in that the amount of manual or guided effort required during the verification phase is relatively low when compared to standard theorem provers. Another advantage of Spin is that its input language, Promela, is a simple notation that can be used to easily model various specifications and specification methods.

In this paper, translation rules and a tool for supporting the analysis of Petri net models using the Spin model checker were described. We have found that by integrating a convenient modeling environment like DOME with the Spin model checker that we have been able to support Petri net analysis with a small amount of development overhead. Several improvements to the tool described in this paper are currently envisioned including investigating how the output from Spin can be used to animate Petri net execution within the DOME environment. In addition, we are interested in exploring how to apply the use of Spin to verify a wider variety of Petri net properties. Finally, we are developing several other translators in order to support the use of the Spin model checker to verify other notations supported by DOME including UML statecharts.

References

[1] *Dome Guide version 5.2.2,* Honeywell Corp., 1999.

[2] G. Holzmann, *"The Model Checker Spin,"* IEEE Transactions on Software Engineering, (23)5, May 1997.

[3] C.A. Petri, "Fundamentals of a theory of asynchronous information flow", In Proc. of the IFIP Congress 62, pp. 386-390, 1963.

[4] T. Murata, "Petri Nets: Properties, Analysis and Applications," Proc. of the IEEE, (77)4, pp. 541-580, April 1989.

[5] Design/CPN Ref. Manual, Meta Soft Corporation, 1992.

[6] B. Grahlmann and C. Pohl, "Profiting from Spin in PEP", Proceedings of the SPIN'98 Workshop, Nov. 1998.