Hindawi Publishing Corporation
Journal of Applied Mathematics
Volume 2014, Article ID 725275, 10 pages
http://dx.doi.org/10.1155/2014/725275

*Research Article*

# Terminal Satisfiability in GSTE

**Yongsheng Xu,[1] Guowu Yang,[1] Zhengwei Chang,[2] Desheng Zheng,[1] and Wensheng Guo[1]**

[1] *School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, Sichuan 611731, China*
[2] *State Grid Sichuan Electric Power Research Institute, Chengdu, Sichuan 610000, China*

Correspondence should be addressed to Guowu Yang; guowu@uestc.edu.cn

Received 9 February 2014; Accepted 10 March 2014; Published 15 May 2014

Academic Editor: Guiming Luo



*Generalized symbolic trajectory evaluation* (GSTE) is an extension of symbolic trajectory evaluation (STE) and a method of model checking. GSTE specifications are given as assertion graphs. There are four efficient methods to verify whether a circuit model obeys an assertion graph in GSTE, Model Checking Strong Satisfiability (SMC), Model Checking Normal Satisfiability (NMC), Model Checking Fair Satisfiability (FMC), and Model Checking Terminal Satisfiability (TMC). SMC, NMC, and FMC have been proved and applied in industry, but TMC has not. This paper gives a six-tuple definition and presents a new algorithm for TMC. Based on these, we prove that our algorithm is sound and complete. It solves the SMC's limitation (resulting in false negative) without extending from finite specification to infinite specification. At last, a case of using TMC to verify a realistic hardware circuit round-robin arbiter is achieved. Avoiding verifying the undesired paths which are not related to the specifications, TMC makes it possible to reduce the computational complexity, and the experimental results suggest that the time cost by SMC is 3.14× with TMC in the case.

## 1. Introduction

Logic errors found in finite state concurrent systems are extremely important problems for both circuit designers and programmers [1] (i.e., sequential circuit designs and communication protocols). Model checking is a technique for verifying finite state concurrent systems [1]. Symbolic trajectory evaluation is a lattice-based model checking technique based on a form of symbolic simulation [2–4]. STE has shown great promise in verifying medium to large scale industrial hardware designs with a high degree of automation at both the gate level and the transistor level [4–7]. *Generalized symbolic trajectory evaluation* [3, 8] is an extension of symbolic trajectory evaluation [9, 10]. STE is very limited in the types of properties that it can specify and verify. GSTE can handle *ω-regular* properties and maintain the efficiency and capacity of STE [9–13]. GSTE is originally developed at Intel and has been used successfully on Intel's next-generation microprocessors [14].

The main disadvantage of model checking is the state explosion, which motivates the need for algorithms such as abstract technology and the data structure of BDD and so forth to alleviate it [2, 15–18]. Although a lot of effort has been spent on improving this weakness, the efficient and effective method has yet to be developed. GSTE is a method of model checking and it also has the problem of state explosion. In GSTE, specifications are given as assertion graph and a model is induced by transition relation. Each edge in the model represents a state transition and a *trace* in the model is a state sequence, while a *path* in the assertion graph is an edge sequence. In SMC, NMC, and FMC, a model satisfies an assertion graph if all the traces (finite or infinite) in the model are accepted by all the paths of the same length in the assertion graph [3, 8, 19]. Results shown in [1, 9, 15, 17, 20–22] indicate that the method of enumeration usually leads to state explosion and increases the complexity of computing. As we know, traces are induced by model, and the number of traces is directly proportional to the size of model. Furthermore, paths are related to specifications. Thus, it will not need to verify each trace or path. Therefore, the key to alleviating the state explosion lies in deleting some undesired paths and traces. Aiming at this situation, this paper puts forward the filtering approach for TMC (Terminal Satisfiability Model Checking).

Previous studies have shown or suggested that a tool is needed to tell the difference between desired paths and undesired paths. One possible method is to choose $E_T$ as the filter condition, where $E_T$ is the set of edges in assertion graph. Our approach only adds some restricted conditions into assertion graphs without changing the structure of models and assertion graphs. Each edge in the assertion graph is labeled by an antecedent ant($e$) and consequent cons($e$). Ant($e$) and cons($e$) are sets of states in the model, so we can assign the states related to specifications to the ant($e$) of edge which is in $E_T$. Then, we can delete undesired paths by $E_T$. A fuller discussion of TMC will appear in a later section.

In this paper, we focus on the need for alleviating state explosion and present a new algorithm for TMC based on GSTE. In Section 2, we introduce the basic definitions in GSTE. In Section 3, some concepts such as terminal assertion graph, terminal path, and terminal satisfiability are defined. After a statement of the basic concepts, related properties and a theorem are given. In Section 4, we present an algorithm for verifying terminal satisfiability with an example which cannot be verified by SMC but by TMC. In the end, we use TMC to successfully verify a hardware circuit round-robin arbiter, and the time cost by SMC is 3.14× with TMC in the case. In Section 5, we prove that our algorithm is sound and complete. Section 6 is the conclusion of the paper.

## 2. Preliminaries

We introduce some basic definitions in GSTE [3, 8]. We assume a universal set of finite states, denoted by $S$.

### 2.1. Model

*Definition 1* (transition relation; see [3, 8]). A relation $T \subseteq S \times S$ is a transition relation if for all $s \in S$, $\exists s' \in S$, $(s, s') \in T$.

*Definition 2* (model; see [3, 8]). The model $M$ induced by the transition relation $T$ is the pair (pre, post), where

(1) the preimage transformer pre: $2^S \to 2^S$ is defined as

$$\text{pre}(Q) = \left\{ s \mid \exists s' \in Q, \left(s, s'\right) \in T \right\} \quad \forall Q \in 2^S, \tag{1}$$

(2) and the postimage transformer post: $2^S \to 2^S$ is defined as

$$\text{post}(Q) = \left\{ s' \mid \exists s \in Q, \left(s, s'\right) \in T \right\} \quad \forall Q \in 2^S. \tag{2}$$

*Definition 3* (trace; see [3, 8]). A trace $\sigma$ in model $M = (\text{pre}, \text{post})$ is a state sequence such that $\sigma[i+1] \in \text{post}(\sigma[i])$ for all $1 \le i \le |\sigma|$.

### 2.2. Assertion Graph

*Definition 4* (assertion graph; see [3, 8]). An assertion graph is a quintuple $G = (V, v_0, E, \text{ant}, \text{cons})$, where $V$ is a finite set of vertices, $v_0$ is the initial vertex, $E \subseteq V \times V$ is a set of edges satisfying for all $u \in V$, $\exists v \in V$, $(u, v) \in E$, ant is a mapping: $E \to 2^S$, and cons is a mapping: $E \to 2^S$.

*Definition 5* (path; see [3, 8]). A path $\rho$ in assertion graph $G$ is an edge sequence such that $\rho[i]$ ends at a vertex from which $\rho[i+1]$ starts for all $1 \le i \le |\rho|$.

*Definition 6* (a trace is accepted by a path; see [3, 8]). An execution trace $\sigma$ in a model is accepted by a path $\rho$ in an assertion graph if (suppose the length of trace and path is $n$)

$$\forall i_{1 \le i \le n} \sigma[i] \in \text{ant}\left(\rho[i]\right) \Longrightarrow \forall i_{1 \le i \le n} \sigma[i] \in \text{cons}\left(\rho[i]\right), \tag{3}$$

where $\sigma[i]$ is the $i$th state of the trace $\sigma$ and $\rho[i]$ is the $i$th edge of the path $\rho$.

### 2.3. Four Kinds of Verifying Algorithms.

Based on Definition 6, GSTE has four kinds of acceptance [19].

*Definition 7* (SMC (model checking strong satisfiability)). A model $M$ strongly satisfies an assertion graph $G$ if for all finite initial paths $\rho$ in $G$ and all finite traces $\sigma$ in $M$ of the same length $\sigma$ is accepted by $\rho$, denoted by $M \models G$.

