Hindawi Publishing Corporation
Journal of Applied Mathematics
Volume 2013, Article ID 709071, 7 pages
http://dx.doi.org/10.1155/2013/709071

Research Article

A Transformation-Based Approach to Implication of GSTE Assertion Graphs

Guowu Yang,[1] William N. N. Hung,[2] Xiaoyu Song,[3] and Wensheng Guo[1]

[1] *School of Computer Science and Engineering, University of Electronic Science and Technology Chengdu, Sichuan 610054, China*
[2] *Synopsys Inc., Mountain View, CA 94043, USA*
[3] *Department of ECE, Portland State University, Portland, Oregon, USA*

Correspondence should be addressed to Guowu Yang; guowuy@126.com
and William N. N. Hung; william_hung@alumni.utexas.net

Received 8 February 2013; Accepted 13 May 2013

Academic Editor: Guiming Luo

Generalized symbolic trajectory evaluation (GSTE) is a model checking approach and has successfully demonstrated its powerful capacity in formal verification of VLSI systems. GSTE is an extension of symbolic trajectory evaluation (STE) to the model checking of ω-regular properties. It is an alternative to classical model checking algorithms where properties are specified as finite-state automata. In GSTE, properties are specified as assertion graphs, which are labeled directed graphs where each edge is labeled with two labeling functions: antecedent and consequent. In this paper, we show the complement relation between GSTE assertion graphs and finite-state automata with the expressiveness of regular languages and ω-regular languages. We present an algorithm that transforms a GSTE assertion graph to a finite-state automaton and vice versa. By applying this algorithm, we transform the problem of GSTE assertion graphs implication to the problem of automata language containment. We demonstrate our approach with its application to verification of an FIFO circuit.

1. Introduction

Generalized symbolic trajectory evaluation (GSTE) [1–4] is a model checking approach which was originally developed at Intel and has successfully demonstrated its powerful capacity in formal verification of VLSI systems [1–7]. GSTE is extended from the lattice-based symbolic trajectory evaluation (STE) [8, 9] which can handle large industrial designs [10–13]. The STE theory consists of a simple specification language, a simulation-based model checking algorithm, and a powerful quaternary abstraction algorithm. Though STE is very efficient in automatic verification of large-scale industrial hardware designs at both gate and transistor levels, it has a limited specification language which only allows the specification of properties over finite time intervals.

In GSTE, all ω-regular properties can be expressed and verified with space and time efficiencies comparable with STE. Assertion graphs are introduced in GSTE as an extension of STE specification language and are the key to the usability of GSTE. An assertion graph is a labeled directed graph where each edge is labeled with two labeling functions: antecedent and consequent [3]. The existing GSTE theory provides an efficient model checking procedure for verifying that a circuit obeys an assertion graph as well as techniques based on abstract interpretation to combat state space explosion [3, 7].

How to establish that one specification implies another is a fundamental problem in formal verification. Hu et al. [6, 7] proposed some algorithms to decide whether one assertion graph implies another through building monitor circuits. Yang and Seger [2] verified assertion graphs through manually refining assertion graphs. Yang et al. [14] gave some conditions under which the assertion graphs implication is decided without reachability analysis if two assertion graphs have the same graph structure. Sebastiani et al. [15] gave the complement relation between an assertion graph and a finite-state automaton. An assertion graph with n states resulted in a nondeterministic automaton with $3 * n$ states.

In this paper, we consider both finite-state automata that accept regular languages and finite-state ω-automata with

Büchi acceptance conditions that accept ω-regular languages. For a finite-state automaton over finite words, acceptance is defined according to the last state visited by a run of the automaton; while for a finite-state automaton with Büchi acceptance conditions, there is no such "the last state" and acceptance is defined according to the set of states that a run visits infinitely often [16].

We present a theory on the complement relation between GSTE assertion graphs and finite-state automata and an algorithm which automatically transforms assertion graphs to their equivalent finite-state automata, and vice versa. For an assertion graph G, we construct a corresponding finite-state automaton M_G such that $L^*(G) = \Sigma^* - L^*(M_G)$ and $L^\omega(G) = \Sigma^\omega - L^\omega(M_G)$, the number of states in M_G is only twice as many as that in G, and a deterministic assertion graph is transformed to a deterministic automaton. On the other hand, for an arbitrary finite-state automaton M, we construct a corresponding assertion graph G_M such that $L^*(G_M) = \Sigma^* - L^*(M)$ and $L^\omega(G_M) = \Sigma^\omega - L^\omega(M)$, the number of states in G_M is the same as that in M, and a deterministic automaton is transformed to a deterministic assertion graph.

We apply the proposed theory and algorithm to the GSTE assertion graphs implication problem by transforming this problem to the automata language containment problem. Since, in many cases, assertion graphs are nondeterministic, their corresponding finite-state automata are also nondeterministic. This may cause exponential state space blowups in checking automata language containment [16–22]. We avoid such blowups in determining the implication relation between a nondeterministic assertion graph and its refinement derived following the refinement strategies given in [2], by transforming the implication relation between nondeterministic assertion graphs to the implication relation between deterministic assertion graphs without adding any states. This application is illustrated with a case study for verifying correctness properties of an FIFO circuit.

The remainder of this paper is organized as follows. In Section 2, we introduce the preliminaries of finite-state automata and GSTE assertion graphs. In Section 3, we present the transformation algorithm and prove its correctness. In Section 4, we propose an application of the transformation algorithm to determine the implication relation between GSTE assertion graphs and demonstrate its effectiveness with a case study. We conclude in Section 5.

2. Preliminaries

In this section, we give the definitions of finite-state automata and GSTE assertion graphs and of their finite and infinite languages. We also show that the fairness edge constraints for GSTE assertion graphs [1] are equivalent to the fairness state constraints.

Definition 1. A finite-state automaton M is a tuple $M = \langle Q, \Sigma, E, LF, q_0, F\rangle$, where Q is a finite set of states, Σ is a finite alphabet, E is a set of directed edges over Q, $LF : E \mapsto 2^\Sigma$ is the labeling function for all edges, $q_0 \in Q$ is the initial state, and F is a subset of Q whose elements are called acceptance states.

