On Transforming Petri Net Model To Moore Machine*

Carl K. Chang and Hsuanwei Huang

Department of Electrical Engineering and Computer Science
University of Illinois at Chicago
Box 4348
Chicago, Illinois 60680

Abstract

Lacking of satisfactory verification methods in verifying the liveness and fairness properties limits the analysis power of Petri net theories. In this paper, an approach is introduced to connect the Petri net model with the Model Checker. A translator is used to transform the reachability graph of Petri net to the Moore machine. The Moore machine together with the behaviors specified by temporal logic are the inputs of Model Checker, which is able to verify the properties of liveness and fairness. During the transformation, local and global behaviors of the Petri net model are separated, which means certain modularity can be achieved. An optimization technique is presented to trim the unnecessary local information from the local reachability graphs. The space complexity of manipulating the global reachability graph, which is generated by combining the trimmed local reachability graph, can be reduced. Moreover, a new approach is proposed to verify the concurrency behavior by using the Model Checker.

1. Introduction

INTEGRAL is a Computer Aided Software Engineering (CASE) environment aimed at supporting the development of real-time distributed software systems. It is integrated because various kinds of analyses have been incorporated into a coherent environment, including system structure (control) analysis, communication analysis, timing analysis, performance analysis, and system-specific property analysis. These analyses can be performed before the coding phase so that a thoroughly verified system specification can be made available before implementation. Moreover, after code has been developed, INTEGRAL extends its power into the testing phase made available before implementation. However, after code has been developed, INTEGRAL extends its power into the testing phase made available before implementation. Moreover, after code has been developed, INTEGRAL extends its power into the testing phase made available before implementation. Moreover, after code has been developed, INTEGRAL extends its power into the testing phase made available before implementation. Moreover, after code has been developed, INTEGRAL extends its power into the testing phase. The details will be described later.

This paper is organized as follows. The transformation model is shown in Section 2. The transformation algorithms are presented in Section 3. A simple example is illustrated in Section 4. In Section 5, the new approach to verify concurrency property by MC will be discussed. Finally, the conclusion will be given at the end of this paper.

2. The Transformation Model

Moore machine is one kind of finite state machines. Different from ordinary finite state machine, Moore machine has output associated with each state. Based on the formal definition given in [5], a Moore machine is a six-tuple \((Q, \Sigma, \Delta, \lambda, q_0)\), where \(Q\) is a set of states, \(\Sigma\) is a set of input alphabets, \(\Delta\) is a set of output alphabets, \(\lambda\) is the set of transition functions, \(\lambda\) is the mapping from \(Q\) to \(\Delta\) giving the output associated with each state, and \(q_0\) is the initial state. The textual form of the Moore machine used by MC can be divided into two parts: declaration part and state transitions. The following attributes are included in the declaration part:

- Moore machine name
- number of states (number of elements in set \(Q\))
- number of arcs (number of \(\Sigma\))
- list of input propositions (\(\Sigma\))
- list of output propositions (\(\Delta\))
- list of Mealy-output propositions

The list of Mealy-output propositions is not used in our transformation process and will be neglected. The input propositions represent the stimuli from the outside world (environments) to the system and the output propositions include the responses of system and the changes of system's inner-states caused by system execution. A proposition means a declarative variable whose value is either true or false, but not both. Three values, "0" (false), "1" (true), and "X" (don't care), are used by the input propositions. The true value of a input proposition means that the corresponding input is present. The false value of input proposition means that the corresponding input is absent. The don't care condition of input proposition means that the presence of the corresponding input will not affect the current state transition. Only two values, "0" and "1", are used by the output propositions and have the same meaning as those of input propositions.

The states transition part describes the transition functions (\(\Delta\)), the state number (\(Q\)), and the output associated with each state (\(\lambda\)). Unlike the ordinary Moore machine, where only the input (output) symbol to be used (generated) is labeled on arc (state), the Moore machine used by MC labels all the input (output) propositions in each arc (state) and uses the values of propositions to discern which propositions are used. Reader can find the textual examples in Appendix A.

* This work is currently supported in part by Fujitsu America, Inc. under contract number 052-80136.
The PN model used by INTEGRAL is Extended Modified Petri Net (EMPN). Hierarchical composition is used in the EMPN model. Two parts are modeled: system kernel and environments. Ports, representing channels to facilitate communication between the system and environments, and between processes, are modeled as places. Ports can be classified as imports and exports. Detailed information of EMPN can be found in [1]. For simplicity, only a two-level hierarchy of the model will be applied in this paper. i.e. system and environments are regarded as processes which requires no further decomposition inside them.

