# Synthesis of Timed Circuits Based on Decomposition

Tomohiro Yoneda, *Member, IEEE*, and Chris J. Myers, *Senior Member, IEEE*

***Abstract*—This paper presents a decomposition-based method for timed circuit design that is capable of significantly reducing the cost of synthesis. In particular, this method synthesizes each output individually. It begins by contracting the timed signal transition graph (STG) to include only transitions on the output of interest and its possible trigger signals. Next, the reachable state space for this contracted STG is analyzed to determine a minimal number of additional signals, which must be reintroduced into the STG to obtain complete state coding. The circuit for this output is then synthesized from this STG. Results show that the quality of the circuit implementation is nearly as good as the one found from the full reachable state space, but it can be applied to find circuits for which full-state-space methods cannot be successfully applied. The proposed method has been implemented as a part of our tool Nii-Utah Timed Asynchronous circuit Synthesis system (nutas), and its first version is available at http://research.nii.ac.jp/~yoneda.**

***Index Terms*—Abstraction, decomposition, synthesis, timed circuits, timed signal transition graphs (STGs).**

## I. Introduction

LOGIC SYNTHESIS [1]–[4] from low-level specification languages is one of the major approaches to the automated synthesis of asynchronous circuits. This approach can potentially synthesize more optimized circuits with higher performance than other methods such as syntax-directed-translation methods [5]–[10]. It, however, usually requires enumeration of the state space of the given specification, and it often suffers from the state-explosion problem. Thus, large specifications expressed in hardware-description languages have usually been synthesized by syntax-directed-translation methods or similar techniques that do not require state-space enumeration, sometimes with local-optimization techniques, such as in [11]. This paper tackles the challenge of using logic synthesis also for large specifications derived from hardware-description languages, as it has the potential of providing further global optimization through timed circuit synthesis [12]. In this approach, a specification written in some high-level language is first translated to a timed signal transition graph (STG), and then, logic synthesis is applied to this timed STG. This method uses a compiler that generates timed STGs with the complete-state-coding (CSC) property. Its preliminary tool is described in [13], and an improved version is described in [14] and [15]. Guaranteeing CSC by such a correct-by-construction method, which may not give optimal solutions in the number of inserted state variables, is practical for large STGs, because automatic CSC solvers sometimes do not handle such large STGs well. Furthermore, by using a special protocol shown in [15] and [16], the performance degradation caused by the inserted state variables can be reduced to an almost negligible amount. A key issue to the success of our approach is a new logic-synthesis technique that is efficient enough to handle large STGs. This paper aims at reducing the average cost for logic synthesis from timed STGs by decomposing (or projecting) a specification to many small subspecifications and running the logic-synthesis procedure for each of them.

The idea for decomposition-based synthesis was first proposed by Chu [17]. In his work, one primary output is chosen and the given STG is modified by replacing each transition for the signal that does not affect the output by a dummy transition. Then, the modified STG is reduced by eliminating selected dummy transitions, while preserving the behavior. A correct circuit can be synthesized from this reduced STG with usually much smaller cost. He, however, left two open problems. First, the reduction of STGs, called *contraction*, was not formalized. For a simple STG such as a marked graph, its contraction is straightforward. But, in the general case, the formalized algorithm was unknown at that time. Second, it was not straightforward to decide if a signal actually affects the output signal or not and no algorithm to make this decision is given in his thesis. As for the first problem, Vogler and Wollowski recently formalized the contraction algorithm using a bisimulation relation in [18], and Zheng *et al.* developed a timed-contraction algorithm in [19]. On the other hand, Puri and Gu tried to solve the second problem in [20]. Their algorithm greedily removes an irrelevant signal (with respect to the output signal) such that the number of CSC violations (CSCV) does not increase by hiding that signal. This algorithm is, however, not so helpful for our purpose, because it needs the state graph of the original STG, which cannot be constructed due to state explosion for very large STGs. Beister *et al.* proposed a similar decomposition-based method for extended-burst-mode machines [21]. Recently, two separate works, one by Carmona and Cortadella [22] and the other by Khomenko *et al.* [23], have been proposed for synthesizing speed-independent circuits efficiently based on an idea similar to that in [20]. Both works first find the necessary input signals (called *support*) for each output by analyzing the original STG and then synthesize each subcircuit with each output individually. Unlike the approach of [20], these works do not use the state graph of the original STG explicitly. That is, the original STG is analyzed using the integer-linear-programming (ILP) technique in the former and the incremental-Boolean-satisfiability (SAT) technique in the

Manuscript received January 23, 2006; revised July 6, 2006. The work of T. Yoneda was supported by JSPS Joint Research Projects. The work of C. J. Myers was supported by NSF Japan Program Award INT-0087281, SRC Grant 2002-TJ-1024, and SRC Grant 2005-TJ-1357. This paper was recommended by Associate Editor S. Nowick.

T. Yoneda is with the National Institute of Informatics, Tokyo 101-8430, Japan (e-mail: yoneda@nii.ac.jp).

C. J. Myers is with the University of Utah, Salt Lake City, UT 84112 USA (e-mail: myers@ece.utah.edu).

Digital Object Identifier 10.1109/TCAD.2006.888269

latter, and hence, their methods are much more efficient than that of [20].

The main contribution of this paper is to propose a new algorithm to find a sufficient set of input signals for a given output for the decomposition-based synthesis approach without using the state graph of the original STG. The algorithm starts with a small set of signals, which are certainly needed for the output signal, and uses only the state graphs of the contracted STGs for determining other necessary input signals. Since the state graphs of the contracted STGs are usually very small, it does not suffer from the state-explosion problem. Furthermore, its decision procedure computes candidates of the necessary signals in many cases more directly than the greedy algorithm in [20], although some cases need heuristics. Since our approach analyzes the state graphs of the contracted STGs explicitly, it is very easy to handle timed STGs and this is the biggest difference between ours and the above ILP/SAT-based approaches. This paper describes the theory and the algorithms extended from [24] for the timed circuit synthesis.

The rest of this paper is organized as follows. Section II shows several definitions needed for this paper. Section III describes the overview of the proposed method, and Section IV mentions how to explore timed state spaces and to check synthesizability. Section V explains in detail how the input sets are determined, which is the main issue of this paper. Section VI discusses some complexity issues. Several experimental results are shown in Section VII, and Section VIII gives our conclusion.

## II. SYNTHESIZABLE STGs

A timed STG $G = (P, T, F, \mathsf{Eft}, \mathsf{Lft}, \mu^0, l, In, Out)$ is a labeled net, where $P$ is a finite set of *places*, $T$ is a finite set of *transitions* ($P \cap T = \emptyset$), $F \subseteq (P \times T) \cup (T \times P)$ is the *flow relation*, $\mathsf{Eft} : T \to \mathbf{Q}^+$, $\mathsf{Lft} : T \to \mathbf{Q}^+ \cup \{\infty\}$ are functions for the *earliest* and *latest firing times* of transitions satisfying $\mathsf{Eft}(t) \leq \mathsf{Lft}(t)$ for all $t \in T$ ($\mathbf{Q}^+$ denotes the set of nonnegative rationals), $\mu^0 \subseteq P$ is the *initial marking*, $l : T \to (In \cup Out) \times \{+, -\} \cup \{\lambda\}$ is the labeling function, and $In$ and $Out$ are the input and output signal sets. Let $\mathsf{sig}(G)$ denote $In \cup Out$. A transition $t$ with $l(t) \in In \times \{+, -\}$ is called an *input transition*, $t$ with $l(t) \in Out \times \{+, -\}$ is called an *output transition*, and $t$ with $l(t) = \lambda$ is called a *dummy transition*. For $w \in \mathsf{sig}(G)$, $w$*-transition* denotes a transition $t$ with $l(t) = w+$ or $w-$. For any transition $t$, $\bullet t = \{p \in P | (p, t) \in F\}$ and $t\bullet = \{p \in P | (t, p) \in F\}$ denote the *source places* and the *destination places* of $t$. For a place $p$, $\bullet p$ and $p\bullet$ are defined similarly. Transitions $t$ and $t'$ such that $\bullet t \cap \bullet t' \neq \emptyset$ are said to be in *conflict*. Let $conflict(t) = \{t' | \bullet t \cap \bullet t' \neq \emptyset\} - \{t\}$. In the rest of this paper, when timed STGs $G$, $G_1$, etc., are considered, their corresponding components $P$, $T$, etc., $P_1$, $T_1$, etc., are implicitly considered. Furthermore, a timed STG is simply called an STG if there is no confusion.

A *marking* $\mu$ of $G$ is any subset of $P$. A transition $t$ is *enabled* in a marking $\mu$ if $\bullet t \subseteq \mu$ (all its source places have tokens in $\mu$); otherwise, it is *disabled*. Let $\mathsf{enabled}(\mu)$ be the set of transitions enabled in $\mu$. A *timed state* $\sigma$ of $G$ is a pair $(\mu, \mathsf{clock})$, where $\mu$ is a marking and clock is a function $T \to \mathbf{R}^+$ ($\mathbf{R}^+$ denotes the set of nonnegative reals). The *initial timed state* $\sigma^0$ is $(\mu^0, \mathsf{clock}^0)$, where $\mathsf{clock}^0(t) = 0$ for all $t \in T$. A timed state changes if time passes or if a transition fires. In timed state $\sigma = (\mu, \mathsf{clock})$, time $\tau \in \mathbf{Q}^+$ can pass, if for all $t \in \mathsf{enabled}(\mu)$, $\mathsf{clock}(t) + \tau \leq \mathsf{Lft}(t)$. In this case, timed state $\sigma' = (\mu', \mathsf{clock}')$ is obtained from $\sigma$ by passing $\tau$, where

1) $\mu' = \mu$ and
2) for all $t \in T$, $\mathsf{clock}'(t) = \mathsf{clock}(t) + \tau$.

In timed state $\sigma = (\mu, \mathsf{clock})$, transition $t_f \in T$ can fire, if $t_f \in \mathsf{enabled}(\mu)$ and $\mathsf{clock}(t_f) \geq \mathsf{Eft}(t_f)$. In this case, timed state $\sigma' = (\mu', \mathsf{clock}')$ is obtained from $\sigma$ by firing $t_f$, where

1) $\mu' = (\mu - \bullet t_f) \cup t_f\bullet$ and
2) for all $t \in T$

$$\mathsf{clock}'(t) = \begin{cases} 0, & \text{if } t \in \mathsf{enabled}(\mu') - \mathsf{enabled}(\mu - \bullet t_f) \\ \mathsf{clock}(t), & \text{otherwise.} \end{cases}$$

That is, firing a transition $t_f$ consumes no time but updates $\mu$ and clock such that the clocks associated with *newly* enabled transitions (i.e., transitions that are enabled in $\mu'$, the final marking obtained when $t_f$ fires, but not enabled in $\mu - \bullet t_f$, the intermediate marking during $t_f$ fires) are reset to zero, and clock values of other transitions (i.e., transitions not affected by $t_f$) are left unchanged. Let $\sigma \xrightarrow{t_f} \sigma'$ denote that $\sigma'$ is obtained from $\sigma$ by first passing some time and, then, firing $t_f$. For a sequence $v = t_1 t_2 \cdots$ of transitions, $\sigma \xrightarrow{v} \sigma'$ is defined similarly ($\sigma$ is equal to $\sigma'$ for an empty $v$). $v$ is called a *trace*, if there exists a $\sigma'$ such that $\sigma^0 \xrightarrow{v} \sigma'$. Let $\mathsf{trace}(G)$ denote the set of all prefix-closed traces of $G$. If there exists a trace that leads to $\sigma'$, $\sigma'$ is called *reachable*. A trace may contain multiple occurrences of the same transition. In this paper, it is assumed that those occurrences of the same transition are distinguished in some appropriate way, such as, by attaching firing counts, but those are omitted for simplicity in this paper. If every reachable timed state $\sigma = (\mu, \mathsf{clock})$ such that there exist $t$ and $\sigma'$ with $\sigma \xrightarrow{t} \sigma'$ satisfies $(\mu - \bullet t) \cap t\bullet = \emptyset$, then $G$ is called *one-safe*. Intuitively, in a one-safe STG, a token is never produced into a place that is already marked. Furthermore, $G$ is *consistent*, if for every trace $v \in \mathsf{trace}(G)$ and every $w \in \mathsf{sig}(G)$ such that $v$ includes two or more $w$ transitions, the last two of them are different (i.e., $w+$ and $w-$, or $w-$ and $w+$).

A reachable timed state is mapped to a *signal state*, which is a binary vector representing the values of signals in $In \cup Out$. Different timed states may be mapped to the same signal state. It is sometimes convenient to annotate a signal state with the information whether the outputs are excited to rise or fall. For this purpose, $R$ or $F$ is used in addition to zero or one in signal states. $R$ represents that the corresponding output signal has the binary value of zero, but it is excited to rise. $F$ indicates the signal has a value of one, but it is excited to fall. When these two notations with or without $R/F$ should be distinguished, we call the former *decorated signal states* and the latter *nondecorated signal states*. For example, suppose that two timed states $\sigma$ and

$\sigma'$ have decorated signal states (1010) and $(101R)$.[1] They have the common nondecorated signal state (1010), but the behavior of the output is different in those timed states. This situation is called a *CSC violation*, and these two timed states are a *CSC violation pair*. If an STG has a CSC violation pair, we say that the STG does not have CSC. Otherwise, it has CSC. If an STG does not have CSC, a circuit cannot be synthesized from the STG without adding a state variable, reducing concurrence, or otherwise changing the behavior of the STG in some way.

This detection of CSCV is, however, a little complicated, if $G$ has dummy transitions. Suppose $\sigma'$ is obtained from $\sigma$ by firing the dummy transition and that an output signal is excited in $\sigma'$ but not in $\sigma$. $\sigma$ and $\sigma'$ have the same nondecorated signal state, while they have different decorated signal states. In this case, however, $\sigma$ and $\sigma'$ cannot be distinguished from the outside (i.e., a dummy transition is invisible), and so, it should not be considered that they cause a CSC violation. In order to define this signal excitation formally, it is useful to define a dummy-free version of a state graph. A *timed state graph* of an STG $G$ is a graph $\langle V, E\rangle$ with an initial timed state $\sigma^0$, denoted by $\mathcal{G}_G = (\langle V, E\rangle, \sigma^0)$, such that $V$ is the set of all reachable timed states of $G$ and $E$ is the timed-state transition relation of $G$, that is, $\{(\sigma, t, \sigma') | \exists v \cdot \sigma^0 \xrightarrow{v} \sigma, \sigma \xrightarrow{t} \sigma'\}$. A *dummy-free timed state graph* of $\mathcal{G}_G$ is a graph $\langle V', E'\rangle$ with an initial timed state $\sigma^{0'}$, denoted by $\mathcal{G}_G^{\mathrm{df}} = (\langle V', E'\rangle, \sigma^{0'})$, satisfying the following:

1) $\sigma^{0'} = \sigma^0$;
2) $V' = \{\sigma | (\sigma', t, \sigma) \in E, t \neq \lambda\} \cup \{\sigma^{0'}\}$;
3) $E' = \{(\sigma, t, \sigma_3) | \sigma \in V', (\sigma, u_1 u_2 \cdots u_n, \sigma_2) \in E^*, n \geq 0, \forall i \cdot u_i = \lambda, (\sigma_2, t, \sigma_3) \in E, t \neq \lambda\}$.

This dummy-free timed state graph is constructed based on the fact that timed-state transitions by a (possibly empty) sequence of dummy transitions followed by a nondummy transition can be replaced by the single nondummy transition. Fig. 1(a) shows a simple timed STG $G$ (the transition labeled by $t$ is a dummy transition), and its timed state graph and dummy-free timed state graph are shown in Fig. 1(b) and (c), respectively. Note that $a+$, for example, can fire at any time between one and two time units after it becomes enabled, and so, there exists an infinite number of timed states reached from $\sigma_0$ by firing $a+$. These figures show for simplicity only timed states that have different markings.

