A New Method for Modeling on Concurrent System

Jing Guo*, Zhongwei Xu and Meng Mei

School of Electronics & Information Engineering, Tongji University, Shanghai, 201804, China

Abstract: Binary decision diagrams (BDDs) are effective means to cope with complex concurrent system. But the size of BDD itself can be relatively large. We study the BDD representation of large synchronous, asynchronous and interleaved processes with communication *via* shared variables. Due to the features of communication, we introduce a novel representation strategy. Based on the model, we continue to model and map up the synchrony, and detect the deadlock errors.

Keywords: Formal verification, Model checking, BDD, Concurrent system.

1. INTRODUCTION

To model the concurrent system is a relatively difficult task due to interaction between concurrently executing processes. The computer-aid verification is a useful and well-accepted method to solve the problem.

Model checking has proved to be a powerful for the verification of concurrent finite state system. The system is described as a model and check whether the specifications are satisfied by the model. In symbolic model checking, the transition relation is described by boolean formulas to find satisfying assignment of the formula. States are represented as sets of boolean formulas. The key is whether the boolean formulas can be efficiently expressed as binary decision diagrams (BDDs). The variables are in form of successive case distinctions. Variables ordering is useful to normal forms. To eliminate the size of BDD is vital to the system efficiently.

Related work on representation of digital circuits [1-3]. The width of circuits is the maximum number of wires, through which any cut go through netlist. It turns out that the communication aspect of BDD is responsible for size of BDD. [3] is also concerned with digital circuits. BDD tree is a appropriate structure that is smaller than BDD itself. BDD can be used to help model checking of another structure [4]. Petri nets grows exponentially in the number of states, so it present a method for representing the state space in the form of BDD. [5] focus on the variable ordering of BDD. The impact and placement of mechanism is integrated into the procedure.

There are several difficulties in modeling BDD [6]: 1) how to represent interleaved, synchronous, asynchronous execution [7] 2) the complexity of the communication between the processes [8] 3) how to plan the concurrent processes [9]. We focused on the above problems. In this paper, we propose a extension of model method to BDD. We use mutually disjoint sets of boolean variables to represent each process. Instead of on process, our analysis constructs a compositional heuristics for variable ordering for entire system. Next we show synchronization can be modeled efficiently. Already have eliminated the detail, deadlock can be find in our reduced model.

The rest of this paper is organized as follows. Section 2 presents the preliminaries. Section 3 presents the novel model of BDD. Section 4 extends the model to model the details of synchronization and extension of it for deadlocks. Section 5 outline the summary and future work.

2. PRELIMINARIES

Definition (Transition System) triple (Init, S, $\rightarrow$) where Init $\subseteq$ S is the set of initial state, $\rightarrow \subseteq S \times S$ is transition relation and S is the set of states. Every transition is labeled with formula, which is enabling condition [10].

Binary decision diagrams (BDDs) [11] are a canonical form representation for a boolean formulas, which are often substantially more compact than traditional normal forms. Its structure is a directed acyclic graph rather than a tree, and variables are ordered as one traverses the graph from root to leaf. A path from the variable to the boolean value 1 or 0 is a form of assignment.

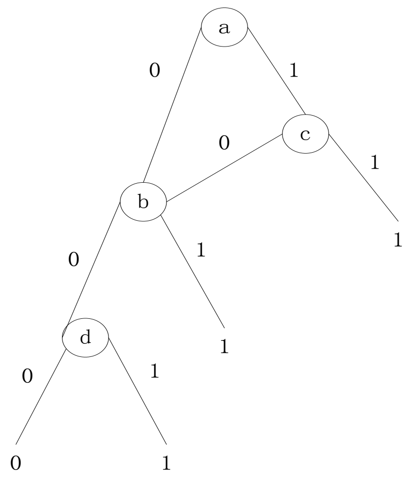

Fig. (1). A binary decision diagram.

3. MODELING CONCURRENT SYSTEMS WITH SHARED DATA VARIABLES

Concurrent processes communicate each other with shared data variables. Each assignment is same as former or the value of variable is update. It is defined as follows:

$$a[x:=v](y)=\begin{cases} a(y) \text{ } if \text{ } x\neq y \\ v \quad if \text{ } x=y \end{cases} \tag{1}$$

Particularly, $a[\varepsilon]=a$.