Actually, we can regard the RG of PN model as one kind of finite state machines. RG is very similar to the Moore machine, except the information labeled on the arcs of RG are the transitions to be fired, instead of the stimulus from the environments. Based on the structure of Moore machine, the modification of the structure of RG is needed. The input and output information, which are based on the declaration of I/O propositions given by the user, must be revealed in the RG to make the transformation easier. The modified RG representation has four types of information labeled on the arc:

- the transition to be fired (T)
- the internal token received from environment (I)
- the message received from other process (R)
- the message sent to other process (S)

The second and third types correspond to the places on the imports and the fourth type corresponds to the tokens placed on the exports. Note that the third type information is not used in the global RG because it can be regarded as local information at the level of a global view. The modified RG and the corresponding Moore states are shown in Figure 1. The transformation of an arc of RG to the Moore machine is described below. For convenience, let "T", "I", "R", and "S" be also used in describing the Moore machine and represent the corresponding propositions relative to the PN model. The first and fourth states of the Moore machine correspond to the first (old-token) and second (new-token) markings in the RG. For a process, the message received from the environments and other processes can be viewed as its input. Thus the information of receiving will remain on the arc. Since the firing of a transition can be viewed as one of the changes of system's inner-states during the system execution, the firing of a transition can be viewed as two special states in the Moore machine: the transition is firing and been fired. When a transition "T" is firing in the PN model, a new Moore state (i.e. the second Moore state) is created to contain the proposition "T" with a true value. After transition "T" is fired, another Moore state (i.e. the third Moore state) is created to contain the proposition "T" with a false value. In the same Moore state, proposition "S" is set to be true, which represents the output of the process after the firing of transition "T". Since "S" will not affect the local behavior of the process, "S" will be reset to false in the fourth Moore state. Note that the "old_token" and "new_token" in RG represent the markings before and after, respectively, the firing of transition T.

3. The Translator

In this Section, the algorithms of the translator are presented. Each step is described in the following subsections.

3.1 Partitioning the System

In the EMPN model, the system itself is a process and the environments can consist of several processes. The imports and exports are the communication channels among processes. Since the name of each place is unique in the PN model, separated PN models can be obtained based on the intersection of imports and exports. Each model represents system kernel or one of the environments. In this step, the place name list and transition name list will be built. The imports and exports remain in each partitioned set.

3.2 Building the Information of I/O Propositions

Recall the model discussed in Section 2, where the input propositions represent the stimuli from the environments and the output propositions represent the responses of the system and the changes of system's inner-states. In order to build the input and output propositions, user should submit the names of places and transitions to the translator for behavior verification. The generation of Moore machine will be based on these information specified in the I/O proposition file. The format of this file is shown below:

- SYSTEM: <system name>
- COMPONENT: <process name>
- INPUT: 
  P(<place name>)
- OUTPUT: 
  P(<place name>)
- LOCAL: 
  P(<place name>)
  T(<transition name>)

In this file, keyword "INPUT" declares which imports of a process will be the input propositions, keyword "OUTPUT" declares which exports of a process will be the output propositions, and keyword "LOCAL" declares which transitions and local places will be the output propositions. <system name>, <process name>, <place name>, and <transition name> are specified by the user and must match with names specified in the EMPN model. If this file is absent, the imports, exports, and the transitions connect to the imports and exports will be the default input and output propositions.

3.3 Building the Local Reachability Graph

Breath first search and net simulation are used in the generation of the local RG. Tokens will be added to each import automatically if it can be used to enable transitions, i.e. all the input places, which are not imports, of the target transition are marked. This action can make the transition firable if it is waiting for the message from other process. The tokens in the imports will also be removed automatically after the exports have been marked. The algorithm of building the local RG is presented below:

Algorithm 3.1 Generate Local Reachability Graph

Input: Production Rules of Processes
Output: Local Reachability Graph
Method:
1. Put initial state to the OPEN LIST.
2. Repeat Steps 3 to 5 until the OPEN LIST is empty.
3. Find the transitions that can be fired based on the following firing rules.
   a. If all of the input places of a transition have tokens.
   b. If not all the input places have tokens but the places without tokens are imports.
4. Fire the enabled transitions, build the arc, and generate the new states.
5. Put the old state to the CLOSE LIST and put the new states to the OPEN LIST. Remove the tokens in the exports.

