# S2N: Model Transformation from SPIN to NuSMV (Tool Paper)

Jiang Yong and Qiu Zongyan

School of Mathematical Sciences, Peking University, Beijing 100871, China
yongjiang043@gmail.com, qzy@math.pku.edu.cn

**Abstract.** SPIN and NuSMV are the two most widely used model checkers. It makes sense to have a translator from SPIN models to NuSMV models, then users can have more choices to build their models and check related properties. Here we describe a tool named S2N which forms a bridge from SPIN to NuSMV.

## 1 Introduction

Model checking [1] is an important verification method, and people have developed many model checkers. SPIN and NuSMV are two of the most widely used model checkers. SPIN [2] provides a higher-level modeling language Promela with a C-like syntax, which makes models easy to build and read. NuSMV [3] provides a relative lower language for describing state transition systems. On the other hand, SPIN supports only LTL to describe properties, while NuSMV can check LTL properties, also CTL properties.

SPIN and NuSMV are impressive and successful in their respective fields. Making their advantage merged will be convenient for modeling in some extent. Here we present an in-developing tool S2N which can translate models in Promela to the language of NuSMV. In this case we get multifold benefits: on one side, modeling complicated systems in NuSMV is hard and error-prone, but with S2N we could have a higher-level language for the work; on the other side, S2N works as it extends SPIN system with the ability to check CTL and other properties NuSMV supports. [5] reported a similar idea, however, the project seems abandoned without producing a real tool. Now we have a preliminary version of S2N, which and its sources are available from `http://code.google.com/p/s2n`. We have carried out some experiments which indicate that for some problems SPIN is more efficient, but sometimes NuSMV works better. The link between the two checkers appears meaningful. Due to the space, we leave more details in a report on the S2N's open-source site.

## 2 Translation Techniques

We use the idea of program counter during the translation. Assume $S$ is a Promela program and $N$ is the resulting program of the translation. We mark statements of $S$ with different integers and use a variable $pc$ in $N$ running over the integers. States in $N$ are changing only with the change of $pc$. The key is making sure the trace of $pc$ is consistent with the flow of $S$. S2N translates processes in $S$ to asynchronous module

instantiations in $N$. There is a top module in $N$ whose variables are defined according to the global variables in $S$. Moreover, we add modules to simulate channels in $S$, and the channel IO becomes the state transition of these modules. S2N produes a set of nested modules with a top module. Each module consists of three parts: variable declaration, initialization and transition, and a constraint expression:

```
MODULE name
  VAR
    ...         -- variable declarations of this module
    pc : ...    -- program counter of associated process
  ASSIGN
    init(...) := ...
    ...         -- initialization of variables in this module
    next(...) := ...
    ...         -- transition of variables
  TRANS
    ...         -- constraints about pc and scheduling
```

*Marking.* Every process in Promela program could be thought as an independent state transition system, so we mark these processes respectively. We mark every elementary statements, including assignment, expression, channel IO, goto statement and break statement. And for nondeterminism, there should be marks before if-selection and do-repetition.

Here is a simple proctype with marks, in which we use the identifiers of the form "M+$mark$" for the marks and separate them from the statements with a '@' sign.

```
mtype = {p1, p2, p3};
chan queue = [3]of{mtype};
active proctype enqueue () {
    mtype ele;
    M1  @ do
          :: M2  @ if
                   :: M3  @ ele = p1;
                   :: M4  @ ele = p2;
                   :: M5  @ ele = p3;
                   fi;
             M6  @ queue ! ele;
          od;
}
```

*Variable Declaration.* We translate the global variables in the Promela program to variables in main module of the resulting NuSMV program, and the local variables to variables in the corresponding module. Each NuSMV module from a process will have an another variable $pc$. For the types, `bit` and `bool` in Promela are translated to boolean type in NuSMV; `mtypes` are naturally translated to enumeration type; we translate `pid`, `byte`, `short`, `int` and `unsigned` all to signed word types. Channel variables are translated to instantiations of some module standing for the channel type.

There are some constraints about types in S2N. Boolean and enumeration type are isolated with other types, thus boolean or enumeration variables can be operated only with variables of the same type.

