Synthesis of Timed Circuits based on Decomposition

Tomohiro Yoneda and Chris Myers

NII-2006-001E
Feb. 2006
Synthesis of Timed Circuits based on Decomposition

Tomohiro Yoneda, National Institute of Informatics, Japan
Chris Myers, University of Utah, USA

Abstract—This report 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 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 CSC. 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 nutas (NII-Utah Timed Asynchronous circuit Synthesis system), and its very first version is available at http://research.nii.ac.jp/~yoneda.

Index Terms—Decomposition, abstraction, synthesis, timed circuits, timed STGs.

I. INTRODUCTION

Logic synthesis [1]–[3] 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 [4]–[9]. However, it 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 [10]. This report 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 [11]. 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 [12], and improved version is described in [13], [14]. 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 [14], [15], the performance degradation caused by the inserted state variables can be reduced to almost negligible amount. A key issue to success of our approach is a new logic synthesis technique that is efficient enough to handle large STGs. This report aims at reducing the average cost for logic synthesis from (timed) STGs by decomposing (or projecting) a specification to many small sub-specifications and running the logic synthesis procedure for each of them.

The idea for decomposition based synthesis is first proposed by Chu [16]. In his work, one primary output is picked up, 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 [17], and Zheng and Myers developed a timed contraction algorithm in [18]. On the other hand, Puri and Gu tried to solve the second problem in [19]. Their algorithm greedily removes an irrelevant signal (with respect to the output signal) such that the number of CSC violations 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, Eckstein, and Wollowski proposed a similar decomposition based method for extended-burst-mode machines [20]. Recently, two separate works, one by Carmona and Cortadella [21] and the other by Khomenjko, Koutny and Yakovlev [22], have been proposed for synthesizing speed-independent circuits efficiently based on an idea similar to that in [19]. Both works first find the necessary input signals (called support) for each output by analyzing the original STG, and then synthesize each sub-circuit with each output individually. Unlike the approach of [19], 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 work of T. Yoneda was supported by JSPS Joint Research Projects. The work of C. Myers was supported by NSF Japan Program award INT-0087281, SRC grant 2002-TJ-1024, and SRC grant 2005-TJ-1357.
the Incremental Boolean Satisfiability (SAT) technique in the latter, and hence, their methods are much more efficient than that of [19].

The main contribution of our work 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 [19], 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 ILPSAT based approaches. This report describes the theory and the algorithms extended from [23] for the timed circuit synthesis.

The rest of this report is organized as follows. Section II gives several definitions needed for this report, and Section III shows the basic theory of our decomposition based synthesis. Section IV describes the overview of the proposed method, and Section V mentions how to explore timed state spaces and to check synthesizable. Section VI explains in detail how the input sets are determined, which is the main issue of this report. Section VII describes the limitations of our method. Several experimental results are shown in Section VIII, and Section IX gives our conclusion.

II. Synthesizable STGs

A timed STG \( G = (P, T, F, Eft, Lft, \mu^0, l, In, Out) \) 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, \( Eft : T \rightarrow Q^+ \), \( Lft : T \rightarrow Q^+ \times \{+,-,\} \) are functions for the earliest and latest firing times of transitions satisfying \( Eft(t) \leq Lft(t) \) for all \( t \in T \) (\( Q^+ \) denotes the set of nonnegative rationals), \( \mu^0 \subseteq P \) is the initial marking, \( l : T \rightarrow (In \cup Out) \times \{+,-,\} \cup \{\lambda\} \) is the labeling function, and \( In \) and \( Out \) are the input and output signal sets. Let \( \text{sig}(G) \) 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 \( w \in \text{sig}(G) \), \( w \)-transition denotes a transition \( t \) with \( l(t) = w + \) or \( w-\). For any transition \( t \), \( \bullet t = \{p \in P \mid \{p, t\} \in F\} \) and \( \bullet t = \{p \in P \mid \{t, p\} \in F\} \) 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 \( \text{conflict}(t) = \{t' \mid \bullet t \cap \bullet t' \neq \emptyset\} - \{t\} \). In the rest of this report, 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 \( \text{enabled}(\mu) \) be the set of transitions enabled in \( \mu \). A timed state \( \sigma \) of \( G \) is a pair \((\mu, \text{clock})\), where \( \mu \) is a marking and \( \text{clock} \) is a function \( T \rightarrow R^+ \) (\( R^+ \) denotes the set of nonnegative reals). The initial timed state \( \sigma^0 \) is \((\mu^0, \text{clock}^0)\), where \( \text{clock}^0(t) = 0 \) for all \( t \in T \). A timed state changes if time passes or if a transition fires. In timed state \( \sigma = (\mu, \text{clock}) \), time \( \tau \in Q^+ \) can pass, if for all \( t \in \text{enabled}(\mu) \), \( \text{clock}(t) + \tau \leq Lft(t) \). In this case, timed state \( \sigma' = (\mu', \text{clock}') \) is obtained from \( \sigma \) by passing \( \tau \), where

