Avoidance of State Explosion Using Dependency Analysis in Model Checking Control Flow Model

Sachoun Park and Gihwon Kwon

Department of Computer Science, Kyonggi University,
San 94-6, Yiui-Dong, Youngtong-Gu, Suwon-Si, Kyonggi-Do, Korea
{sachem, khkwon}@kyonggi.ac.kr

Abstract. State explosion problem is a major huddle in model checking area. The model described in the temporal model checking is mainly control flow model. The *f*FSM is a model for describing the control flow aspects in PeaCE(Ptolemy extension as a Codesign Environment), which is a hardware/ software codesign environment to support complex embedded systems. *f*FSM, like a Statecharts, supports concurrency, hierarchy and global variables. But due to lack of their formality, we defined step semantics for this model and developed its verification tool in the previous work. In this paper, we present the model reduction technique based on dependency analysis to avoid the state explosion problem. As a result, the model, which couldn't be verified before applying the technique, is verified.

Keywords: State explosion problem, Dependency analysis, Model reduction, Model checking.

1 Introduction

Control flow model like a Finite State Machine (FSM) is widely used in specifying system behavior. The PeaCE[1] is the Hardware/software codesign environment to support complex embedded systems. The specification uses synchronous dataflow (SDF) model for computation tasks, extended FSM model for control tasks and task-level specification model for system level coordination of internal models (SDF and FSM).

The *f*FSM is another variant of Harel's Statecharts, which supports concurrency, hierarchy and internal event as Statecharts does. Also it includes global variables as memories in a system. This model is influenced from STATEMATE of i-Logix inc.[2] and the Ptolemy[3] approaches. But the formal semantics of *f*FSM was not defined. The absence of a formal semantics caused problems such as confidence for simulation, correctness of code generation, and validation of a system specification. In the previous work, in order to solve those problems we defined step semantics of *f*FSM and we developed simulation and verification tool, Stepper, by means of the formal semantics, which was defined by flatten model of *f*FSM. SMV model checker was used in verification part, so this tool had a translation module from flatten *f*FSM into input language of SMV. In our tool, to be convenient for user to check some

* This work was supported by grant No.(R01-2005-000-11120-0) from the Basic Research Program of the Korea Science & Engineering Foundation.

M. Gavrilova et al. (Eds.): ICCSA 2006, LNCS 3984, pp. 905 – 911, 2006.

important properties, those are automatically generated. According to the users' choice in the property window, one of six properties can be checked. Built-in properties are such as unused components, unreachable guards, ambiguous transitions, deadlock, divergent behaviors, and race condition violation[4].

However, because of the use of variables in *f*FSM model, the state explosion problem occurs. Therefore, in this paper, to overcome the problem, we introduce model reduction technique into the Stepper, which is focusing on components of the model that are referred to in the property to be checked via dependency analysis. This technique is sometimes kwon as *cone of influence* which syntactically decrease the size of the state transition graph. Chan showed effective results of the model checking of Statecharts model using this technique[5]. Lind-Nielsen[6] proposed dependency analysis on the HSEM(Hierarchical State Event Model) to perform the compositional model checkingWe attempted to apply these techniques to *f*FSM model to tackle the state explosion problem. Through experimental results, we show that the Stepper is improved on the scalability and give that system consisted of loosely coupled components is very effective in model checking.

The rest of the paper is structured as follows. In the next section, we overview the reduction technique so called cone-of-influence. In Section 3, we show reduction technique with dependency analysis in the control flow model. The experimental results present in section 4, and then we conclude the paper in section 5.

2 Background

Cone of influence technique attempts to decrease the size of the control flow model by focusing on the variables of the system that are referred to in the properties for model checking. In this chapter, we will summarize the cone of influence abstraction explained in [7].