Every process is modeled as form of transition system (Init, S, $\rightarrow$). However, states are in the form of $A_p \times L_p$

where A_p is a assignment and L_p is a location. Transitions must be $(g,l)\xrightarrow[\varphi]{\alpha}(g',l')=(g,l)\xrightarrow[\varphi]{\varepsilon \text{ } or \text{ } x:=v}(g',l')$. $\varepsilon \text{ } or \text{ } x:=v$ is assignment related to shared variables, and φ is a boolean formula. The finite set of the variables is V_a , the set of variables is X. T_x is the set of updates. Processes are indexed by i. Processes can be in form of $P_1\cdots,P_i,\cdots,P_n$. For L_i, L_j , if $i\neq j$ then $L_i\cap L_j=\varnothing$. Processes communicate each other through shared variables, so $X_i\cap X_j\neq\varnothing$. Global assignment is a function that maps the variable to a value. g_i denoted the global assignment take place in Process i. The set of global assignment is GA. The set of indexes of processes that share x is given as In_x . Processes are considered as cartesian product. If only one process is active at a time, the composition of processes is interleaving. It is a transition system $C_{|}=(Init,S,\rightarrow_{|})$ where

1) $S=GA\times L$ (A global assignment and local location states put together a state.)

2) Init= $\left\{\left(g,\vec{l}\right)\middle|\left(\left(g_{i,}\vec{l_i}\right)\in Init_i \text{ } for \text{ } all \text{ } i\right)\right\}$.(In the process i, it is the initial state of that process.)

3) $\left(g_i,\vec{l_i}\right)\xrightarrow[\varphi]{\alpha}\left(g_i',\vec{l_i'}\right)$. (The interleaved global transition arises from local transition by P_i .

If some process were active at a time, the situation should be asynchronous composition. $C_{|||}=(Init,S,\rightarrow_{|||})$ where Init and S is same as interleaving, but the set $In\in[n]$.

For each $i\in In$, $\left(g_i,\vec{l_i}\right)\xrightarrow[\varphi]{\alpha}\left(g_i',\vec{l_i'}\right)$.

If $i\in[n]\setminus In$, $\vec{l_i'}=\vec{l_i}$.

There may be some conflicts, which are processes want to update the same variable at the same time. Therefore, we map out our strategy. The conflict solution for variable x is a binary relation $\multimap_x$ between the set of α_i. α_i must be in the process where the update exists. The relation resolve which update take place. If there were updates, all the update should meet the requirements.

If all of processes were active at a time, it would be synchrony. It is similar to synchrony apart from that $In=[n]$. And we use $C_{||}=(Init,S,\rightarrow_{||})$ to represent the composition.

BDDs is boolean relation, but the transition system is transition relation between states been labeled. We unify the representation by trying to use boolean formula to represent the transition.

Any variables can be a vector of $\left\{\vec{x_1},\cdots,\vec{x}_{length\,x}\right\}$. If it updated, then $\vec{x}^{up\,i}=\left\{\vec{x_1}^{up\,i},\cdots,\vec{x}^{up\,i}_{length\,x}\right\}$. A member of vector is true, the relevant boolean vector is 1. Otherwise, it is 0. We define X as next operator, which encode state changes. LX encodes local variable update. Update flag U_x^i denotes the i-th transition updates x. The i-th transition could be transform to boolean formula as: $R(\rightarrow_i)=\vee(\mathrm{a},\mathrm{l})\xrightarrow[\varphi]{\alpha}{}_i(a',\mathrm{l}')$.

$$=\begin{bmatrix} \left(\underset{x\in x_i}{\wedge}\begin{cases}\neg U_x^i \text{ } if \text{ } \alpha\notin T_x \\ U_x^i \quad if \text{ } \alpha\in T_x\end{cases}\right) \\ \wedge\begin{cases} TRUE \text{ } if \alpha=\varepsilon \\ LX\,\vec{x}^{up\,i}\bullet\vec{v} \text{ } if \alpha:=v\end{cases} \\ \wedge\varphi' \\ \underset{h\in L_i}{\wedge\wedge}\left(\begin{matrix}\begin{cases}\neg h \text{ } if \text{ } h\neq l \\ h \quad if \text{ } h=l\end{cases} \\ \wedge\begin{cases}\neg Xh \text{ } if \text{ } h\neq l' \\ Xh \quad if \text{ } h=l'\end{cases}\end{matrix}\right)\end{bmatrix} \tag{2}$$

We judge whether the value is true, the related variable exist. The thinking is through the structure. In order to remark asynchronous and interleaved situation, we need to define three particular formula: idle, sched and communication. Idle describe the situation that the process remains at current location and does not carry out any variable update:

$$Idle_i=\left(\underset{x\in x_i}{\wedge}\neg\alpha_x^i\right)\wedge\left(\underset{l\in L_i}{\wedge}xl\equiv l\right) \tag{3}$$