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

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

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

\[
\text{clock}'(t) = \begin{cases} 
0 & \text{if } t \in \text{enabled}(\mu') - \text{enabled}(\mu - \bullet t_f) \\
\text{clock}(t) & \text{else}.
\end{cases}
\]

That is, firing a transition \( t_f \) consumes no time, but updates \( \mu \) and \( \text{clock} \) such that the clocks associated with newly enabled transitions (i.e., transitions that are enabled in \( \mu' \) but not in \( \mu - \bullet t_f \)) are reset to 0, 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 (i.e., \( \sigma \) is equal to \( \sigma' \) for an empty \( v \)). \( v \) is called a trace, if there exists a \( \sigma' \) such that \( \sigma \xrightarrow{\cdot} \sigma' \). Let \( \text{trace}(G) \) denote the set of all 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 report, it is assumed that those occurrences of the same transition are distinguished by some appropriate way, such as, by attaching firing counts, but those are omitted for simplicity in this report. If every reachable timed state \( \sigma = (\mu, \text{clock}) \) such that there exist \( t \) and \( \sigma' \) with \( \sigma \xrightarrow{t} \sigma' \) satisfies \( (\mu - \bullet t) \cap \bullet t = \emptyset \), 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 \( v \in \text{trace}(G) \) and every \( w \in \text{sig}(G) \) such that \( v \) includes two or more \( w \)-transitions, the last two of them are different (i.e., \( w+ \) and \( w- \), or \( w- \) and \( w+ = \frac{1}{w} \)).

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 0 or 1 in signal states. \( R \) represents that the corresponding output signal has the binary value of 0, but it is excited to rise. \( F \) indicates the signal has a value of 1, 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 undecorated signal states. For example, suppose

\[1\]Remember that \( \text{trace}(G) \) is prefix closed.
that two timed states $\sigma$ and $\sigma'$ have decorated signal states (1010) and (101R) \(^2\). 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 concurreny, or otherwise changing the behavior of the STG in some way.

This detection of CSC violations 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 $(V, E)$ with an initial timed state $\sigma^0$, denoted by $G_0 = ((V, E), \sigma^0)$, 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, $\{ (\sigma, \sigma') \mid \exists \tau, \sigma^0 \xrightarrow{\tau} \sigma, \sigma \rightarrow \sigma' \}$. A dummy-free timed state graph $G_{DF}$ is a graph $(V', E')$ with an initial timed state $\sigma'^0$, denoted by $G_{DF} = ((V', E'), \sigma'^0)$, satisfying,

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

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. Figure 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 Figures 1 (b) and (c), respectively. Note that $a+$, for example, can fire at any time between 1 and 2 after it becomes enabled, and so, there exist 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 $G_{DF} = ((V', E'), \sigma'^0)$. An output signal $w$ is excited in a timed state $\sigma$ if $\exists \sigma', (\sigma', t, \sigma) \in E'$ 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 Figure 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 Figure 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 information instead of the existence of outgoing edges in the state graph. This is complicated by the fact that either $x+$ or $y+$ is not yet enabled in $\sigma_1$.

The definition of signal excitation proposed in this report is as follows. An output signal $w$ is excited in a timed state $\sigma$ of $G_{DF} = ((V', E'), \sigma'^0)$, if there exists a (possibly empty) sequence $u_1 u_2 \cdots u_n$ of dummy transitions such that $(\sigma, u_1 u_2 \cdots u_n, \sigma_2) \in E^*$, $\sigma_2 = (\mu_2, \text{clock}_2)$, and $t \in \text{enabled}(\mu_2)$ with $l(t) = w+$ or $l(t) = w-$, where $G_0 = ((V, E), \sigma^0)$. Let out$_{\text{excited}}(\sigma)$ 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 $w \in \text{out}_{\text{excited}}(\sigma)$ and the binary value of $w$ in $\sigma$ is 1 (or 0). For example, the decorated signal state of $\sigma_1$ in the previous example is $(a, x, y) = (1RR)$. Figure 1 (d) shows decorated signal states on the dummy-free timed state graph. Based on this definition of decorated signal states, the detection of CSC violations can be done in the same way as STGs without dummy transitions.