Now, the signal excitation can be defined on this $\mathcal{G}_G^{\mathrm{df}} = (\langle V', E'\rangle, \sigma^{0'})$. An output signal $w$ is excited in a timed state $\sigma$, if $\exists \sigma' \cdot (\sigma, t, \sigma') \in E'$ with $l(t) = w+$ or $l(t) = w-$. For example, $x$ is excited to rise in $\sigma_1$. This straightforward definition is, however, not sufficient for the timed case. Consider $\sigma_2$ of Fig. 1(b). In this timed state, both $x+$ and $y+$ are enabled but only $x+$ can fire in this state, because the earliest firing time of $y+$ is larger than the latest firing time of $x+$. Thus, $\sigma_1$ has only one successor state reached by firing $x+$ in Fig. 1(c). It is, however, necessary to define that $y$ is also excited in $\sigma_1$ in order to synthesize a circuit for the output $y$ correctly, because $y$ is triggered by $a+$, not by $x+$, as shown in the STG. Therefore, the signal excitation should be defined based on the enabledness

[1]Note that in (1010) some input may be excited, but only outputs are decorated in our definition.

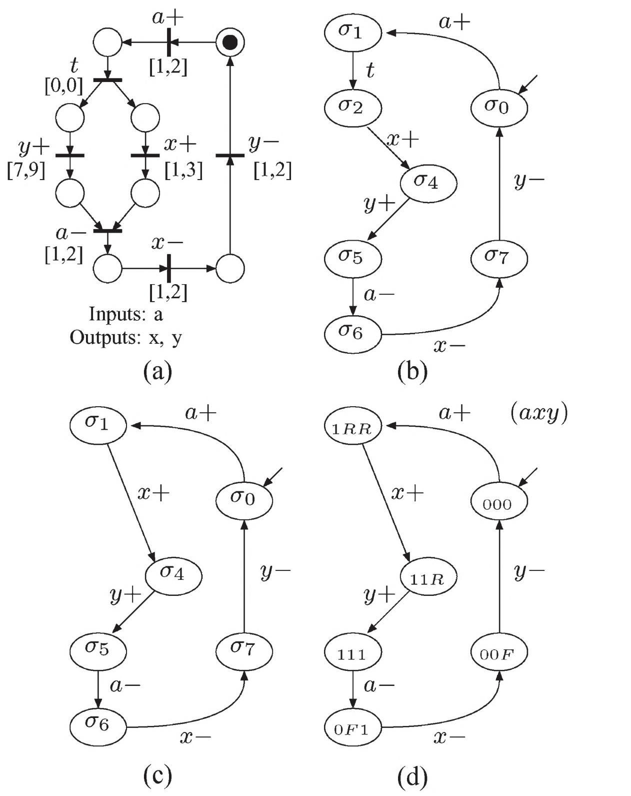


Fig. 1. (a) Simple STG with a dummy transition. (b) Timed state graph. (c) Dummy-free timed state graph. (d) Dummy-free timed state graph with decorated signal states.

information instead of the existence of outgoing edges in the state graph. This is complicated by the fact that neither $x+$ nor $y+$ is yet enabled in $\sigma_1$.

The definition of signal excitation proposed in this paper is as follows. An output signal $w$ is *excited* in a timed state $\sigma$ of $\mathcal{G}_G^{\mathrm{df}} = (\langle V', E'\rangle, \sigma^{0'})$, if there exists a (possibly empty) sequence $u_1 u_2 \cdots u_n$ of dummy transitions such that $(\sigma, u_1 u_2 \cdots u_n, \sigma_2) \in E^*$, $\sigma_2 = (\mu_2, \mathsf{clock}_2)$, and $t \in \mathsf{enabled}(\mu_2)$ with $l(t) = w+$ or $l(t) = w-$, where $\mathcal{G}_G = (\langle V, E\rangle, \sigma^0)$. Let $\mathsf{out\_excited}(\sigma)$ be a set of output signals that are excited in $\sigma$. Then, it is defined that a timed state $\sigma$ has $R$ (or $F$) for an output $w$ in its decorated signal state, if and only if $w \in \mathsf{out\_excited}(\sigma)$ and the binary value of $w$ in $\sigma$ is zero (or one). For example, the decorated signal state of $\sigma_1$ in the previous example is $(a, x, y) = (1RR)$. Fig. 1(d) shows decorated signal states on the dummy-free timed state graph. Based on this definition of decorated signal states, the detection of CSCV can be done in the same way as STGs without dummy transitions.

The property called output semimodularity is also necessary to synthesize a circuit from an STG. For the untimed case, this property is formally stated as follows. An STG $G$ is output semimodular, if its dummy-free state graph $\mathcal{G}_G^{\mathrm{df}} = (\langle V', E'\rangle, \sigma^{0'})$ satisfies that for any $(\sigma, t, \sigma') \in E'$ with $l(t) = x+$ or $l(t) = x-$, if a signal $w$ $(\neq x)$ is excited in $\sigma$, but not in $\sigma'$, then signals $w$ and $x$ are both input. This definition again has a problem in the timed case. Consider the STG shown in Fig. 2(a), where $t_1$ and $t_2$ are dummy transitions. It has the timed state graph and dummy-free timed state graph, as shown in Fig. 2(b) and (c). According to the excitation defined

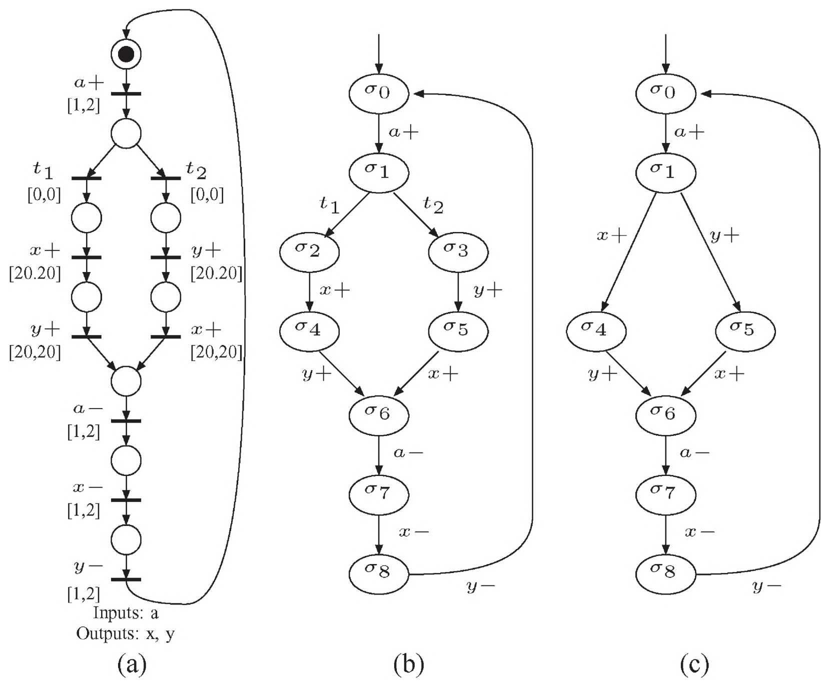


Fig. 2. (a) STG with conflicting dummy transitions. (b) Timed state graph. (c) Dummy-free timed state graph.

above, in the dummy-free timed state graph, $x$ is excited in both $\sigma_1$ and $\sigma_5$ as well as $y$ is excited in both $\sigma_1$ and $\sigma_4$. Thus, this STG satisfies the above property. Hence, if only the untimed behavior of this STG is considered, a circuit such that $a+$ triggers both $x+$ and $y+$ is synthesized from it. However, it is impossible to find any delay assignment to this circuit, under which the circuit satisfies the timed behavior of the STG, because when $x+$ fires later than $y+$, $x+$ should fire at a total of 40 time units later than the firing of $a+$; otherwise, it should fire only 20 time units later than the firing of $a+$. Hence, this timed STG should not be considered to be synthesizable. In this paper, we use the following simplified definition of output semimodularity for timed STGs. A timed STG $G$ is *output semimodular*, if for any conflicting transitions that are enabled in the same timed state of $\mathcal{G}_G$, every (possibly empty) path of dummy transitions starting from each of them on the STG ends with an input transition. For the STG shown in Fig. 2(a), $t_1$ and $t_2$ that are in conflict with each other are enabled in $\sigma_1$, and the path of dummy transitions from $t_1$ on the STG, which is $t_1$ itself, ends with an output transition $x+$. Thus, this STG is not output semimodular in our definition. Although this definition is sometimes unnecessarily too strong (e.g., the case that both $x+$ and $y+$ have [0,0] delays in the above STG.), it is, in our experience, not a practical problem.

Although one-safeness of STGs is not required for synthesis, our timed state-space enumeration algorithm supports only one-safe STGs. Furthermore, consistency significantly simplifies the analysis and synthesis algorithms. Thus, we say that an STG $G$ is *synthesizable*, if $G$ is one-safe, consistent, output semimodular, and has CSC.

There is another property needed especially for timed circuit synthesis. The timed circuit synthesis method assumes that a synthesized logic function for an output is implemented with a delay within the firing-time bounds (i.e., $[\mathsf{Eft}(t), \mathsf{Lft}(t)]$) of the corresponding output transitions in the given timed STG. This assumption, however, may not work, if the output transitions related to the same output signal have different firing-time bounds, or even a dummy transition that precedes those output transitions has a nonzero delay. In order to simplify the problem, this paper considers a class of timed STGs satisfying the following timed implementability. A timed STG $G$ is *timed implementable*, if for every output signal $x$ of $G$, every $x$ transition has the same firing-time bounds, and in any path of dummy transitions on $G$ that ends with an output transition, all dummy transitions have [0,0] bounds. If a timed STG does not satisfy the timed implementability due to dummy transitions with nonzero bounds in the paths to output transitions and the dummy transitions satisfy the conditions for exact contraction mentioned in the next section, then it can be converted to a new timed STG with timed implementability without changing the semantics by collapsing those nonzero time bounds to the time bounds of the output transitions. Note that the requirement that the output transitions related to the same output signal have identical time bounds is mainly for simplification of the presentation, and it can be relaxed, for example, such that the rising transitions and the falling transitions related to the same output signal can have different time bounds.

## III. Decomposition-Based-Synthesis Overview

The top level algorithm for the proposed decomposition-based synthesis is shown in Fig. 3. For a given synthesizable and timed-implementable STG $G$, our algorithm tries to compute an abstraction $G_{\text{abs}}$ for each output signal $x$ of $G$, such that a correct circuit for $x$ can be synthesized from it. Then, a timed circuit synthesis algorithm is applied to $G_{\text{abs}}$.

The algorithm for obtaining such an abstraction is shown in Fig. 4. It first constructs the initial input set $V$ for $x$ by taking $x$

```
decomposition_based_synthesis($G$) {
  forall $x \in Out$ {
    $G_{abs}$ = obtain_abs($G$, $x$);
    $C_x$ = timed_logic_synthesis($G_{abs}$);
  }
}
```

Fig. 3. Top-level algorithm for synthesis.

```
obtain_abs($G$, $x$) {
  $V = \{x\} \cup$ trigger($x$);
  while(true) {
    $G_{abs}$ = contract_STG($G$, $V$);
    ($res$, $ssg$) = check_synthesizable($G_{abs}$);
    if ($res$ == "synthesizable") return $G_{abs}$;
    if ($res$ == "one-safeness violation") {
      if (every transition is already flagged) abort;
      flag some transitions to disallow contraction;
      continue;
    }
    if ($res$ == "consistency or O.S.M violation") abort;
    $CSCV$ = obtain_CSC_violation_trace_set($G_{abs}$, $ssg$);
    forall $g \in CSCV$ {
      $candidate$ =
        analyze_CSCV_trace($g$, $G$, $V$);
      if ($candidate$ == "not found") abort;
      add_constraints_matrix($candidate$);
    }
    $newV$ = solve_covering_problem();
    $V = V \cup newV$;
  }
}
```

Fig. 4. Algorithm to obtain an abstraction.

and its possible trigger signals. A signal $w$ is a *possible trigger signal* for an output $x$, if one of its corresponding transitions can reach on $G$ some $x$ transitions either directly or through only dummy transitions (i.e., without passing any other signal transitions). Let $\mathsf{trigger}(x)$ denote the set of all possible trigger signals for $x$.

The algorithm next contracts dummy transitions in $G'$, if possible, where $G'$ is an STG obtained from $G$ by replacing transitions related to signals in $\mathsf{sig}(G) - V$ by dummy transitions. This contraction of transitions produces a reduced STG that behaves similarly to the original STG with respect to the remaining signals. More formally, there exists a simulation from $G'$ to the contracted STG. It is proved in Theorem 3 of the Appendix that such a reduced STG is correct[2] with respect to the original STG $G$, if the reduced STG has CSC and the input signal set contains $\mathsf{trigger}(x)$.

There are several definitions of correctness in the context of the decomposition-based synthesis. In [17], it is defined as the property that the parallel composition of the contracted STGs for all outputs has a state graph that is isomorphic to that of the original STG. The correctness defined in [18] is similar to the conformance used in [25]. Although this correctness is less strict than Chu's, only deterministic (i.e., dummy-free) STGs are considered in their definition. Our correctness is similar to the latter in the case that the given STG is untimed and deterministic. Ours is, however, different from these two, because the parallel composition is not used for the correctness definition, and STGs with dummy transitions can be naturally handled by defining correctness based on sets of signal states that are obtained from dummy-free timed state graphs.

[2]More precisely, it is "cover-correct" in our terminology. See the Appendix.

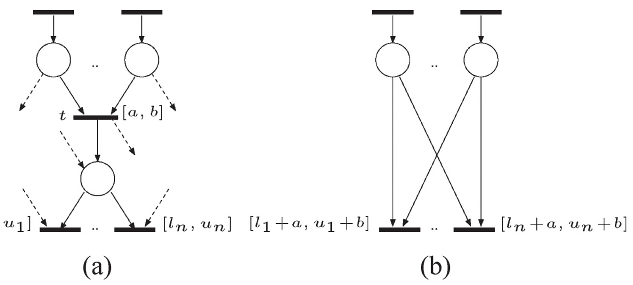


Fig. 5. Exact contraction.

Such correct contraction of timed STGs can be done in a way similar to that is shown in [19]. Our method, however, applies the contraction to only a dummy transition $t$ with a net structure as shown in Fig. 5(a), where arcs from $\bullet t$ and $t$ (shown by dashed arrows) are not allowed and, similarly, arcs to $t\bullet$ and $t\bullet\bullet$ (also shown by dashed arrows) are not allowed either. Then, contracting $t$ results in the sub-STG shown in Fig. 5(b). The untimed behavior of transitions except for $t$ is the same in both STGs, and the time spent in $t$ is just transferred to those in $t\bullet\bullet$. Thus, these two timed STGs have exactly the same dummy-free timed state graph. We prefer this *exact contraction*, because it synthesizes more optimal circuits than the general contraction that preserves the correctness in a more conservative fashion. Note that if some of the dashed arcs in Fig. 5(a) exist, the behavior of the STG cannot be preserved in general by any modification of time bounds. For example, if transitions in $t\bullet\bullet$ have two or more source places, adding $a$ and $b$ to their time bounds, as shown in Fig. 5(b), does not capture the correct timed behavior in the case that the firing of $t$ is early and its firing time does not determine those of $t\bullet\bullet$. From the equivalence of dummy-free timed state graphs, it is straightforward that our exact contraction preserves consistency. On the other hand, contracting transitions violates output semimodularity with respect to an output transition $x$, if some nonoutput transition $y$ that conflicts with some other transition, say $z$, has a path of transitions ending with $x$, and every transition (including $y$) in the path from $y$ to $x$ is contracted, because $x$ conflicts with $z$ in the resultant STG. This is, however, impossible in our case, because transitions related to the possible trigger signals, which are the nondummy transitions nearest to $t$ in the path, are considered to be input signals as mentioned above, and so, they are never contracted. Thus, output semimodularity is also preserved. Further details about the contraction algorithm are omitted in this paper. Note that even our exact contraction (and any contraction algorithm of timed STGs) cannot preserve one-safeness. Also, note that the exact contraction may result in STGs that still contain dummy transitions.