The second situation is only one variable is true at any time:

$$Sched = \bigvee_{i=1}^{n} \bigwedge_{j=1}^{n} \begin{cases} \neg s_j \ \ if \ j \neq i \\ s_j \ \ \ if \ j = i \end{cases} \tag{4}$$

From now on we continue to define the third situation, local copies the value from the global variable:

$$Communication = \bigwedge_{x \in X} \bigwedge_{x \in own_i} \vec{x} \equiv \vec{x}^{\,up\,i} \tag{5}$$

own_i is the set of index of processes where the variable update. For the conflict situation, we define the relation $®_x$:

$$\mathcal{R}\left(\multimap_x\right) = \bigwedge_{x \in X} \left[\begin{array}{l} \bigvee_{(x:=v_i)_{i \in In}} \multimap_x x := v \\ \left[\begin{array}{l} \left(\left(\bigwedge_{x \in In} \alpha_x^i \wedge \left(LX \vec{x}^{\,up\,i} \bullet \vec{v}_i \right) \right) \right) \\ \wedge \left(\bigwedge_{i \in own_x \backslash In} \neg U_x^i \right) \\ \wedge \left(X \vec{x} \bullet \vec{v} \right) \end{array} \right] \\ \vee \left[\left(\bigwedge_{i \in own_x} \neg U_x^i \right) \wedge \left(X \vec{x} \equiv \vec{x} \right) \right] \end{array} \right] \tag{6}$$

The first disjunction over all elements of $\multimap_x$ and the second covers the circumstance of no update.

Interleaving relation use to denote idle transition and illustrate that exactly one s_i is true at any time.

$$R\left(\rightarrow_{|}\right) = \left(\begin{array}{l} \bigwedge_{i=1}^{n} \left(s_i \wedge \left(R\left(\rightarrow_i\right) \right) \vee \left(\neg s_i \wedge Idle_i \right) \right) \\ \wedge Sched \wedge Communication \wedge \mathcal{R}\left(\multimap_x\right) \end{array} \right) \tag{7}$$

$$R\left(\rightarrow_{|||}\right) = \left(\begin{array}{l} \bigwedge_{i=1}^{n} \left(R\left(\rightarrow_i\right) \vee Idle_i \right) \\ \wedge Communication \wedge \mathcal{R}\left(\multimap_x\right) \end{array} \right) \tag{8}$$

$$R\left(\rightarrow_{||}\right) = \left(\begin{array}{l} \bigwedge_{i=1}^{n} R\left(\rightarrow_i\right) \\ \wedge Communication \wedge \mathcal{R}\left(\multimap_x\right) \end{array} \right) \tag{9}$$

In interleaving, the conflict does not occur, the $\mathcal{R}\left(\multimap_x\right)$ merely account for local and global update.

4. DETAILED MODEL OF SYNCHRONY

We assume that processes communication with each other only *via* global variables. In real world, concurrent synchronization is a regime where multiple groups of whole system are synchronized. We establish a mechanism for model the situation better. The variable m_i has two value state: L(locked) and U(Unlocked).

$$t_i \bullet m_i = \begin{cases} t_i \ \ if \ m_i = L \\ \neg t_i \ if \ m_i = U \end{cases} \tag{10}$$

If we model the process explicitly, the result would be a complicated model. Fortunately, we neglect the details. When it ignores the internal implementation, two executions have the same states: $\pi \approx_m \pi'$.

$$D = \bigwedge_{i \in In} \left(t_i \bullet m_i \wedge X\left(t_i \bullet m_i \right) \right) \tag{11}$$

Lock wait m is unlocked, and changes the state. Unlock is similar.

Conditions are classified into three: wait, signal and broadcast. Signal awakens one of executions that are waiting for this condition. Broadcast awakens all of the executions that are wait for this condition. Wait use m(Locked, Unlocked) as parameter. cond[i] flag is true, there is a wait in i-th process.

$$\begin{array}{l} wait = \left(cond\left[i\right] = 1 \wedge s_i \bullet m_i \Rightarrow s_i \right) \wedge \\ \left(\left(Idle \, until \, con\left[i\right] == 0 \right) \wedge s_i \bullet m_i = s_i \right) \end{array} \tag{12}$$

$$\begin{array}{l} signal = \\ \left(choose \, i \right) \wedge \left(\begin{array}{l} \left(\left(cond\left[i\right] == 1 \right) \vee \left(cond == 0 \right) \right) \\ \Rightarrow cond\left[i\right] = 0 \end{array} \right) \end{array} \tag{13}$$