Let V be the set of variables of a given synchronous circuits, which can be described by a set of equations: $v_i^{'} = f_i(V)$, for each $v_i \in V$, where f_i is a boolean function. Suppose that a set of variables $V' \subseteq V$ are of interest with respect to the required property. We want to simplify the model by referring only to these variables. However, the values of variables in V' might depend on values of variables not in V'. Therefore, we define the cone of influence C for V' and use C in order to reduce the description of the model. The cone of influence C of V' is the minimal set of variables such that

- $V' \subseteq C$
- If for some $v_l \in C$ its f_l depends on v_j, the $v_j \in C$.

We will next show that the cone of influence reduction preserves the correctness of specifications in Computation Tree Logic(CTL) if they are defined over variables (atomic propositions) in C.

Let $V = \{v_1, \cdots, v_n\}$ be a set of Boolean variables and let $M = \{S, R, S_0, L\}$ be the model of a synchronous circuit defined over V where,

$S = \{0,1\}^n$ is the set of all valuation of V.

$R = \wedge_{i=1}^{n}[v_i^{'} = f_i(V)]$.

$L(s) = \{v_i \mid s(v_i) = 1 \ \ for\ 1 \le i \le n\}$, $S_0 \subseteq S$.

Suppose we reduce the circuit with respect to the cone of influence $C = \{v_1, \cdots, v_k\}$ for some $k \le n$. The reduced model $\hat{M} = (\hat{S}, \hat{R}, \hat{S}_0, \hat{L})$ is defined by

$\hat{S} = \{0,1\}^k$ is the set of all valuations of $\{v_1, \cdots, v_k\}$

$\hat{R} = \wedge_{i=1}^{k}[v_i^{'} = f_i(V)]$

$\hat{L}(\hat{s}) = \{v_i \mid \hat{s}(v_i) = 1 \ \ for\ 1 \le i \le k\}$

$\hat{S}_0 = \{(\hat{d}_1, \cdots, \hat{d}_k \mid there\ is\ a\ state\ (d_1, \cdots, d_n)\} \in S_0\ such\ that\ \hat{d}_1 = d_1 \wedge \cdots \wedge \hat{d}_k = d_k\}$

Let $B \subseteq S \times \hat{S}$ be the relation defined as follow:

$((d_1, \cdots d_k), (\hat{d}_1, \cdots, \hat{d}_k)) \in B \Leftrightarrow d_i = \hat{d}_k \ \ for\ all\ 1 \le i \le k$

According to the proof in [7] B is a bisimulation between M and $\hat{M}$. Thus, $M \equiv \hat{M}$. As a result, we can obtain the following theorem:

Let f be a CTL formula with atomic proposition in C. Then $M \models f \Leftrightarrow \hat{M} \models f$.

3 Reduction of Control Flow Model

In this section, we explain our reduction method with below example. Figure 1 shows *f*FSM of mole game, where a player can hit the moles which move up and down after he/she inserts the coin. This game is a kind of reflex game. Whenever the player hit the risen mole, the score increase. During a time unit, presented *time* event, player can hit one mole once.