The reduced STG $G_{\mathrm{abs}}$ obtained by contraction is then checked if it is synthesizable or not. This process needs to enumerate its timed state space and construct the decorated signal state graph $ssg$ that corresponds to its dummy-free timed state graph. If $G_{\mathrm{abs}}$ is synthesizable, the algorithm returns it. If it has a one-safeness violation, some transitions that may cause the one-safeness violation are flagged to show that they should not be contracted. Selecting those transitions depends on the contraction algorithm, but for the exact contraction, the following idea usually works. Suppose that a safety violation is detected when a transition $u$ fires. If the original STG is one-safe, then this one-safe violation must be caused by contracting some transition $t$ of the original STG, satisfying $t \in u \bullet \bullet$. Thus, those transitions in $u \bullet \bullet$ can be selected. In the case that every transition is already flagged, the original STG is not one-safe. Thus, the algorithm aborts. Otherwise, the contraction process is restarted. Please note that the above selected $t$ may cause a one-safeness violation. Then, the transitions in $t \bullet \bullet$ are selected by the above process. If the original STG is one-safe, this process certainly terminates. If consistency or output semimodularity is violated (indicated by "consistency or O.S.M violation"), it is also violated in the original STG, because our contraction is exact. Thus, the algorithm aborts.

The remaining case is that $G_{\mathrm{abs}}$ does not simply have CSC. This happens when the input-signal set does not contain some relevant signals. In this case, some set of traces of $G_{\mathrm{abs}}$ that cause CSCV is extracted[3] from $ssg$. The algorithm then analyzes each $g \in$ CSCV and tries to find candidate inputs to be added in order to resolve the CSC violation. It fails to find the candidate inputs (indicated by "not found") when the original STG does not have CSC. In this case, the algorithm aborts. Otherwise, the set candidate contains a set of requirements such that each requirement, which is a set of signals, is satisfied if at least one of the signals in the requirement is added to $V$. In order to resolve the CSC violation, every requirement must be satisfied. Those requirements are added in the constraint matrix to set up a covering problem. This process is repeated for every CSC violation trace in CSCV. Finally, the covering problem is solved for those requirements, and the optimal set of signals is added to $V$. This $V$ is used to compute a new $G_{\mathrm{abs}}$, and the algorithm repeats the above process.

The formal discussion about the correctness of this method is shown in the Appendix. The Appendix first defines the correctness of an STG with respect to another STG, and then proves that the $G_{\mathrm{abs}}$, which is finally obtained in the above process when it has CSC, is correct with respect to the original STG $G$. As mentioned in the Appendix, an STG defines a set of covers and a cover defines a circuit. Thus, an STG defines a set of circuits. Let $C_G$ be this set defined by an STG $G$. Then, the correctness of $G_1$ with respect to $G$ implies $C_{G_1} \subseteq C_G$. Hence, from the above fact that $G_{\mathrm{abs}}$ is correct with respect to the original STG $G$, $C_{G_{\mathrm{abs}}} \subseteq C_G$ is derived, which implies the correctness of our method.

## IV. Checking Synthesizability

The reduced STG $G_{\mathrm{abs}}$ is checked if it is synthesizable or not in **check_synthesizable**, as shown in Fig. 4. This is done by exploring the timed state space of $G_{\mathrm{abs}}$ and obtaining its corresponding decorated signal state graph. Since the timed state space of a timed STG is potentially infinite, equivalence classes of timed states are actually explored. Let $I$ be a set of inequalities of the form $t - u \leq c$, where $t$ and $u$ are variables to represent the next firing times of transitions $t$ and $u$, and $c$ is a constant. For a given marking $\mu$, if $I$ over the variables related to the transitions enabled in $\mu$ is considered, then $I$ determines the bounds of the firing-time separation of those transitions. Thus, $(\mu, I)$ represents an equivalence class of timed states, which is called a *timed state class*. Let $\alpha_0 = (\mu_0, I_0)$ be the initial timed state class, and for a timed state class $\alpha$, $\mathsf{firable}(\alpha)$ denotes the set of *firable* transitions, i.e., the set of transitions that can fire earlier than any other transition in $\alpha$. It is known that the timed state class space is finite [26], [27], and so, its space enumeration is done by firing every firable transition from $\alpha_0$ until no new timed state classes are reached. The inclusion of timed state classes is considered in this process. A timed state class $(\mu, I)$ *includes* another timed state class $(\mu', I')$, if $\mu = \mu'$ and the solution set of $I$ includes that of $I'$. If a timed class $\alpha$ that is newly generated is included by some timed class $\alpha'$ that is generated previously, the traversal from $\alpha$ is stopped. On the other hand, if $\alpha$ includes $\alpha'$, then $\alpha'$ is removed, the arcs from the predecessors of $\alpha'$ to $\alpha'$ are reconnected to $\alpha$, and the traversal from $\alpha$ is continued.

When enumerating the equivalence classes, one-safeness can be easily checked. Once the graph of these equivalence classes is constructed, its dummy-free version is obtained by the way explained in Fig. 1. This modified graph is then projected to a decorated signal state graph by considering only decorated signal states. It is straightforward to check the consistency, output semimodularity, and CSC on this decorated signal state graph.

The above timed state class enumeration can be improved using the ideas of the partial order reduction and POSET method, which are similar to that proposed in [28] and [29]. Since the firing order of dummy transitions does not affect the dummy-free timed state class graph if they are concurrent with any other transitions, the state-space explored can be reduced by only considering a single interleaving of firing those dummy transitions. This is effective especially in our case, because the exact timed contraction can contract only a restricted class of dummy transitions, and many dummy transitions sometimes remain in $G_{\mathrm{abs}}$. In other words, the penalty of the synthesis cost due to using the exact contraction can be decreased by these techniques. As mentioned later, some large circuits in our examples could be synthesized only by using these techniques.

## V. Analyzing CSC Violation Traces

If $G_{\mathrm{abs}}$ does not have CSC, a set of CSC violation traces is constructed by **obtain_CSC_violation_trace_set** from the decorated signal state graph. Each of such CSC violation traces is analyzed by **analyze_CSCV_trace**, which is the core part of this paper.

The algorithm **analyze_CSCV_trace** first generates a concrete trace of the original STG $G$ which corresponds to the given CSC violation trace of $G_{\mathrm{abs}}$. This is done by a technique similar to the one developed for partial order reduction, which

[3]In our current implementation, one shortest trace is selected for each CSC violation pair because using all CSC violation traces is very expensive.

```
analyze_CSCV_trace($g$, $G$, $V$) {
    $f$ = guided_sim($g$, $G$, $V$);
    $can$ = find_inputs($f$, $G$, $V$);
    if ($can == false$) return "not found";
    else return $can$;
}
```

Fig. 6. Algorithm for analyzing CSC violation trace.

we call *guided simulation*. Then, it finds a set of requirements for an appropriate input set by analyzing the concrete CSC violation trace. The overall procedure is shown in Fig. 6.

This section first discusses the algorithm to analyze the concrete CSC violation traces, because guided simulation is strongly related to this algorithm. Guided simulation is then discussed. In the following, an *interface signal* means the signals used in $G_{\rm abs}$, i.e., the signals in $V$, and a *noninterface signal* means the remaining signals of $G$, i.e., the signals in $D = \mathsf{sig}(G) - V$. The corresponding transitions are called similarly.

Please note that the proposed algorithm currently has the following restrictions on the class of timed STGs to be handled. Any loop in the given timed STG must contain at least one transition with nonzero delay (i.e., its earliest firing time is greater than zero). This restriction is necessary to guarantee the termination of the second phase of guided simulation (see Section V-C). In the untimed case, this termination is not guaranteed if a loop formed only by noninterface or dummy transitions exists [24]. In the timed case, however, even if such a loop exists, time certainly passes in the loop from the above restriction, and eventually, other transitions are fired. Thus, guided simulation can terminate. For every two reachable timed state classes of the STG, either one is reachable from the other. This restriction is necessary, because every CSC violation pair is found along a single path from the initial timed-state class in our algorithm. This limitation should be eliminated in the future. However, our method can be successfully applied to the standard benchmark suites shown in Table II. Furthermore, this method is intended to handle the timed STGs generated by a compiler to translate high-level specification languages to STGs in a high-level synthesis flow. Those generated STGs usually satisfy the above condition. Hence, we do not consider this as a severe limitation.

## A. Regular Concrete Traces

Each CSC violation trace $g$ of $G_{\rm abs}$, constructed by **obtain_CSC_violation_trace_set**, is assumed to be of the form $g = \langle g_0, g_1, g_2\rangle$, $s_0 \xrightarrow{g_0} s_1$, $s_1 \xrightarrow{g_1} s_2$, and $s_2 \xrightarrow{g_2} s_3$, where $s_0$ is the initial signal state of $G_{\rm abs}$, $s_1$ and $s_2$ are the first two signal states that correspond to the CSC violation pair, and $g_2$ contains exactly one (interface) transition.[4] For this $g$, the corresponding concrete trace $f$ is constructed by guided simulation shown later, where $f = \langle f_0, f_1, f_2\rangle$ such that for $0 \le i \le 2$, projecting out noninterface and dummy transitions from $f_i$ is equal to $g_i$. It is assumed that $f$ is

[4]Choosing those $g_0$, $g_1$, and $g_2$ such that only the first two CSC violation signal states in the reduced STG are captured may cause poor results in the subsequent analysis. Some improvement may be possible in this choice, but currently, we have not tried this.

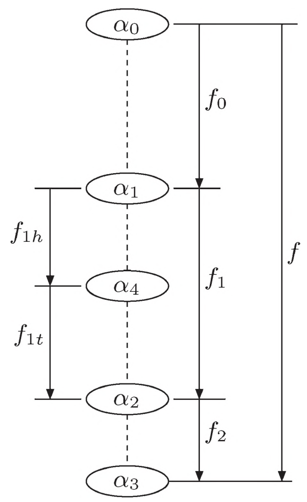


Fig. 7. Labeling of a concrete trace.

separated into $f_i$ when interface transitions fire, i.e., each $f_i$ ends with an interface transition. Let $\alpha_1$, $\alpha_2$, and $\alpha_3$ denote the timed state classes of $G$ obtained by $f_0$, $f_1$, and $f_2$, respectively (see Fig. 7). Since some noninterface or dummy transitions fire concurrently with the interface transitions, there exist many such concrete traces that correspond to $g$. Thus, we first define an equivalence class of traces based on the causality relation.

For two interface transitions $a$ and $b$ in $f$, if $a$ fires before $b$ without any interface transitions between them, it is denoted by $(a, b) \in R_1^f$. For any two transitions $t_1$ and $t_2$ in $f$, if $t_2$ fires by consuming the token produced by the firing of $t_1$, it is denoted by $(t_1, t_2) \in R_2^f$. In the untimed case, the actual causality relation for $f$ is defined by the transitive closure of the union of $R_1^f$ and $R_2^f$, i.e., $(t_1, t_2) \in (R_1^f \cup R_2^f)^*$. In the timed case, however, it is further needed to consider *timed causality*, which is defined in [30]. For any two transitions $t$ and $u$ with $(t, u) \in (R_2^f)^*$, let $\mathsf{path}_f(t, u)$ denote a set of paths from $t$ to $u$ defined by the relation $R_2^f$, where each path is represented by a set of transitions. That is,

$$\mathsf{path}_f(t,u) = \begin{cases} \emptyset, & \text{if } t = u \\ \left\{\{u\} \cup q \mid (t', u) \in R_2^f, q \in \mathsf{path}_f(t, t')\right\}, & \text{otherwise.}\end{cases}$$

For example, if $f$ is a concrete trace obtained from the STG shown in Fig. 9(a), then $\mathsf{path}_f(c+, x-)$ is $\{\{a+, x+, a-, x-\}, \{a+, b-, a-, x-\}\}$. Furthermore, let

$$\mathsf{max\_eft}_f(t,u) = \max_{q \in \mathsf{path}_f(t,u)} \left(\sum_{r \in q} \mathsf{Eft}(r)\right)$$

$$\mathsf{max\_lft}_f(t,u) = \max_{q \in \mathsf{path}_f(t,u)} \left(\sum_{r \in q} \mathsf{Lft}(r)\right).$$

Finally, for any two transitions $t_1$ and $t_2$ in $f$, if there exists a transition $t_3$ in $f$ such that $(t_3, t_1) \in (R_2^f)^*$, $(t_3, t_2) \in (R_2^f)^*$, and $\mathsf{max\_lft}_f(t_3, t_1) < \mathsf{max\_eft}_f(t_3, t_2)$, then it is denoted by $(t_1, t_2) \in R_3^f$. Intuitively, $(t_1, t_2) \in R_3^f$ implies that $t_1$ and $t_2$

has a common ancestor $t_3$, and the maximal time separation between $t_3$ and $t_1$ is smaller than the minimal time separation between $t_3$ and $t_2$. Thus, $t_2$ cannot fire before $t_1$ under the timing constraints associated with the causality relation $R_2^f$. This timed causality relation can be computed using a time-separation algorithm such as shown in [12].

Using the above relations, we say that $t_1$ is an *ancestor* of $t_2$ in $f$, denoted by $[t_1 \rightsquigarrow_f t_2]$, if $t_1$ and $t_2$ are related by the transitive closure of the union of $R_1^f$, $R_2^f$, and $R_3^f$. This ancestor relation represents an actual causality relation with respect to the specific abstracted trace $g$. In the rest of this paper, we use the following terminology.

- $t_1$ *causes* $t_2$, if $[t_1 \rightsquigarrow_f t_2]$ holds.
- $t_1$ and $t_2$ are *ordered*, if either $[t_1 \rightsquigarrow_f t_2]$ or $[t_2 \rightsquigarrow_f t_1]$ holds.
- $t_1$ and $t_2$ are *concurrent*, if they are not ordered.

This actual causality relation defines equivalent classes of traces of $G$. For a trace $f$, let $\|f\|_G$ denote a set of traces of $G$ (including $f$) such that for any $f' \in \|f\|_G$, $(R_1^{f'} \cup R_2^{f'} \cup R_3^{f'})^* = (R_1^f \cup R_2^f \cup R_3^f)^*$ holds.

For a given abstracted trace $g$, our algorithm constructs one particular concrete trace $f$ satisfying a property which we call regularity and analyzes it to handle all traces in $\|f\|_G$. A trace $f$ is *regular*, if for every interface transition $t$ in $f$, all noninterface or dummy transitions that are concurrent with $t$ fire before $t$ in $f$. This regularity is necessary for the following reason. As shown later, for a transition $t$, which is an ancestor of an interface transition $x$ in $f$, it is necessary to find a noninterface signal $w$ such that every $w$ transition in $f$ is ordered with $t$. If every $w$ transition appears in $f$, it is easy to check the above property, i.e., this can actually be done by constructing a data structure similar to an occurrence net [31]. Otherwise, however, it is not clear when the generation of $f$ should be terminated to check the above property with respect to every $w$ transition, if $f$ is nonregular, because a concurrent $w$ transition may fire a long time later. On the other hand, if $f$ is regular, every transition concurrent with $x$ is fired before $x$. Hence, if a regular trace $f$ is generated up to $x$, it can be decided whether every $w$ transition is ordered with $t$ or not, because a $w$ transition that does not appear in $f$, if it exists, is caused by $x$, and so, it is caused by $t$ from $[t \rightsquigarrow_f x]$.

## B. Determining the Input Set

The idea to resolve the CSC violation between $\alpha_1$ and $\alpha_2$ is to add to the input set a noninterface signal $w$ such that $f_1$ contains odd number of $w$ transitions. If a $w$ transition fires in $f_1$ in odd times, then the signal takes different values in $\alpha_1$ and $\alpha_2$, and so, the CSC violation is resolved by adding $w$ to the input set. However, such $w$ may not work for other traces in $\|f\|_G$, unless the causality relation guarantees that it fires certainly odd times in traces in $\|f\|_G$. Thus, we need to define the following notions.

For a (sub)trace $h$, let $\mathsf{final}(h)$ denote the last transition in $h$, and $\mathsf{before}(h)$ the transition fired just before the first transition of $h$ in a currently designated trace. When $h$ starts from the initial timed state class, $\mathsf{before}(h)$ is the virtual transition $v$ that is assumed to cause the first transition of every trace. For a trace $f$, $(l_1, l_2)$ is a *nested subtrace pair* of $f$, if $\mathsf{before}(l_1)$, $\mathsf{before}(l_2)$, $\mathsf{final}(l_2)$, and $\mathsf{final}(l_1)$ are distinct, and they are caused in this order, i.e., $[\mathsf{before}(l_1) \rightsquigarrow_f \mathsf{before}(l_2)]$, $[\mathsf{before}(l_2) \rightsquigarrow_f \mathsf{final}(l_2)]$, $[\mathsf{final}(l_2) \rightsquigarrow_f \mathsf{final}(l_1)]$ (see Fig. 8).