Definition 2. A finite word (or string) $X = x_1x_2\cdots x_n \in \Sigma^*$ is accepted by a finite-state automaton M (denoted as $X \vDash_* M$) if there exists a path $\rho = q_0q_1\cdots q_n$ such that $(q_{i-1}, q_i) \in E$, $q_n \in F$, and $x_i \in LF((q_{i-1}, q_i))$, denoted as $X \vDash_{LF} \rho$.

Definition 3. An infinite word (or string) $X = x_1x_2\cdots \in \Sigma^\omega$ is accepted by a finite-state automaton M (denoted as $X \vDash_\omega M$) if there exists an infinite path $\rho = q_0q_1\cdots$ such that $(q_{i-1}, q_i) \in E$, $\mathrm{Inf}(\rho) \cap F \neq \emptyset$, where $\mathrm{Inf}(\rho)$ is the set of states occurring infinitely often in ρ, and $x_i \in LF((q_{i-1}, q_i))$, denoted as $X \vDash_{LF} \rho$.

Definition 4. The finite language $L^*(M)$ of M is $L^*(M) = \{X \in \Sigma^* \mid X \vDash_* M\}$. The infinite language $L^\omega(M)$ of M is $L^\omega(M) = \{X \in \Sigma^\omega \mid X \vDash_\omega M\}$.

Definition 5. Given a finite-state automaton $M = \langle Q, \Sigma, E, LF, q_0, F\rangle$, $\mathrm{In}(q)$ and $\mathrm{Out}(q)$ for $q \in Q$ are called the set of incoming edges of q and the set of outgoing edges of q, respectively; that is, $\mathrm{In}(q) = \{(q', q) \mid (q', q) \in E\}$ and $\mathrm{Out}(q) = \{(q, q') \mid (q, q') \in E\}$.

Definition 6. If for all $q \in Q$ and for all $e, e' \in \mathrm{Out}(q)$, $e \neq e'$ and $LF(e) \cap LF(e') = \emptyset$, M is called a deterministic finite-state automaton (DFA). Otherwise, M is called a nondeterministic finite-state automaton (NFA).

Traditionally, an NFA may have a set of initial states. The expressiveness of an NFA with multiple initial states is the same as that of an NFA with a unique initial state. The transformation is easy.

Definition 7. An assertion graph G is a tuple $\langle Q, \Sigma, E, q_0, \mathrm{ant}, \mathrm{cons}, F\rangle$, where Q is a finite set of states, Σ is a finite alphabet, E is a set of directed edges over Q, $\mathrm{ant} : E \mapsto 2^\Sigma$ and $\mathrm{cons} : E \mapsto 2^\Sigma$ are the labeling functions for all edges, $q_0 \in Q$ is the initial state, and F is a subset of Q and elements of F are called acceptance states.

Definition 8. A finite word (or string) $X = x_1x_2\cdots x_n \in \Sigma^*$ is accepted by an assertion graph G (denoted as $X \vDash_* G$) if for each finite path $\rho = q_0q_1\cdots q_n$ with $(q_{i-1}, q_i) \in E$ (where $1 \leq i \leq n$) and $q_n \in F$ such that $X \vDash_{\mathrm{ant}} \rho \Rightarrow X \vDash_{\mathrm{cons}} \rho$, where $X \vDash_{\mathrm{ant}} \rho$ denotes $x_i \in \mathrm{ant}((q_{i-1}, q_i))$ and $X \vDash_{\mathrm{cons}} \rho$ denotes $x_i \in \mathrm{cons}((q_{i-1}, q_i))$ for $1 \leq i \leq n$. An infinite word (or string) $X = x_1x_2\cdots \in \Sigma^\omega$ is accepted by G (denoted as $X \vDash_\omega G$) if for all infinite path $\rho = q_0q_1\cdots$ with $(q_{i-1}, q_i) \in E$ (where $i \geq 1$) and $\mathrm{Inf}(\rho) \cap F \neq \emptyset$ such that $X \vDash_{\mathrm{ant}} \rho \Rightarrow X \vDash_{\mathrm{cons}} \rho$. The finite language $L^*(G)$ of G is $L^*(G) = \{X \in \Sigma^* \mid X \vDash_* G\}$. The infinite language $L^\omega(G)$ of G is $L^\omega(G) = \{X \in \Sigma^\omega \mid X \vDash_\omega G\}$.

Remark 9. In [1], an assertion graph $G = \langle Q, \Sigma, E, q_0, \mathrm{ant}, \mathrm{cons}, F_e\rangle$ has a set F_e of acceptance (fair) edges. The final edge of a finite path must be in F_e and for an infinite path; at least one edge in F_e must appear in the path infinitely often. This can be transformed to the above definitions by the following construction. For any $q \in Q$, if the incoming edges $\mathrm{In}(q)$ have both acceptance edges and nonacceptance edges, q is substituted with q' and q'' such that $\mathrm{In}(q') = \mathrm{In}(q) \cap F_e$, $\mathrm{In}(q'') = \mathrm{In}(q) - \mathrm{In}(q')$; that is, the incoming

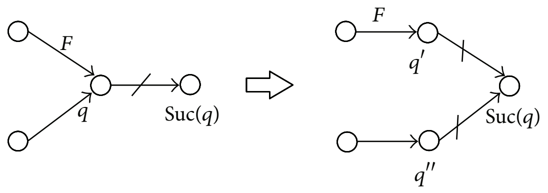

Figure 1: Fair edges to fair states.

edges of q' are all acceptance edges, the incoming edges of q'' are nonacceptance edges (See Figure 1). The outgoing edges of q' and q'' are the same as the outgoing edges of q, and the labeling function ant and cons are the same. All such $q's$ are included the acceptance state set F in the transformed assertion graph.

Definition 10. Given an assertion graph $G = \langle Q, \Sigma, E, q_0, \text{ant}, \text{cons}, F\rangle$, if for all $q \in Q$, for all $e, e' \in \text{Out}(q)$ and $e \neq e'$, $\text{ant}(e) \cap \text{ant}(e') = \emptyset$, G is called a deterministic assertion graph (DAG). Otherwise, G is called a nondeterministic assertion graph (NAG).