*Definition 8* (NMC (model checking normal satisfiability)). A model $M$ normally satisfies an assertion graph $G$ if for every infinite initial path $\rho_\omega$ in $G$ and every infinite trace $\sigma_\omega$ in $M$ of the same length $\sigma_\omega$ is accepted by $\rho_\omega$, denoted by $M \models G$.

*Definition 9* (FMC (model checking fair satisfiability)). A model $M$ fairly satisfies an assertion graph $G$ under $F$ (fairness constraint) if for every infinite fair path $\rho_\omega$ in $G$ and every infinite trace $\sigma_\omega$ in $M$, $\sigma_\omega$ is accepted by $\rho_\omega$, denoted by $M \models_F G$.

*Definition 10* (TMC (model checking terminal satisfiability)). A model $M$ terminally satisfies an assertion graph $G$ under $E_T$ (terminal edge set) if for all finite terminal paths $\rho_T$ in $G$ and all finite traces $\sigma$ in $M$ of the same length $\sigma$ is accepted by $\rho_T$, denoted by $M \models_T G$.

We use the following notations in the rest of the paper, $\rho = [e_1, e_2, \ldots]$ be an arbitrary sequence of elements, $|\rho|$ be the length of the sequence, and $\rho[i]$ be the $i$th element $e_i$ in the sequence. Further, $(\rho : e)$ denotes the sequence by appending element $e$ to the end of $\rho$ when $\rho$ is finite, and $(e : \rho)$ is the sequence by appending element $e$ to the head of $\rho$.

## 3. Terminal Satisfiability

In this section, we define some concepts such as terminal assertion graph and terminal path and give some related properties on them.

In GSTE, the set of states in model $M$ is finite ($|S|$ is finite), but each state has its successor states so that traces in the model can be extended to infinite length. Each edge in the assertion graph has its successor edges so that assertion graphs can specify infinite properties. It should be noted that in our daily life, most of the properties which have been verified are finite rather than infinite. In particular, some properties are terminated at some states and these states are not involved. In GSTE existing theories, the SMC algorithm is the only one used to handle finite properties [3, 8]; it cannot

stop at some edges whose labels ant($e$) are not related to the specifications and there are many redundant calculations. The program proves that we can predict the behavior of the model; these observations lead us to hypothesize that finite specifications will terminate at some states which we can predict. Corresponding to terminal specifications, the paths will also terminate at some edges, which are named terminal edge. Our algorithm will make the labels ant($e$) on the edges, which cannot reach terminal edges, equal to $\emptyset$. The method outlined here can be used to reduce the complexity of computation.

For convenience, we expanded the assertion graph into six-tuple.

*Definition 11* (terminal assertion graph). A terminal assertion graph is a six-tuple $G = (V, v_0, E, \text{ant}, \text{cons}, E_T)$, where $V$, $v_0$, $E$, ant, and cons are the same as in preliminaries; $E_T \subseteq E$ is a set of terminal edges.

*Definition 12* (terminal path). A terminal path is

$$\rho_T = e_0 e_1 e_2 \cdots e_{|\rho_T|} \quad \left(e_i \in E,\ i = 0, 1, 2, \ldots, \left|\rho_T\right|\right), \tag{4}$$

where $e_0$ is an initial edge, it starts from $v_0$, and $e_{|\rho_T|}$ is a terminal edge, $e_{|\rho_T|} \in E_T$. The last edge of path $\rho_T$ must be terminal edge.

*Definition 13* (a trace is accepted by a terminal path; see [3, 8, 19]). Let $G = (V, v_0, E, \text{ant}, \text{cons}, E_T)$ be a terminal assertion graph and $M = (\text{pre}, \text{post})$ be a model. Given an edge labeling $\gamma : E \to 2^S$ where $\gamma$ is either ant or cons, a trace $\sigma$ in $M$ satisfies a terminal path $\rho_T$ of the same length under $\gamma$ if for every $1 \le i \le |\sigma|$, $\sigma[i] \in \gamma(\rho_T[i])$, denoted by

$$(M, \sigma) \vDash_\gamma (G, \rho_T) \quad (\gamma = \text{ant or cons}). \tag{5}$$

The trace $\sigma$ is accepted by the terminal path $\rho_T$ if

$$(M, \sigma) \vDash_{\text{ant}} (G, \rho_T) \Longrightarrow (M, \sigma) \vDash_{\text{cons}} (G, \rho_T), \tag{6}$$

denoted by $(M, \sigma) \vDash_T (G, \rho_T)$.

*Definition 14* (model satisfies terminal assertion graph). A model $M$ satisfies a terminal assertion graph $G$ if for all terminal paths $\rho_T$ in $G$ and all traces $\sigma$ in $M$ of the same length one has $(M, \sigma) \vDash_T (G, \rho_T)$, denoted by

$$M \vDash_T G. \tag{7}$$

*Definition 15* (model strongly satisfies assertion graph; see [3, 8]). A model $M$ strongly satisfies an assertion graph $G$ ($E_T = \emptyset$) if for all finite paths $\rho$ in $G$ and all traces $\sigma$ in $M$ of the same length one has $(M, \sigma) \vDash (G, \rho)$, denoted by

$$M \vDash G. \tag{8}$$

*The Difference between Finite Path and Terminal Path*. Note that there is a key difference between finite path and terminal path; unlike terminal path, which must be end up with a terminal edge $e \in E_T$, finite path can end up with any edge $e \in E$. Given our analysis, this suggests a terminal path will be a finite path; however, a finite path may not be a terminal path.

Let $A = \{\rho \mid \rho$ is a finite initial path and the length of $\rho$ is $|\rho|$, where $|\rho| = 1, 2, \ldots\}$ and $B = \{\rho_T \mid \rho_T$ is a terminal path and the length of $\rho_T$ is $|\rho_T|$, where $|\rho_T| = 1, 2, \ldots\}$. The last edge of $\rho_T$ must be a terminal edge, $\rho_T[|\rho_T|] \in E_T$; however, the last edge of $\rho$ does not have this restriction, it just needs to meet $\rho[|\rho|] \in E$, $E_T \subseteq E$, so one has $B \subseteq A$.

**Theorem 16.** *For any assertion graph $G = (V, v_0, E, ant, cons, E_T)$ and any model $M$, one has*

$$M \vDash G \Longrightarrow M \vDash_T G. \tag{9}$$

*Proof.* According to Definition 15

$$\begin{aligned} M \vDash G &\Longleftrightarrow \forall \rho \in A, \quad (M, \sigma) \vDash (G, \rho) \\ &\Longrightarrow \forall \rho_T \in B, \quad (M, \sigma) \vDash_T (G, \rho_T) \quad (\text{using } B \subseteq A) \\ &\Longleftrightarrow M \vDash_T G \quad (\text{using Definition 14}). \end{aligned} \tag{10}$$

□

Theorem 16 describes the relationship between SMC and TMC. If SMC returns true, then TMC returns true, but not vice versa. When $E_T = E$, terminal satisfiability behaves as strong satisfiability. When $E_T \subseteq E$, $E_T \neq E$, we can remove some undesired paths in the assertion graph.

## 4. Model Checking with Terminal Satisfiability

In this section, we describe a model checking method for verifying terminal satisfiability. The concept of terminal satisfiability was proposed in [19] but was not given any algorithm. In this paper, we present an algorithm and prove it.

*4.1. The TMC Algorithm.* In [3, 8], the algorithms for SMC, NMC, and FMC were given. Our TMC algorithm is similar. First, we should define the backward simulation sequence.