The property called output semi-modularity 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 semi-modular, if its dummy-free state graph $G_{DF} = ((V', E'), \sigma'^0)$ 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 Figure 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 Figures 2 (b) and (c).

\(^2\)Note that in (1010), some inputs may be excited, but only outputs are decorated in our definition.
According to the excitation defined 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 $\sigma_3$ 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 totally 40 time units later than the firing of $a^+$, while 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 report, we use the following simplified definition of output semi-modularity for timed STGs. A timed STG $G$ is output semi-modular, if for any conflicting transitions that are enabled in the same timed state of $G$, 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 Figure 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 semi-modular 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 like other tools such as petrify [1] and attacks [24]. Furthermore, the consistency significantly simplifies the analysis and synthesis algorithms. Thus, we say that an STG $G$ is synthesizable, if $G$ is one-safe, consistent, output semi-modular, 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., $[\text{Eft}(t), \text{Lft}(t)]$) 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 non-zero delay. In order to simplify the problem, this report 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.

III. DECOMPOSITION THEORY

There are several definitions of correctness in the context of the decomposition based synthesis. In [16], it is defined that the parallel composition of the contracted STGs for all outputs has the state graph isomorphic to that of the original STG. In [17], it is similar to the conformance used in [25]. Although this correctness is less strict than the Chu’s one, STGs must be deterministic (e.g., dummy-free). 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. This section gives the formal definition of our correctness and several lemmas and a theorem that are important for our method. It is assumed that the given STG $G$ is synthesizable and timed implementable.

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

For $x \in \text{Out}$, $\text{ES}(x^+)$ denotes a set of nondecorated signal states mapped from reachable timed states of a dummy-free timed state graph $G_{\text{df},G}$, where their decorated signal states have $R$ for $x$, and $\text{QS}(x^+)$ denotes a set of similar nondecorated signal states except that their decorated signal states have 1 for $x$. $\text{ES}(x^-)$ and $\text{QS}(x^-)$ are defined similarly. The other nondecorated signal states are unreachable, and this set is denoted by $\text{UR}$. From the definition of CSC, if and only if an STG has CSC, its $\text{ES}(x^+), \text{QS}(x^+), \text{ES}(x^-),$ and $\text{QS}(x^-)$ are disjoint for each $x \in \text{Out}$.

A circuit is defined by a set of logic functions (i.e., the technology mapping is beyond the scope of this report), and a logic function is specified by a cover, which is a set of nondecorated signal states where the logic function takes the value 1. In this report, the implementation technologies considered are atomic gates and generalized-C (gC) elements. In the atomic gate implementation, an STG $G$ defines for each $x \in \text{Out}$ 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

$$ES(x^+) \subseteq C(x^+) - UR \subseteq ES(x^+),$$

$$ES(x^-) \subseteq C(x^-) - UR \subseteq ES(x^-) \cup QS(x^-).$$
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 $C_1(x) - UR_1 = ES_1(x^+)$ must satisfy $C_1(x) - UR = ES(x^+) ∪ QS(x^+)$.

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., $|Eft(t), Lift(t)|$) 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 if and only if $G$ 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 $G_1$, the delay-correctness of $G_1$ is easily achieved by disallowing the contraction for the transitions in trigger($x$) $∪ \{x\}$ for $x ∈ Out_1$. Thus, the rest of this section focuses on the cover-correctness.

For STGs $G_1$ and $G_2$ with $Out_1 = Out_2$ and $In_1 = In_2$, a simulation from $G_1$ to $G_2$ is a relation $S$ between timed states of $G_{G_1}^s = (\{(V'_1, E'_1), σ'_0\})$ and $G_{G_2}^s = (\{(V'_2, E'_2), σ'_2\})$ satisfying

- $(σ'_0, σ'_2) ∈ S$,
- for any $(σ_1, σ_2) ∈ S$, outexcited($σ_1$) = outexcited($σ_2$)
- for any $(σ_1, σ_2) ∈ S$ and any $(σ_1, t, θ, σ'_1) ∈ E'_1$, there exists some $t_2$ and $σ'_2$ such that $l(t_2) = l(t_1)$, $(σ_2, t_2, θ, σ'_2) ∈ E'_2$, and $(σ'_1, σ'_2) ∈ S$ 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 1:** For STGs $G_1$ and $G_2$, if $G_1 \rightsquigarrow G_2$ and $G_2$ has CSC, then for $x ∈ Out_1$, the following holds.

- $ES_1(x^+) ⊆ ES_2(x^+)$,
- $ES_2(x^+) - ES_1(x^+) ⊆ UR_1$
- $QS_1(x^-) ⊆ QS_2(x^-)$,
- $QS_2(x^-) - QS_1(x^-) ⊆ UR_1$,
- $UR_1 ⊆ UR_2$.

