Simple Promela Verification Model Translation Method based on Relative SysML State Machine Diagrams

Bo Wang\(^1\), Takahiro Ando\(^1\), Kenji Hisazumi\(^2\), Weiqiang Kong\(^3\), Akira Fukuda\(^1\)
Yasutaka Michiura\(^4\), Keita Sakemi\(^4\), Michihiro Matsumoto\(^4\)

\(^{1}\)Graduate School/Faculty of Information Science and Electrical Engineering, Kyushu University, Fukuoka, Japan
\(^{2}\)System LSI Research Center, Kyushu University, Fukuoka, Japan
\(^{3}\)School of Software, Dalian University of Technology, Dalian, China
\(^{4}\)Japan Manned Space Systems Corporation, Tsukuba, Ibaraki, Japan

Abstract—In this study, we developed a method for converting relative SysML state machine diagrams into Promela models that can be verified using the SPIN model checking tool. When analyzing a system, it is common for there to be plural state machines in the system. Some of these state machines are complex interrelated. Our goal was to develop a method to translate these related state machine diagrams to Promela models that show the communication between state machines. This study was based on the translation method we developed previously. In this study, we improved the method so that it could not only show the influence of outside events on the system, but also the influence of inside messages between the related state machines. In this paper, we describe the rules to convert the related SysML state machine diagrams to a simple verification model in Promela.

Keywords: State Machine Diagrams, SysML, SPIN, Model Checking, Translation Method

1. Introduction

With the development of information and communications technologies, software systems have become much more complex and embedded in all aspects of life, such as smartphones, automobiles, home automation, and space equipment. The more complex the embedded systems become, the more reliability is desired. Therefore, rigorous formal verifications are required during each stage of system development.

Model checking [1] is a well-known formal verification technique to check the correctness of software designs. SPIN [2], NuSMV [3], and UPPAAL [4] are the most widely used model checkers. In model checking, one must build a model in the formal description languages corresponding to the desired model checker. The models written in description language are executable. The behaviors of the system can be simulated and exhaustively explored by the model checker to verify whether the system meets the required properties.

Costs and verification process efficiency are the primary issues when model checking techniques are used to verify the target system. During the early stages of the verification process, building a simple model for limited range verification can be a great way to reduce verification costs. Moreover, the quality of the model is also important because it affects the efficiency of the verification processes. However, in traditional model checking, the quality of the model depends on the verification engineer’s skill, which is often problematic. Therefore, automatic verification model generation independent of the verification engineer’s skill level is desired. Additionally, because analyzing the verification results tend to be complex, a simplified method for this process is needed.

In this study, we improved the automatic generation of simple verification models from a SysML state machine diagram [5] to a SPIN model we developed previously [6]. In particular, we expanded the translation objects from one single SysML state machine diagram to plural relative state machine diagrams, and proposed to show the interactions between related state machines. Using our translation method, not only the behaviors inside each state machine diagram, but also the information of interactions between each relative state machine can be translated into a simple verification model in Promela, the input language used by SPIN. We demonstrated the feasibility of our method by applying it to verify a simple system.

2. Related Work

Much research has been conducted on the application of formal verification techniques to analyze state machine diagrams. Bhaduri and Ramesh [7] carried out a comprehensive survey of studies that applied model checking to state machines, in which various model checkers including SPIN [2], SMV [8], and FDR [9] were used.

We previously developed an automatically generated simple verification model from a SysML state machine diagram to a SPIN model [6]. Using the translation method, the state machine diagram could be translated into a simple verification model that could show the behavior of the state machine. However, the converting object of the translation method
was a single state machine diagram. Thus, the verification model could not show the interactions between relative state machines. Consequently, to verify a system with plural relative state machine diagrams, we had to translate the diagrams individually and verify each simple verification model repeatedly. Moreover, we could not translate the interactions between the state machines, as the verification models did not address them.

3. Proposed Translation Method

In this section, we describe our translation method from relative SysML state machine diagrams to the verification model. The verification model in Promela was written as simple processes that include the behaviors in state machines and the interaction information between related state machines. In this way, the verification model runs like a system rather than separate components. We used a macro description to express each element of the model at the same time in order to simply recognize the correspondence between the elements of the state machine diagram.

In our verification model, each relative state machine runs in parallel, but the parallel behaviors in the state machine are transformed into one sequential process in Promela. Although our verification model cannot express all the behaviors of the diagram with parallel processes, it could prevent state explosion in the early verification stage and reduce verification cost.