*Definition 17.* Given an assertion graph $G = (V, v_0, E, \text{ant}, \text{cons}, E_T)$ and a model $M = (\text{pre}, \text{post})$, the backward simulation sequence is as below

$$[\varphi_1, \varphi_2, \varphi_3, \ldots], \tag{11}$$

where $\varphi_n : E \to 2^S$ $(n \ge 1)$ is the $n$-step backward simulation relation. It is defined as

$$\varphi_1(e) = \begin{cases} \bigcup_{e^+ \in \text{out}(e) \cap E_T} \left(\text{pre}\left(\text{ant}\left(e^+\right)\right) \cap \text{ant}(e)\right) & \forall e \in E - E_T, \\ \text{ant}(e) & \forall e \in E_T, \end{cases}$$

$$\varphi_n(e) = \begin{cases} \varphi_n(e) \cup \left( \bigcup_{e^+ \in \text{out}(e)} \left(\text{pre}\left(\varphi_n\left(e^+\right)\right) \cap \text{ant}(e)\right) \right) & \forall e \in E - E_T, \\ \text{ant}(e) & \forall e \in E_T. \end{cases} \tag{12}$$

Based on this result, when we verify finite properties, we simply verify those traces that can reach some states in ant($e$), where $e \in E_T$. These states, which are related to the property, are assigned to the label ant of the terminal edge. Then we emulate the model's behavior backwards starting from the terminal edge. The $n$-step backward simulation relation $\varphi_n$ is the set of the states that can reach the terminal edge within $n$-step.

Obviously, the backward simulation relation sequence is nondecreasing and has a least fix-point upper limit bounded by ant. We denote the least fix-point by $\varphi_T^*$. Now we propose a terminal satisfiability model checking algorithm that first performs a series of graph transformations and then checks strong satisfiability on the resulting graph.

*4.2. The Application of TMC.* In order to describe TMC clearly, let us look at the following example [23].

*Example 18.* Consider the model $M$ in Figure 1, where $S = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6\}$.

Suppose we want to specify the following property: if the system is in the state $s_5$, then it must have been in $s_3$ the last time when it was neither in $s_5$ nor $s_6$. The model satisfies this property. Now, let us catch this property using the assertion graph in Figure 2.

The fact that SMC cannot verify this property is due to undesired paths, such as the finite initial path $\rho_1 = [(v_0, v_1)]$ and the finite trace $\sigma_1 = [s_4]$, $\sigma_1 \vDash_{\text{ant}} \rho_1$, but the first state $s_4$ of $\sigma_1$ is not in the cons$((v_0, v_1)) = \{s_3\}$. Thus, the model does not strongly satisfy the assertion graph. If we use SMC, the SMC algorithm will return false. In order to deal with this situation in the former, [3, 8] introduce NMC and FMC. As we know, NMC and FMC have been used to deal with *ω-regular* properties. If we want to solve the SMC's limitation, we should expand finite specifications to infinite specifications. This is the key that leads to the increasing complexity of the model checking. However, the algorithm of using TMC to verify this property is more suitable. TMC can be accomplished without expanding finite specifications to infinite specifications. The following example is constructed only for the purpose of illustrating the computational procedure discussed.

*Step 1.* Construct the terminal assertion graph and figure out the terminal edges (Figure 3).

*Step 2.* First, the fix-point $\varphi_T^*$ of every edge is computed according to Definition 17; then the label ant of every edge is replaced by the corresponding $\varphi_T^*$; lastly, the resulting assertion graph $G'$ is gotten.

*Step 3.* This step behaves as the algorithm SMC : SMC($M$, $G'$). The edge which is marked by the blue line is the terminal edge. Supposing $E_T = \{(v_1, v_2)\}$, there is only one terminal edge in the case (Figure 3). Thus, the terminal path can be grouped into two cases (as follows):

$$\begin{aligned} \rho_{T_1} &= \left[(v_0, v_1)(v_1, v_2)\right]\left(\left|\rho_{T_1}\right| = 2\right), \\ \rho_{T_2} &= \left[(v_0, v_1)(v_1, v_1)\cdots(v_1, v_1)(v_1, v_2)\right]\left(\left|\rho_{T_2}\right| \geq 3\right). \end{aligned} \tag{13}$$

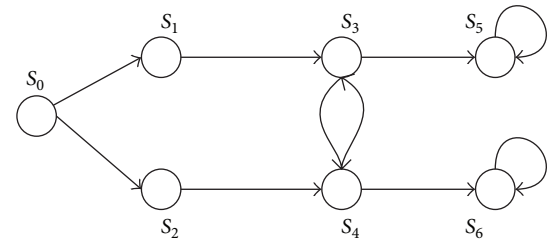


Figure 1: A simple model.

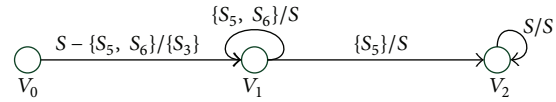


Figure 2: An assertion graph.

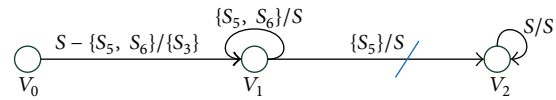


Figure 3: A terminal assertion graph.

Because the terminal path must end up with the terminal edge, there is only one terminal edge in Figure 3 and the vertex $v_2$ to $v_1$ is unreachable. Thus, if a path reaches the edge $(v_2, v_2)$ that is not a terminal path, then all terminal paths are finite. As we know, the length of a terminal path must be greater than or equal to 2. Thus, we can eliminate the path $\rho_1$ and the trace $\sigma_1$, which will lead to SMC returning false negative, since $|\rho_1| = |\sigma_1| = 1$. The trace in model $M$ (Figure 1), if not starting from $s_3$, ultimately would violate the label ant$((v_1, v_2))$ on the edge $(v_1, v_2)$; therefore the model $M$ (Figure 1) terminally satisfies the terminal assertion graph $G$ (Figure 3). The calculation is as follows (after one iteration, every edge can be reached to the fix-point).

The first iteration (initialization):

$$\begin{aligned} \varphi_1(v_1, v_2) &= \text{ant}((v_1, v_2)) = \{s_5\}, \\ \varphi_1(v_1, v_1) &= \text{pre}(\text{ant}((v_1, v_2))) \cap \text{ant}((v_1, v_1)) \\ &= \text{pre}(\{s_5\}) \cap \{s_5, s_6\} = \{s_3, s_5\} \cap \{s_5, s_6\} = \{s_5\}, \\ \varphi_1(v_0, v_1) &= \text{pre}(\text{ant}((v_1, v_2))) \cap \text{ant}((v_0, v_1)) \\ &= \{s_3, s_5\} \cap \{s_0, s_1, s_2, s_3, s_4\} \\ &= \{s_3\}, \\ \varphi_1(v_2, v_2) &= \emptyset. \end{aligned} \tag{14}$$

The second iteration:

$$\begin{aligned} \varphi_2(v_1, v_2) &= \{s_5\}, \\ \varphi_2(v_1, v_1) &= \varphi_1(v_1, v_1) \cup (\text{pre}(\varphi_1(v_1, v_1)) \cap \text{ant}((v_1, v_1))) \\ &\cup (\text{pre}(\varphi_1(v_1, v_2)) \cap \text{ant}((v_1, v_1))) \\ &= \{s_5\} \cup (\text{pre}\{s_5\} \cap \{s_5, s_6\}) \\ &= \{s_5\}, \end{aligned}$$

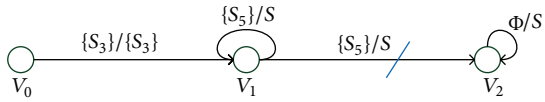


Figure 4: The resulting assertion graph $G'$.