Note that deadlock of process execution and overflow of the places will be checked in Steps 3 and 4, respectively.

3.4 Building the Moore Machine of Each Process

The transformation model has been presented in the last section. In this section, a detailed algorithm is presented. In this algorithm, "label" means to label the propositions, corresponding to the transition or place names declared in the I/O proposition file, to the Moore machine by setting the proposition values to either true or false.
Algorithm 3.2 Build Moore Machine
Input: Reachability Graph
Output: Moore Machine
Method:
1. Get the marking of a state \( S \) of the reachability graph \( G \).
2. Label the state based on the marking.
3. If the input list and receive list of the outgoing arcs are not empty, label the arc based on the union of input and receive lists.
4. If the fired transition is found in the declared output propositions then create a new state which is labeled with the name of fired transition.
5. If some ports are marked after the firing of transition (the send action), then create a new state which is labeled with the marking of exports and newly generated inner-process marking.
6. Create a new state which is labeled with the marking of state \( S' \); the next state of \( S \) after firing and \( S \neq S' \) of \( G \).

After the Moore machine has been built, it will be converted to the textual form and stored in a file. Note that the Moore machine generated in this step can be used to verify the local behaviors of the corresponding process.

3.5 Optimizing the Reachability Graph

Based on the consideration of modularity, the local information of each process is not useful from the global view. A transition is a local transition if it is not connected to imports, exports, or both. A place of an EMPN model is local if it is not import or export. By filtering out the local transitions and places of each process, we can greatly reduce the state number of global RG when we combine all the local RGs together.

In the optimization process, we regard each arc, which contains the local transition, to be an \( \varepsilon \)-move. Building the closure of the \( \varepsilon \)-move, we can change the RG with \( \varepsilon \)-move to a RG without \( \varepsilon \)-move. Unlike the algorithm described in [5], which uses the input symbol to generate the new state and the new state is the union of all the reachable states of the states contained in the \( \varepsilon \)-CLOSURE, we instead use the union of the states first and then generate the new arcs and new states. This is because that each transition is unique in the PN model and the firing of a transition must have the same marking. If the states of RG have the same marking, they must be of the same state. Instead of using the input symbol as the starter to find the new states [5], we use the old state as the starter to find the new arcs and new states. The algorithm to build the optimized RG is shown below.

Algorithm 3.3 Optimize Reachability Graph
Input: Reachability Graph
Output: Optimized Reachability Graph
Method:
1. Compute the \( \varepsilon \)-CLOSURE of each state.
2. Use the initial state of the reachability graph as the initial state of the optimized state and put it in the OPEN LIST.
3. While OPEN LIST is not empty, do the following steps:
   a. Union the reachable states based on the \( \varepsilon \)-CLOSURE.
   b. Copy the arcs of the reachable states which are not \( \varepsilon \)-move to the newly generated state.
   c. Based on the non-\( \varepsilon \)-move arcs, create the new states.
   d. If the new state has the same \( \varepsilon \)-CLOSURE as the states in the OPEN LIST or CLOSE LIST, then point the arc to that state. Otherwise put the new state to the OPEN LIST.

One simple example is shown in Figure 2. In this example, \( \varepsilon \)-CLOSURE (\( S1 \)) = \( \{S1, S2, S3, S4\} \), \( \varepsilon \)-CLOSURE (\( S2 \)) = \( \{S2, S3, S4\} \), \( \varepsilon \)-CLOSURE (\( S3 \)) = \( \{S3, S4\} \), and \( \varepsilon \)-CLOSURE (\( S4 \)) = \( \{S3, S4\} \).

3.6 Generating the Global Reachability Graph

After the optimization, the local RG of each process no longer contains the local information. The generation of global RG are based on the synchronization of send and receive actions. The major difference between local RG and global RG is the marking style. The marking of global RG is based on the marking of ports, where the local RG is based on the marking of local places. Net simulation is also used in the generation of global RG. The algorithm is described in the following.