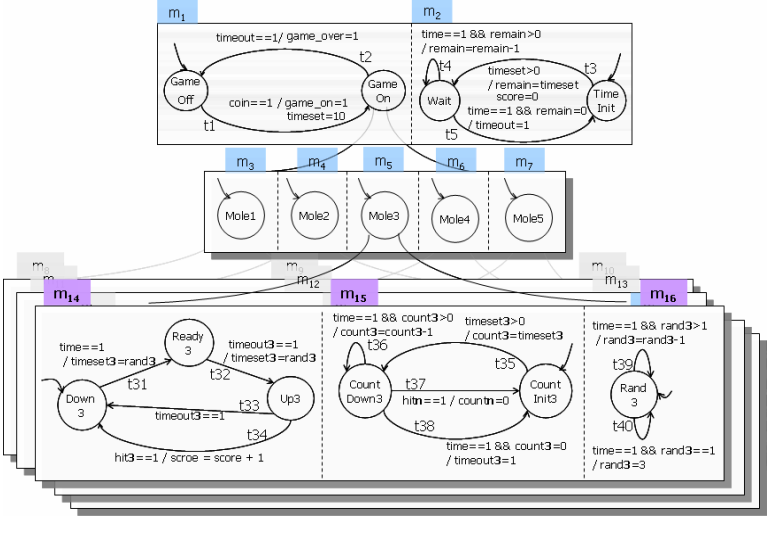

Fig. 1. *f*FSM model of mole game

3.1 Syntax of Control Flow Model

To explain the reduction technique, we define flatten machine of *f*FSM. There exit events, global variables, states, and transition. *I, O,* and *IT* are sets of input events, output events, and internal events, respectively.

Definition 1 (*f*FSM). $fFSM = (I, O, IT, M, \gamma, V)$, where *I*, *O*, *IT* are set of events, *V* is a set of global variables, and $M = \{m_1, ..., m_n\}$ is the set of *simple* FSM. Let $\Sigma = \bigcup_{i=1}^{n} S_i$ be the set of all states in *M*, hierarchical relation γ maps a state to the set of machines which belong to the state: $\gamma : \Sigma \rightarrow 2^M$.

The hierarchical function γ has three properties: there exist a unique root machine, every non-root machine has exactly one ancestor state, and the composition function contains no cycles. Let $sub : \Sigma \rightarrow 2^{\Sigma}$, and $sub(s) = \{s' | M_i \in \gamma(s) \wedge s' \in S_i\}$ is another function to relate between a super state and its sub states. sub^+ denotes the transitive closureof *sub* and *sub** denotes the reflexive transitive closure of *sub*.

Definition 2 (*Simple* FSM). $m_i = (S_i, s_i^0, T_i, scr_i)$

1. $S_i = \{s_i^0, s_i^1, ..., s_i^n\}$ is the finite set of states of m_i,
2. s_i^0 is a initial state,
3. T_i is the set of transition of m_i, and a transition $t \in T_i = (s, g, A, s')$ is composed of source and target states s, $s' \in S_i$, guarding condition g which is Boolean expr ession, and set of actions A,
4. $scr_i : S_i \rightarrow 2^{Script}$ is a function to map a set of script into a state.

Guards that include variables and events have the following minimal grammar.

$$G ::= true \mid \neg G \mid G_1 \wedge G_2 \mid e < Exp \mid e = Exp \mid v < Exp \mid v = Exp$$

$$Exp ::= n \mid v \mid Exp_1 \bullet Exp_2,$$

where n is an integer constant, $v \in V$ is a global variable, $\bullet \in \{+, -, \times, /\}$ represents a set of binary operators.

3.2 Dependency Analysis in Control Flow Model

Although this example is a small, however it has 20 events and 18 variables, which cause state explosion during model checking. So we focused the feature of this model and found out that the relationship among moles is loosely coupled. Actually if we want to check any properties about the *mole3*, there is no need to concern with other moles, because the behavior of the *mole3* is not affected by other moles. In order to verify the model reduced by means of only components referred in a property, we defined dependency between components in a given model, where the component is one of machine, event, and variable, because each component is translated by one variable in SMV input language according to translation rules mentioned in previous works[4], we can say that this reduction technique preserves the correctness of properties written CTL if they are defined over variables (atomic propositions)

corresponding to components referred in properties. In this subsection we will explain how to generate dependency graph of given *f*FSM model.