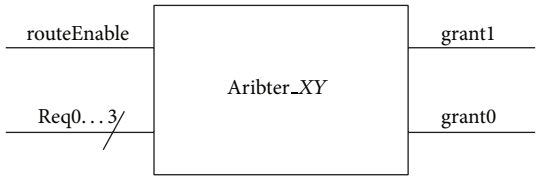


Figure 5: An arbiter with 4 inputs.

$$\begin{aligned}\varphi_2(v_0,v_1) &= \varphi_1(v_0,v_1) \cup (\mathrm{pre}(\varphi_1(v_1,v_1)) \cap \mathrm{ant}((v_0,v_1)))\\ &\cup (\mathrm{pre}(\varphi_1(v_1,v_2)) \cap \mathrm{ant}((v_0,v_1)))\\ &= \{s_3\} \cup (\mathrm{pre}\{s_5\} \cap \{s_0,s_1,s_2,s_3,s_4\})\\ &\cup (\mathrm{pre}\{s_5\} \cap \{s_0,s_1,s_2,s_3,s_4\})\\ &= \{s_3\},\\ \varphi_2(v_2,v_2) &= \emptyset.\end{aligned} \tag{15}$$

After replacing the label ant of each edge with the corresponding fix-point $\varphi_T^*$, the resulting assertion graph $G'$ is shown as Figure 4.

As can be seen, ant of each edge has been reached by the minimum, and TMC returns true. Thus, the model satisfies the terminal assertion graph.

*4.3. A Hardware Verification Case of TMC.* In this section, we present a case study of TMC on hardware verification. The circuit we choose for this study is a round-robin arbiter, which is the core component in many real network systems [24]. An arbiter has $N$ inputs and the output is a vector encoding of arbitration results. Request with highest priority in round-robin order is granted in each cycle, and the priority of the request which is granted in last arbitration will become the lowest in the next round; this protocol guarantees a dynamic priority assignment to requestors without starvation. We consider the arbiter with 4 inputs (shown in Figure 5). First, the priority of inputs is placed in descending order from req[0] to req[3] [25, 26]. Thus, req[0] has the highest priority, req[1] has the next priority, and so on. Each input of the arbiter in Figure 5 is connected to a switch cell; only one of the four inputs will succeed each clock cycle. Others are rejected and must retry later. routeEnable is enable signal.

We model this arbiter as a finite state machine (shown in Figure 6) according to its truth table in paper [24]. The input req[3 : 0] is a vector of ternary-valued variables; each variable can take 0, 1, or $X$. $X$ indicates that variable can take 0 or 1. Figure 6 is the model of this arbiter. Consider

trans_0_0 = req1 = 0&req2 = 0&req3 = 0&routeEnable = 1,

trans_0_1 = req1 = 1&ruoteEnable = 1,

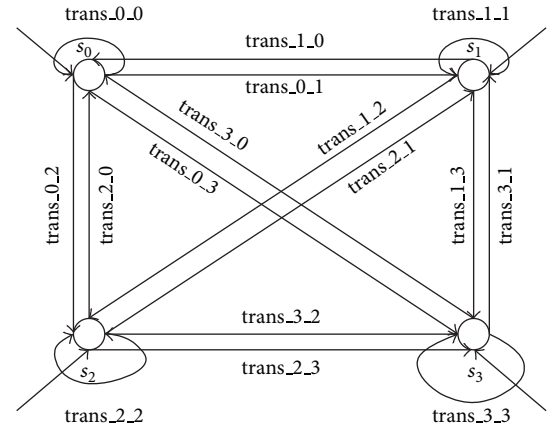


Figure 6: The behavior of arbiter.

trans_0_2 = req1 = 0&req2 = 1&routeEnable = 1,

trans_0_3 = req1 = 0&req2 = 0&req3 = 1&routeEnable = 1,

trans_1_1 = req2 = 0&req3 = 0&req0 = 0&routeEnable = 1,

trans_1_2 = req2 = 1&routeEnable = 1,

trans_1_3 = req2 = 0&req3 = 1&routeEnable = 1,

trans_1_0 = req2 = 0&req3 = 0&req0 = 1&routeEnable = 1,

trans_2_2 = req3 = 0&req0 = 0&req1 = 0&routeEnable = 1,

trans_2_3 = req3 = 1&routeEnable = 1,

trans_2_0 = req3 = 0&req0 = 1&routeEnable = 1,

trans_2_1 = req3 = 0&req0 = 0&req1 = 1&routeEnable = 1,

trans_3_3 = req0 = 0&req1 = 0&req2 = 0&routeEnable = 1,

trans_3_0 = req0 = 1&routeEnable = 1,

trans_3_1 = req0 = 0&req1 = 1&routeEnable = 1,

trans_3_2 = req0 = 0&req1 = 0&req2 = 1&routeEnable = 1. (16)

Safety property: once a request reqi is set high from a state and kept high, then the request will be granted after several cycles. Suppose a state where the value of grant is [1, 0], which means that the last request granted is req2; if the request req2 is set high again and kept high, then the request will be granted after at most 4 cycles. We use the terminal assertion graph to specify this property (shown as in Figure 7) [24]. Consider

ant_$V_I$_$V$set = req0 = 0&req1 = 0&req2 = 1&req3 = 0,

ant_$V$set_$V_1$ = req2 = 1&req3 = 0&req0 = 0&req1 = 1,

ant_$V$set_$V_3$ = req2 = 1&req3 = 1,

ant_$V$set_$V_0$ = req2 = 1&req3 = 0&req0 = 1,

ant_$V_1$_$V_2$ = req2 = 1,

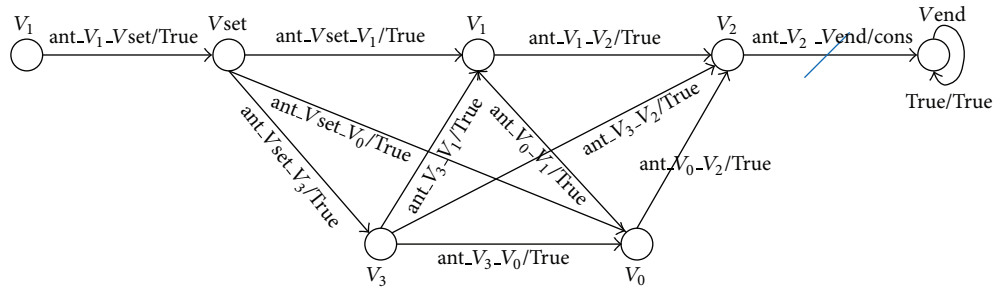


Figure 7: The terminal assertion graph.

$$\begin{aligned}
\text{ant_}V_0\text{_}V_1 &= \text{req2} = 1\&\text{req1} = 1,\\
\text{ant_}V_0\text{_}V_2 &= \text{req2} = 1\&\text{req1} = 0,\\
\text{ant_}V_3\text{_}V_0 &= \text{req2} = 1\&\text{req0} = 1,\\
\text{ant_}V_3\text{_}V_1 &= \text{req2} = 1\&\text{req0} = 0\&\text{req1} = 1,\\
\text{ant_}V_3\text{_}V_2 &= \text{req2} = 1\&\text{req0} = 0\&\text{req1} = 0,\\
\text{ant_}V_2\text{_}V\text{end} &= \text{grant1} = 0\&\text{grant0} = 1,\\
\text{cons} &= \text{grant1} = 0\&\text{grant0} = 1.
\end{aligned} \tag{17}$$