For any assertion graph $G = \langle Q, \Sigma, E, q_0, \text{ant}, \text{cons}, F\rangle$, we can assume that G is a nonrestarting assertion graph (i.e., $\text{In}(q_0) = \emptyset$); otherwise, we construct a nonrestarting assertion graph $G' = \langle Q', \Sigma, E', q_0, \text{ant}, \text{cons}, F'\rangle$ from G: $Q' = Q \cup \{q'\}$ (add a new state q' to Q), $\text{In}(q') = \{(q, q') \mid (q, q_0) \in E)\}$, $\text{Out}(q') = \{(q', q) \mid (q_0, q) \in E)\}$, $\text{ant}((q', q)) = \text{ant}((q_0, q))$, $\text{cons}((q', q)) = \text{cons}((q_0, q))$, $\text{ant}((q, q')) = \text{ant}((q, q_0))$, $\text{cons}((q, q')) = \text{cons}((q, q_0))$. Basically, we copy the edges and the related labeling functions ant and cons of q_0 to q' and delete the incoming edges of q_0. The other edges and labeling functions remain the same.

3. Transformations between GSTE Assertion Graphs and Finite-State Automata

In this section, we present the transformations between a GSTE assertion graph and a finite-state automaton. Given an assertion graph G, we build a finite-state automaton M_G such that $L^*(G) = \Sigma^* - L^*(M_G)$ and $L^\omega(G) = \Sigma^\omega - L^\omega(M_G)$. And if G is a deterministic assertion graph, then M_G is also a deterministic finite-state automaton. Let $M_G = \langle Q', \Sigma, E', LF, q'_0, F'\rangle$:

(1) $Q' = (Q - \{q_0\}) \times \{0, 1\} \cup \{(q_0, 0)\}$, $q'_0 = (q_0, 0)$;

(2) $E' = \{((q_1, k_1), (q_2, k_2)) \mid (q_1, q_2) \in E$; if $k_1 = 0$, then $k_2 \in \{0, 1\}$; if $k_1 = 1$, then $k_2 = 1\}$ (See Figure 2);

(3) $LF((q_1, 0), (q_2, 0)) = \text{ant}((q_1, q_2)) \cap \text{cons}((q_1, q_2))$, $LF((q_1, 0), (q_2, 1)) = \text{ant}((q_1, q_2)) \cap \neg\text{cons}((q_1, q_2))$, $LF((q_1, 1), (q_2, 1)) = \text{ant}((q_1, q_2))$;

(4) $F' = \{(q, 1) \mid q \in F\}$.

Intuitively, M_G is constructed to accept all strings (or words) that are not accepted by G. Such a string (or word) satisfies the antecedent of each edge traversed while violating the consequent of at least of such edge.

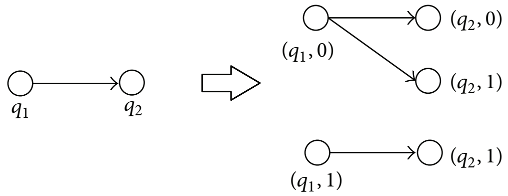

Figure 2: Assertion graph to finite-state automaton.

Theorem 11. *If G is a DAG, then M_G is a DFA.*

Proof. We prove the following two cases.

(i) For any state $(q, 1)$ in M_G and any two outgoing edges $((q, 1), (q_1, 1))$ and $((q, 1), (q_2, 1))$, $LF(((q, 1), (q_1, 1))) \cap LF(((q, 1), (q_2, 1))) = \text{ant}((q, q_1)) \cap \text{ant}((q, q_2))$ (the construction of M_G) $= \emptyset$ (the definition of DAG).

(ii) For any state $(q, 0)$ in M_G and any two outgoing edges $((q, 0), (q_1, k_1))$ and $((q, 0), (q_2, k_2))$.

(a) If $q_1 = q_2$, then $k_1 = 0, k_2 = 1$, $LF(((q, 0), (q_1, 0))) \cap LF(((q, 0), (q_1, 1))) = \text{ant}((q, q_1)) \cap \text{cons}((q, q_1)) \cap \text{ant}((q, q_1)) \cap \neg\text{cons}((q, q_1))$ (the construction of M_G) $= \emptyset$.

(b) If $q_1 \neq q_2$, then $LF(((q, 1), (q_1, 1))) \cap LF(((q, 1), (q_2, 1))) \subseteq \text{ant}((q, q_1)) \cap \text{ant}((q, q_2))$ (the construction of M_G) $= \emptyset$ (the definition of DAG).

Therefore, M_G is a DFA. □

Theorem 12. $L^*(M_G) = \Sigma^* - L^*(G)$.

Proof. First, we show $L^*(M_G) \supseteq \Sigma^* - L^*(G)$.

Suppose $X = x_1 x_2 \cdots x_n \in \Sigma^* - L^*(G)$.

Then there exists a finite path $\rho = q_0 q_1 \cdots q_n$ in G with $(q_{i-1}, q_i) \in E$, $1 \le i \le n$ and $q_n \in F$ such that $x_i \in \text{ant}((q_{i-1}, q_i))$ for $1 \le i \le n$; and $\exists l$, $1 \le l \le n$, when $1 \le h < l$, $x_h \in \text{ant}((q_{i-1}, q_i)) \cap \text{cons}((q_{i-1}, q_i))$, $x_l \in \text{ant}((q_{l-1}, q_l)) \cap \neg\text{cons}((q_{i-1}, q_i))$.

Let a path ρ' of M_G be as follows:

$$\rho' = (q_0, 0)(q_1, 0) \cdots (q_{l-0}, 0)(q_l, 1) \cdots (q_n, 1). \tag{1}$$

Then $x_i \in LF((q_{i-1}, k_{i-1}), (q_i, k_i))$, where $k_i = 0$ if $1 \le i < l$, $k_i = 1$ if $l \le i \le n$; and $(q_n, 1) \in F'$.

Therefore, $X \in L^*(M_G)$.

Second, we show $L^*(M_G) \subseteq \Sigma^* - L^*(G)$.