**Proof:** For $s_1 ∈ ES_1(x^+)$, let $σ_1$ denote a timed state of $G_{G_1}^s$, from which $s_1$ is mapped, and suppose that $σ_1$ is reached from $σ'_0$ by a sequence $v_1$ of transitions on $G_{G_1}^s$. From $G_1 \rightsquigarrow G_2$, a timed state, denoted by $σ_2$, is reachable on $G_{G_2}^s$ by a sequence $v_2$ of transitions such that $l(v_1) = l(v_2)$, where $l(t_1, t_2, ...)$ = $l(t_1)l(t_2)⋯$ (note that $λ$ is deleted in this sequence). Thus, $σ_2$ is mapped to also $s_1$. Since outexcited($σ_1$) = outexcited($σ_2$) and $s_1 ∈ ES_1(x^+)$ hold, $s_2 ∈ ES_2(x^+)$ also holds. Hence, $ES_1(x^+) ⊆ ES_2(x^+)$ holds. The other three cases can be proved similarly. Next, suppose that $s_2 ∈ ES_2(x^+) - ES_1(x^+)$ holds. Since the value for $x$ is 0 in $s_2$, $s_2$ is included in either $QS_1(x^-)$ or $UR_1$. If $s_2 ∈ QS_1(x^-)$ holds, then $s_2 ∈ QS_2(x^-)$ holds from $QS_1(x^-) ⊆ QS_2(x^-)$. This, however, violates that $G_2$ has CSC from $s_2 ∈ ES_2(x^+)$. 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 $UR_1$, $UR_1 ⊆ UR_2$ holds.

The key of this proof is that the corresponding timed states in $G_{G_1}^s$ and $G_{G_2}^s$ have the same set of excited signals. For untimed methods, the trace equivalence relation is used to guarantee this (e.g. in [21]). For timed methods, however, such a relation on traces are not helpful, because excited transitions does not necessarily fire as shown in Figure 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 $CD(D)$, 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 $s = (abc)$ and $D = \{a, b\}$, $CD(s) =$ \{0001, 0101, 1001, 1101\} and its core is $(cd) = (01)$. The mappings from $D$-closure $CD(s)$ to its core $s'$ and its inverse are defined by $proj_D(CD(s))$ and $proj_D^{-1}(s')$. Note that both are the one-to-one mappings. The $D$-closure and these mappings are extended to sets as follows:

$$CD(S) = \bigcup_{s ∈ S} CD(s), \quad proj_D(CD(S)) = \{proj_D(CD(s)) | s ∈ S\}, \quad proj_D^{-1}(S') = \bigcup_{s' ∈ S'} proj_D^{-1}(s')$$

For an STG $G$ and $x ∈ Out$, a set $D$ of signals is an irrelevant input set for $x$, if

1. $D \subseteq In ∪ Out - \{x\}$,
2. $CD(EQ(x^+)) - UR = ES(x^+)$,
3. $CD(EQ(x^-)) - UR = ES(x^-)$.

From this definition, the following lemma holds.

**Lemma 2:** For $x ∈ Out$ and any irrelevant input set $D$ for $x$, the following hold.

- $CD(QS(x^+)) - UR = QS(x^+)$,
- $CD(QS(x^-)) - UR = QS(x^-)$.

**Proof:** Suppose $CD(QS(x^+)) - UR \neq QS(x^+)$. From $QS(x^+) ⊆ CD(QS(x^+))$, this means that for some $s ∈ CD(QS(x^+)) - UR$, $s \notin QS(x^+)$ holds. From $x \notin D$, such $s$ must be in $ES(x^-)$. Since $s ∈ CD(QS(x^+))$ holds, there exists $s_2 ∈ QS(x^+)$ such that $s ∈ CD(s_2)$. From $s ∈ CD(s)$, $CD(s_2)$ and $CD(s_2)$ have a common element, which implies $CD(s) = CD(s_2)$ and so $s_2 ∈ CD(s)$. From $s ∈ ES(x^-)$ and $CD(EQ(x^-)) - UR = ES(x^-)$, $s_2 ∈ ES(x^-)$ 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 $G$, $x ∈ Out$ 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 $sig(G) - D - \{x\}$ and the output signal set $\{x\}$. Let $ES_1$, $QS_1$ and so on be for $G_{D,x}$.