The edge $(V_2, V\text{end})$ is the terminal edge. We only verify those paths that end up with the edge $(V_2, V\text{end})$. If we use SMC to verify this property, we must verify all finite paths. If using TMC, we only need to verify terminal paths. In the terminal assertion graph (Figure 7), all finite paths are as follows:

$$\begin{aligned}
\rho_1 &= [V_I, V\text{set}],\\
\rho_2 &= [V_I, V\text{set}]\,[V\text{set}, V_1],\\
\rho_3 &= [V_I, V\text{set}]\,[V\text{set}, V_3],\\
\rho_4 &= [V_I, V\text{set}]\,[V\text{set}, V_0],\\
\rho_5 &= [V_I, V\text{set}]\,[V\text{set}, V_1]\,[V_1, V_2],\\
\rho_6 &= [V_I, V\text{set}]\,[V\text{set}, V_3]\,[V_3, V_1],\\
\rho_7 &= [V_I, V\text{set}]\,[V\text{set}, V_0]\,[V_0, V_1],\\
\rho_8 &= [V_I, V\text{set}]\,[V\text{set}, V_3]\,[V_3, V_0],\\
\rho_9 &= [V_I, V\text{set}]\,[V\text{set}, V_3]\,[V_3, V_2],\\
\rho_{10} &= [V_I, V\text{set}]\,[V\text{set}, V_3]\,[V_3, V_1]\,[V_1, V_2],\\
\rho_{11} &= [V_I, V\text{set}]\,[V\text{set}, V_3]\,[V_3, V_0]\,[V_0, V_1],\\
\rho_{12} &= [V_I, V\text{set}]\,[V\text{set}, V_3]\,[V_3, V_0]\,[V_0, V_1]\,[V_1, V_2],\\
\rho_{13} &= [V_I, V\text{set}]\,[V\text{set}, V_3]\,[V_3, V_0]\,[V_0, V_2],\\
\rho_{14} &= [V_I, V\text{set}]\,[V\text{set}, V_0]\,[V_0, V_1]\,[V_1, V_2],\\
\rho_{15} &= [V_I, V\text{set}]\,[V\text{set}, V_0]\,[V_0, V_2],\\
\rho_{16} &= [V_I, V\text{set}]\,[V\text{set}, V_1]\,[V_1, V_2]\,[V_2, V\text{end}],\\
\rho_{17} &= [V_I, V\text{set}]\,[V\text{set}, V_0]\,[V_0, V_1]\,[V_1, V_2]\,[V_2, V\text{end}],\\
\rho_{18} &= [V_I, V\text{set}]\,[V\text{set}, V_0]\,[V_0, V_2]\,[V_2, V\text{end}],\\
\rho_{19} &= [V_I, V\text{set}]\,[V\text{set}, V_3]\,[V_3, V_1]\,[V_1, V_2]\,[V_2, V\text{end}],\\
\rho_{20} &= [V_I, V\text{set}]\,[V\text{set}, V_3]\,[V_3, V_2]\,[V_2, V\text{end}],\\
\rho_{21} &= [V_I, V\text{set}]\,[V\text{set}, V_3]\,[V_3, V_0]\,[V_0, V_2]\,[V_2, V\text{end}],\\
\rho_{22} &= [V_I, V\text{set}]\,[V\text{set}, V_3]\,[V_3, V_0]\,[V_0, V_1]\,[V_1, V_2]\\
&\quad \times [V_2, V\text{end}].
\end{aligned} \tag{18}$$

If we use SMC, we must verify 22 paths: $\rho_1, \ldots, \rho_{22}$. But we use TMC only to verify $\rho_{16}, \ldots, \rho_{22}$. Therefore, the time cost by SMC is 3.14× with TMC in this case.

## 5. Correctness Proof

In this section, we formally prove that the TMC algorithm is both sound and complete.

*5.1. The Logic of Algorithm.* $M \vDash_T G \Leftrightarrow$ for all $\rho_T$ in $G$, for all $\sigma$ in $M$ of the same length, and $(M, \sigma) \vDash_T (G, \rho_T) \Leftrightarrow$ for all $\rho_T$ in $G$ and for all $\sigma$ in $M$ of the same length,

$$(M, \sigma) \vDash_{\text{ant}} (G, \rho_T) \Longrightarrow (M, \sigma) \vDash_{\text{cons}} (G, \rho_T). \tag{19}$$

By analyzing algorithm TMC, in the end, we use SMC$(M, G')$. Consider

$$\text{SMC}(M, G') \text{ returns true} \Longrightarrow M \vDash G'. \tag{20}$$

It indicates that we can only get

$$\begin{aligned}
&\forall \rho \text{ in } G, \quad \forall \sigma \text{ in } M \text{ of the same length},\\
&(M, \sigma) \vDash_{\varphi_T^*} (G, \rho) \Longrightarrow (M, \sigma) \vDash_{\text{cons}} (G, \rho).
\end{aligned} \tag{21}$$

But we are aiming at (according to (19))

$$\begin{aligned}
&\forall \rho_T \text{ in } G, \quad \forall \sigma \text{ in } M \text{ of the same length},\\
&(M, \sigma) \vDash_{\text{ant}} (G, \rho_T) \Longrightarrow (M, \sigma) \vDash_{\text{cons}} (G, \rho_T).
\end{aligned} \tag{22}$$

Then, if we want to prove TMC is correct, we must prove that (according to (21) and (22))

$$\begin{aligned} &\forall \rho_T \text{ in } G, \quad \forall \sigma \text{ in } M \text{ of the same length,} \\ &(M,\sigma) \vDash_{\text{ant}} (G,\rho_T) \Longrightarrow (M,\sigma) \vDash_{\varphi_T^*} (G,\rho_T). \end{aligned} \tag{23}$$

*5.2. The TMC Algorithm Is Sound.* According to Definition 17, we get the following lemma.

**Lemma 19.** *For a terminal assertion graph $G = (V, v_0, E, ant, cons, E_T)$ and a model $M = (pre, post)$, let $[\varphi_1, \varphi_2, \varphi_3, \ldots]$ be the backward simulation sequence towards $E_T \subseteq E$. Then for any $n \geq 2$, for all finite terminal paths $\rho_T$, and all finite traces $\sigma$ of length $n+1$, such that*

(1) $\rho_T[n+1] \in E_T$

(2) *when $\sigma^2 \vDash_{ant} \rho_T^2$ and $|\sigma^2| = |\rho_T^2| = n$, we have $\sigma[i] \in \varphi_{n+1-i}(\rho_T[i])$ $(i = 2, 3, \ldots, n)$ (note: $\rho_T^2$ represents all paths of length $n$ which are obtained by traversing backwards starting from the terminal edge. For example, if $\rho_T = e_1 e_2 \cdots e_n e_{n+1}$ and $e_{n+1} \in E_T$, then $\rho_T^2 = e_2 e_3 \cdots e_n e_{n+1}$, $\sigma^2$ represents all traces of length $n$ obtained by traversing backwards starting from the last state of $\sigma$)*

(3) *when $\sigma \vDash_{ant} \rho_T$, we have $\sigma[1] \in \varphi_n(\rho_T[1])$, $\sigma[n+1] \in \varphi_1(\rho_T[n+1])$.*

*Proof* (mathematical induction). Suppose $|\rho_T| = |\sigma| = k+1$.

*First Step* ($k = 2$). (1) $\rho_T[3] \in E_T$ since $\rho_T$ is a terminal path.

(2) Let us prove $\sigma[2] \in \varphi_1(\rho_T[2])$:

$$\sigma^2 \vDash_{\text{ant}} \rho_T^2 \Longrightarrow \sigma[2] \in \text{ant}(\rho_T[2]), \quad \sigma[3] \in \text{ant}(\rho_T[3]), \tag{24}$$

$$\sigma[2] \in \text{pre}(\sigma[3]) \overset{(24)}{\Longrightarrow} \sigma[2] \in \text{pre}(\text{ant}(\rho_T[3])). \tag{25}$$

According to (24)-(25),

$$\sigma[2] \in \text{ant}(\rho_T[2]) \cap \text{pre}(\text{ant}(\rho_T[3])) \subseteq \varphi_1(\rho_T[2]). \tag{26}$$

(3) Let us prove $\sigma[1] \in \varphi_2(\rho_T[1])$, $\sigma[3] \in \varphi_1(\rho_T[3])$:

$$\sigma[1] \in \text{pre}(\sigma[2]) \overset{(26)}{\Longrightarrow} \sigma[1] \in \text{pre}(\varphi_1(\rho_T[2])) \tag{27}$$

$$\sigma \vDash_{\text{ant}} \rho_T \Longrightarrow \sigma[1] \in \text{ant}(\rho_T[1]). \tag{28}$$

According to (27)-(28),

$$\begin{aligned} \sigma[1] &\in \text{ant}(\rho_T[1]) \cap \text{pre}(\varphi_1(\rho_T[2])) \\ &\subseteq \varphi_1(\rho_T[1]) \\ &\cup \left( \bigcup_{e^+ \in \text{out}(\rho_T[1])} (\text{pre}(\varphi_1(e^+)) \cap \text{ant}(\rho_T[1])) \right) \\ &= \varphi_2(\rho_T[1]), \end{aligned} \tag{29}$$

$$\sigma \vDash_{\text{ant}} \rho_T \Longrightarrow \sigma[3] \in \text{ant}(\rho_T[3]) = \varphi_1(\rho_T[3]).$$

Then, the conclusions (1), (2), and (3) are true for $k = 2$.