Suppose $X = x_1 x_2 \cdots x_n \in L^*(M_G)$.

Then there exists a path as follows:

$$\rho' = (q_0, 0)(q_1, 0) \cdots (q_{l-0}, 0)(q_l, 1) \cdots (q_n, 1), \tag{2}$$

such that $x_i \in LF((q_{i-1}, k_{i-1}), (q_i, k_i))$, where $k_i = 0$ if $1 \le i < l$, $k_i = 1$ if $l \le i \le n$; and $(q_n, 1) \in F'$.

According to the construction of M_G from G, we know that $\rho = q_0 q_1 \cdots q_n$ is a path of G; if $1 \le i < l$, $x_i \in LF((q_{i-1}, k_{i-1}), (q_i, k_i)) = LF((q_{i-1}, 0), (q_i, 0)) = \text{ant}((q_{i-1}, q_i)) \cap \text{cons}((q_{i-1}, q_i)) \subseteq \text{ant}((q_{i-1}, q_i))$.

$x_l \in LF((q_{l-1}, 0), (q_l, 1)) = \text{ant}((q_{l-1}, q_l)) \cap \neg\text{cons}((q_{l-1}, q_l))$, that is, $x_l \in \text{ant}((q_{l-1}, q_l))$, but $x_l \notin \text{cons}((q_{l-1}, q_l))$.

If $l < i \le n$, $x_i \in LF((q_{i-1}, k_{i-1}), (q_i, k_i)) = LF((q_{i-1}, 1), (q_i, 1)) = \mathrm{ant}((q_{i-1}, q_i))$.

Therefore, $X \notin L^*(G)$. □

Theorem 13. $L^\omega(M_G) = \Sigma^\omega - L^\omega(G)$.

Proof. First, we prove $L^\omega(M_G) \supseteq \Sigma^\omega - L^\omega(G)$.

Suppose $X = x_1 x_2 \cdots \in \Sigma^\omega - L^\omega(G)$.

Then there exists an infinite path $\rho = q_0 q_1 \cdots$ in G with $(q_{i-1}, q_i) \in E$ and $\mathrm{Inf}(\rho) \cap F \neq \emptyset$ such that $x_i \in \mathrm{ant}((q_{i-1}, q_i))$ for $1 \le i$; and $\exists l, 1 \le l$, $x_l \notin \mathrm{cons}((q_{l-1}, q_l))$. Let a path ρ' of M_G be $\rho' = (q_0, 0)(q_1, 0) \cdots (q_{l-1}, 0)(q_l, 1)(q_{l+1}, 1) \cdots$. This path ρ' will visit infinitely often every $(q, 1) \in F'$ if $q \in \mathrm{Inf}(\rho) \cap F$. And $x_i \in LF((q_{i-1}, k_{i-1}), (q_i, k_i))$, where $k_{i-1} = 0$ if $1 \le i \le l$, $k_i = 1$ if $l < i$.

Thus $X \in L^\omega(M_G)$.

Second, we prove $L^\omega(M_G) \subseteq \Sigma^\omega - L^\omega(G)$.

Suppose $X = x_1 x_2 \cdots \in L^\omega(M_G)$.

Then there exists an infinite path $\rho' = (q_0, 0)(q_1, 0) \cdots (q_{l-1}, 0)(q_l, 1)(q_{l+1}, 1) \cdots$ in M_G with $(q_{i-1}, q_i) \in E$ and $\mathrm{Inf}(\rho') \cap F' \neq \emptyset$ such that $x_i \in LF((q_{i-1}, k_{i-1}), (q_i, k_i))$, where $k_i = 0$ if $1 \le i < l$, $k_i = 1$ if $l \le i \le n$.

According to the construction of M_G from G, we know that $\rho = q_0 q_1 \cdots$ is a path of G, and $\mathrm{Inf}(\rho) \cap F \neq \emptyset$; $x_i \in \mathrm{ant}((q_{i-1}, q_i))$ for $1 \le i$, and $x_l \in \mathrm{ant}((q_{l-1}, q_l)) \cap \neg\mathrm{cons}((q_{l-1}, q_l))$, that is, $x_l \notin \mathrm{cons}((q_{l-1}, q_l))$.

Therefore, $X \nvDash_\omega G$, that is, $X \in \Sigma^\omega - L^\omega(G)$. □

Given a finite-state automaton $M = \langle Q, \Sigma, E, LF, q_0, F\rangle$, build an assertion graph G_M as $G_M = \langle Q, \Sigma, E, q_0, \mathrm{ant}, \mathrm{cons}, F\rangle$ where $\mathrm{ant}(e) = LF(e)$, $\mathrm{cons}(e) = \emptyset$ for all $e \in E$. This construction is the same as the construction proposed in [15].

Theorem 14. *$L^*(G_M) = \Sigma^* - L^*(M)$ and $L^\omega(G_M) = \Sigma^\omega - L^\omega(M)$.*

Proof. Directly from the construction of G_M and the language definitions in [15]. □

Theorem 15. *If M is a DFA, then G_M is a DAG.*

Proof. Directly from the construction of G_M and the deterministic definition. □

4. Application to GSTE Assertion Graphs Implication

4.1. Transforming GSTE Assertion Graphs Implication to Automata Language Containment. The complement relation between an assertion graph and a finite-state automaton can be applied to determine the implication relation between assertion graphs by transforming the GSTE assertion graphs implication to the finite-state automata language containment.

Definition 16. Given two assertion graphs G_1 and G_2, $G_1 \Rightarrow_* G_2$ if and only if $L^*(G_1) \subseteq L^*(G_2)$ and $G_1 \Rightarrow_\omega G_2$ if and only if $L^\omega(G_1) \subseteq L^\omega(G_2)$.

Let M^{*c} be an automaton whose finite language is the complement of the finite language of an automaton M; that is, $L^*(M^{*c}) = \Sigma^* - L^*(M)$. Let $M^{\omega c}$ be an automaton whose infinite language is the complement of the infinite language of an automaton M; that is, $L^\omega(M^{\omega c}) = \Sigma^\omega - L^\omega(M)$.