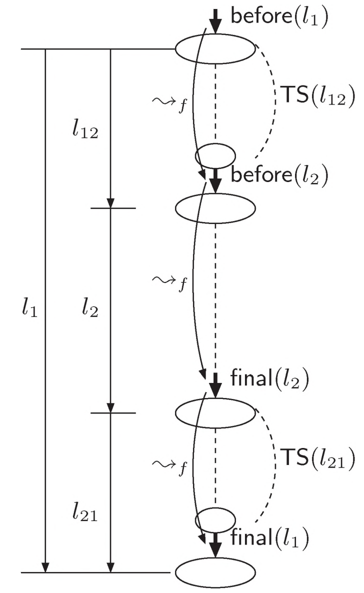


Fig. 8. Nested subtrace pair.

For a signal $w$ and a nested subtrace pair $(l_1, l_2)$ of $f$, $w$ is *semiessential* with respect to $(l_1, l_2)$ in $f$, if

- none of $\mathsf{before}(l_1)$, $\mathsf{before}(l_2)$, $\mathsf{final}(l_2)$, and $\mathsf{final}(l_1)$ is a $w$-transition and
- every $w$ transition in $f$ is ordered with $\mathsf{before}(l_1)$, $\mathsf{before}(l_2)$, $\mathsf{final}(l_2)$, and $\mathsf{final}(l_1)$.

Similarly, $w$ is *essential* with respect to $(l_1, l_2)$ in $f$, if $w$ is semiessential with respect to $(l_1, l_2)$ in $f$, and $l_2$ contains an odd number of $w$ transitions while neither $l_{12}$ nor $l_{21}$ contains any $w$ transition, where $l_{12}$ and $l_{21}$ are the subtraces of $l_1$ before $l_2$ and after $l_2$, as shown in Fig. 8. As mentioned previously, it is easy to check whether $w$ is (semi)essential or not, if $f$ is regular and contains interface transitions at the end of $l_1$ or later.

For $f' \in \|f\|_G$, a nested subtrace pair $(l_1', l_2')$ of $f'$ that corresponds to $(l_1, l_2)$ in $f$ is the nested subtrace pair defined by using $\mathsf{before}(l_1)$, $\mathsf{before}(l_2)$, $\mathsf{final}(l_2)$, and $\mathsf{final}(l_1)$. For a subtrace $l$, let $\mathsf{TS}(l)$ denote a set of timed state classes in which the transitions in $l$ fire (see Fig. 8). $\mathsf{SS}(l)$ denotes the corresponding signal-state set. The following lemma holds.

*Lemma 1:* Let $(l_1, l_2)$ be a nested subtrace pair of $f$ and $w$ be essential with respect to $(l_1, l_2)$. For any $f' \in \|f\|_G$ and the nested subtrace pair $(l_1', l_2')$ of $f'$ that corresponds to $(l_1, l_2)$, the signal states in $\mathsf{SS}(l_{12}')$ are distinguished from those in $\mathsf{SS}(l_{21}')$ by the signal $w$.

*Proof:* The proof is straightforward from the definition of the essentialness. ■

Consider the concrete trace shown in Fig. 7. Since $\alpha_1$ and $\alpha_2$ cause a CSC violation, there exist at least two interface transitions in $f_1$. Note that if there are no interface transitions between $\alpha_1$ and $\alpha_2$, $\alpha_2$ cannot cause CSCV in the dummy-free version of the timed state class graph. Thus, the first interface

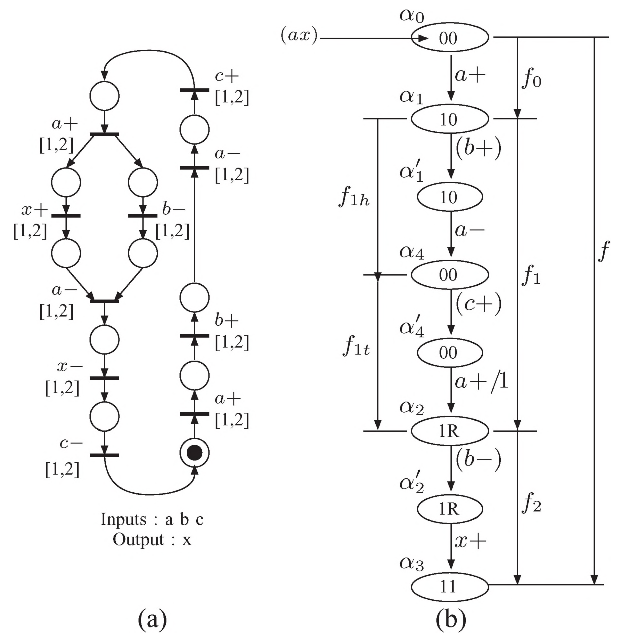


Fig. 9. (a) Simple STG. (b) Its CSC violation trace.

transition in $f_1$, say $y$, exists in the middle of $f_1$. Divide $f_1$ into $f_{1h}$ and $f_{1t}$ with $y$, i.e., $f_1 = \langle f_{1h}, f_{1t}\rangle$ and $f_{1h}$ ends with $y$. Fig. 7 shows the relation among $f_0 \cdots f_2$ as well as $f_{1h}$ and $f_{1t}$. Let $l_1 = \langle f_1, f_2\rangle$ and $l_2 = f_{1t}$. Then, $l_{12} = f_{1h}$ and $l_{21} = f_2$ hold. From the definitions of $f_0$, $f_{1h}$, $f_1$, and $f_2$, they end with interface transitions. Thus, each of them is different, and they are caused in this order. Hence, $(l_1, l_2) = (\langle f_1, f_2\rangle, f_{1t})$ is a nested subtrace pair of $f$. The following theorem holds.

*Theorem 1:* The CSC violation with respect to $f' \in \|f\|_G$ with $f = \langle f_0, f_1, f_2\rangle$ is resolved by adding a noninterface signal $w$ to the input set, if $w$ is essential with respect to $(l_1, l_2) = (\langle f_1, f_2\rangle, f_{1t})$ in $f$.

*Proof:* For any $f' \in \|f\|_G$ and the nested subtrace pair $(l'_1, l'_2)$ of $f'$ that corresponds to $(l_1, l_2)$, the CSC violation is caused between the timed state classes in $\mathsf{TS}(l'_{12})$ and those in $\mathsf{TS}(l'_{21})$. Those signal states are distinguished by $w$ from Lemma 1. ■

For example, consider an STG shown in Fig. 9(a). The output $x$ has the possible trigger signal $a$, and so, the initial $V$ is $\{a, x\}$. Then, the reduced STG with interface signals $a$ and $x$ has one CSC violation trace, and its corresponding concrete trace $f$ is shown in Fig. 9(b). This trace is regular, because a noninterface transition $b-$ is concurrent with an interface transition $x+$, and $b-$ fires before $x+$ in this trace. $\|f\|_G$ contains another trace $f'$ obtained by swapping $b-$ and $x+$ in $f$. $(\langle f_1, f_2\rangle, f_{1t})$ is a nested subtrace pair of $f$, and the noninterface signal $c$ is essential with respect to it. The noninterface signal $b$ is not semiessential, because it is not ordered with $\mathsf{final}(l_1) = x+$. $\mathsf{TS}(l_{12}) = \{\alpha_1, \alpha'_1\}$ and $\mathsf{TS}(l_{21}) = \{\alpha_2, \alpha'_2\}$ cause the CSC violation, and it is resolved by adding the essential signal $c$ to $V$. Actually, for this new $V = \{a, c, x\}$, the reduced STG has CSC, and a circuit for $x$ can be synthesized from it.

If there is no essential signal with respect to $(\langle f_1, f_2\rangle, f_{1t})$ in $f$, the CSC violation cannot be resolved by adding a single noninterface signal. However, CSCV can be resolved by adding two or more noninterface signals to the input set. There are two cases depending on the existence of semiessential noninterface signals.

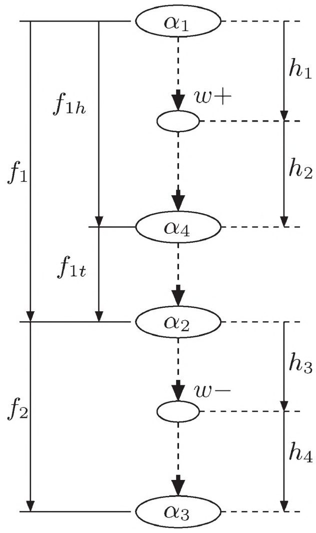
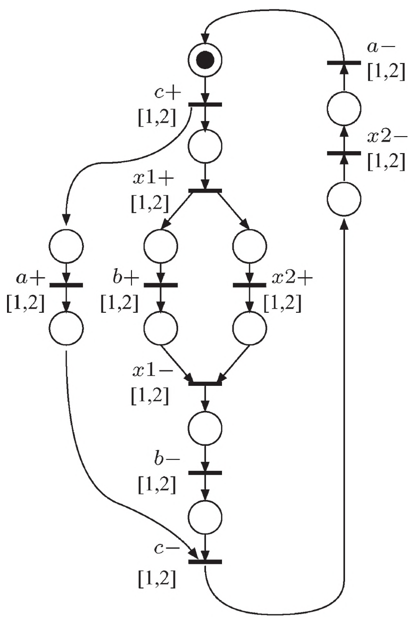


Fig. 10. Nested subtrace pair reduced from the original nested subtrace pair.

If there exist in $f_{1h}$ or $f_2$, noninterface signals that are semiessential with respect to $(\langle f_1, f_2\rangle, f_{1t})$, the problem can be reduced to several subproblems that can be solved by the above approach. Suppose that such a semiessential noninterface signal $w$ changes, as shown in Fig. 10. Then, by adding $w$ to the input set, the timed state classes that cause CSCV (i.e., those in $\mathsf{TS}(f_{1h})$ and $\mathsf{TS}(f_2)$) are divided into two groups: one is $\mathsf{TS}(h_1)$ and $\mathsf{TS}(h_4)$ and the other is $\mathsf{TS}(h_2)$ and $\mathsf{TS}(h_3)$. The first group can be handled by considering a nested subtrace pair $(\langle f_1, f_2\rangle, \langle h_2, f_{1t}, h_3\rangle)$. Even if there exists no essential noninterface signal for $(\langle f_1, f_2\rangle, f_{1t})$, this $(\langle f_1, f_2\rangle, \langle h_2, f_{1t}, h_3\rangle)$ may have it, because $\langle h_2, f_{1t}, h_3\rangle$ is longer than $f_{1t}$. Similarly, the second group can be handled by a nested subtrace pair $(\langle h_2, f_{1t}, h_3\rangle, f_{1t})$, and it may have an essential noninterface signal, because $\langle h_2, f_{1t}, h_3\rangle$ is shorter than $\langle f_1, f_2\rangle$. We say that $(\langle f_1, f_2\rangle, \langle h_2, f_{1t}, h_3\rangle)$ or $(\langle h_2, f_{1t}, h_3\rangle, f_{1t})$ is *reduced* from $(\langle f_1, f_2\rangle, f_{1t})$ with respect to a semiessential signal $w$. As aforementioned, it is important that a reduced nested subtrace pair may have an essential noninterface signal, even if the original nested subtrace pair does not. If every reduced nested subtrace pair has an essential noninterface signal, adding those essential signals as well as $w$ to the input set resolves the CSC violation in $\|f\|_G$. If there exists no essential noninterface signal for some reduced nested subtrace pair $(l_1, l_2)$, the above process can be applied to it as long as semiessential noninterface signals exist for $(l_1, l_2)$, which may solve the CSC violation using more noninterface signals.

Second, if there exists in $f_{1h}$ or $f_2$ no noninterface signal that is semiessential with respect to $(\langle f_1, f_2\rangle, f_{1t})$, a more complicated process is necessary to resolve the CSC violation in $\|f\|_G$. Suppose that the nested subtrace pair that is considered here is $(l_1, l_2)$. Our algorithm finds for a noninterface transition $t$ fired in $f$, which is concurrent with at least one transition among $\mathsf{before}(l_1)$, $\mathsf{before}(l_2)$, $\mathsf{final}(l_2)$, and $\mathsf{final}(l_1)$. Let $u$ denote such a transition among them, with which $t$ is concurrent. Our algorithm then generates two traces $f'$ and $f''$ from $f$ by



Inputs : a, b
Output : c, x1, x2

Fig. 11. STG that has no essential signals.

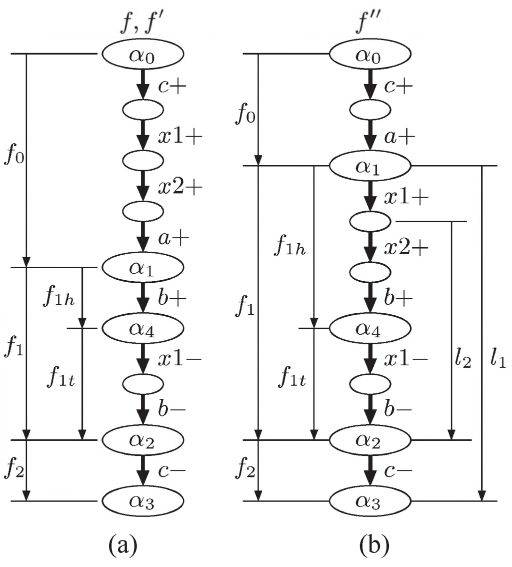


Fig. 12. (a) Original traces. (b) Newly generated traces.

interleaving $t$ and $u$. It means that $f'$ and $f''$ are obtained from $f$ by adding the ancestor relation such that $t$ causes $u$ in $f'$, and $u$ causes $t$ in $f''$. More precisely, suppose that $t$ and $u$ fire in this order in $f$. The new trace $f'$ is obtained simply by adding $[t \rightsquigarrow_f u]$ to the ancestor relation of $f$. Similarly, $f''$ is obtained by adding $[u \rightsquigarrow_f t]$ to the ancestor relation of $f$, but in order to make the firing order of $f''$ consistent with this modified ancestor relation, it is necessary to move $t$ next to $u$, because from $[u \rightsquigarrow_{f''} t]$, $t$ cannot exist before $u$ in $f''$. In addition, the noninterface transitions $v$ that are caused by $t$ and fired before $u$ should also be moved with $t$, because from $[u \rightsquigarrow_{f''} t]$ and $[t \rightsquigarrow_{f''} v]$, $v$ cannot exist before $u$ in $f''$ either. The idea is that our algorithm tries to predict from those interleavings the situation where the noninterface signal $w$ that is related to $t$ is added to the input set and every transition is ordered with the other interface transitions, before actually regenerating the state space of the reduced STG for the modified input set.

For example, consider the STG shown in Fig. 11 and its output $c$. The possible trigger signals for $c$ are $a$ and $b$. For $V = \{a, b, c\}$, $G_{\text{abs}}$ has a CSC violation trace $g = c{+}\ a{+}\ b{+}\ b{-}\ c{-}$ with $g_0 = c{+}\ a{+}$, $g_1 = b{+}\ b{-}$ and $g_2 = c{-}$. Guided simulation generates $f_0 = c{+}\ x1{+}\ x2{+}\ a{+}$, $f_1 = b{+}\ x1{-}\ b{-}$, and $f_2 = c{-}$, as shown in Fig. 12(a). Our algorithm first looks for a semiessential noninterface signal for $(\langle f_1, f_2\rangle, f_{1t})$, but both noninterface transitions $x1{+}$ and $x2{+}$ are concurrent with $\mathsf{before}(\langle f_1, f_2\rangle) = a{+}$. Thus, this STG has no semiessential signals. Here, choose $x1$ and $\mathsf{before}(\langle f_1, f_2\rangle) = a{+}$, and generate $f'$ and $f''$ by interleaving them. It is not easy to illustrate these traces in a figure like Fig. 12, because it does not show the ancestor relation precisely, but assume that in Fig. 12, $x1{+}$ causes $a{+}$ in $f'$ while $a{+}$ causes $x1{+}$ in $f''$. Note that in $f''$, $x1{+}$ and $x2{+}$ (which is caused by $x1{+}$ and fired before $a{+}$) are moved next to $a{+}$. In $f'$, $x1$ is now essential with respect to $(\langle f_1, f_2\rangle, f_{1t})$, because every $x1$ transition is ordered with $a{+}$, $b{+}$, $b{-}$, $c{-}$, and only one $x1{-}$ exists in $f_{1t}$. Thus, this CSC violation can be resolved in $\|f\|_G$ by adding $x1$ in the input set. In $f''$, $x1$ is semiessential, but not essential, because $x1$ fires in $f_{1h}$. This corresponds to the first case above, and can be handled by considering one reduced nested subtrace pair $(l_1, l_2)$ as shown in Fig. 12(b), because only $\alpha_1$ and $\alpha_2$ cause the CSC violation after adding $x1$. Then, our algorithm looks for another noninterface transition that is essential with respect to $(l_1, l_2)$ in $f''$, which is $x2$ in this case. In other words, $x2$ can resolve the CSC violation between $\mathsf{TS}(l_{12}) = \{\alpha_1\}$ and $\mathsf{TS}(l_{21}) = \{\alpha_2\}$. Therefore, the CSC violation with respect to $f''$ can be resolved in $\|f''\|_G$ by adding both $x1$ and $x2$.