Algorithm 3.4 Generate Global Reachability Graph
Input: a Set of Local Reachability Graph
Output: Global Reachability Graph
Method:
1. Combine each initial state of the local RG as the initial state of global RG with the initial marking in each port.
2. A transition is fireable if some of the following conditions are satisfied:
   a. The transition is enabled by receiving the input from environments.
   b. The transition is a sending transition.
   c. The transition is a receiving transition and the imports are marked.
3. Fire the fireable transition and create the new state.
4. If the new state is found in the OPEN LIST or CLOSE LIST, then discard it. Otherwise, it to the OPEN LIST.

Note that deadlock and overflow are also checked in the generation of global RG (in steps 4 and 5). The generation of Moore machine from the global RG is the same as that performed to the local RG and is omitted here.

3.7 The Complexity Issue

It is well-known that the RG generation of PN model is \( \text{PSPACE Hard} \). Generating the whole RG of the system directly is very time consuming. By generating the RG of each process separately and then combining them together can save much more time and space. For example, if the system has two processes and each process has \( a \) and \( b \) transitions, respectively. The total time to generate the system reachability graph is \( a^2b^2 \). Using the algorithm presented in this paper, the time complexity becomes \( a^2b^2 \), which is much less than in the original case. In addition, the local and global behaviors can be separated and verified independently. Verifying the local behavior and the global behavior separately are much easier than verifying them all at once.

4. An Example

In this paper, we use the dining philosopher problem [4] to demonstrate the translation process. The PN model is shown in Figure 3. In this figure, the places with dotted in-arc and out-arcs are ports. The Petri net model contains three processes: a door control and two philosophers. Initially no one enters the dining room and the philosophers can request to enter the dining room. The door control may grant or deny the request, but it prohibits two philosophers entering the room at the same time.

By applying the translator presented in Section 3, the local RG of each process is shown in Figure 4, the optimized local reachability graphs are shown in Figure 5, and the global reachability graph is shown in Figure 6. The Moore machine of the door control is shown in Appendix A.

After the Moore machines of the local processes and the system are created, Model Checker can be used to check their behavior specified with temporal operators. Let the fairness constraints of the system be:

\[ T5 \& T17. \]
\[ T13 \& T17. \]
The fairness constraints state that if the philosopher issues the enter request and the dining room is available then the philosopher can get the grant admission or be denied infinity often. The following temporal formulas are used to check the safety and liveness properties.

\[ AG(\text{enter1} \land \text{enter2}), \]
\[ AG(\text{request1} \rightarrow AF(\text{deny1} \lor \text{enter1})), \]
\[ AG(\text{request1} \rightarrow EF(\text{deny1} \lor \text{enter1})). \]

The notation "AG" means that the property must hold by all the states in all the paths. The notation "AF" ("EF") means that for all (some) paths the formula must be satisfied eventually. The first formula means that the two philosophers will not enter the room at the same time. The second formula states that once philosopher1 issues the enter request, he will eventually obtain the reply from door control under all circumstances. The third formula states that once philosopher2 issues the enter request, he will obtain the reply from door control under some circumstances. Since the second formula gets the negative result, we know that the PN model of dining philosopher problem may run into starvation. This means that philosopher1 or philosopher2 may never enter the room even they request to enter it. The execution sequences of running the Model Checker is shown in Appendix B.

5. The Approach to Verifying Concurrency Property

As mentioned in Section 1, the generation of RG and the inference mechanism used with temporal-logic verification are based on execution interleaving. Execution interleaving is inexact in verifying the concurrency property of the concurrent system. During the development of our translator, we discovered an interesting new way to verify the concurrency property of the PN model by the MC. Since the MC only checks the true and false values of the input and output propositions, there is no difference to check the Moore states which contains one or two propositions having the true values. If we fire two transitions, say \( t_1 \) and \( t_2 \), concurrently, we will have a Moore state which has the propositions corresponding to \( t_1 \) and \( t_2 \) to be true at the same time. Based on this observation, we can say that the MC can be used to check the concurrency behavior of the PN model model if the RG of PN model is generated via concurrent firing. The transformation model described in this is still useful if we have a RG generated via concurrent firing. The only difference is to convert "T" into "T1 and T2" in this figure.

The algorithms in generating RG can also be modified to perform concurrent firing, where the maximum concurrent firing is applied. The maximum concurrent firing means that all the enabled transitions will be fired at once. Due to space limitations, only the rules used in concurrent firing are described in this paper. A set of transitions can be fired concurrently if and only if the following conditions are satisfied.