*Initialization and Transition.* In the first state all variables should be initialized. The state transition happens only in assignments and channel IOs, while the other statements have no effect on variables. Every transition has at least one condition which is the variable $pc$ should equal to the mark of the corresponding statement.

As the example above, part of the translation of variable $ele$ will be

```
init(ele) := p1;
next(ele) := case
               pc = 3 : p1;
               pc = 4 : p2;
               pc = 5 : p3;
               ...
             esac;
```

*Constraint Expression.* The constraint expression usually has the shape as follows:

```
  (  running & (
        pc = ... & ... & next(pc) = ...
      ...
      | pc = ... & ... & next(pc) = ...
     )
   | !running & next(pc) = pc ) -- constraint of program counter
& (  pc = ... & ...
   | pc = ... & ... & next(pc) = ...
   ...
  ) -> running
& (  pc = ... & ...
   | pc = ... & ... & next(pc) = ...
   ...
  ) -> !running             -- constraint of scheduling
```

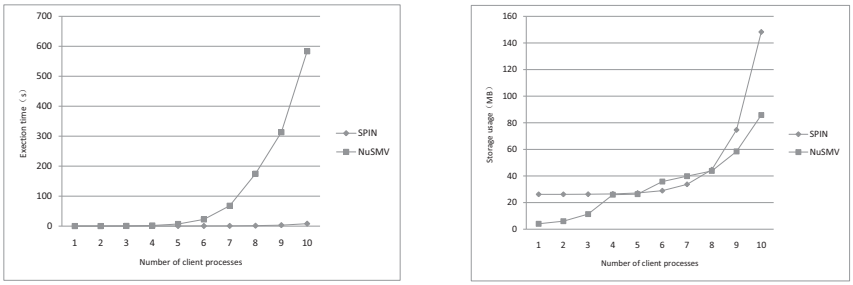
The expression is a conjunction of three literals. The first defines the transition of variable $pc$ while the rest is for scheduling. The transition rules for $pc$ are built based on the flow of the Promela program. When current module is not running, $pc$ keeps no change. For atomic sequences in SPIN , the output module should keep running among the sequences as atomic steps. On contrary, modules should not be scheduled to run in some cases, like when the current statement is not executable.

## 3 Tool and Experiments

### 3.1 Features supported

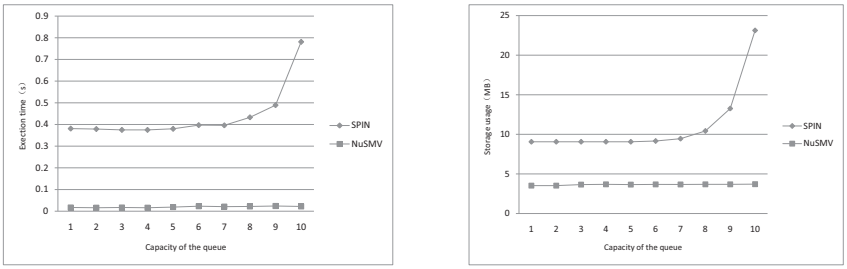
Currently S2N has supported most features of Promela. It translates one `proctype` at a time, so the complexity of translation is linear to the number of `proctype`s in SPIN model. Features as follows are supported by current version.

**Types and Variables.** S2N supports all data types in Promela. Channels can be either global or local, but they could not be passed as parameters or message data in other channels. In other words, users of S2N should declare channels with initialization and only use them to send and receive message. S2N supports rendezvous communication.




Execution Time Memory Usage

**Fig. 1.** Execution with different numbers of clients in Sleep-Wakeup Process Scheduling




Execution Time Memory Usage

**Fig. 2.** Execution with different capacities in Queue

**Expressions.** S2N supports the usual boolean and arithmetic operations, const, variable reference, and operations about channels including `empty`, `nempty`, `full`, `nfull`, `len`. S2N also supports Dynamic processes by `run`.

**Statements.** S2N supports assignment, standard channel IO, expression, `break`, `goto`, `skip`, as well as flow controlling including selection, repetition, sequence, labeling, `atomic`, and `unless`. It also supports weak `d_step`, which has the same affect with `atomic` except that it can't escape from the sequence of `d_step` in `unless`.