Definition 3 (Dependency Graph). Dependency graph is consist of set of nodes N and set of links L. N is a set of all components of given model and when m is a simple machine, $e \in I \cup O \cup IT$ is an event, $v \in V$ is a variable, L is a set of the dependency relation defined as follows:

1. Machine m depends on machine m' if m is defined in a sub-machine of m'. Mach ine m depends on event e or variable v if there is at least one guard in m that has a syntactic reference to e or v.
2. Event e depends on machine m if there is at least one occurring e in m. Event e d epends on event e'or variable v if e' or v is used a guard to occur e. Event e depe nds on variable v if e is used an action and there is a syntactic reference to v in th e right-hand side of the assignment of the action.
3. Variable v depends on machine m if there is at least one assignment to v in m. Va riable v depends on event e or variable v' if e or v' is used a guard to update v an d is used in the right-hand side of the assignment to v.

Below figure 2 shows a generated DG by means of Definition 3 about machine m_1 and m_2 in Figure 1. DG is obtained by fixed point calculation from components referr ed in properties to be checked. Machine m_1 and m_2 do not directly depend on each oth er, but through events *timeset* and *timeout* depend on each other implicitly.

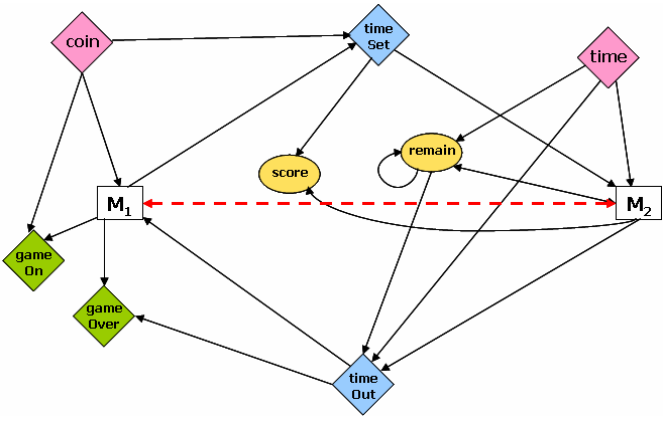

Fig. 2. Dependency graph of m_1 and m_2 in Figure 1

3.3 Correctness of Specification in Reduced Model

With the DG, if we want to check about the machine m_2, then events *gameOn* and *gameOver* may not be used to translate the model into SMV. This technique is a kind of *cone of influence* mentioned in previous chapter. Due to our translation rule, each component in DG corresponds to variable in SMV and variables in SMV are implicitly represented by Boolean variables. So we regard the component as some

Boolean variables like variables in synchronous circuits. It is proved in [7] that the cone of influence preserves the correctness of specifications in CTL if they are defined over variables (atomic propositions) corresponding to components referred in properties. According to the proof, since the original model M and the reduced model M' are bisimulation relation, for any CTL formula ϕ written in referred components in a given property, $M \models \phi \Leftrightarrow M' \models \phi$.

4 Experimental Results

4.1 Verification on Reduced Model

Using simulation, ambiguous transitions was found machine m_{15} in Figure 1. The event trace is $\langle\{coin\},\{time\},\{time\},\{time\},\{time\},\{time, hit_3\}\rangle$. It means that when the first rising the third mole, if player hits the mole, then in state the *CountDown3*, transition t_{36} and t_{37} are executed simultaneously, that is ambiguous transitions. But, by original model, we cannot terminate the verification procedure in Stepper. To overcome this state explosion problem, we apply the reduction technique based on DG explained in previous section, like figure 3. So we can detect this error within 8.25 second and 159027 BDD size. However, the trace generated by model checking is not the same by simulation. The result trace is $\langle\{coin\},\{time\},\{time, hit_3\}\rangle$. It means that there exist serious flaw in the mole game *f*FSM.