In general, the above process should be repeated to generate new traces by interleaving some concurrent transitions. Furthermore, there are usually many choices for noninterface signals. Thus, generating as many such combinations as possible is desirable. The whole process for both cases is described by the pseudocode shown in Fig. 13. This procedure constructs a Boolean expression $E$ over a set of noninterface signals such that for each feasible assignment of $E$, the CSC violation is resolved in $\|f\|_G$ by adding the noninterface signals that have one in the assignment. Then, it is converted to a conjunctive normal form (CNF). Thus, one noninterface signal in each clause of the CNF should be added to resolve the CSC violation. Hence, this CNF represents a set of requirements, and solving the covering problem that all these requirements are satisfied obtains the optimal set of signals to be added, which is done in **obtain_abs** as mentioned previously.

The following theorem holds.

*Theorem 2:* The algorithm shown in Fig. 13 returns "false," only when the given STG have no CSC.

*Proof:* The algorithm returns "false," either when a pair $(t, u)$ does not exist, or when some call of **find_essential** in the last "forall" loop returns "false." The latter case happens again when either of the above two cases happens in the recurred **find_essential**. Since the recursion eventually terminates, the former case should happen in some recursion. When the former case happens, there exists no noninterface transition that is

```
find_inputs($f$, $G$, $V$) {
  $A$ = build_ancestor_relation($f$, $G$);
  $\langle f_0, f_1, f_2\rangle = f$;
  $\langle f_{1h}, f_{1t}\rangle = f_1$;
  $E$ = find_essential($\langle f_1, f_2\rangle$, $f_{1t}$, $f$, $A$);
  return convert_to_CNF($E$);
}

find_essential($l_1$, $l_2$, $f$, $A$) {
  $E = false$;
  for each noninterface signal $w$ {
    if ($w$ is essential w.r.t. $(l_1, l_2)$ in $f$)
      $E = E \vee w$;
  }
  if ($E \neq false$) return $E$;
  for each noninterface signal $w$ {
    if ($w$ is semi-essential w.r.t. $(l_1, l_2)$ in $f$) {
      $F = w$;
      $N$ = reduce_nested_subtrace_pairs($w$, $l_1$, $l_2$, $f$);
      forall $(h_1, h_2) \in N$ {
        $F'$ = find_essential($h_1$, $h_2$, $f$, $A$);
        $F = F \wedge (F')$;
      }
      $E = E \vee F$;
    }
  }
  if ($E \neq false$) return $E$;
  $(t, u)$ = pick a transition $t$ in $f$ concurrent with
                $\exists u \in \{\mathsf{before}(l_1), \mathsf{before}(l_2), \mathsf{final}(l_2), \mathsf{final}(l_1)\}$;
  if (such $(t, u)$ does not exist) return $false$;
  $N$ = generate_interleavings($t$, $u$, $f$);
  $E = true$;
  forall $(l_1', l_2', f', A') \in N$ {
    $F$ = find_essential($l_1'$, $l_2'$, $f'$, $A'$);
    $E = E \wedge (F)$;
  }
  return $E$;
}
```

Fig. 13. Algorithm for determining the input set.

concurrent with some of $\mathsf{before}(l_1)$, $\mathsf{before}(l_2)$, $\mathsf{final}(l_2)$, and $\mathsf{final}(l_1)$. It implies that every noninterface signal except for the signals related to $\mathsf{before}(l_1)$, $\mathsf{before}(l_2)$, $\mathsf{final}(l_2)$, and $\mathsf{final}(l_1)$ is semiessential. Since no essential signal exists and no reduced nested subtrace pair works, such that every noninterface signal included in $\langle f_1, f_2\rangle$ must appear in $f_{1t}$ even times. Hence, the CSC violation in $\mathsf{TS}(f_{1h})$ and $\mathsf{TS}(f_2)$ cannot be resolved, even if every noninterface signal is used. This implies that the given STG has no CSC. ■

```
guided_sim($g$, $G$, $V$) {
  $h$ = guided_sim_phase1($g$, $G$, $V$, $\alpha_0$, $null$);
  $f$ = guided_sim_phase2($h$, $G$, $V$, $\alpha_0$);
  return $f$;
}

guided_sim_phase1($g$, $G$, $V$, $\alpha$, $h$) {
  if ($g$ is empty) return $h$;
  if (($g, \alpha$) is already visited) return "backtrack";
  $g_1$ = head($g$);
  $nec$ = necessary($\alpha$, $g_1$);
  $dep = \emptyset$;
  forall $t \in nec$
    $dep = dep \cup$ dependent($\alpha$, $t$);
  forall $t \in dep$ {
    $\alpha'$ = fire($\alpha$, $t$);
    if ($t == g_1$) $g'$ = tail($g$);
    else $g' = g$;
    $h'$ = append($h$, $t$);
    $result$ = guided_sim_phase1($g'$, $G$, $V$, $\alpha'$, $h'$);
    if ($result \neq$ "backtrack") return $result$;
  }
  return "backtrack";
}

dependent($\alpha$, $t$) {
  $E = new = \{t\}$;
  while(true) {
    $new' = \emptyset$;
    forall $x \in new$
      forall $y \in conflict(x) \cap NonIF\_Dum$
        $new' = new' \cup$ necessary($\alpha$, $y$) $- E$;
    if ($new' == \emptyset$) break;
    $E = E \cup new'$;
    $new = new'$;
  }
  return $E$;
}

necessary($\alpha$, $t$) {        /* $\alpha = (\mu, I)$ */
  if ($t$ is already visited) return $\emptyset$;
  if ($t \in \mathsf{enabled}(\mu)$)
    if ($t \in \mathsf{firable}(\alpha)$) return $\{t\}$;
    else return {choose_one($\mathsf{firable}(\alpha) \cap NonIF\_Dum$)};
  $E = \emptyset$;
  $p$ =choose_one($\bullet t - \mu$);
  forall $x \in \bullet p \cap NonIF\_Dum$
    $E = E \cup$ necessary($\alpha, x$);
  return $E$;
}
```

Fig. 14. Algorithm for guided simulation (1).

## C. Guided Simulation

For a given abstracted trace $g$, guided simulation obtains a regular trace $f$ of $G$ such that a trace obtained by projecting out the noninterface and dummy transitions from $f$ is equal to $g$. The algorithm is shown in Fig. 14. It consists of two phases. Phase 1 generates a concrete trace $h$ that satisfies the projection condition but not the regularity. Actually, for each interface transition $t$ appearing in $g$, the noninterface and dummy transitions that cause $t$ (by $R_2^f$ and $R_3^f$) are certainly contained before $t$ in $h$, but those that are concurrent with $t$ may either appear after $t$ or not appear in $h$. In phase 2, the concurrent noninterface or dummy transitions are added to $h$ or moved in order to satisfy regularity.

In **guided_sim_phase1**, if $g$ is nonempty, it picks the first transition of $g$, denoted by $g_1$, and computes its necessary set, i.e., the set of transitions that should be fired to fire $g_1$, using **necessary**. In **necessary**, if $t$ is not enabled, one of its empty source places is traversed upward along the noninterface or dummy transitions (*NonIF_Dum* denotes the set of all noninterface and dummy transitions) recursively. If $t$ is enabled and firable, it is returned. If $t$ is enabled, but not firable, some firable transition must precede $g_1$, and so, one of the firable noninterface or dummy transitions chosen by **choose_one** is returned. In the net shown in Fig. 15, where $t_1 \cdots t_5$ are noninterface transitions, and $g_1$ and $g_2$ are interface transitions, the necessary set of $g_1$ is $\{t_1\}$. **guided_sim_phase1**

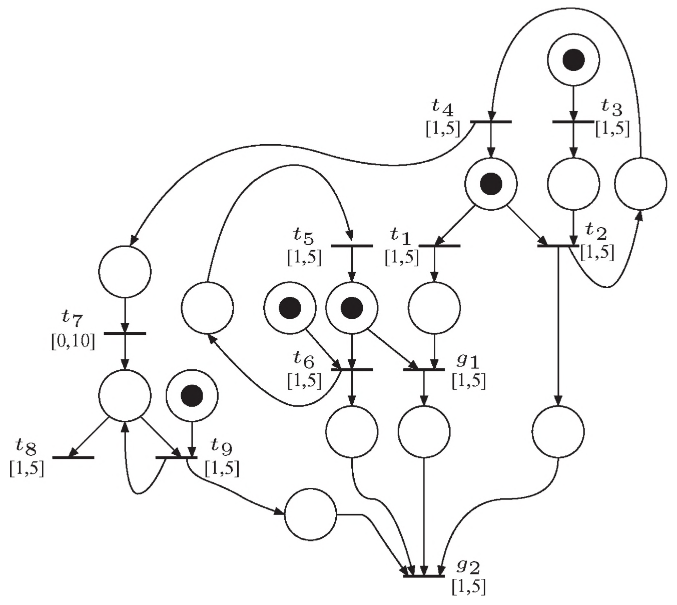


Fig. 15. Examples for guided simulation.

then computes a dependent set of each necessary transition. The dependent set of a transition $t$ is a set of transitions whose firings may be necessary before $t$ in order to avoid missing the possible concrete traces. For example, consider generating a concrete trace of the net shown in Fig. 15 for an abstracted trace $g_1g_2$. As shown above, the necessary set of $g_1$ is $\{t_1\}$. If $t_1$ is fired from the current marking, $g_1$ becomes enabled, but $g_2$ can never be fired. In this case, firing $t_3$, $t_2$, $t_4$, and $t_1$ in this order leads to the correct concrete trace. It is, however, not easy to find such a correct firing sequence directly. Instead, our algorithm guarantees to generate the correct concrete trace by a backtracking mechanism that fires a sufficient set of transitions in each step. Such a sufficient set of transitions is the dependent set. It is formally stated that a dependent set of transition $t$ is a set $E$ of transitions, satisfying $t \in E$, and for each $x \in E$, the necessary transitions of $conflict(x) \cap NonIF\_Dum$ are also in $E$. The dependent set can be defined as a closure, and so, the while loop in Fig. 14 computes it. In our example, the dependent set of $t_1$ is $\{t_1, t_3\}$. Hence, even if **guided_sim_phase1** fires $t_1$ first, it eventually can fire $t_3$ and, then, $t_2$ after several backtrackings. After firing $t_2$, it is straightforward to fire $t_4$ and $t_1$, because they are the necessary transitions for $g_1$. When $g_1$ becomes enabled, its necessary set is $\{g_1\}$ and its dependent set is $\{g_1, t_6\}$. The correct transition to be fired here is $t_6$ and is again guaranteed to be found by the backtracking mechanism. The fired transitions are appended to the trace $h$, and it is returned when $g$ becomes empty. Then, the trace $h$ is passed to **guided_sim_phase2** in order to generate a regular trace based on $h$. Note that it is possible that the algorithm happens to choose a correct transition in the dependent set in the first trial. In this case, backtrackings are not necessary, because the expected trace is obtained. In other words, every transition in the dependent set is not necessarily tried. This is why **guided_sim_phase1** terminates when a result that is not "backtrack" is obtained in its second "forall" loop.

In **guided_sim_phase2**, each transition determined by **find_firing_trans** is fired until $h$, which is also updated by **find_firing_trans**, becomes empty as shown in Fig. 16.

```
guided_sim_phase2($h$, $G$, $V$, $\alpha$) {
  $f$ = null;
  fired = null;
  while(true) {
    $(t, h)$ =find_firing_trans($\alpha$, $h$);
    $f$ = append($f$, $t$);
    $\alpha$ = fire($\alpha$, $t$);
    if ($h$ is empty) break;
  }
  return $f$;
}

find_firing_trans($\alpha$, $h$) {
  while(true) {
    $(t, h')$ = (head($h$), tail($h$));
    if ($t$ is interface)
      return find_concur_trans($\alpha$, $t$, $h$, $h'$);
    else
      if ($t \notin fired$) return $(t, h')$;
      else {
        $fired = fired - \{t\}$;
        $h = h'$;
      }
  }
}

find_concur_trans($\alpha$, $t$, $h$, $h'$) {
  find $x \in$ firable($\alpha$) $\cap$ NonIF_Dum
        s.t. $conflict(x) \cap (\text{prefix}(h - fired, x) = \emptyset$;
  if (such $x$ exists) {
    $fired = fired \cup \{x\}$;
    return $(x, h)$;
  }
  else return $(t, h')$;
}
```

Fig. 16. Algorithm for guided simulation (2).

In **find_firing_trans**, if the first transition $t$ of $h$ is an interface transition, it implies that $t$ is enabled in the current marking because all its necessary transitions are supposed to be fired. In order to satisfy regularity, however, firable noninterface or dummy transitions that are concurrent with $t$ should be fired before $t$, if they exist. This should be done carefully, if such a noninterface or dummy transition $x$ is in conflict with some other transition. In such a case, an appropriate transition should be chosen such that it is consistent with the rest of $h$ (up to the point where $x$ fires). Let prefix$(h, x)$ denote a prefix of $h$ before the occurrence of $x$. It is equal to $h$, if $x$ is not included in $h$. Then, if $conflict(x) \cap \text{prefix}(h, x) \neq \emptyset$, it implies that some other transition conflicting with $x$ fires before $x$ in $h$, and so, $x$ should not be fired in the current timed state class. Thus, it is necessary to find a noninterface or dummy firable transition $x$ such that $conflict(x) \cap \text{prefix}(h, x) = \emptyset$. If such an $x$ is found, it is returned with the nonupdated $h$ keeping the interface transition in its head. In this case, $x$ is added to a global variable fired to avoid firing it again when it is in the top of $h$. Considering the set fired, the above condition should be rewritten to $conflict(x) \cap \text{prefix}(h - \text{fired}, x)$. If such $x$ does not exist, either there are no firable noninterface or dummy transitions, or every such transition conflicts with some interface transition in $h$. In either case, no firable noninterface or dummy transitions are concurrent with $t$. Hence, $t$ and $h' = \text{tail}(h)$ are returned.[5]

If $t$ is noninterface or dummy, regularity just requires that $t$ should be fired. But, it may be already fired as mentioned above to generate regular traces when it is concurrent with some interface transition. Thus, if $t \in$ fired, it is just removed from fired, and the next transition of $h$ is processed. Otherwise, $t$ and $h' = \text{tail}(h)$ are returned.