*Second Step*. We already saw that the conclusions are true for $k = 2$. Assume this, for an arbitrary $n$, the above lemma is true for $k = n$, then we get the following.

($\sigma$ represents all traces of length $n+1$ and $\rho_T$ represents all terminal paths of the same length) (1) $\rho_T[n+1] \in E_T$. (2) When $\sigma^2 \vDash_{\text{ant}} \rho_T^2$, we have $\sigma[i] \in \varphi_{n+1-i}(\rho_T[i])$ $(i = 2, 3, \ldots, n)$ (where $\rho_T^2$ represents all paths of length $k$ obtained by traversing backwards starting from the terminal edge; the terminal edge is the last edge of the terminal path; $\sigma^2$ is similar), (3) when $\sigma \vDash_{\text{ant}} \rho_T$, we have $\sigma[1] \in \varphi_n(\rho_T[1])$, $\sigma[n+1] \in \varphi_1(\rho_T[n+1])$.

*Third Step* (let us derive the lemma is true for $k = n+1$ from above assumption). We must consider all terminal paths $\rho_T$ of length $n+2$ and all traces of the same length.

(1) $\rho_T[n+2] \in E_T$ since $\rho_T$ is a terminal path.

(2) We must prove $\sigma^2 \vDash_{\text{ant}} \rho_T^2 \Rightarrow \sigma[i] \in \varphi_{n+2-i}(\rho_T[i])$ $(i = 2, 3, \ldots, n, n+1)$ at this step, $|\rho_T^2| = |\sigma^2| = n+1$

$$\sigma^2 \vDash_{\text{ant}} \rho_T^2 \Longrightarrow \sigma^3 \vDash_{\text{ant}} \rho_T^3, \tag{30}$$

where $\rho_T^3$ represents all paths of length $n$ obtained by traversing starting from the terminal edge and $\sigma^3$ is obtained by deleting the first two states of $\sigma$, $|\sigma| = n+2 \Rightarrow |\sigma^3| = n$, and $|\rho_T| = n+2 \Rightarrow |\rho_T^3| = n$. By the induction hypothesis we have the following conclusion:

$$\begin{aligned} \sigma^3 \vDash_{\text{ant}} \rho_T^3 \Longrightarrow \sigma[i] \in \varphi_{n+1-i}(\rho_T[i]) \subseteq \varphi_{n+2-i}(\rho_T[i]), \\ (i = 3, 4, \ldots, n, n+1). \end{aligned} \tag{31}$$

According to (31),

$$\sigma[3] \in \varphi_{n-1}(\rho_T[3]), \tag{32}$$

$$\sigma[2] \in \text{pre}(\sigma[3]) \overset{(32)}{\Longrightarrow} \sigma[2] \in \text{pre}(\varphi_{n-1}(\rho_T[3])), \tag{33}$$

$$\begin{aligned} \sigma[2] &\in \text{ant}(\rho_T[2]) \overset{(33)}{\Longrightarrow} \sigma[2] \in \text{ant}(\rho_T[2]) \\ &\cap \text{pre}(\varphi_{n-1}(\rho_T[3])) \\ &\subseteq \varphi_{n-1}(\rho_T[2]) \\ &\cup \left( \bigcup_{e^+ \in \text{out}(\rho_T[2])} (\text{pre}(\varphi_{n-1}(e^+)) \cap \text{ant}(\rho_T[2])) \right) \\ &= \varphi_n(\rho_T[2]), \end{aligned} \tag{34}$$

namely, $\sigma^2 \vDash_{\text{ant}} \rho_T^2 \Rightarrow \sigma[i] \in \varphi_{n+2-i}(\rho_T[i])$ $(i = 2, 3, \ldots, n, n+1)$.

(3) Consider

$$\sigma \vDash_{\text{ant}} \rho_T \Longrightarrow \sigma[1] \in \text{ant}(\rho_T[1]), \tag{35}$$

$$\sigma \vDash_{\text{ant}} \rho_T \Longrightarrow \sigma^2 \vDash_{\text{ant}} \rho_T^2 \overset{(34)}{\Longrightarrow} \sigma[2] \in \varphi_n(\rho_T[2]), \tag{36}$$

$$\sigma[1] \in \text{pre}(\sigma[2]) \overset{(36)}{\Longrightarrow} \sigma[1] \in \text{pre}(\varphi_n(\rho_T[2])). \tag{37}$$

According to (35)–(37),

$$\begin{aligned}\sigma[1] &\in \operatorname{ant}(\rho_T[1]) \cap \operatorname{pre}(\varphi_n(\rho_T[2])) \\ &\subseteq \varphi_n(\rho_T[1]) \\ &\cup \left( \bigcup_{e^+ \in \operatorname{out}(\rho_T[1])} (\operatorname{pre}(\varphi_n(e^+)) \cap \operatorname{ant}(\rho_T[1])) \right) \\ &= \varphi_{n+1}(\rho_T[1]), \\ \sigma \vDash_{\text{ant}} \rho_T &\Longrightarrow \sigma[n+2] \in \operatorname{ant}(\rho_T[n+2]) = \varphi_1(\rho_T[n+2])\end{aligned} \tag{38}$$

which exactly means that the lemma holds for $k = n + 1$. Therefore, the lemma is true for all $n$ starting with 2. □

The conclusions of Lemma 19 have been used to prove Theorem 20. At the same time, Lemma 19 gives the conclusion that the backward simulation sequence $\varphi_n$ is a convergent sequence. If the length of the path is $n$, then the simulation sequence of the $i$th edges will reach the fixed point through at least $n + 1 - i$ iterations.

Suppose $\varphi_T^*$ is the fix-point of the backward simulation sequence based on the terminal set $E_T$; after replacing the label ant of each edge with the corresponding fix-point $\varphi_T^*$, the resulting assertion graph is denoted by $G'$.

**Theorem 20.** *Consider*

$$M \vDash_T G \Longleftrightarrow M \vDash_T G'. \tag{$*$}$$

*Proof.* According to Section 5.1 (the logic of algorithm), we know $(*)$ holds equivalent to the following $(**)$. For any $n \geq 2$, $\rho_T$ represents all terminal paths of length $n$ in assertion graph $G$ ($\rho_T$ is also the terminal path in assertion graph $G'$) and $\sigma$ represents all traces of the same length in model $M$. Consider

$$\sigma \vDash_{\text{ant}} \rho_T \Longleftrightarrow \sigma \vDash_{\varphi_T^*} \rho_T. \tag{$**$}$$

The direction $\Leftarrow$ is obvious, since $\varphi_T^*(e) \subseteq \operatorname{ant}(e)$ for any $e \in E$.

The direction $\Rightarrow$: when $\sigma \vDash_{\text{ant}} \rho_T$, according to Lemma 19,

$$\sigma[i] \in \varphi_T^*(\rho_T[i]) \quad (i = 1, 2, \ldots, n); \tag{39}$$

therefore, $\sigma \vDash_{\text{ant}} \rho_T \Rightarrow \sigma \vDash_{\varphi_T^*} \rho_T$. □

$G'$ can be obtained through a series of transformations on $G$. Theorem 20 guarantees that these transformation operations will not change the model's features on terminal satisfiability.