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

- $proj_D^{-1}(ES_1(x^+)) = CD(ES(x^+))$,
- $proj_D^{-1}(ES_1(x^-)) = CD(ES(x^-))$,
- $proj_D^{-1}(QS_1(x^+)) = CD(QS(x^+))$,
- $proj_D^{-1}(QS_1(x^-)) = CD(QS(x^-))$,
- $proj_D^{-1}(UR_1) ⊆ UR$. 


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 $G_G$ and $G_{G_{D,x}}$. In the dummy-free timed state graph $G^d_G$, however, some timed states that are in $G^d_{G_{D,x}}$ are deleted from the construction of the dummy-free timed state graphs, as shown in Figure 3. Note that $G^d_G$ and $G_{G,D,x}$ also have a one-to-one mapping if $G$ have no dummy transitions, and Figure 3 shows this case.

The proof of this lemma first shows $ES_1(x) \subseteq \proj_D(C_D(ES(x))))$. Suppose that $s_1 \in ES_1(x)$, and let $s_1$ be the timed state in $G^d_{G_{D,x}}$ that is mapped to $s_1$. This $s_1$ exists also in $G^d_G$ from the above relation between $G_G$ and $G^d_{G_{D,x}}$, and let $s$ be its signal state. Then, $s_1 = \proj_D(C_D(s))$. Furthermore, $s \in ES(x)$ holds from the following reason. From the signal excitation of $G^d_{G_{D,x}}$, $x$ must be excited to rise in some timed state reached only by transitions related to the signals in $D$, such as $s_2$ shown in Figure 3. Let $s_2$ be its signal state. Then, $s_2 \in ES(x)$ and $s \in C_D(s_2)$, so $s \in ES(x)$ holds. Hence, $s_1 = \proj_D(C_D(ES(x))))$ is derived.

Next, $ES_1(x) \supseteq \proj_D(C_D(ES(x))))$ is proved by showing that for $s \in ES(x)$, $s_1 = \proj_D(C_D(s))$ and $s_1 = \proj_D(\proj_D(C_D(s)))$, and so, $s \in ES(x)$ holds. Hence, $s_1 = \proj_D(C_D(ES(x))))$ is derived.

In the latter case, from the construction of the dummy-free state graphs, there exists a timed state $s_1$ included in both $G^d_G$ and $G^d_{G_{D,x}}$, from which $s$ is reached by transitions related to the signals in $D$ (see Figure 3). From this relation between $s_1$ and $s$, and $s_1$ has the signal state $s' \in C_D(s)$, and $s_1 = \proj_D(C_D(s))$ in $G^d_{G_{D,x}}$, since $D$ is the irrelevant input set, $C_D(ES(x)) - UR = ES(x)$ holds, and so, $s' \in ES(x)$ holds. Thus, $x$ is excited to rise in $s'$ in $G^d_G$, so $s_1 \in ES_1(x)$ is derived.

Hence, $ES_1(x) = \proj_D(C_D(ES(x))))$ is shown. Applying $\proj_D$ derives the final property. The proofs for the other three properties are similar.

Furthermore, from the above discussion, if $s$ mapped to $s$ is reached in $G^d_G$, then some $s'$ mapped to $s_1 = \proj_D(C_D(s))$ is also reached in $G^d_{G_{D,x}}$. Thus, if $s_1 \in UR_1$ holds, then every $s \in \proj^{-1}_D(s_1)$ is also in $UR$. Hence, the final property holds.

Lemma 4: For $x \in Out$ and a set $D$ of signals with $D \subseteq In \cup Out - \{x\}$, if $G_{D,x}$ has CSC, and $D \cap trigger(x) = \emptyset$ 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 $s' \in C_D(s) - UR$ such that $s' \notin ES(x)$, or there exist signal states $s \in ES(x)$ and $s' \in C_D(s) - UR$ 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 $s$ and $s'$ in $G^d_G$ whose signal states are $s$ and $s'$, respectively. From the construction of the dummy-free timed state graphs, there exist timed state $s_1$ and $s_1'$ in $G^d_{G_{D,x}}$ whose signal states are $s_1$ and $s_1'$, satisfying $s_1 = \proj_D(C_D(s))$ and $s_1' = \proj_D(C_D(s'))$ (see Figure 3). Note that $s_1 = s_1'$ holds from $\proj_D(C_D(s)) = \proj_D(C_D(s'))$. These $s_1$ and $s_1'$ cannot be the same from the following reason. $s$ and $s'$ can be the same, only when one of them is reached from the other on $G^d_G$ 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 semi-modularity of $G$, or $x$ is enabled without firing any transition in trigger($x$) from $D \cap trigger(x) = \emptyset$, which is impossible. Thus, $s_1$ and $s_1'$ must be different. Since $x$ is excited to rise in $s_1$, so is it in $s_1'$ of $G^d_{G_{D,x}}$. On the other hand, although $x$ is not excited in $s_1'$, $x$ may be considered to be excited to rise in $s_1'$ of $G^d_{G_{D,x}}$ from the excitation definition, if $x$ is enabled in some timed state of $G_{G_{D,x}}$ (such as $s_2$ in Figure 3) that is reached from $s_1'$ by only dummy transitions. But, this cannot happen, because $x$ is disabled in $s'$ and $D \cap trigger(x) = \emptyset$ holds. Therefore, $x$ is not excited in $s_1'$ of $G^d_{G_{D,x}}$, either. This contradicts that $G_{D,x}$ has CSC, because $x$ is excited in $s_1'$, not excited in $s_1$, and $s_1 = s_1'$ holds. Hence, $D$ should be an irrelevant input set.