Theorem 17. *$G_1 \Rightarrow_* G_2$ iff $L^*((M_{G_1})^{*c} \times M_{G_2}) = \emptyset$.*
$G_1 \Rightarrow_\omega G_2$ iff $L^\omega((M_{G_1})^{\omega c} \times M_{G_2}) = \emptyset$.

Proof. $(G_1 \Rightarrow_* G_2) \Leftrightarrow L^*(G_1) \subseteq L^*(G_2) \Leftrightarrow L^*(M_{G_1}) \supseteq L^*(M_{G_2}) \Leftrightarrow L^*((M_{G_1})^{*c} \times M_{G_2}) = \emptyset$.

$(G_1 \Rightarrow_\omega G_2) \Leftrightarrow L^\omega(G_1) \subseteq L^\omega(G_2) \Leftrightarrow L^\omega(M_{G_1}) \supseteq L^\omega(M_{G_2}) \Leftrightarrow L^\omega((M_{G_1})^{\omega c} \times M_{G_2}) = \emptyset$. □

In GSTE, because of its aggressive abstraction, abstraction refinement is often necessary [2]. Verifying whether a circuit C satisfies an assertion graph G, $C \models G$, using the GSTE engine may produce false negative if G is too abstract. Thus, it is necessary to refine G into a refined assertion graph G^{ref}. If we can verify $C \models G^{\mathrm{ref}}$, then we can conclude $C \models G$. But how can we guarantee that $(C \models G^{\mathrm{ref}}) \Rightarrow (C \models G)$ if the assertion graph is complex? We must establish $G^{\mathrm{ref}} \Rightarrow G$. We can verify $G^{\mathrm{ref}} \Rightarrow G$ by transforming it to an automata language containment test using Theorem 17.

A critical difficulty in checking the language containment, $L(M_1) \subseteq L(M_2)$, is the construction of a complement finite-state automaton M_2^c of M_2. Complementing a nondeterministic automaton with n states resulted in an automaton with $2^{2^{O(n)}}$ [23]. Some optimized complementing constructions were introduced with $2^{O(n \log n)}$ [16–22]. However, the complexity is still exponential.

Although we can transform deterministic assertion graphs to deterministic finite-state automata, in many cases, assertion graphs are nondeterministic. This may cause state space blowups when checking language containment between automata corresponding to these assertion graphs. We observe that in determining the implication relation between an assertion graph and its refinements constructed using the refinement strategies in [2], we can transform the nondeterministic assertion graph G to deterministic assertion graph G_{det} by adding only one variable without adding any states and then apply the refinements to G_{det}. The requirement of our method is that the refinements applied to G_{det} will not cause nondeterminism. The refinement strategies in [2] satisfy our requirement. The language containment of deterministic finite state automata can be checked in polynomial time [19]. Therefore, the state space blowup problem can be avoided in determining the implication relation between a refined assertion graph and an original assertion graph.

For any given assertion graph $G = \langle Q, \Sigma, E, q_0, \mathrm{ant}, \mathrm{cons}, F\rangle$, let G^{ref} be a refined assertion graph of G. If G is a nondeterministic assertion graph, we add an additional variable "det" to constrain the overlapped antecedents of the outgoing edges of every state in G and the resulting assertion graph is denoted as G_{det}; that is, $G_{\mathrm{det}} = \langle Q, \Sigma \times \mathrm{det}, E, q_0, \mathrm{ant} \times \mathrm{det}, \mathrm{cons} \times \mathrm{det}, F\rangle$. The value domain of "det" is from 1 to the maximal number n that we need to distinguish the overlapped antecedents of the outgoing edges of every state in the assertion graph G, denoted as $D(\mathrm{det})$.

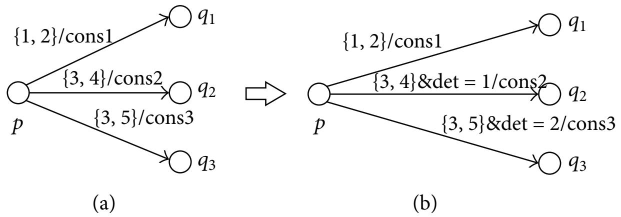

Figure 3: Assertion graph determinization.

Figure 3 illustrates the determinization process: there are three outgoing edges from state p: (p, q_1), (p, q_2), and (p, q_3). Two of these three antecedents $\mathrm{ant}((p, q_2)) = \{3, 4\}$ and $\mathrm{ant}((p, q_3)) = \{3, 5\}$ in Figure 3(a) are overlapped. So, it is a non-deterministic transition. We constrain these two antecedents by adding a variable "det" with a domain of $\{1, 2\}$ and changing $\mathrm{ant}((p, q_2))$ to $\{3, 4\} \times \{\mathrm{det} = 1\}$ and $\mathrm{ant}((p, q_3))$ to $\{3, 5\} \times \{\mathrm{det} = 2\}$ in Figure 3(b), the determinized assertion graph G_{det}. For simplification, the consequents and the antecedents which are not needed to be constrained will not be followed by $\times\mathrm{det}$.

The projection of the (finite or infinite) language of G_{det} is defined as $L^*_{\mathrm{proj}}(G_{\mathrm{det}}) = \{x_1 x_2 \cdots x_n \in \Sigma^* \mid \exists d_i \in D(\mathrm{det}), \text{s.t. } (x_1, d_1)(x_2, d_2) \cdots (x_n, d_n) \in L^*(G_{\mathrm{det}})\}$, $L^{\omega}_{\mathrm{proj}}(G_{\mathrm{det}}) = \{x_1 x_2 \cdots \in \Sigma^{\omega} \mid \exists d_i \in D(\mathrm{det}), \text{s.t. } (x_1, d_1)(x_2, d_2) \cdots \in L^{\omega}(G_{\mathrm{det}})\}$.