**Theorem 21.** *For any terminal assertion graph $G = (V, v_0, E, ant, cons, E_T)$ and any model $M$,*

$$TMC(M, G) \text{ returns true} \Longrightarrow M \vDash_T G. \tag{40}$$

*Proof* ($G'$ is defined in Theorem 20). Consider

$$\begin{aligned}&\text{TMC returns true} \\ &\Longleftrightarrow \text{SMC}(M, G') \text{ returns true} \\ &\Longrightarrow M \vDash G' \quad (\text{SMC is complete } [3, 8]) \\ &\Longrightarrow M \vDash_T G' \quad (\text{Theorem 16}) \\ &\Longleftrightarrow M \vDash_T G \quad (\text{Theorem 20}).\end{aligned} \tag{41}$$

Theorem 21 proves that the TMC algorithm is sound. □

*5.3. The TMC Algorithm Is Complete.* Now let us prove the completeness of the algorithm. First, we need the following lemma.

**Lemma 22** (see [3, 8]). *For any $n \geq 1$, $e \in E$, and any $s \in \psi_n(e)$, there is a finite initial path $\rho$ and a finite trace $\sigma$ of some length $l \leq n$ such that*

$$\rho[l] = e, \qquad \sigma[l] = s, \qquad \sigma \vDash_{ant} \rho. \tag{42}$$

Lemma 22 provides a counterexample for the proof of Theorem 25. We can find a path $\rho$ and a trace $\sigma$ satisfying $\sigma \vDash_{\text{ant}} \rho$ but $\sigma \nvDash_{\text{cons}} \rho$ when TMC returns false. $[\psi_1, \psi_2, \ldots, \psi_n, \ldots]$ is the simulation sequence used in SMC algorithm; for more, please read [3, 8].

**Lemma 23** (see [3, 8]). *For all $n \geq 1$ and $e \in E$,*

$$\psi_n(e) \subseteq \psi_{n+1}(e), \qquad \psi_n(e) \subseteq ant(e). \tag{43}$$

According to the results of Lemma 23, we can conclude that $\psi_n(e) \subseteq \varphi_T^*(e)$ when $\operatorname{ant}(e) = \varphi_T^*(e)$.

**Lemma 24.** *For a terminal assertion graph $G = (V, v_0, E, ant, cons, E_T)$ and a model $M = (pre, post)$, let $[\varphi_1, \varphi_2, \varphi_3, \ldots]$ be the backward simulation sequence towards $E_T \subseteq E$. Then, for any $n \geq 1$, for any $e \in E$, and any state $s \in \varphi_n(e)$, there is a finite path $\rho_e$ which starts from e and a finite trace $\sigma_s$ which starts from s of some length $l$ ($1 \leq l \leq n + 1$) such that*

$$\rho_e[l] \in E_T, \qquad \sigma_s \vDash_{ant} \rho_e. \tag{44}$$

*Proof* (mathematical induction). $k$ is a subscripted variable of the simulation sequence $\varphi$.

(1) ($k = 1$): when $e \in E_T$, we have $\varphi_1(e) = \operatorname{ant}(e)$; for all $s \in \varphi_1(e)$, there is a path $\rho_e = [e]$ of length 1 and a trace $\sigma_s = [s]$ such that

$$\rho_e[1] \in E_T, \qquad \sigma_s \vDash_{\text{ant}} \rho_e. \tag{45}$$

When $e \in E - E_T$, we have two cases.

*Case 1.* $\operatorname{out}(e) \cap E_T = \emptyset$, $\varphi_1(e) = \emptyset$, the conclusion is obvious.

**Input**: the model $M$ and terminal assertion graph $G$.
**Output**: true: model terminally satisfies the terminal assertion graph.
false: model doesn't satisfy the terminal assertion graph.
(1) repeat
(2) $G_{\text{old}} := G$;
(3) $\varphi_T^*$:= the fix-point of the backward simulation sequence towards $E_T$;
(4) $G$:= replace the antecedent function in $G$ with $\varphi_T^*$;
(5) until $G = G_{\text{old}}$;
(6) SMC$(M, G)$;
(7) end.

Algorithm 1: TMC$(M, G)$.

*Case 2.* $\text{out}(e) \cap E_T \neq \emptyset$ and $\varphi_1(e) \neq \emptyset$; according to Definition 17:

$$\forall s \in \varphi_1(e), \quad \exists e^+ \in \text{out}(e) \cap E_T,$$
$$\text{s.t. } s \in \text{ant}(e) \cap \text{pre}\left(\text{ant}\left(e^+\right)\right) \Longrightarrow s \in \text{ant}(e), \tag{46}$$
$$s \in \text{pre}\left(\text{ant}\left(e^+\right)\right);$$
$$\overset{(46)}{\Longrightarrow} \exists s' \in \text{ant}\left(e^+\right), \quad \text{s.t. } s \in \text{pre}\left(s'\right), \tag{47}$$

namely, there is a path $\rho_e = [ee^+]$ of length 2 and a trace $\sigma_s = [ss']$ such that

$$\rho_e[2] = e^+ \in E_T, \qquad \sigma_s \vDash_{\text{ant}} \rho_e. \tag{48}$$

(2) Suppose the lemma is true for $k = n$ ($n$ is an arbitrary integer), we can get for all $e \in E$, for all $s \in \varphi_n(e)$, there is a finite path $\rho_e$ starting from $e$ and a trace $\sigma_s$ starting from $s$ of some length $l$ $(1 \leq l \leq n+1)$ such that

$$\rho_e[l] \in E_T, \qquad \sigma_s \vDash_{\text{ant}} \rho_e. \tag{49}$$

(3) Suppose $k = n+1$ for all $e \in E$ and for all $s \in \varphi_{n+1}(e)$, according to Definition 17, we also have two cases.

*Case 1.* $s \in \varphi_n(e)$; the lemma is obviously true according to induction hypothesis.

*Case 2.* Consider

$$\exists e^+ \in \text{out}(e), \quad \text{s.t. } s \in \text{pre}\left(\varphi_n\left(e^+\right)\right) \cap \text{ant}(e)$$
$$\Longrightarrow s \in \text{ant}(e), \quad s \in \text{pre}\left(\varphi_n\left(e^+\right)\right); \tag{50}$$
$$\overset{(50)}{\Rightarrow} \exists s' \in \varphi_n\left(e^+\right) \quad \text{s.t. } s \in \text{pre}\left(s'\right) \tag{51}$$

by the induction hypothesis, there exists a path $\rho_{e^+}$ of length $l$ $(1 \leq l \leq n+1)$ which starts from $e^+$ and a trace $\sigma_{s'}$ which starts from $s'$ such that

$$\rho_{e^+}[l] \in E_T, \qquad \sigma_{s'} \vDash_{\text{ant}} \rho_{e^+}. \tag{52}$$

According to (51), there is a path $\rho_e = (e : \rho_{e^+})$ of length $l+1$ and a trace $\sigma_s = (s : \sigma_{s'})$ of the same length such that

$$\rho_e[l+1] \in E_T, \qquad \sigma_s \vDash_{\text{ant}} \rho_e. \tag{53}$$

(1), (2), and (3) show that the lemma is true. □

Lemma 24 ensures that the counterexample path which we find is the terminal path.

**Theorem 25.** *Consider*

$$TMC(M, G) \text{ returns } false \Longrightarrow M \nVdash_T G. \tag{54}$$

*Proof.* Consider

$$\text{TMC}(M, G) \text{ returns false} \Longrightarrow \text{SMC}\left(M, G'\right) \text{ returns false}; \tag{55}$$

according to the SMC algorithm, there exists an edge $e_1 \in E$ such that $\psi^*(e_1) \not\subseteq \text{cons}(e_1)$. Let $s_1 \in \psi^*(e_1) - \text{cons}(e)$, according to the Lemma 22, there exists a finite path $\rho_1$ of length $l_1$ and a trace $\sigma_1$ of the same length such that

$$\rho_1[l_1] = e_1, \qquad \sigma_1[l_1] = s_1,$$
$$\sigma_1 \vDash_{\text{ant}} \rho_1, \quad \text{but } \sigma_1 \nvDash_{\text{cons}} \rho_1. \tag{56}$$

When $e_1 \in E_T$, $\rho_1$ is the terminal path. Namely, there exists an initial terminal path $\rho_1$ and a trace $\sigma_1$ of the same length such that $\sigma_1 \vDash_{\text{ant}} \rho_1$, $\sigma_1 \nvDash_{\text{cons}} \rho_1$; according to Definition 14, we get $M \nVdash_T G$.