**Others.** S2N supports C-style macros and `init`, while `inline`, `_pid` are not supported yet. The array indexes must be numeric constants. S2N does not support translation of the property claims yet, so users should add manually their properties to the NuSMV model.

### 3.2 Experiments

We have done many experiments. Here we give two examples with statistic data to show the usage and features of S2N. We take the examples from existing SPIN work.

*Sleep-Wakeup Process Scheduling.* The algorithm was proposed by Ruane [4]. In implementation [6], several client processes are going to compete some resource, and one server takes responsible for waking up sleep processes when the resource is available. We transform this model to NuSMV model, and check the property that the resource is accessed by at most one process all the time. The statistics is depicted in Fig. 1.

In this example, SPIN is faster than NuSMV from seven clients onwards, and the memory usage of NuSMV increases slower than SPIN from eight clients onwards. This is probably because of SPIN 's on-the-fly and partial order reduction strategy, and NuSMV stores states with BDD trading memory use for the run time. It shows that for small models, NuSMV model transformed from S2N could work as well as SPIN .

*Queue.* Queue is a widely used data structure. To model a queue with some fixed capacity, it is not easy using NuSMV as we have to consider the move of elements after dequeue or enqueue. However, this is convenient using SPIN as its channel feature. Moreover, with the C-style macro we can change the queue's capacity without rebuild the model. We implement in SPIN a queue model and transform it to NuSMV model using S2N, and compare the building time and the memory usage in two systems. The results are depicted in Fig. 2.

As shown, NuSMV is much faster and consumes less memory than SPIN in building the same model. For bigger queues, the differences become even large. This is because SPIN has to traverse the states resulting the DFS stack takes up more memory set, though nothing checked here. By contrast, the NuSMV's BDD reduces the storage. This example shows that our translation for channels to module is efficient.

## 4 Conclusion and Future Work

This paper is not intend to entail that SPIN can be replaced by NuSMV. Instead, S2N provides a method for users to choose the right model checker for their problems. There are two cases where S2N may help: (1) If we build an SPIN model and want to check some properties that is beyond the function of SPIN but within NuSMV's domain. (2) If the model is difficult to build in NuSMV, we can consider building the model using SPIN 's modeling language Promela first.

S2N can be improved in several ways. One of the future work is to extend S2N to support the remaining features of Promela, such as specifications. In fact, S2N is similar as compiler from a high-level language to a targeting low-level language. There should be many possibilities for optimizations. In addition, there are still many work on exploring the language design and implementation techniques for the formal modeling, as the cases in the programming world.

## References


1. Stephan Merz. Model checking: A tutorial overview. In Modeling and Verification of Parallel Processes. Lecture Notes in Computer Science, Vol. 2067, 3–38. Springer-Verlag, 2001.
2. Gerard J.Holzmann. The SPIN Model Checker: Primer and Reference Manual. Addison-Wesley Professional, September 2003.
3. NuSMV tutorial. http://nusmv.fbk.eu/NuSMV/tutorial/index.html.
4. L.M. Ruane. Process synchronization in the UTS kernel. Computing Systems, Vol. 3, No. 3, 387–421. Usenix 1990.
5. Michael Baldamus1, Jochen Schröder-Babo. p2b: A translation utility for linking promela and symbolic model checking (tool paper). In LNCS, Vol. 2057, 183–191. Springer 2001.
6. Ki-Seok Bang, Jin-Young Choi, Chuck Yoo. Comments on "The Model Checker SPIN ". IEEE transactions on software engineering, Vol. 27, No. 6, 573–576, 2001.

## Presentation Plan

We plan to have three parts in the presentation. First, we give a brief review of SPIN and NuSMV with their features, leading to the motivation of our work. Second, we plan to give an overview of the idea and procedure of the transformation, and lastly a demo of the tool S2N with some examples.

1. Motivation (2 minutes)
   In this part, we will introduce briefly NuSMV, tell the difference between SPIN and NuSMV, and the advantage of the tool S2N.
2. Transformation Techniques (8 minutes)
   In this part we present the basic idea of the S2N work, and some main rules of translation from Promela to the language of NuSMV.
3. Tool S2N Demonstration (5-10 minutes)
   We will demonstrate the tool S2N in this part, including its usage and supported features. Of course some examples will be shown to explain the second part and indicate how well the tool works.