Lemma 18. *$L^*(G) = L^*_{proj}(G_{det})$, $L^{\omega}(G) = L^{\omega}_{proj}(G_{det})$.*

Proof. First, suppose $X = x_1 x_2 \cdots x_n \in L^*(G)$. For any finite path $\rho = q_0 q_1 \cdots q_n$ in G_{det}, it is also a path in G according to the construction of G_{det} (because it does not change the graph structure of the assertion graph G). If $X \models_{\mathrm{ant}} \rho$, that is, $x_i \in \mathrm{ant}((q_{i-1}, q_i))$ for $1 \le i \le n$, then $X \models_{\mathrm{cons}} \rho$, namely, $x_i \in \mathrm{cons}((q_{i-1}, q_i))$ for $1 \le i \le n$. According to the definition of G_{det}, there exists $d_i \in D(\mathrm{det})$ such that $\mathrm{ant}((q_{i-1}, q_i)) \times d_i$ is the antecedent of the edge (q_{i-1}, q_i) in G_{det} if there is a constraint $\mathrm{det} = d_i$ for this antecedent, or a subset of the antecedent of the edge (q_{i-1}, q_i) in G_{det} if there is no constraint for this antecedent. Therefore, $X_{\mathrm{det}} = (x_1, d_1)(x_2, d_2) \cdots (x_n, d_n) \models_{\mathrm{ant} \times \mathrm{det}} \rho$, and $X_{\mathrm{det}} \models_{\mathrm{cons} \times \mathrm{det}} \rho$ because there is no constraint for the consequents. Thus, $X_{\mathrm{det}} \in L^*(G_{\mathrm{det}})$. According to the definition of the projection language of G_{det}, $X \in L^*_{\mathrm{proj}}(G_{\mathrm{det}})$.

Second, suppose that $X = x_1 x_2 \cdots x_n \in L^*_{\mathrm{proj}}(G_{\mathrm{det}})$. Then there exists $d_i \in D(\mathrm{det})$ such that $X_{\mathrm{det}} = (x_1, d_1)(x_2, d_2) \cdots (x_n, d_n) \in L^*(G_{\mathrm{det}})$, which implies that if $(x_i, d_i) \in \mathrm{ant}((q_{i-1}, q_i)) \times d_i$ for $1 \le i \le n$, then $(x_i, d_i) \in \mathrm{cons}((q_{i-1}, q_i)) \times \mathrm{det}$. And $(x_i, d_i) \in \mathrm{ant}((q_{i-1}, q_i)) \times d_i$ implies $x_i \in \mathrm{ant}((q_{i-1}, q_i))$, $(x_i, d_i) \in \mathrm{cons}((q_{i-1}, q_i)) \times \mathrm{det}$ implies $x_i \in \mathrm{cons}((q_{i-1}, q_i))$.

Therefore $X \in L^*(G)$.

Combining these two cases, we have $L^*(G) = L^*_{\mathrm{proj}}(G_{\mathrm{det}})$.

Similarly, $L^{\omega}(G) = L^{\omega}_{\mathrm{proj}}(G_{\mathrm{det}})$ is also true. □

Theorem 19. *$(G^{ref}_{det} \Rightarrow G_{det}) \Rightarrow (G^{ref} \Rightarrow G)$.*

Proof. $(G^{\mathrm{ref}}_{\mathrm{det}} \Rightarrow G_{\mathrm{det}}) \Leftrightarrow (L(G^{\mathrm{ref}}_{\mathrm{det}}) \subseteq L(G_{\mathrm{det}}))$ (definition of implication).

$\Rightarrow (L_{\mathrm{proj}}(G^{\mathrm{ref}}_{\mathrm{det}}) \subseteq L_{\mathrm{proj}}(G_{\mathrm{det}}))$ (definition of projection language).

$\Rightarrow (L(G_{\mathrm{ref}}) \subseteq L(G))$ (Lemma 18).

$\Rightarrow (G^{\mathrm{ref}} \Rightarrow G)$ (definition of implication). □

In the above proof, the language $L(G)$ can be finite language $L^*(G)$ or infinite language $L^{\omega}(G)$. Thus, Theorem 19 holds for both finite and infinite languages, respectively.

4.2. Case Study. An FIFO is a common circuit within a microprocessor design. The design requirements of an FIFO can be fairly complex due to the variable lengths, different rates of data throughput, and timing. Therefore, it makes a good practice to verify the FIFO design against these requirements. In general, the behavior of an FIFO must meet the following requirements: (1) correctness of full and empty flags, and (2) enqueued data must be dequeued in the correct order while maintaining uncorrupted data. The FIFO has 0 entry after the reset. If an enqueue only operation occurred, the total number of entries increases by 1. On the other hand, if a dequeue only operation occurred, the total number of entries decreases by 1 when the number of entries is bigger than 0. The empty flag is set when the number of entries is 0 and the full flag is set when contents reach the depth of the FIFO.

An assertion graph of a 3-deep FIFO circuit is shown in Figure 4 without determinization variable det.

The top part of the assertion graph specifies the number of filled entries. However, it cannot guarantee whether the enqueued data is corrupted or not. In order to overcome this situation, an enqueued vector of distinct symbolic constants is used at an arbitrary time as shown in the bottom portion of the graph.

The assertion graph shown in Figure 4 without determinization variable det has the potential problem of overapproximation. In order to overcome this problem, a refined assertion graph shown in Figure 5 without determinization variable det is introduced.

It unfolds the graph at the states where the precision is lost.

Using the GSTE engine, we can verify the refined assertion graph on the FIFO circuit. To conclude that the original assertion graph also holds on the circuit, we must establish that the refined assertion graph implies the original assertion graph. Since both assertion graphs are non-deterministic, we first determinize the original assertion graph. A variable "det" whose domain is $\{1, 2\}$ is added to determinize the original assertion graph. We add "det = 1" in the antecedents on the edges (i-filled, $(i + 1)$-filled) and "det = 2" in the antecedents on the edges (i-filled, i-ahead), respectively, for $i = 0, 1, 2$. And We add "det = 1" in the antecedents on the edges (i-filled, i-filled) and "det = 2" in the antecedents on the edges (i-filled, $(i - 1)$-ahead), respectively, for $i = 1, 2$. The determinized assertion graph is shown in Figure 4. We then refine the assertion graph in Figure 4 by applying the same refinements that refine the original assertion graph to the refined assertion graph and get the determinized and refined assertion graph shown in Figure 5.