In the previous example, for an abstracted trace $g_1 g_2$

$$t_3\, t_2\, t_4\, t_1\, t_6\, t_5\, g_1\, t_7\, t_9\, g_2$$

is obtained by **guided_sim_phase1**. This does not satisfy regularity, because $t_7$ and $t_9$ fire after $g_1$. When $h = g_1\ t_7\ t_9\ g_2$ is first given to **find_firing_trans**, $(t_7, h)$ is returned with fire $= \{t_7\}$. After firing $t_7$, both $t_8$ and $t_9$ become firable. $t_8$ is not chosen, because $conflict(t_8) = \{t_9\}$ and $\text{prefix}(h, t_8) = \{t_9\}$. After firing $t_9$, however, $t_8$ is chosen and fired, because it is concurrent with $g_1$, and so, it must be fired before $g_1$ to obtain a regular trace. Note that $t_8$ is not fired in **guided_sim_phase1**, because it is not necessary to fire $g_2$ (Remember that **guided_sim_phase1** fires only transitions that are necessary to obtain the expected trace, and **guided_sim_phase2** adds some other transitions to the trace to obtain a regular trace). Finally, $g_1$ and $g_2$ are fired, and the following regular trace is obtained:

$$t_3\, t_2\, t_4\, t_1\, t_6\, t_5\, t_7\, t_9\, t_8\, g_1\, g_2.$$

Note that this second phase is deterministic (backtracking is not necessary), because every causal transition needed for interface transitions in $g$ is found in the phase 1. An alternative of guided simulation with only one phase is possible, but from our experience, it causes more backtracking than the two-phase guided simulation shown above.

## VI. Discussion

While it is expected that the average case complexity of the proposed method is much smaller than that of the nondecomposition-based logic-synthesis methods, our method does not improve the worst case complexity. This is obvious if one considers a timed STG, in which every input and output signal in the STG is necessary for a subcircuit of each output signal in the STG. Our method finally obtains the whole set of signals as the input set and explores the full state space to synthesize each subcircuit. Thus, in this case, the synthesis cost is even larger than the nondecomposition-based method due to the overhead to compute the input sets. For more practical cases where the above situation does not occur, it is also necessary to discuss the complexity of the techniques used in our method. First of all, the number of CSC violation traces is exponential in the size of the reduced STG. This number can be kept below some (given) upper bound at the sacrifice of the optimality of the synthesized circuits. For each CSC violation trace, the worst case complexity for finding a regular trace is exponential in the size of the STG, because phase 1 of the guided simulation may have to generate every firing sequence associated with each conflicting transition using the backtracking mechanism. The number of these sequences is exponential in the size of the STG in worst cases. This high worst case complexity is, however, not a big problem, because STGs do not usually contain so many conflicting transitions. To obtain the timed causality relation, $\text{max\_eft}_f(t, u)$ and $\text{max\_lft}_f(t, u)$ need to be computed for each pair $(t, u)$ of transitions and a regular concrete trace, each of which can be done with cost linear in the size of $f$. Thus, the whole computation needs the complexity $O(|f|^3)$. Note that $|f|$ is linear in the size of the STG, because a regular concrete trace does not contain any repeated behavior. Finally, the worst case cost to find the candidates of input signals for each $f$ is exponential in $|f|$, because the number of interleavings created by concurrent transitions is in general exponential in $|f|$. Note that this cost can also be controlled by limiting the recursion depth of **find_essential** at the sacrifice of the optimality.

TABLE I
Comparison Between SI-Circuit-Synthesis Tools

| Circuits | Synthesis time (sec.) | | | Literal Counts | | |
|---|---|---|---|---|---|---|
| | Prop. | moebius | csat† | Prop. | moebius | csat |
| PpWkCsc(3,9) | 0.2 | 2.0 | 0.1 | 156 | 130 | 156 |
| PpWkCsc(3,12) | 0.3 | 13.0 | 0.3 | 210 | 173 | 210 |
| TangramCsc(3,2) | 0.4 | 3.0 | 0.2 | 106 | 103 | 100 |
| TangramCsc(4,3) | 12.7 | 39.0 | 1.3 | 248 | 247 | 244 |
| ArtCsc(10,9) | 298.6 | 6240.0 | 7545.6 | 1423 | 2128 | 1283 |

†: SAT instance generation times are not included.

## VII. Experimental Results

The proposed method has been naively implemented using the C language.[6] This section evaluates the potential performance of the proposed method and the area overhead of the synthesized circuits. The experiments here have been done on a 2.8-GHz Pentium-4 workstation with 4 GB of memory.

For speed-independent circuit synthesis, tools moebius [22] and csat [23] use a similar decomposition-based synthesis. Our tool has an optional mode to handle untimed STGs and synthesize speed-independent circuits by just using untimed state-space exploration and untimed trace analysis. Using this untimed mode of our tool, this section first compares the performance of speed-independent circuit synthesis between our method and the above tools. Since moebius is not currently available to the public due to a licensing problem, we used their benchmark suites and compared our results with the experimental results from their latest paper [32]. Their experiments were done on a 2.53-GHz Pentium 4 with 512-MB memory. As for csat, Khomenko sent us the experimental version of their tool. Since it works on a Windows machine, our experiments use a 3.06-GHz Pentium 4 with 1-GB memory. The results in Table I show this comparison. It seems that the literal counts shown in [32] were obtained from logic equations optimized in some way

[5]If transitions conflicting with interface transitions form loops, then there can exist many nonequivalent concrete traces that correspond to $g$. In our current implementation, one shortest regular concrete trace is selected.

[6]The interleaving generation part in **find_essential** is not fully implemented currently. It means that some noninterface signal that can actually resolve the CSC violation may not be found by our current implementation. This may lead to larger input sets, and so, a larger number of literals may be needed in the synthesized circuits.

TABLE II
EXPERIMENTAL RESULTS (1)

| Circuit | Literal counts | |
|---|---|---|
| | atacs | Proposed |
| alloc-outbound | 16 | 16 |
| atod | 9 | 9 |
| chu150 | 11 | 11 |
| chu172 | 6 | 6 |
| converta | 12 | 12 |
| dff | 8 | 8 |
| master-read | 26 | 26 |
| mp-forward-pkt | 16 | 16 |
| nak-pa | 20 | 20 |
| nowick | 18 | 18 |
| pe-rcv-ifc | 50 | 51 |
| pe-send-ifc | 60 | 61 |
| ram-read-sbuf | 20 | 20 |
| rcv-setup | 8 | 8 |
| sbuf-ram-write | 19 | 19 |
| sbuf-read-ctl | 13 | 13 |
| sbuf-send-pkt2 | 19 | 19 |
| sendr-done | 5 | 5 |
| trimos-send | 21 | 21 |
| vbe10b | 21 | 21 |
| vbe4a | 6 | 6 |
| vbe6a | 17 | 17 |
| vmebus-arb | 9 | 9 |
| wrdata | 11 | 11 |
| wrdatab | 29 | 26 |
| rappid | 152 | 152 |

TABLE III
COMPARISON BETWEEN UNTIMED AND TIMED-CIRCUIT SYNTHESIS

| Circuits | Synthesis time (sec.) | | Literal counts | |
|---|---|---|---|---|
| | Untimed | Timed | Untimed | Timed |
| IIR_a | 13.9 | 18.1 | 449 | 391 |
| IIR_b | 19.2 | 65.0 | 567 | 462 |
| FIR_a | 215.5 | 268.0 | 1155 | 923 |
| FIR_b | 285.7 | 952.7 | 1604 | 1116 |
| DCT_a | 2952.0 | 3700.3 | 2207 | 1870 |
| DCT_b | 2752.2 | 3345.9 | 2496 | 1902 |

(e.g., gate sharing). Thus, their values are sometimes smaller. It should also be noted that csat currently builds all the speed-independent solutions with minimal supports and that there can be combinatorially many of them.

For the timed circuit synthesis, in order to evaluate the area overhead of the proposed method, small timed specifications, which are obtained from the standard benchmarks by adding the fixed lower bound and upper bounds to each transition ([3], [5] for output transitions, [8], [10] for input transitions), are synthesized by the proposed method and a timed circuit synthesis tool atacs [33], and the literal counts of the synthesized circuits are compared. Table II, except for the last line, shows these results. These results show that the quality at least with respect to the area size is not badly affected, even though our method uses restricted information for synthesizing subcircuits, and so may choose nonoptimal input sets. Since these example are small, the CPU times for both methods are almost the same.

The last example shown in Table II is the control circuit for RAPPID. This example is larger, and so, atacs cannot complete the synthesis on the flat specification without hierarchical decomposition. The literal count shown for atacs in the table is obtained using hierarchical decomposition. The proposed method synthesizes it within 15 s.

Table III shows the results for much larger examples, which are taken from [15]. They are specifications for IIR filters, FIR filters, and the first phase of the discrete-cosine-transform (DCT) circuit obtained from SpecC/Balsa high-level specifications (slightly modified versions are used for this paper due to some improvement of our Balsa compiler). Those with "_b" are allowed to use more operational units. Thus, they have more concurrence than those with "_a." In order to evaluate the performance of the timed circuit synthesis, this table also shows the performance of synthesis of the untimed version (speed independent) of the above specifications. Note that partial order reduction and POSET techniques are used for these examples, because otherwise, timed circuit synthesis does not terminate due to many dummy transitions left by the exact timed net contraction (the untimed circuit synthesis has not improved much by these techniques). Although the timed circuit synthesis takes a longer time especially for the specifications with more concurrence, much more compact circuits (compared with the untimed versions) are successfully synthesized (by using the above techniques) without significant performance penalty.

The untimed version of IIR_a is synthesized by atacs in about 200 s, but it runs out of memory for the other specifications. Since there is no hierarchy information for those designs, the hierarchal synthesis of atacs does not work. The only circuit synthesized by atacs from the untimed version of IIR_a has the same literal count 449 as the one synthesized by our method.

Finally, in order to discuss the limits of the proposed method, this section considers one more example. This is a circuit also for the first phase of DCT, but four ALUs, four multipliers, and two two-port memories are used to obtain maximal performance utilizing the maximal concurrence of the DCT algorithm. This circuit is taken from our ongoing project, which aims at the precise comparison between synchronous and asynchronous practical implementations of the same algorithm on the same platform (e.g., a field programmable gate array). Since its timed STG is much more concurrent than those of previous examples, its synthesis cost is very high. Actually, our method could not synthesize subcircuits for several outputs from the original STG due to memory overflow. Thus, in order to reduce the concurrence of those output signals, two auxiliary signals are inserted to the original STG by hand. It makes it possible to synthesize all subcircuits successfully. The total time for synthesis was 36 206 s. This paper has suggested that one of the limits in our method is decided by the concurrence. To make it clearer, a simple scalable STG shown in Fig. 17 is considered. Table IV shows the CPU times and the amount of memory needed for the synthesis. Our method cannot handle the STG with $n = 20$ due to memory overflow. This is because our method uses an explicit timed-state representation. This problem may be avoided to some degree by using a symbolic timed-state representation like the one proposed in [34], but it is not clear how difficult it is to implement the CSC-violation-trace analysis on it.

## VIII. CONCLUSION

This paper presents a decomposition-based method for efficient synthesis of large timed circuits. The idea proposed for

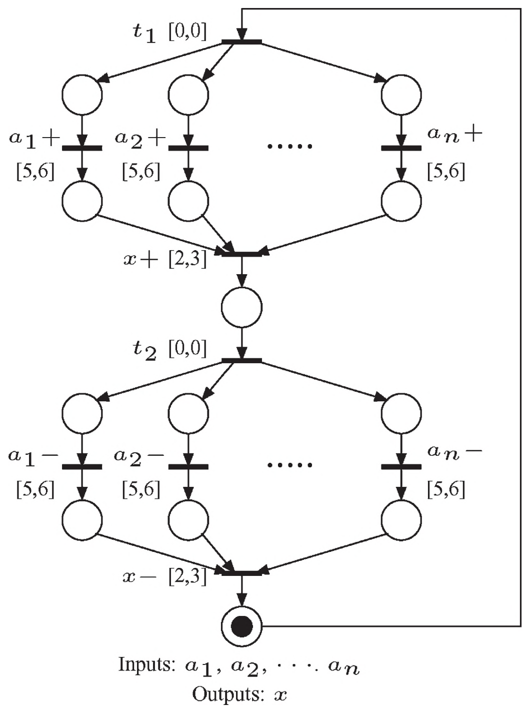


Fig. 17. Simple scalable STG.

TABLE IV
SYNTHESIS TIMES AND AMOUNT OF MEMORY NEEDED FOR VARIOUS $n$

| $n$ | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|
| Synthesis times (s) | 40 | 152 | 581 | 2248 | 8924 | - |
| Memory (MB) | 88 | 180 | 374 | 781 | 1637 | ($> 3GB$) |

speed-independent circuits [24] has been extended for timed circuit synthesis. Since the state spaces of the original timed STGs are not needed to be explored, the proposed method allows for the synthesis of large timed circuits that could not be synthesized using conventional flat synthesis methods. Although this method does have some area overhead for small circuits, the experimental results show that the overhead appears to be very small.

Our future work includes an extension of the proposed method such that it can handle specifications without CSC. Since the cost for solving CSC is usually very high, it would be nice if a CSC-solving algorithm could be applied to reduced STGs. A state variable insertion is one of the major approaches to automatic CSC solving. It is, however, allowed only before the output-signal transitions on an STG, because the interface behavior should not be changed. This restriction makes the state variable insertion in the reduced STG difficult, because almost all transitions in the reduced STG are for input signals in our definition. Actually, some input signals in the reduced STG are output signals in the original STG. Therefore, our algorithm first needs to be extended to handle reduced STGs containing multiple output signals. Then, finding appropriate output signals before which state variables for solving CSC are inserted can be done based on the idea similar to the CSC-violation analysis proposed in this paper. It is very interesting to see how this decomposition-based CSC-solving approach improves the performance compared to the traditional full-state-based CSC solvers.

## APPENDIX
## DECOMPOSITION THEORY

In this appendix, it is assumed that the given STG $G$ is synthesizable and timed implementable.

For $x \in Out$, $ES(x+)$ denotes a set of nondecorated signal states mapped from reachable timed states of a dummy-free timed state graph $\mathcal{G}_G^{\text{df}}$, where their decorated signal states have $R$ for $x$, and $QS(x+)$ denotes a set of similar nondecorated signal states except that their decorated signal states have one for $x$. $ES(x-)$ and $QS(x-)$ are defined similarly. The other nondecorated signal states are unreachable, and this set is denoted by $UR$. From the definition of CSC, if and only if an STG has CSC, its $ES(x+)$, $QS(x+)$, $ES(x-)$, and $QS(x-)$ are disjoint for each $x \in Out$.

A circuit is defined by a set of logic functions (i.e., the technology mapping is beyond the scope of this paper), and a logic function is specified by a *cover*, which is a set of nondecorated signal states where the logic function takes the value of one. In this paper, the implementation technologies considered are *atomic gates* and *generalized-C (gC) elements*. In the atomic-gate implementation, an STG $G$ defines for each $x \in Out$ a cover, denoted by $C(x)$, satisfying

$$C(x) - UR = ES(x+) \cup QS(x+).$$

In the gC implementation, $G$ defines two covers $C(x+)$ and $C(x-)$ satisfying

$$ES(x+) \subseteq C(x+) - UR \subseteq ES(x+) \cup QS(x+)$$
$$ES(x-) \subseteq C(x-) - UR \subseteq ES(x-) \cup QS(x-).$$

An STG $G_1$ is *cover-correct* with respect to $G$, if for each output signal of $G_1$, the covers $C_1(x)$ or $C_1(x+)$, $C_1(x-)$ for $G_1$ satisfy the above conditions of the covers for $G$. For example, in the case of the atomic-gate implementation, $C_1(x)$ satisfying $C_1(x) - UR_1 = ES_1(x+) \cup QS_1(x+)$ must satisfy $C_1(x) - UR = ES(x+) \cup QS(x+)$.

In order that a correct delay can be assigned to the synthesized circuit, another property is needed for the correctness of $G_1$. An STG $G_1$ is *delay-correct* with respect to $G$, if $G_1$ is timed implementable, and for every output signal $x$ of $G_1$, every $x$ transition of $G_1$ has the same firing-time bounds (i.e., $[\mathsf{Eft}(t), \mathsf{Lft}(t)]$) as $x$ transitions in $G$.

If $G_1$ is both cover-correct and delay-correct with respect to $G$, $G_1$ is *correct* with respect to $G$. Intuitively, a circuit synthesized from $G_1$ behaves as expected in $G$ in a sense that the logic function takes the value one in the states where $G$ expects the output to be excited or stable high, as long as the circuit is in the context that $G$ considers. Note that from the timed implementability of $G$, the delay-correctness of $G_1$ is easily achieved by disallowing the contraction for the transitions in $\mathsf{trigger}(x) \cup \{x\}$ for $x \in \text{Out}_1$. Thus, the rest of this appendix focuses on the cover-correctness.