- \( I(t_1) \cap I(t_2) = \emptyset \).
- \( O(t_1) \cap O(t_2) = \emptyset \).
- \( I(t_1) \cap I(t_2) = \emptyset \), but the intersection place(s) have enough tokens to enable the firing of two transitions.
- \( O(t_1) \cap O(t_2) = \emptyset \), but the intersection place(s) have large enough bound to accept the tokens from these two transitions.

Since we focus the discussion on the verification of the PN properties by MC in this paper, ways of reducing the complexity caused by the generation of RG, such as applying the execution constraints, is out of the scope of this paper and can be found in [2].

6. Conclusions

The transformation algorithms which transformed the Petri net model to Moore machines have been presented in this paper. It has been implemented in C language under the UNIX operating system. Since each arc of the reachability graph is transformed to two Moore states and other Moore states are totally identical to the states in the reachability graph, the Moore machine will have identical properties held by the reachability graph of the PN model. The space complexity of creating the global reachability graph can also be reduced when the local information of the local reachability graph is removed. During the development of our translator, we found out that if we can build the concurrency behavior in the reachability graph, then the concurrency behavior of the Petri net model can be verified by Model Checker. However, the impact of discarding the interleaving inference of temporal logic is yet to be investigated. Moreover, application of our approach to the temporal-logic based inference in the automatic reasoning system is another research subject for further study.

Finally, with the rapid advances of microprocessor technology, real-time systems become more complex and more distributed. Model Checker does not have the capability to verify the timing properties of real-time systems. Currently, developing a real-time Model Checker is a major issue in our research agenda.

Appendix A
The Moore machine of door control

NAME = control;
STATES = 16;
CUBES = 20;
INPUTS = P5, P7, P13, P15;
MOORE-OUTPUTS = P20, P6, P14, P21, accept1, deny1, return1, grant1, accept2, deny2, return2;

<table>
<thead>
<tr>
<th>0</th>
<th>000000000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>000100000000</td>
</tr>
<tr>
<td>2</td>
<td>000000000000</td>
</tr>
<tr>
<td>3</td>
<td>000000010000</td>
</tr>
<tr>
<td>4</td>
<td>000000000000</td>
</tr>
<tr>
<td>5</td>
<td>000001000000</td>
</tr>
<tr>
<td>6</td>
<td>000000000000</td>
</tr>
<tr>
<td>7</td>
<td>000000010000</td>
</tr>
<tr>
<td>8</td>
<td>010000000000</td>
</tr>
<tr>
<td>9</td>
<td>000000000000</td>
</tr>
<tr>
<td>10</td>
<td>000000000000</td>
</tr>
<tr>
<td>11</td>
<td>000000000000</td>
</tr>
<tr>
<td>12</td>
<td>000000000000</td>
</tr>
<tr>
<td>13</td>
<td>000100000000</td>
</tr>
<tr>
<td>14</td>
<td>000000010000</td>
</tr>
<tr>
<td>15</td>
<td>000000000000</td>
</tr>
<tr>
<td>END</td>
<td>000000000000</td>
</tr>
</tbody>
</table>

Appendix B
The Verification of Philosopher Problem

% mcb dining.fsm
CTL MODEL CHECKER (version B1.0)
Reading dining...
Fairness constraint: 'I(P5 & P7).
Fairness constraint: 'I(P15 & P17).
Fairness constraint: ...
time: (777)
\[ I = AG \{ \text{enter1} & \text{enter2} \}. \]
The formula is TRUE.

time: (02)
\[ I = AG \{ \text{request} \} \rightarrow AF \{ \text{enter1 | denied1} \}. \]
The formula is FALSE.

time: (263)
\[ I = AG \{ \text{request} \} \rightarrow EF \{ \text{enter1 | denied1} \}. \]
The formula is TRUE.

time: (183)
\[ I = AG \{ \text{request} \} \rightarrow EF \{ \text{enter1 | denied1} \}. \]
The formula is TRUE.

time: (183)
\[ I = . \]
End of session.

References


(a) Modified RS of PN model

(b) Moore machine

Figure 1: The transformation model

Figure 2: Example of \(\varepsilon\)-CLOSURE optimization

Figure 3: Petri net model of Dining Philosopher Problem

Figure 4: The reachability graph of each local process
Figure 5: The optimized local reachability graphs

Figure 6: The global reachability graph