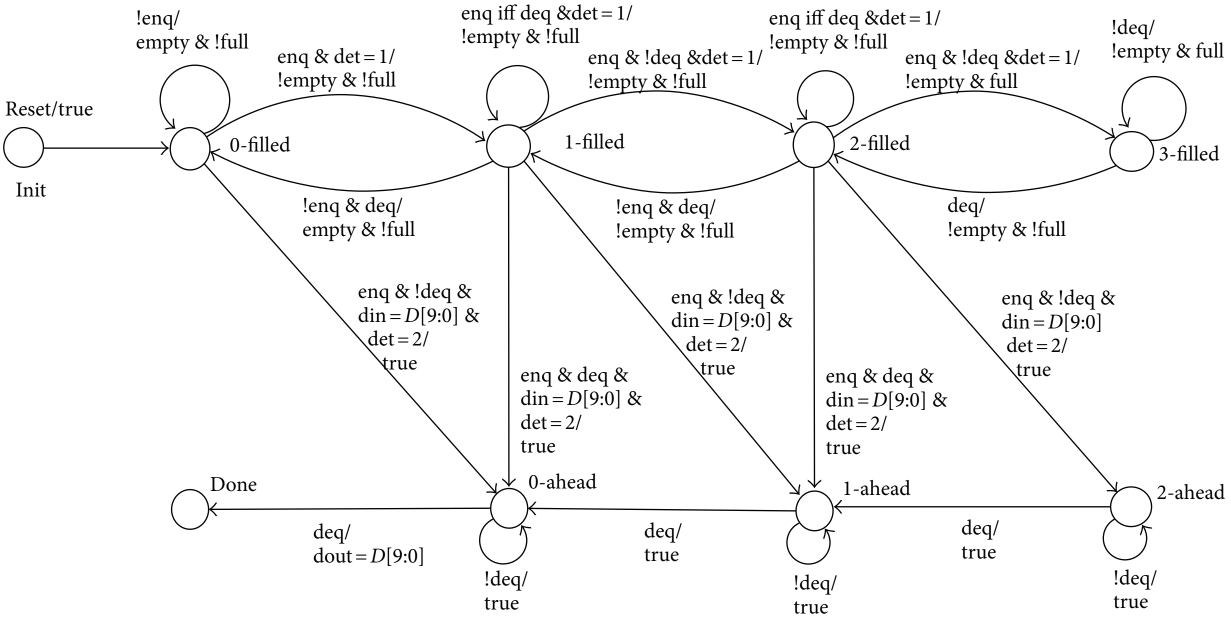

Figure 4: Determinized assertion graph for FIFO.

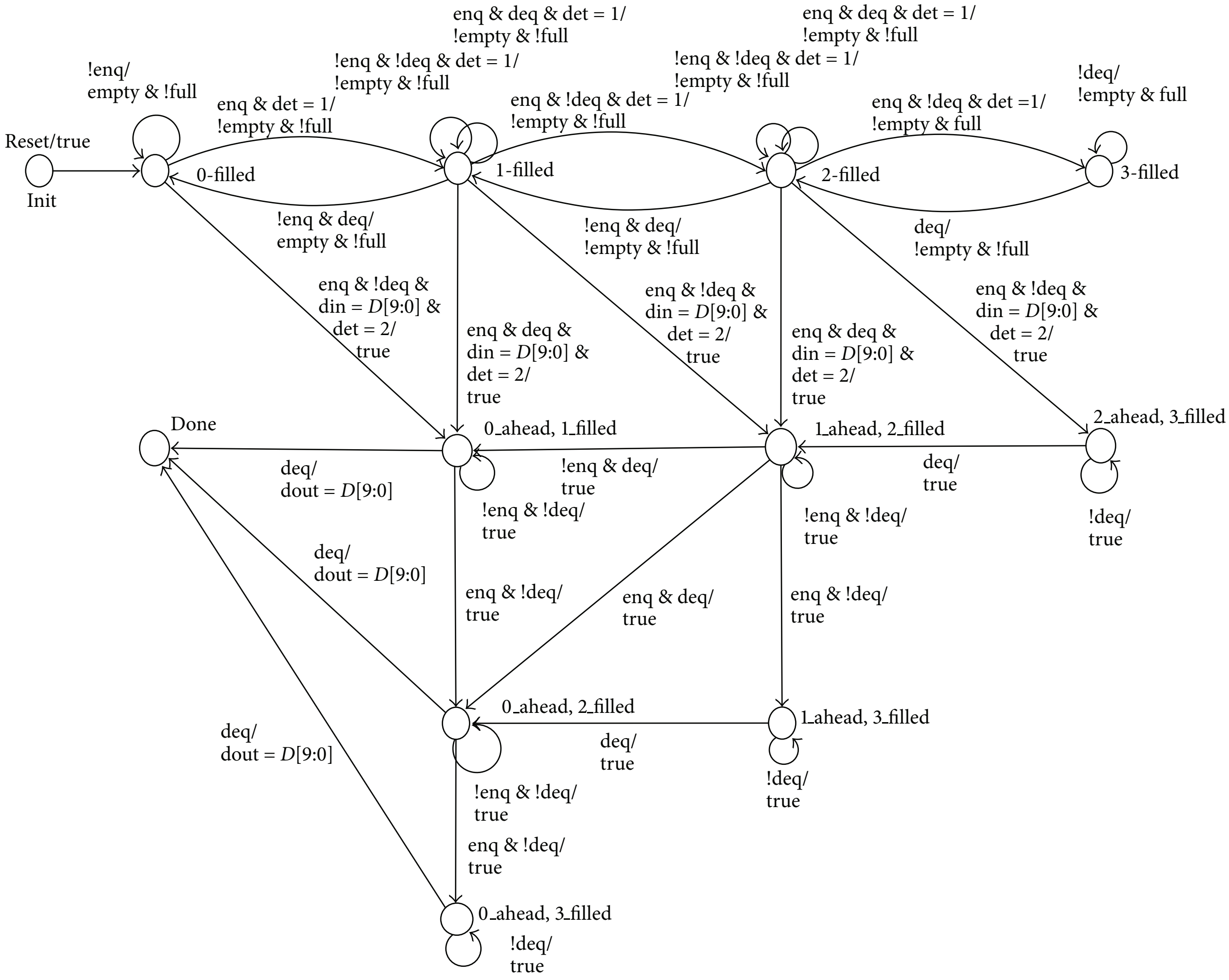

Figure 5: Determinized and refined assertion graph for FIFO.