Furthermore, we allowed the event reception channels in each state machine to receive the trigger events in order to show the communication status of each state machine. We defined two channels for each state machine in the verification model: one channel only receives input events outside the system and the other channel only receives interact signals from other state machines inside the system. The types of received messages by the two channels are shown in Fig. 1. Thus, we can easily recognize the communication status of each state machine.

3.1 Overview

The inputs to our translation method are the SysML state machine diagrams, the information about their variables, and the event-channel table. Information about the SysML state machines is stored in an XMI file, and information about their variables is stored in a CSV file. The event-channel table expresses the correspondence between each event and channel. The translation rules shown below use two different channels to receive interact signals and outside events and translate elements of the original diagrams into their corresponding description in Promela.

1) The state names in the diagram are translated into ntype values in Promela.
2) An ntype variable that represents the current state is provided for each region in the diagram.
3) The event names in the diagram are translated into ntype values.
4) The channels for receiving input events and interact signals are provided as channel variables for each state machine.
5) Based on the input variable information, the corresponding variables in Promela are provided for the variables in the statement or guard condition in the diagram.
6) The initial pseudo state is translated into an inline macro description.
7) The states, including composite states, are translated into three types of inline macros.
8) The outgoing transitions of a state are translated into one inline macro.
9) The regions are translated into two types of inline macros.
10) The final state in each region is translated into an inline macro.
11) The event occurrence model is described as an inline macro.
12) The behavior of the original state machine is described by one process in Promela.

In the following subsections, we explain our translation rules in detail along with translation examples. Each of the following examples were obtained by translating the corresponding element of the two relative state machine diagrams shown in Fig. 2.

3.2 Declaration of Values and Variables

Each state name is translated into an mtype value in Promela (lines 1–12).

Each variable representing the current state in a state machine is declared an mtype variable (lines 13–16).

Each variable in guard conditions and actions in the state machine diagram is translated into a variable in Promela using the variables information in the CSV file (lines 17–20).

```
#controller_top_init, controller_top_autocruise, controller_top_goal, controller_top_emergency, controller_top_final;
#motor_top_init, motor_top_idle, motor_top_rotation, motor_top_overheat, motor_top_final;

mtype = (controller_top_init,
controller_top_autocruise,
controller_top_goal,
controller_top_emergency,
controller_top_final);

mtype = (motor_top_init,
motor_top_idle,
motor_top_rotation,
motor_top_overheat,
motor_top_final);

mtype controller_topState = controller_top_init;
mtype motorTopState = motor_top_init;

int x = 0;
int x_MAX = 100;
int temp = 20;
int temp_MAX = 80;
```

3.3 Declaration of Events and Channels

Table 1 is an event-channel table for the example system (Fig. 2).

<table>
<thead>
<tr>
<th>Event</th>
<th>Channel</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>go</td>
<td>cinput_event</td>
<td>outside</td>
</tr>
<tr>
<td>turnoff</td>
<td>cinput_event</td>
<td>outside</td>
</tr>
<tr>
<td>emergency_stop</td>
<td>cinput_event</td>
<td>outside</td>
</tr>
<tr>
<td>off</td>
<td>m_event</td>
<td>controller</td>
</tr>
<tr>
<td>stop</td>
<td>m_event</td>
<td>controller</td>
</tr>
<tr>
<td>rotate</td>
<td>m_event</td>
<td>controller</td>
</tr>
<tr>
<td>overheat</td>
<td>c_event</td>
<td>motor</td>
</tr>
</tbody>
</table>

According to the event-channel table, each input event is translated into an mtype value in Promela (line 1). In order to receive input events, the corresponding channels are declared as a channel variable in Promela for each state machine (lines 2–3).

Each interact signal to the state machines is translated into an mtype value in Promela (lines 4, 7). The channel for receiving interact signals for each state machine is declared as a channel variable in Promela (lines 5, 8).

In order to make a distinction between the interact signals and input events for each state machine. The channels use different nomenclatures. Additionally, we used a 0 size channel in order to preserve the synchronization of each process.

```
mtype = (go, turnoff, emergency_stop);
chan cinput_event = [0] of { mtype };
mtype = (rotate, stop, off );
chan m_event = [0] of { mtype };
mtype = (overheat );
chan c_event = [0] of { mtype };
```

3.4 Translation of Initial Pseudo States

A initial pseudo state points to the state which a state machine starts from. In each state machine, initial pseudo states are translated into inline macros, where each has an inline macro call for the state entry behavior.

```
inline T_controller_top_init() {
S_controller_top_start_entry()
}
```