When $e_1 \in E - E_T$, $\rho_1$ is not the terminal path, we know $s_1 \in \psi^*(e_1)$, $s_1 \notin \text{cons}(e_1)$. By analyzing Algorithm 1, we find that the terminal assertion graph $G$ is already updated to $G'$ when we call the SMC, and every edge in $G'$ is labeled with $\varphi_T^*(e)/\text{cons}(e)$. In $G'$, for all $e \in E$, $\text{ant}(e) = \varphi_T^*(e)$. According to Lemma 23, we get $s_1 \in \psi^*(e_1) \subseteq \varphi_T^*(e_1)$, $s_1 \notin \text{cons}(e_1)$. According to Lemma 24, there exists a finite path $\rho_{e_1}$ of length $l_2$ which starts from $e_1$ and a trace $\sigma_{s_1}$ of the same length which starts from $s_1$ such that

$$\rho_{e_1}[l_2] \in E_T, \qquad \sigma_{s_1} \vDash_{\text{ant}} \rho_{e_1}. \tag{57}$$

Since $\rho_1[l_1] = \rho_{e_1}[1] = e_1$, $\rho_1$ and $\rho_{e_1}$ can be spliced into one path $\rho$ of length $l_1 + l_2 - 1$. Since $\sigma_1[l_1] = \sigma_{s_1}[1] = s_1$ is true, $\sigma_1$ and $\sigma_{s_1}$ can be spliced into one trace $\sigma$ whose length is $l_1 + l_2 - 1$. $\rho$ is a terminal path and such that

$$\rho[l_1 + l_2 - 1] \in E_T, \qquad \sigma \vDash_{\text{ant}} \rho, \qquad \sigma \nvDash_{\text{cons}} \rho. \tag{58}$$

According to Definition 14, we get $M \nVdash_T G$. □

Theorem 25 proves that the TMC algorithm is complete.

## 6. Conclusions

This paper has presented a theoretical and experimental study of the TMC process and related concepts, such as terminal assertion graph, terminal path, and terminal satisfiability. Under the basis of the concept mentioned above, to improve the efficiency of the method SMC and solve its limitation, this paper presented an algorithm TMC. The approach TMC is explained and discussed thoroughly in the body of the paper. Then, we use the hardware circuit round-robin arbiter to specify that TMC can be used in industry successfully. In the end, this paper proves that our approach is sound and complete. The method outlined here can be used to deal with finite specifications. It remains to be determined whether our approach will be suitable for infinite specifications.

## Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

## Acknowledgments

This work was supported by NSFC (nos. 60973016 and 61272175), 973 Program (no. 2010CB328004), and RFSGCC (no. SGSCDK00DWKJ1300510). The authors also would like to express their sincere thanks to Stephen Akobre and Lixiu Lin for useful discussions.

## References

[1] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*, MIT press, 1999.

[2] S. Hazelhurst and C.-J. H. Seger, "Simple theorem prover based on symbolic trajectory evaluation and BDD's," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 4, pp. 413–422, 1995.

[3] S. Hazelhurst and C.-J. H. Seger, "Symbolic trajectory evaluation," in *Formal Hardware Verification*, pp. 3–78, Springer, New York, NY, USA, 1997.

[4] S. Owre, S. Rajan, J. M. Rushby, N. Shankar, and M. Srivas, "Pvs: combining specification, proof checking, and model checking," in *Computer Aided Verification*, pp. 411–414, Springer, 1996.

[5] C.-J. H. Seger and R. E. Bryant, "Formal verification by symbolic evaluation of partially-ordered trajectories," *Formal Methods in System Design*, vol. 6, no. 2, pp. 147–189, 1995.

[6] M. Pandey, R. Raimi, R. E. Bryant, and M. S. Abadir, "Formal verification of content addressable memories using symbolic trajectory evaluation," in *Proceedings of the 34th Design Automation Conference*, pp. 167–172, June 1997.

[7] M. Pandey and R. E. Bryant, "Exploiting symmetry when verifying transistor-level circuits by symbolic trajectory evaluation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 7, pp. 918–935, 1999.

[8] J. Yang and C.-J. H. Seger, "Introduction to generalized symbolic trajectory evaluation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 3, pp. 345–353, 2003.

[9] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Progress on the state explosion problem in model checking," in *Informatics*, pp. 176–194, Springer, 2001.

[10] C.-T. Chou, "The mathematical foundation of symbolic trajectory evaluation," in *Computer Aided Verification*, pp. 196–207, Springer, 1999.

[11] C. Baier and J.-P. Katoen, *Principles of Model Checking*, MIT Press, Cambridge, Mass, USA, 2008.

[12] Y. Ramakrishna, C. Ramakrishnan, I. Ramakrishnan, S. A. Smolka, T. Swift, and D. S. Warren, "Efficient model checking using tabled resolution," in *Computer Aided Verification*, pp. 143–154, Springer, 1997.

[13] P. Gammie and R. Van Der Meyden, "Mck: model checking the logic of knowledge," in *Computer Aided Verification*, pp. 479–483, Springer, 2004.

[14] B. Bentley, "High level validation of next-generation microprocessors," in *Proceedings of the 7th IEEE International High-Level Design Validation and Test Workshop*, pp. 31–35, IEEE, 2002.

[15] A. Platzer, E. M. Clarke, and P. Zuliani, "Bayesian statistical model checking with application to Simulink/Stateflow verification," in *Proceedings of the 13th ACM international conference on Hybrid systems*, pp. 243–252, 2010.

[16] E. M. Clarke and P. Zuliani, "Statistical model checking for cyber-physical systems," in *Automated Technology for Verification and Analysis*, pp. 1–12, Springer, 2011.

[17] E. M. Clarke, O. Grumberg, and D. E. Long, "Model checking and abstraction," *ACM Transactions on Programming Languages and Systems*, vol. 16, no. 5, pp. 1512–1542, 1994.

[18] W. Visser, K. Havelund, G. Brat, S. Park, and F. Lerda, "Model checking programs," *Automated Software Engineering*, vol. 10, no. 2, pp. 203–232, 2003.

[19] A. J. Hu, J. Casas, and J. Yang, "Reasoning about gste assertion graphs," in *Correct Hardware Design and Verification Methods*, pp. 170–184, Springer, 2003.

[20] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang, "Symbolic model checking: $10^{20}$ states and beyond," *Information and Computation*, vol. 98, no. 2, pp. 142–170, 1992.

[21] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu, "Bounded model checking," *Advances in Computers*, vol. 58, pp. 118–149, 2003.

[22] E. Clarke, A. Biere, R. Raimi, and Y. Zhu, "Bounded model checking using satisfiability solving," *Formal Methods in System Design*, vol. 19, no. 1, pp. 7–34, 2001.

[23] J. Yang and C. J. H. Seger, "Generalized symbolic trajectory evaluation," Intel SCL Tech. Rep, 2000.

[24] Y. Li, N. Zeng, W. N. N. Hung, and X. Song, "Enhanced symbolic simulation of a round-robin arbiter," in *Proceedings of the 29th IEEE International Conference on Computer Design (ICCD '11)*, pp. 102–107, November 2011.

[25] E. S. Shin, V. J. Mooney III, and G. F. Riley, "Round-robin arbiter design and generation," in *Proceedings of the 15th International Symposium on System Synthesis*, pp. 243–248, October 2002.

[26] Y. Li, S. Panwar, and H. Jonathan Chao, "On the performance of a Dual Round-Robin switch," in *Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies*, pp. 1688–1697, April 2001.