For STGs $G_1$ and $G_2$ with $\mathrm{Out}_1 = \mathrm{Out}_2$ and $\mathrm{In}_1 = \mathrm{In}_2$, a *simulation* from $G_1$ to $G_2$ is a relation $S$ between timed states of $\mathcal{G}^{\mathrm{df}}_{G_1} = (\langle V'_1, E'_1\rangle, \sigma^0_1)$ and $\mathcal{G}^{\mathrm{df}}_{G_2} = (\langle V'_2, E'_2\rangle, \sigma^0_2)$, satisfying

1) $(\sigma^0_1, \sigma^0_2) \in S$;
2) for any $(\sigma_1, \sigma_2) \in S$, $\mathsf{out\_excited}(\sigma_1) = \mathsf{out\_excited}(\sigma_2)$ holds;
3) for any $(\sigma_1, \sigma_2) \in S$ and any $(\sigma_1, t_1, \sigma'_1) \in E'_1$, there exists some $t_2$ and $\sigma'_2$ such that $l(t_2) = l(t_1)$, $(\sigma_2, t_2, \sigma'_2) \in E'_2$, and $(\sigma'_1, \sigma'_2) \in S$ hold.

Let $G_1 \rightsquigarrow G_2$ denote that $G_1$ and $G_2$ have the same input- and output-signal sets and that there exists a simulation from $G_1$ to $G_2$.

*Lemma 2:* For STGs $G_1$ and $G_2$, if $G_1 \rightsquigarrow G_2$ and $G_2$ has CSC, then for $x \in \mathrm{Out}_1$, the following hold:

$$\begin{aligned} &ES_1(x+) \subseteq ES_2(x+), && ES_2(x+) - ES_1(x+) \subseteq UR_1 \\ &QS_1(x+) \subseteq QS_2(x+), && QS_2(x+) - QS_1(x+) \subseteq UR_1 \\ &ES_1(x-) \subseteq ES_2(x-), && ES_2(x-) - ES_1(x-) \subseteq UR_1 \\ &QS_1(x-) \subseteq QS_2(x-), && QS_2(x-) - QS_1(x-) \subseteq UR_1 \\ &\quad UR_1 \supseteq UR_2. \end{aligned}$$

*Proof:* For $s_1 \in ES_1(x+)$, let $\sigma_1$ denote a timed state of $\mathcal{G}^{\mathrm{df}}_{G_1}$ from which $s_1$ is mapped, and suppose that $\sigma_1$ is reached from $\sigma^0_1$ by a sequence $v_1$ of transitions on $\mathcal{G}^{\mathrm{df}}_{G_1}$. From $G_1 \rightsquigarrow G_2$, a timed state, denoted by $\sigma_2$, is reachable on $\mathcal{G}^{\mathrm{df}}_{G_2}$ by a sequence $v_2$ of transitions such that $l(v_1) = l(v_2)$, where $l(t_1 t_2 \cdots) = l(t_1)l(t_2)\cdots$ (note that $\lambda$ is deleted in this sequence). Thus, $\sigma_2$ is also mapped to $s_1$. Since $\mathsf{out\_excited}(\sigma_1) = \mathsf{out\_excited}(\sigma_2)$ and $s_1 \in ES_1(x+)$ hold, $s_1 \in ES_2(x+)$ also holds. Hence, $ES_1(x+) \subseteq ES_2(x+)$ holds. The other three cases can be proved similarly. Next, suppose that $s_2 \in ES_2(x+) - ES_1(x+)$ holds. Since the value for $x$ is zero in $s_2$, $s_2$ is included in either $QS_1(x-)$ or $UR_1$. If $s_2 \in QS_1(x-)$ holds, then $s_2 \in QS_2(x-)$ holds from $QS_1(x-) \subseteq QS_2(x-)$. This, however, violates that $G_2$ has CSC from $s_2 \in ES_2(x+)$. Hence, $s_2$ must be in $UR_1$. The remaining three cases can be proved similarly. Finally, since additional reachable signal states in $G_2$ are in $UR_1$, $UR_1 \supseteq UR_2$ holds. ■

The key of this proof is that the corresponding timed states in $\mathcal{G}^{\mathrm{df}}_{G_1}$ and $\mathcal{G}^{\mathrm{df}}_{G_2}$ have the same set of excited signals. For untimed methods, the trace-equivalence relation is used to guarantee this (e.g., in [22]). For timed methods, however, such a relation on traces are not helpful, because excited transitions does not necessarily fire as shown in Fig. 1, i.e., the traces defined by transition firings do not give sufficient excitation information. Hence, the simulation relation defined above is necessary.

For a nondecorated signal state $s$ and a set $D$ of signals, the *D-closure* of $s$, denoted by $\mathcal{C}_D(s)$, is a set of all nondecorated signal states, including $s$, such that their binary vectors are the same if the signals in $D$ are projected out. The *core* of a $D$-closure is the common binary vector obtained by projecting out the signals in $D$. For example, for $s = (abcd) = (1101)$ and $D = \{a, b\}$, $\mathcal{C}_D(s) = \{0001, 0101, 1001, 1101\}$ and its core is $(cd) = (01)$. The mappings from $D$-closure $\mathcal{C}_D(s)$ to its core $s'$ and its inverse are defined by $\mathsf{proj}_D(\mathcal{C}_D(s))$ and $\mathsf{proj}^{-1}_D(s')$. Note that both are the one-to-one mappings. The $D$-closure and these mappings are extended to sets as follows: $\mathcal{C}_D(S) = \bigcup_{s \in S} \mathcal{C}_D(s)$, $\mathsf{proj}_D(\mathcal{C}_D(S)) = \{\mathsf{proj}_D(\mathcal{C}_D(s)) | s \in S\}$, and $\mathsf{proj}^{-1}_D(S') = \bigcup_{s' \in S'} \mathsf{proj}^{-1}_D(s')$.

For an STG $G$ and $x \in Out$, a set $D$ of signals is an *irrelevant input set* for $x$, if:

1) $D \subseteq In \cup Out - \{x\}$;
2) $\mathcal{C}_D(ES(x+)) - UR = ES(x+)$;
3) $\mathcal{C}_D(ES(x-)) - UR = ES(x-)$.

From this definition, the following lemma holds.

*Lemma 3:* For $x \in Out$ and any irrelevant input set $D$ for $x$, the following hold:

$$\begin{aligned} \mathcal{C}_D\left(QS(x+)\right) - UR &= QS(x+) \\ \mathcal{C}_D\left(QS(x-)\right) - UR &= QS(x-). \end{aligned}$$

*Proof:* Suppose $\mathcal{C}_D(QS(x+)) - UR \neq QS(x+)$. From $QS(x+) \subseteq \mathcal{C}_D(QS(x+))$, this means that for some $s \in \mathcal{C}_D(QS(x+)) - UR$, $s \notin QS(x+)$ holds. From $x \notin D$, such that $s$ must be in $ES(x-)$. Since $s \in \mathcal{C}_D(QS(x+))$ holds, there exists $s_2 \in QS(x+)$ such that $s \in \mathcal{C}_D(s_2)$. From $s \in \mathcal{C}_D(s)$, $\mathcal{C}_D(s)$ and $\mathcal{C}_D(s_2)$ have a common element, which implies $\mathcal{C}_D(s) = \mathcal{C}_D(s_2)$ and, so, $s_2 \in \mathcal{C}_D(s)$. From $s \in ES(x-)$ and $\mathcal{C}_D(ES(x-)) - UR = ES(x-)$, $s_2 \in ES(x-)$ is derived, which however, contradicts that $G$ has CSC, and so, $ES(x-)$ and $QS(x+)$ are disjoint. The remaining case can be proved similarly. ■

For an STG $G$, $x \in Out$ and a set $D$ of signals with $x \notin D$, let $G_{D,x}$ denote an STG obtained from $G$ by making transitions related to signals in $D$ dummy, which has the input-signal set $\mathsf{sig}(G) - D - \{x\}$ and the output-signal set $\{x\}$. Let $ES_1$, $QS_1$, and so on be for $G_{D,x}$.

*Lemma 4:* For $x \in Out$ and any irrelevant input set $D$ for $x$, the following hold:

$$\begin{aligned} \mathsf{proj}^{-1}_D\left(ES_1(x+)\right) &= \mathcal{C}_D\left(ES(x+)\right) \\ \mathsf{proj}^{-1}_D\left(ES_1(x-)\right) &= \mathcal{C}_D\left(ES(x-)\right) \\ \mathsf{proj}^{-1}_D\left(QS_1(x+)\right) &= \mathcal{C}_D\left(QS(x+)\right) \\ \mathsf{proj}^{-1}_D\left(QS_1(x-)\right) &= \mathcal{C}_D\left(QS(x-)\right) \\ \mathsf{proj}^{-1}_D(UR_1) &\subseteq UR. \end{aligned}$$

*Proof:* Since $G_{D,x}$ has the same flow relation as $G$ except that some transitions are dummy in $G_{D,x}$, there exists a one-to-one mapping between timed states of $\mathcal{G}_G$ and $\mathcal{G}_{G_{D,x}}$. In the dummy-free timed state graph $\mathcal{G}^{\mathrm{df}}_{G_{D,x}}$, however, some timed states that are in $\mathcal{G}^{\mathrm{df}}_G$ are deleted from the construction of the dummy-free timed state graphs, as shown in Fig. 18. Note that $\mathcal{G}^{\mathrm{df}}_G$ and $\mathcal{G}_{G_{D,x}}$ also have a one-to-one mapping if $G$ have no dummy transitions, and Fig. 18 shows this case.

The proof of this lemma first shows $ES_1(x+) \subseteq \mathsf{proj}_D(\mathcal{C}_D(ES(x+)))$. Suppose that $s_1 \in ES_1(x+)$, and let $\sigma_1$ be the timed state in $\mathcal{G}^{\mathrm{df}}_{G_{D,x}}$ that is mapped to $s_1$. This $\sigma_1$ exists also in $\mathcal{G}^{\mathrm{df}}_G$ from the above relation between $\mathcal{G}^{\mathrm{df}}_G$ and $\mathcal{G}^{\mathrm{df}}_{G_{D,x}}$, and let $s$ be its signal state. Then, $s_1 = \mathsf{proj}_D(\mathcal{C}_D(s))$ holds. Furthermore, $s \in ES(x+)$ holds from the following

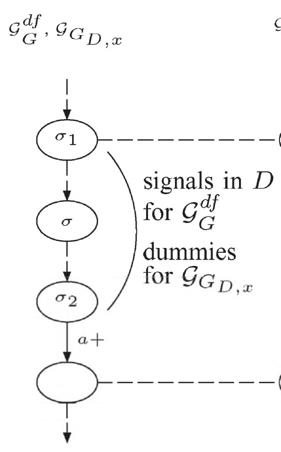

$\mathcal{G}_G^{df}, \mathcal{G}_{G_{D,x}}$
$\mathcal{G}_{G_{D,x}}^{df}$
$\sigma_1$
$\sigma$
$\sigma_2$
signals in $D$ for $\mathcal{G}_G^{df}$
dummies for $\mathcal{G}_{G_{D,x}}$
$a+$

Fig. 18. Relation between $\mathcal{G}_G^{\rm df}$ and $\mathcal{G}_{G_{D,x}}^{\rm df}$.

reason. From the signal excitation of $\mathcal{G}_{G_{D,x}}^{\rm df}$, $x+$ must be excited to rise in some timed state reached only by transitions related to the signals in $D$, such as $\sigma_2$ shown in Fig. 18. Let $s_2$ be its signal state. Then, $s_2 \in ES(x+)$ and $s \in \mathcal{C}_D(s_2) - UR$ hold. Since $D$ is an irrelevant input set, $\mathcal{C}_D(ES(x+)) - UR = ES(x+)$ holds, and so, $s \in ES(x+)$ holds. Hence, $s_1 \in \mathsf{proj}_D(\mathcal{C}_D(ES(x+)))$ is derived.

Next, $ES_1(x+) \supseteq \mathsf{proj}_D(\mathcal{C}_D(ES(x+)))$ is proved by showing that for $s \in ES(x+)$, $s_1 = \mathsf{proj}_D(\mathcal{C}_D(s))$ satisfies $s_1 \in ES_1(x+)$. Let $\sigma$ be the timed state in $\mathcal{G}_G^{\rm df}$ that is mapped to $s$. This $\sigma$ may or may not exist in $\mathcal{G}_{G_{D,x}}^{\rm df}$. In the former case, it has the signal state $s_1 = \mathsf{proj}_D(\mathcal{C}_D(s))$ in $\mathcal{G}_{G_{D,x}}^{\rm df}$, and $x$ is excited to rise in $\sigma$. Hence, $s_1 \in ES_1(x+)$ is derived. In the latter case, from the construction of dummy-free state graphs, there exists a timed state $\sigma_1$ included in both $\mathcal{G}_G^{\rm df}$ and $\mathcal{G}_{G_{D,x}}^{\rm df}$, from which $\sigma$ is reached only by transitions related to the signals in $D$ (see Fig. 18). From this relation between $\sigma$ and $\sigma_1$, $\sigma_1$ has the signal state $s' \in \mathcal{C}_D(s) - UR$ and $s_1 = \mathsf{proj}_D(\mathcal{C}_D(s))$ in $\mathcal{G}_{G_{D,x}}^{\rm df}$. Since $D$ is an irrelevant input set, $\mathcal{C}_D(ES(x+)) - UR = ES(x+)$ holds, and so, $s' \in ES(x+)$ holds. Thus, $x$ is excited to rise in $\sigma'$ in $\mathcal{G}_G^{\rm df}$, and therefore, $s_1 \in ES_1(x+)$ is derived.

Hence, $ES_1(x+) = \mathsf{proj}_D(\mathcal{C}_D(ES(x+)))$ is shown. Applying $\mathsf{proj}_D^{-1}$ derives the first property. The proofs for the other three properties are similar.

Furthermore, from the above discussion, if $\sigma$ mapped to $s$ is reached in $\mathcal{G}_G^{\rm df}$, then some $\sigma'$ mapped to $s_1 = \mathsf{proj}_D(\mathcal{C}_D(s))$ is also reached in $\mathcal{G}_{G_{D,x}}^{\rm df}$. Thus, if $s_1 \in UR_1$ holds, then every $s \in \mathsf{proj}_D^{-1}(s_1)$ is also in $UR$. Hence, the final property holds. ■

*Lemma 5:* For $x \in Out$ and a set $D$ of signals with $D \subseteq In \cup Out - \{x\}$, if $G_{D,x}$ has CSC, and $D \cap \mathsf{trigger}(x) = \emptyset$ holds, then $D$ is an irrelevant input set.

*Proof:* Suppose that $D$ is not an irrelevant input set. Then, there exist signal states $s \in ES(x+)$ and $s' \in \mathcal{C}_D(s) - UR$ such that $s' \notin ES(x+)$, or there exist signal states $s \in ES(x-)$ and $s' \in \mathcal{C}_D(s) - UR$ such that $s' \notin ES(x-)$. In this proof, the former case is considered, but the latter case can be proved similarly. There exist timed states $\sigma$ and $\sigma'$ in $\mathcal{G}_G^{\rm df}$ whose signal states are $s$ and $s'$, respectively. From the construction of the dummy-free timed state graphs, there exist timed state $\sigma_1$ and $\sigma_1'$ in $\mathcal{G}_{G_{D,x}}^{\rm df}$ whose signal states are $s_1$ and $s_1'$, satisfying $s_1 = \mathsf{proj}_D(\mathcal{C}_D(s))$ and $s_1' = \mathsf{proj}_D(\mathcal{C}_D(s'))$ (see Fig. 18). Note that $s_1 = s_1'$ holds from $\mathsf{proj}_D(\mathcal{C}_D(s)) = \mathsf{proj}_D(\mathcal{C}_D(s'))$. These $\sigma_1$ and $\sigma_1'$ cannot be the same for the following reason. $\sigma$ and $\sigma_1$ can be the same, only when one of them is reached from the other on $\mathcal{G}_G^{\rm df}$ only by transitions related to the signals in $D$. In this case, however, from $s \in ES(x+)$ and $s' \notin ES(x+)$, either $x+$ is disabled without firing $x+$, which violates the output semimodularity of $G$, or $x+$ is enabled without firing any transition in $\mathsf{trigger}(x)$ from $D \cap \mathsf{trigger}(x) = \emptyset$, which is impossible. Thus, $\sigma_1$ and $\sigma_1'$ must be different. Since $x$ is excited to rise in $\sigma$, so is it in $\sigma_1$ of $\mathcal{G}_{G_{D,x}}^{\rm df}$. On the other hand, although $x$ is not excited in $\sigma'$, $x$ may be considered to be excited to rise in $\sigma_1'$ of $\mathcal{G}_{G_{D,x}}^{\rm df}$ from the excitation definition, if $x+$ is enabled in some timed state of $\mathcal{G}_{G_{D,x}}$ (such as $\sigma_2$ in Fig. 18) that is reached from $\sigma'$ by only dummy transitions. But, this cannot happen because $x+$ is disabled in $\sigma'$ and $D \cap \mathsf{trigger}(x) = \emptyset$ holds. Therefore, $x$ is not excited in $\sigma_1'$ of $\mathcal{G}_{G_{D,x}}^{\rm df}$ either. This contradicts that $G_{D,x}$ has CSC, because $x$ is excited in $\sigma'$, not excited in $\sigma_1'$, and $s_1 = s_1'$ holds. Hence, $D$ should be an irrelevant input set. ■

*Lemma 6:* For $x \in Out$ and any irrelevant input set $D$ for $x$, suppose that $G'$ satisfies $G_{D,x} \rightsquigarrow G'$. If $G'$ has CSC, then $G'$ is cover-correct with respect to $G$.