Lemma 5: For $x \in Out$ and any irrelevant input set $D$ for $x$, suppose that $G'$ satisfies $G_{D,x} \sim G'$. 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 $G_{D,x}$, $ES_2$, $QS_2$, and $UR_2$ be for $G'$.

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

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

(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

$$ES(x) \subseteq \proj^{-1}_D(C_2(x)) - UR \subseteq ES(x) + QS(x)$$

(2)

to show the above, this proof first shows

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

(3)

and then (2) is shown.

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

$$ES_2(x) - UR_2 \subseteq C_2(x) - UR_2 - UR_1 \subseteq ES_2(x) + QS_2(x) - UR_1$$

(4)
From $UR_1 \supseteq UR_2$ as shown in Lemma 1,
\[
C_2(x) - UR_2 - UR_1 = C_2(x) - UR_1
\]
holds. Furthermore, $ES_2(x) - UR_1$ is rewritten to $ES_1(x)$ as follows using Lemma 1.
\[
ES_2(x) - UR_1 = ES_1(x) + (ES_2(x) - ES_1(x)) - UR_1 = ES_1(x) - UR_1 = ES_1(x).
\]
Similarly, $QS_2(x) - UR_1 = QS_1(x)$ holds. Hence, (4) is rewritten to (3).

Next, applying $\text{proj}_D^{-1}$ to (3) derives
\[
\text{proj}_D^{-1}(ES_1(x)) \subseteq \text{proj}_D^{-1}(C_2(x) - UR_1) \subseteq \text{proj}_D^{-1}(ES_1(x) + QS_1(x)).
\]

From Lemma 3, this is rewritten as
\[
C_D(ES(x)) \subseteq \text{proj}_D^{-1}(C_2(x) - UR_1) \subseteq C(D(ES(x)) + C_D(QS(x))),
\]
and, by removing $UR$,.
\[
C_D(ES(x)) - UR \subseteq \text{proj}_D^{-1}(C_2(x) - UR_1) - UR \subseteq (C_D(ES(x)) - UR) \cup (C_D(QS(x)) - UR)
\]
is obtained. Since $D$ is an irrelevant input set, $C_D(ES(x)) - UR = ES(x)$ holds, and from Lemma 2, $C_D(QS(x)) - UR = QS(x)$ holds. Thus,
\[
ES(x) \subseteq \text{proj}_D^{-1}(C_2(x) - UR_1) - UR \subseteq ES(x) + QS(x)
\]
holds. The above $\text{proj}_D^{-1}(C_2(x) - UR_1) - UR$ can be rewritten as follows from $\text{proj}_D^{-1}(UR_1) \subseteq UR$ by Lemma 3.
\[
\text{proj}_D^{-1}(C_2(x) - UR_1) - UR = \text{proj}_D^{-1}(C_2(x)) - \text{proj}_D^{-1}(UR_1) - UR = \text{proj}_D^{-1}(C_2(x)) - UR.
\]

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

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

**Theorem 1:** If $\text{abs}(G, V, x)$ has CSC for $V$ with trigger$(x) \subseteq V$, then $\text{abs}(G, V, x)$ is cover-correct with respect to $G$.

**Proof:** From trigger$(x) \subseteq V$, $D = \text{sig}(G) - V$ satisfies $D \cap \text{trigger}(x) = \emptyset$. $\text{abs}(G, V, x)$ has CSC. Thus, from Lemma 4, $D$ is an irrelevant input set for $x$. From the above definition, $G_{D,x} \sim \text{abs}(G, V, x)$ holds, and $\text{abs}(G, V, x)$ has CSC. Hence, from Lemma 5, $\text{abs}(G, V, x)$ is cover-correct with respect to $G$.