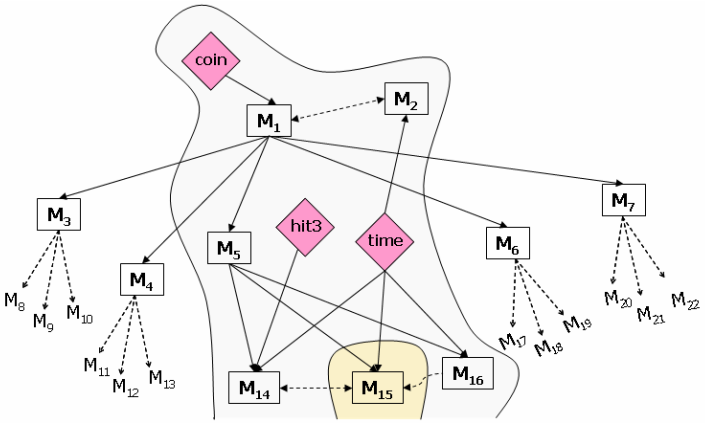

Fig. 3. Selected machine for verifying ambiguous transitions in m_{15}

4.2 Correction of the Model Error

The model error is that when still the control is in *Ready3* state, if player hits the third mole, since the model react the event hit_3, t_{36} and t_{34} are executed simultaneously, but it is just a model error, that is ugly model, not any semantic error. Even though there is a little concern of observation, we can understand that the machine m_5 becomes stuck after that trace. However, through the detection of local deadlock, this error is not detected by semantic analysis because eventually when the super state of machine

m_5 is transferred, the stuck situation is forced to be resolved by semantics. In this case, like Figure 4, designer must modify the model.

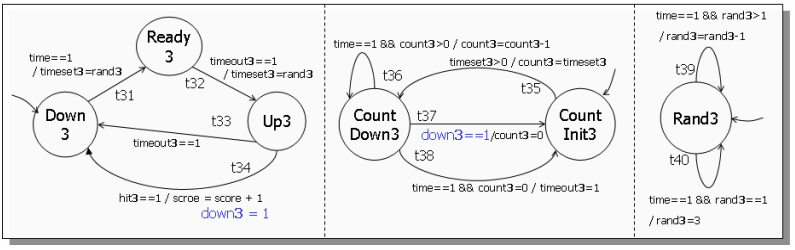

Fig. 4. Modified sub-model under the machine m_5

5 Conclusions

The *f*FSM is a model for describing the control flow aspects in PeaCE(Ptolemy extension as a Codesign Environment), which is a hardware/software codesign environment to support complex embedded systems[1]. We defined step semantics for the control flow model and developed its verification tool in the previous work[4]. In this paper, in order to avoid the state explosion problem, we introduce the model reduction technique based on dependency analysis[5,6,7]. As a result, the model, which couldn't be verified before applying the technique, is verified.

Now we are interesting in applying this reduction technique to software model checking. Because software model used in model checking is one of control flow model and its size is huge.

References

1. D. Kim, "System-Level Specification and Cosimulation for Multimedia Embedded Systems," Ph.D. Dissertation, Computer Science Department, Seoul National University, 2004.
2. iLogix: http://www.ilogix.com/
3. http://ptolemy.eecs.berkeley.edu/
4. S. Park, K. G. Kwon, and S. Ha, "Formalization of fFSM Model and Its Verification," in the Proceedings of the ICESS, LNCS 3820, Springer, pp.361-372, 2005.
5. W. Chan, "Symbolic Model checking for Large software Specification," Dissertation, Computer Science and Engineering at University of Washington, 1999.
6. J. B. Lind-Nielsen, "Verification of Large State/Event Systems," Ph.D. Dissertation, Department of Information Technology, Technical University of Denmark, 2000.
7. E. M. Clarke, O. Grumberg and D. Peled, Model Checking, MIT Press, 1999.
8. J. Lind-Nielsen, H. R. Andersen, H. Hulgaard, G. Behrmann, K. J. Kristoffersen, K. G. Larsen, "Verification of Large State/Event Systems Using Compositionality and Dependency Analysis", FMSD, pp. 5-23, 2001.
9. E. M. Clarke, W. Heinle, "Modular translation of Statecharts to SMV," Technical Report CMU-CS-00-XXX, CMU School of Computer Science, August 2000.