Deadlock may be take place all the processes are waiting. Based on lock and wait, we add a global variable P_in_wait to counter the number of processes in a wait state. We want to lock the process then find the process is already in state Locked. We increase the variable. P_in_wait is not more than N. Otherwise, a deadlock is detected. After we set up the cond, increase or decrease the P_in_wait. We use dd to illustrate that the deadlock is founded.

$$\begin{array}{l} lock_{new} = \\ dd \wedge \left(\begin{array}{l} \left(\left(t_i == U \right) \Rightarrow \left(t_i \bullet m_i = t_i \right) \right) \vee \\ \left(\left(t_i == L \right) \Rightarrow P\_in\_wait \uparrow \right) \end{array} \right) \\ \wedge \left(\left(wait \ t_i == L \right) \wedge \left(P\_in\_wait == N \Rightarrow dd = 1 \right) \right) \end{array} \tag{14}$$

$$\begin{array}{l} wait_{new} = \\ dd \wedge cond\left[i\right] = 1 \wedge t_i == U \wedge P\_in\_wait \uparrow \\ \wedge \left(P\_in\_wait == N \Rightarrow dd = 1 \right) \\ \wedge \left(cond\left[i\right] == 0 \Rightarrow P\_in\_wait \downarrow \right) \end{array} \tag{15}$$

CONCLUSION

In summary, we present a new model of interleaved, synchronous and asynchronous concurrent systems. This model shows improvement in the size. To better model synchronization, a mechanism is introduced. We hold the new trends of the concurrent processes, and plan the execution, so can detect the deadlock. For further work, we will discuss other "regular" errors and calculate the up bounds for BDDs sizes.

CONFLICT OF INTEREST

The authors confirm that this article content has no conflict of interest.

ACKNOWLEDGEMENTS

The work is support by the National Natural Science Foundation of China under grant No. 60674004, No. 61075002, the National "Twelfth Five-Year" Plan for Science & Technology Support under grant No.2011BAG-01B03, the railway ministry basic research program of China under grant No.2012AA112801, the National High Technology Research and Development Program 863 under grant No. 2012AA112801

REFERENCES

[1] R. Bryant, "Symbolic boolean manipulation with ordered binary decision diagrams", *ACM Computing Surveys*, vol. 41, pp. 686-698, June 1992.

[2] K. McMillan, "*Symbolic Model Checking: An Approach to the State Explosion Problem*," Technical Report CMU–CS–92–131, PhD Thesis, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, 1992.

[3] K. McMillan, *Hierarchical Representations of Discrete Functions, with Application to Model Checking. In Computer Aided Verification*, LNCS: Springer–Verlag, Germany, pp. 41-54, 1994.

[4] P. Miczulski, "Calculating State Spaces of Hierarchical Petri Nets Using BDD", In: *Design of Embedded Control Systems*, 2005, pp. 85-94

[5] R. Goré, and J. Thomson, "An improved BDD method for intuitionistic propositional logic: BDDIntKt system description", *Lecture Notes in Computer Science*, vol. 7898, pp. 275-281, 2013.

[6] T. K. Nguyen, J. Sun, Y. Liu, J. S. Dong, and Y. Liu, "Improved BDD-based discrete analysis of timed systems", *Lecture Notes in Computer Science* , vol. 7436, pp. 326–340, 2012.

[7] J. Sun, Y. Liu, J. S. Dong, Y. Liu, L. Shi, and E. Andr´e, "Modeling and verifying hierarchical real-time systems using stateful timed CSP", *ACM Transactions on Software Engineering and Methodology*, vol. 22, no. 1, pp. 3.1-3.29, 2013.

[8] J. Sun, Y. Liu, J. S. Dong, and J. Pang, "PAT: Towards flexible verification under fairness", *Lecture Notes in Computer Science*, vol. 5643 of Springer, pp.709-714, 2009.

[9] J. Sun, Y. Liu, J. S. Dong, and X. Zhang, " Verifying stateful timed CSP using ımplicit clocks and zone abstraction", *Lecture Notes in Computer Science*, vol. 5885, pp. 581-600, 2009.

[10] F. Herbreteau, B. Srivathsan, and I. Walukiewicz, "Efficient emptiness check for timed b¨uchi automata", *Lecture Notes in Computer Science*, vol. 6174, pp. 148-161, 2010.

[11] D. Beyer, C. Lewerentz, A. Noack, " Rabbit: A tool for BDD-based verification of real-time systems", *Lecture Notes in Computer Science*, vol. 2725, pp. 122-125, 2003.

Received: September 16, 2014 Revised: December 23, 2014 Accepted: December 31, 2014