**IV. DECOMPOSITION BASED SYNTHESIS OVERVIEW**

The top level algorithm for the proposed decomposition based synthesis is shown in Figure 4. For a given synthesizable and timed implementable STG $G$, our algorithm tries to compute an abstraction $G_{abs}$ for each output signal $x$ of $G$.

$$\text{decomposition_based_synthesis}(G) \{$$

\[
\text{forall } x \in \text{Out} \{
G_{abs} = \text{obtain}_{\text{abs}}(G, x);
C_x = \text{timedLogicSynthesis}(G_{abs});
\}
\]

$$\}

Fig. 4. Top-level algorithm for synthesis.

**obtain_{\text{abs}}(G, x) \{**

\[
V = \{x\} \cup \text{trigger}(x);
\]

while(true) {

\[
G_{abs} = \text{contract}_{\text{STG}}(G, V);
(res, ssg) = \text{check}_{\text{synthesizable}}(G_{abs});
\]

if $(res = \text{"synthesizable") return $G_{abs};$

if $(res = \text{"one-safeness violation") \{$

if $(\text{no transition can be contracted)}$ abort;

disallow contraction of some transitions; continue;

$\}

if $(res = \text{"consistency or O.S.M violation") abort;

$CSCV = \text{obtain}_{\text{CSCV}}(\text{trigger} \text{trace}_{\text{set}}(G_{abs}, ssg));$

\forall g \in CSCV \{$

\[
\text{candidate} =
\text{analyze}_{\text{CSCV}}(g, V, x);
\]

if $(\text{candidate == \text{"not found") abort; add}\text{constraints}_{\text{matrix}}(\text{candidate});$

\}

\text{newV} = \text{solve}_{\text{covering}_{\text{problem}}};$

\text{V} = \text{V} \cup \text{newV};$

\}

$$\}

Fig. 5. Algorithm to obtain an abstraction.

such that a correct circuit for $x$ can be synthesized from it. Then, a timed circuit synthesis algorithm is applied to $G_{abs}$.

The algorithm for obtaining such an abstraction is shown in Figure 5. It first constructs the initial input set $V$ for $x$ by taking $x$ and its possible trigger signals. As shown in Theorem 1, the input signal set needs to include those trigger signals in order to obtain a correct abstraction.

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 $\text{sig}(G) - V$ by dummy transitions. This contraction of transitions should produce a reduced STG $G''$ such that $G' \sim G''$ and $G''$ is delay-correct with respect to $G'$. Such contraction of timed STGs can be done in a way similar to that shown in [18]. Our method, however, applies the contraction to only transitions with a restricted class of the flow relation, such that consistency and output semi-modularity are preserved. We prefer this exact contraction 3, because it synthesizes more optimal circuits than general contraction. The further details about the contraction algorithm are omitted in this report. Note that the resultant STG may contain dummy transitions due to the exact contraction.

The reduced STG $G_{abs}$ obtained by the contraction is then checked if it is synthesizable or not. This process needs to enu-

3In order to preserve consistency, it is necessary to keep the timing information exactly. That’s why it is called exact contraction. Note that it (or any contraction algorithm of timed STGs) cannot preserve one-safeness in general.
merate its timed state space and construct the decorated signal state graph $ssg$ that corresponds to its dummy-free timed state graph. If $G_{abs}$ is synthesizable, the algorithm returns it, because it is correct with respect to the original STG from Theorem 1. If it has 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 it is not difficult. 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. If consistency or output semi-modularity is violated (indicated by ‘consistency or O.S.M violation’), it is violated also in the original STG, because our contraction is exact. Thus, the algorithm aborts.

The remaining case is that $G_{abs}$ 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 $G_{abs}$ that cause CSC violations, $CSCV$, is extracted 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 are added to $V$. This $V$ is used to compute a new $G_{abs}$, and the algorithm repeats the above process.

V. CHECKING SYNTHESISABILITY

The reduced STG $G_{abs}$ is checked if it is synthesizable or not in check_synthesizable as shown in Figure 5. This is done by exploring the timed state space of $G_{abs}$ 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 $\alpha$, fireable($\alpha$) denotes the set of fireable 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 fireable 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 this equivalence classes is constructed, its dummy free version is obtained by the way explained in Figure 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 semi-modularity, 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 fireable 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 $G_{abs}$.

VI. ANALYZING CSC VIOLATION TRACE

If $G_{abs}$ 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 report.

The algorithm analyze_CSCV_trace first generates a concrete trace of the original STG $G$ which corresponds to the given CSC violation trace of $G_{abs}$. This is done by a technique similar to the one developed for the partial order reduction, which 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 Figure 6.