We select COSPAN [24, 25] as the model checking engine to perform language containment test. We code the automata corresponding to the assertion graphs in Figures 4 and 5 in S/R, the input language of COSPAN, and use COSPAN to verify that the language of the automaton corresponding to the assertion graph in Figure 4 is contained in the language of the assertion graph in Figure 5. COSPAN establishes the language containment using 0.156 megabytes memory and 0.02 CPU seconds on a Pentium IV 2.8 GHz computer.

5. Conclusion

In this paper, we have established the complement relation between GSTE assertion graphs and finite-state automata with expressiveness of regular languages and ω-regular languages. We present an algorithm that transforms a GSTE assertion graph to a finite-state automaton and vice versa. Using this algorithm, we transform the problem of GSTE assertion graphs implication to the problem of automata language containment. We avoid the exponential state space blowups in checking language containment of nondeterministic finite-state automata when determining the implication relation between a nondeterministic assertion graph and its refinement derived following the refinement strategies given in [2]. This avoidance is achieved by transforming the implication relation between nondeterministic assertion graphs to the implication relation between deterministic assertion graphs without adding any states. This application has been illustrated with a case study for verifying properties of an FIFO circuit.

Acknowledgments

This paper is partially supported by the National Natural Science Foundation of China (no. 60973016 and no. 61272175), and the National Basic Research Program of China (973 Program: no. 2010CB328004).

References

[1] J. Yang and C. J. H. Seger, "Generalized symbolic trajectory evaluation," Technical Report, 2002.

[2] J. Yang and C. J. H. Seger, "Generalized symbolic trajectory evaluation-abstraction in action," in *FMCAD*, vol. 2517, pp. 70–87, 2002.

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

[4] J. Yang and C. J. H. Seger, "Compositional specification and model checking in GSTE," in *Computer-Aided Verification*, vol. 3114, pp. 216–228, 2004.

[5] B. Bentley, "High level validation of next generation microprocessors," in *IEEE High Level Design Validation and Test Workshop*, 2002.

[6] A. J. Hu, J. Casas, and J. Yang, "Efficient generation of monitor circuits for GSTE assertion graphs," in *IEEE/ACM International Conference on Computer-Aided Design*, pp. 154–159, 2003.

[7] A. J. Hu, J. Casas, and J. Yang, "Reasoning about GSTE assertion graphs," in *Proceedings of the 12th Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME '03)*, pp. 170–184, 2003.

[8] C. T. Chou, "The mathematical foundation of symbolic trajectory evaluation," in *Computer-Aided Verification*, vol. 1633, pp. 196–207, 1999.

[9] 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. 147–190, 1995.

[10] M. Aagaard, R. B. Jones, and C. J. H. Seger, "Combining theorem proving and trajectory evaluation in an industrial environment," in *Design Automation Conference*, pp. 538–541, 1998.

[11] P. Bjesse, T. Leonard, and A. Mokkedem, "Finding bugs in an α microprocessor using satisfiability solvers," in *Computer-Aided Verification*, pp. 454–464, 2001.

[12] K. L. Nelson, A. Jain, and R. E. Bryant, "Formal verification of a superscalar execution unit," in *Design Automation Conference*, pp. 161–167, 1997.

[13] M. Pandey, R. Raimi, D. L. Beatty, and R. E. Bryant, "Formal verification of PowerPC arrays using symbolic trajectory evaluation," in *Design Automation Conference*, pp. 649–654, 1996.

[14] G. Yang, J. Yang, W. N. N. Hung, and X. Song, "Implication of assertion graphs in GSTE," in *Asia South Pacific Design Automation Conference*, pp. 1060–1063, 2005.

[15] R. Sebastiani, E. Singerman, S. Tonetta, and M. Y. Vardi, "GSTE is partitioned model checking," in *Computer-Aided Verification*, vol. 3114, pp. 229–241, Springer, Berlin, Germany, 2004.

[16] E. Friedgut, O. Kupferman, and M. Vardi, "Büchi complementation made tighter," in *Proceedings of the 2nd International Symposium on Automated Technology for Verification and Analysis*, pp. 64–78, 2004, Lecture Notes in Computer Science.

[17] N. Klarlund, "Progress measures for complementation of ω-automata with applications to temporal logic," in *Proceedings of the 32nd Annual Symposium of Foundations of Computer Science*, pp. 358–367, 1991.

[18] O. Kupferman and M. Y. Vardi, "From complementation to certification," in *Proceedings of the Tools and algorithms for the construction and analysis of systems (TACAS '04)*, vol. 2988, pp. 591–606, 2004, Lecture Notes on Computer Science.

[19] R. P. Kurshan, "Complementing deterministic Büchi automata in polynomial time," *Journal of Computer and System Sciences*, vol. 35, no. 1, pp. 59–71, 1987.

[20] M. Michel, "Complementation is more difficult with automata on infinite words," in *CNET*, Paris, France, 1988.

[21] S. Safra, "On the complexity of omega-automata," in *Foundations of Computer Science*, pp. 319–327, 1988.

[22] S. Tasiran, R. Hojati, and R. K. Brayton, "Language containment of non-deterministic omega-automata," in *Proceedings of the IFIP WG 10. 5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME '95)*, pp. 261–277, Springer, London, UK, 1995.

[23] J. R. B. Büchi, "On a decision method in restricted second order arithmetic," in *International Congress For Logic, Methodology and Philosophy of Science*, pp. 1–12, 1962.

[24] K. Fisler and R. P. Kurshan, "Verifying vhdl designs with cospan," in *Formal Hardware Verification—Methods and Systems in Comparison*, pp. 206–247, Springer, London, UK, 1997.

[25] R. P. Kurshan, *Computer-Aided Verification of Coordinating Processes*, Princeton Series in Computer Science, Princeton University Press, Princeton, NJ, USA, 1994, The automata-theoretic approach.