3.5 Translation of States

Each state is divided and translated into three inline macros: entry part (lines 1–5), do part (lines 6–9), and exit part (lines 10–12). The entry part represents the entry behavior of the state, the do part represents the do behavior, and the exit part represents the exit behavior. In addition, the behavior of the sending signals may be included in each part of the state and the action part of the transition (line 3). The signals become trigger events for the other state machines.

```
inline S_motor_top_overheat_entry() {
    motorTopState = motor_top_overheat;
c_event!overheat
}
inline S_motor_top_overheat() {
skip
}
inline S_motor_top_overheat_exit() {
skip
}
```
3.6 Translation of Outgoing Transitions

Our transition rules translate all outgoing transitions of a state into an inline macro in a group. The internal transitions are dealt with in a manner similar to the outgoing transitions and they are translated into the same macro as outgoing transitions. In this macro, the transitions are described as conditional branches by the trigger event occurrences. The inline macro call of the source state exit behavior is placed before the actions of each transition. The inline macro call of the target state entry behavior is placed after the actions. Different channels can exist in the same inline macro in order to receive different trigger events from different sources (lines 10, 19). These behaviors are obtained from the semantics of the state machine diagrams.

```plaintext
:: (motor_topState == motor_top_rotation) ->
S_motor_top_rotation();
T_motor_top_rotation();
```

3.8 Translation of Final State

The final state is translated into an inline macro that terminates the operations of the behavior in the region.

```plaintext
inline S_controller_top_final_entry() {
controller_topState = controller_top_final
}
```

3.9 The Event Occurrence Model

In our verification model, we adopted the following input event occurrence model for each state machine. In the model, input events that can occur are sent to each corresponding channel as triggers for state machines. When the state machine reaches the final state, the event occurrence model terminates.

```plaintext
active proctype cinput_eventOccur() {
do
:: cinput_event!go
:: cinput_event!turnoff
:: cinput_event!emergency_stop
:: (controller_topState == controller_top_final) -> break
od
```

3.10 Process for State Machine Behavior

In our Promela verification model, only one process is present in each state machine. This process represents the overall operation of the original state machine diagram. It starts from the inline macro call of the initial pseudo state and executes the transitions and behaviors in the region.
active proctype controller_stm() {
    T_controller_top_init();
    do
        :: R_controller_top()
    od
}

4. Case Study

In this section, we confirm the feasibility of our translation method for SPIN model checking. We applied our translation method to the related state machine diagrams shown in Fig. 2 and generated the corresponding Promela code. The interaction information of each state machine is implemented in the Promela code. Moreover, LTL verification can be conducted using the code as the input to the SPIN model checker. We used the following LTL formula, which indicates that the controller state will not permanently become the goal state, to verify the reachability of the Promela code.

\[ \neg (\text{controller\_topState == controller\_top\_goal}) \]

The result of this verification case study is shown in Fig. 3. The figure shows that an error has been found, which means that a transition sequence has reached the controller’s goal state. This is an appropriate result for this verification case study, showing the generated model can be used for SPIN model checking.

A counterexample for this case study is shown in Fig. 4. The green region in the figure clearly shows the interactions between each state machine. The yellow region in the figure represents the counterexample sequence. When \((x > x_{\text{MAX}})\), the controller state machine sends a "stop" signal to the motor state machine and the controller’s state becomes the goal state. The motor state machine receives the "stop" signal and the motor changes to the idle state. The red region in the figure shows the values of each variable when the sequence ends, at which time the controller has reached the goal state. Thus, we can easily observe the code corresponding the verification model to state machines. In addition, the code relating the verification model to the interactions between each process can also be observed.

These results reveal that our translation method can be useful in the model checking process.

5. Conclusions

In this study, we improved our previously proposed translation method. The translation objects have extended from a single SysML state machine to plural relative SysML state machine diagrams. Using our method, relative state machine diagrams can be converted into a simple SPIN verification model. Using inline Promela macros in our simple verification model, we can easily recognize the correspondence between Promela codes and each component of the original diagram. Moreover, we can also identify interact signals between relative state machines using channel variables in Promela. Thus, we can more easily recognize the source of each trigger event and clearly analyze the verification results. The flexibility of the translation method is also increased.

In future work, we intend to further some other representations of the message communication in the translation method. For example, we aim to provide only one channel for each state machine, receiving both input events and interact messages. This may help increase the verification process efficiency. We also plan to apply our method to various examples and refine our translation rules based on feedback attained while doing so. Moreover, we plan to develop a translation method that translates state machine diagrams into parallel verification models rather than simple sequential models. Further, we plan to develop an automatic translation system that can translate the target objects into different types of models to meet different needs.

References

Fig. 3: Result of Verification Case Study Using the SPIN Model Checker

Fig. 4: Counterexample of a Verification for the System Using the SPIN Model Checker