*Proof:* This proof focuses on the gC implementation, and furthermore, only the cover for $x+$ is considered, because the proofs for the other cases can be done similarly. Let $ES_1$, $QS_1$, and $UR_1$ be for $G_{D,x}$, and $ES_2$, $QS_2$, and $UR_2$ be for $G'$.

Let $C_2(x+)$ denote the cover for $G'$ and $x+$. From the definition of covers

$$ES_2(x+) \subseteq C_2(x+) - UR_2 \subseteq ES_2(x+) \cup QS_2(x+) \quad (1)$$

holds. This proof shows that $C_2(x+)$ is also a cover of $G$. Since $C_2(x+)$ should be considered in the signal state space of $G$, this is shown by

$$ES(x+) \subseteq \mathsf{proj}_D^{-1}(C_2(x+)) - UR \subseteq ES(x+) \cup QS(x+). \quad (2)$$

To show the aforementioned equation, this proof first shows

$$ES_1(x+) \subseteq C_2(x+) - UR_1 \subseteq ES_1(x+) \cup QS_1(x+) \quad (3)$$

and then (2) is shown.

The above (1) is rewritten by removing $UR_1$ as follows:

$$\begin{aligned} ES_2(x+) - UR_1 &\subseteq C_2(x+) - UR_2 - UR_1 \\ &\subseteq ES_2(x+) \cup QS_2(x+) - UR_1. \quad (4) \end{aligned}$$

From $UR_1 \supseteq UR_2$ as shown in Lemma 2

$$C_2(x+) - UR_2 - UR_1 = C_2(x+) - UR_1$$

holds. Furthermore, $ES_2(x+) - UR_1$ is rewritten to $ES_1(x+)$ as follows using Lemma 2:

$$\begin{aligned} &ES_2(x+) - UR_1 \\ &\quad = ES_1(x+) \cup (ES_2(x+) - ES_1(x+)) - UR_1 \\ &\quad = ES_1(x+) - UR_1 \\ &\quad = ES_1(x+). \end{aligned}$$

Similarly, $QS_2(x+) - UR_1 = QS_1(x+)$ holds. Hence, (4) is rewritten to (3).

Next, applying $\mathsf{proj}_D^{-1}$ to (3) derives

$$\begin{aligned} \mathsf{proj}_D^{-1}(ES_1(x+)) &\subseteq \mathsf{proj}_D^{-1}(C_2(x+) - UR_1) \\ &\subseteq \mathsf{proj}_D^{-1}(ES_1(x+) \cup QS_1(x+)). \end{aligned}$$

From Lemma 4, this is rewritten as

$$\begin{aligned} \mathcal{C}_D(ES(x+)) &\subseteq \mathsf{proj}_D^{-1}(C_2(x+) - UR_1) \\ &\subseteq \mathcal{C}_D(ES(x+)) \cup \mathcal{C}_D(QS(x+)) \end{aligned}$$

and by removing $UR$

$$\begin{aligned} \mathcal{C}_D(ES(x+)) - UR &\subseteq \mathsf{proj}_D^{-1}(C_2(x+) - UR_1) - UR \\ &\subseteq (\mathcal{C}_D(ES(x+)) - UR) \cup (\mathcal{C}_D(QS(x+)) - UR) \end{aligned}$$

is obtained. Since $D$ is an irrelevant input set, $\mathcal{C}_D(ES(x+)) - UR = ES(x+)$ holds, and from Lemma 3, $\mathcal{C}_D(QS(x+)) - UR = QS(x+)$ holds. Thus

$$\begin{aligned} ES(x+) &\subseteq \mathsf{proj}_D^{-1}(C_2(x+) - UR_1) - UR \\ &\subseteq ES(x+) \cup QS(x+) \qquad (5) \end{aligned}$$

holds. The above $\mathsf{proj}_D^{-1}(C_2(x+) - UR_1) - UR$ can be rewritten as follows from $\mathsf{proj}_D^{-1}(UR_1) \subseteq UR$ by Lemma 4:

$$\begin{aligned} &\mathsf{proj}_D^{-1}(C_2(x+) - UR_1) - UR \\ &\quad = \mathsf{proj}_D^{-1}(C_2(x+)) - \mathsf{proj}_D^{1}(UR_1) - UR \\ &\quad = \mathsf{proj}_D^{-1}(C_2(x+)) - UR. \qquad (6) \end{aligned}$$

Hence, from (5) and (6), (2) is obtained. ■

For an STG $G$, $x \in Out$, and $V \subseteq \mathsf{sig}(G)$ such that $x \in V$, let $\mathsf{abs}(G, V, x)$ be any STG such that $G_{D,x} \rightsquigarrow \mathsf{abs}(G, V, x)$ with $D = \mathsf{sig}(G) - V$. The main theorem is as follows.

*Theorem 3:* If $\mathsf{abs}(G, V, x)$ has CSC for $V$ with $\mathsf{trigger}(x) \subseteq V$, then $\mathsf{abs}(G, V, x)$ is cover-correct with respect to $G$.

*Proof:* From $\mathsf{trigger}(x) \subseteq V$, $D = \mathsf{sig}(G) - V$ satisfies $D \cap \mathsf{trigger}(x) = \emptyset$. $\mathsf{abs}(G, V, x)$ has CSC. Thus, from Lemma 5, $D$ is an irrelevant input set for $x$. From the above definition, $G_{D,x} \rightsquigarrow \mathsf{abs}(G, V, x)$ holds, and $\mathsf{abs}(G, V, x)$ has CSC. Hence, from Lemma 6, $\mathsf{abs}(G, V, x)$ is cover-correct with respect to $G$. ■

## ACKNOWLEDGMENT

The authors would like to thank W. Vogler and M. Schaefer for their helpful comments. Furthermore, they would like to thank J. Carmona and J. Cortadella for giving them the benchmark suites and the copy of their latest paper and V. Khomenko and A. Yakovlev for making their experimental software available.

## REFERENCES

[1] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, "Petrify: A tool for manipulating concurrent specifications and synthesis of asynchronous controllers," *IEICE Trans. Inf. Syst.*, vol. E80-D, no. 3, pp. 315–325, Mar. 1997.

[2] P. A. Beerel, C. J. Myers, and T. H.-Y. Meng, "Covering conditions and algorithms for the synthesis of speed-independent circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 13, no. 3, pp. 205–219, Mar. 1998.

[3] R. M. Fuhrer, S. M. Nowick, M. Theobald, N. K. Jha, B. Lin, and L. Plana, "Minimalist: An environment for the synthesis, verification and testability of burst-mode asynchronous machines," Columbia Univ., New York, Tech. Rep. TR CUCS-020-99, Jul. 1999.

[4] C. J. Myers, *Asynchronous Circuit Design*. Hoboken, NJ: Wiley, 2001.

[5] S. M. Burns and A. J. Martin, "Syntax-directed translation of concurrent programs into self-timed circuits," in *Advanced Research in VLSI*, J. Allen and F. Leighton, Eds. Cambridge, MA: MIT Press, 1988, pp. 35–50.

[6] K. van Berkel, J. Kessels, M. Roncken, R. Saeijs, and F. Schalij, "The VLSI-programming language Tangram and its translation into handshake circuits," in *Proc. EDAC*, 1991, pp. 384–389.

[7] E. Kim, J.-G. Lee, and D.-I. Lee, "Automatic process-oriented control circuit generation for asynchronous high-level synthesis," in *Proc. Int. Symp. Adv. Res. Asynchronous Circuits and Syst.*, Apr. 2000, pp. 104–113.

[8] J. Kessels and A. Peeters, "The Tangram framework: Asynchronous circuits for low power," in *Proc. Asia and South Pacific Des. Autom. Conf.*, Feb. 2001, pp. 255–260.

[9] D. Edwards and A. Bardsley, "Balsa: An asynchronous hardware synthesis language," *Comput. J.*, vol. 45, no. 1, pp. 12–18, 2002.

[10] A. Bystrov and A. Yakovlev, "Asynchronous circuit synthesis by direct mapping: Interfacing to environment," in *Proc. Int. Symp. Adv. Res. Asynchronous Circuits Syst.*, Apr. 2002, pp. 127–136.

[11] T. Chelcea and S. M. Nowick, "Resynthesis and peephole transformations for the optimization of large-scale asynchronous systems," in *Proc. ACM/IEEE Des. Autom. Conf.*, Jun. 2002, pp. 405–410.

[12] C. J. Myers and T. H.-Y. Meng, "Synthesis of timed asynchronous circuits," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 1, no. 2, pp. 106–119, Jun. 1993.

[13] T. Yoneda and C. J. Myers, "Synthesizing timed circuits from high level specification languages," Nat. Inst. Informatics, Tokyo, Japan, NII Tech. Rep., NII-2003-003E, 2003.

[14] A. Matsumoto, "High level synthesis of asynchronous circuits (in Japanese)," M.S. thesis, Tokyo Inst. Technol., Tokyo, Japan, 2004.

[15] T. Yoneda, A. Matsumoto, M. Kato, and C. J. Myers, "High level synthesis of timed asynchronous circuits," in *Proc. Int. Symp. Adv. Res. Asynchronous Circuits Syst.*, 2005, pp. 178–189.

[16] N. Sretasereekul, H. Saito, M. Imai, E. Kim, M. Ozcan, K. Thongnoo, H. Nakamura, and T. Nanya, "A zero-time-overhead asynchronous four-phase controller," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2003, vol. 5, pp. 205–208.

[17] T.-A. Chu, "Synthesis of self-timed VLSI circuits from graph-theoretic specifications," Ph.D. dissertation, MIT Lab. Comput. Sci., Cambridge, MA, Jun. 1987.

[18] W. Vogler and R. Wollowski, "Decomposition in asynchronous circuit design," in *Concurrency and Hardware Design*, vol. 2549, J. Cortadella, A. Yakovlev, and G. Rozenberg, Eds. New York: Springer-Verlag, 2002, pp. 152–190.

[19] H. Zheng, E. Mercer, and C. J. Myers, "Modular verification of timed circuits using automatic abstraction," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 22, no. 9, pp. 1138–1153, Sep. 2003.

[20] R. Puri and J. Gu, "A modular partitioning approach for asynchronous circuit synthesis," in *Proc. ACM/IEEE Des. Autom. Conf.*, Jun. 1994, pp. 63–69.

[21] J. Beister, G. Eckstein, and R. Wollowski, "From STG to extended-burst-mode machines," in *Proc. Int. Symp. Adv. Res. Asynchronous Circuits Syst.*, Apr. 1999, pp. 145–158.

[22] J. Carmona and J. Cortadella, "ILP models for the synthesis of asynchronous control circuits," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2003, pp. 818–825.

[23] V. Khomenko, M. Koutny, and A. Yakovlev, "Logic synthesis for asynchronous circuits based on petri net unfoldings and incremental SAT," in *Proc. ACSD*, 2004, pp. 16–25.

[24] T. Yoneda, H. Onda, and C. J. Myers, "Synthesis of speed independent circuits based on decomposition," in *Proc. Int. Symp. Adv. Res. Asynchronous Circuits Syst.*, Apr. 2004, pp. 135–145.

[25] D. L. Dill, *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits*. Cambridge, MA: MIT Press, 1988.

[26] B. Berthomieu and M. Diaz, "Modeling and verification of time dependent systems using time Petri nets," *IEEE Trans. Softw. Eng.*, vol. 17, no. 3, pp. 259–273, Mar. 1991.

[27] T. Yoneda and H. Schlingloff, "Efficient verification of parallel real-time systems," *Form. Method Syst. Des.*, vol. 11, no. 2, pp. 187–215, Aug. 1997.

[28] T. Yoneda, E. G. Mercer, and C. J. Myers, "Modular synthesis of timed circuits using partial order reduction," in *Proc. 10th Workshop Synthesis Syst. Integr Mixed Technol.*, 2001, pp. 127–134.

[29] E. G. Mercer, C. J. Myers, and T. Yoneda, "Improved POSET timing analysis in timed Petri nets," in *Proc. 10th Workshop Synthesis Syst. Integr Mixed Technol.*, 2001, pp. 151–158.

[30] S. T. Jung and C. J. Myers, "Direct synthesis of timed circuits from free-choice STGs," *IEEE Trans. Comput.-Aided Design Integr Circuits Syst.*, vol. 21, no. 3, pp. 275–290, Mar. 2002.

[31] K. McMillan, "Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits," in *Proc. Int. Workshop Comput. Aided Verification*, G. V. Bochman and D. K. Probst, Eds. New York: Springer-Verlag, 1992, vol. 663, pp. 164–177.

[32] J. Carmona, J. Colom, J. Cortadella, and F. García-Vallés, "Synthesis of asynchronous controllers using integer linear programming," *IEEE Trans. Comput.-Aided Design Integr Circuits Syst.*, vol. 25, no. 9, pp. 1637–1651, Sep. 2006.

[33] C. J. Myers, "Computer-aided synthesis and verification of gate-level timed circuits," Ph.D. dissertation, Dept. Elect. Eng., Stanford Univ., Stanford, CA, Oct. 1995.

[34] S. A. Seshia and R. E. Bryant, "Unbounded, fully symbolic model checking of timed automata using boolean methods," in *Proc. Int. Conf Comput.-Aided Verification*, 2003, vol. LNCS 2725, pp. 154–166.

**Tomohiro Yoneda** (M'85) received the B.E., M.E., and Dr.Eng. degrees in computer science from the Tokyo Institute of Technology, Tokyo, Japan, in 1980, 1982, and 1985, respectively.

In 1985, he joined the staff of Tokyo Institute of Technology. From 1990 to 1991, he was a Visiting Researcher at Carnegie Mellon University, Pittsburgh, PA. In 2002, he joined with the National Institute of Informatics, Tokyo, where he is currently a Professor. His research activities currently focus on formal verification of hardware and synthesis of asynchronous circuits.

Dr. Yoneda is a member of the Institute of Electronics, Information, and Communication Engineers of Japan, and Information Processing Society of Japan.

**Chris J. Myers** (S'91–M'96–SM'04) received the B.S. degree in electrical engineering and Chinese history from the California Institute of Technology, Pasadena, in 1991, and the M.S.E.E. and Ph.D. degrees from Stanford University, Stanford, CA, in 1993 and 1995, respectively.

He is currently a Professor in the Department of Electrical and Computer Engineering, University of Utah, Salt Lake City. He is the author of over 60 technical papers and the textbook entitled *Asynchronous Circuit Design*. He is also a coinventor on four patents. His research interests include algorithms for the analysis of real-time concurrent systems, analog error control decoders, formal verification, asynchronous circuit design, and the modeling and analysis of genetic regulatory circuits.

Dr. Myers received an NSF Fellowship in 1991, an NSF CAREER award in 1996, and best paper awards at Async1999 and Async2007.