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

A. Regular concrete traces

Each CSC violation trace $g$ of $G_{abs}$, constructed by obtain_CSC_violation_trace_set, is assumed to be of the form $g = (g_0, g_1, g_2)$, $s_0 \overset{\gamma_0}{\rightarrow} s_1$, $s_1 \overset{\gamma_1}{\rightarrow} s_2$, and $s_2 \overset{\gamma_2}{\rightarrow} s_3$, where $s_0$ is the initial signal state of $G_{abs}$, $s_1$ and $s_2$ are the first two signal states that correspond to the CSC violation pair, and
analyze_CSCV_trace(g, G, V, x) {
    f = guided_sim_phase1(g, G, V, x, α0, null);
    can = find_inputs(f, G, V, x);
    if (can == false) return "not found";
    else return can;
}

Fig. 6. Algorithm for analyzing CSC violation trace.

Fig. 7. Labeling of a concrete trace.

g2 contains exactly one (interface) transition. For this g, the corresponding concrete trace f is constructed by the guided simulation shown later, where f = (f0, f1, f2) such that for 0 ≤ i ≤ 2, projecting out noninterface and dummy transitions from fi is equal to gi. It is assumed that f is separated into fi’s when interface transitions fire, i.e., each fi ends with an interface transition. Let α1, α2, and α3 denote the timed state classes of G obtained by f0, f1, and f2, respectively (see Figure 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) ∈ R1f. For any two transitions t1 and t2 in f, if t2 fires by consuming the token produced by the firing of t1, it is denoted by (t1, t2) ∈ R2f. In the untimed case, the actual causality relation for f is defined by the transitive closure of the union of R1f and R2f, i.e., (t1, t2) ∈ (R1f ∪ R2f)∗. 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) ∈ (R2f)∗, let pathf(t, u) denote a set of paths from t to u defined by the relation R2f, where each path is represented by a set of transitions. That is,

\[
\text{path}_f(t, u) = \begin{cases} 
\emptyset, & \text{if } t = u, \\
\{u\} \cup q | (t', u) \in R_2^f, q \in \text{path}_f(t, t') & \text{else.}
\end{cases}
\]

For example, if f is a concrete trace obtained from the STG shown in Figure 9 (a), then pathf(c+, x−) is \{\{a+, x+, a−, x−\}, \{a+, b−, a−, x−\}\}. Furthermore, let

\[
\begin{align*}
\text{max}_{eft} f(t, u) &= \max_{q \in \text{path}_f(t, u)} \left( \sum_{t \in q} \text{Eft}(t) \right), \\
\text{max}_{lft} f(t, u) &= \max_{q \in \text{path}_f(t, u)} \left( \sum_{t \in q} \text{Lft}(t) \right).
\end{align*}
\]

Finally, for any two transitions t1 and t2 in f, if there exists a transition t3 in f such that (t3, t1) ∈ (R2f)∗, (t3, t2) ∈ (R2f)∗, and maxeft_f(t3, t1) < maxeft_f(t3, t2), then it is denoted by (t1, t2) ∈ R2f. Intuitively, (t1, t2) ∈ R2f implies that t1 and t2 has a common ancestor t3, and the maximal time separation between t3 and t1 is smaller than the minimal time separation between t3 and t2. Thus, t3 cannot fire before t2 under the causality relation R2f. This timed causality relation can be computed using a time separation algorithm such as shown in [11].

Using the above relations, we say that t1 is an ancestor of t2 in f, denoted by [t1 \sim_f t2], if t1 and t2 are related by the transitive closure of the union of R1f, R2f, and R3f. This ancestor relation represents an actual causality relation with respect to the specific abstracted trace g. In the rest of this report, we use the following terminology.

- t1 causes t2, if [t1 \sim_f t2] holds.
- t1 and t2 are ordered, if either [t1 \sim_f t2] or [t2 \sim_f t1] holds.
- t1 and t2 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’ ∈ ||f||G, (R1f ∪ R2f ∪ R3f)∗ 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 \sim_f x].

B. Determining the input set

The idea to resolve the CSC violation between α1 and α2 is to add to the input set a noninterface signal w such that f1 contains odd number of w-transitions. If w-transition fires in f1 in odd times, then the signal takes different values in α1 and α2, 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 final($h$) denote the last transition in $h$, and 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, 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 before($l_1$), before($l_2$), final($l_2$), and final($l_1$) are distinct, and they are caused in this order, i.e., before($l_1$) $\sim_f$ before($l_2$), before($l_2$) $\sim_f$ final($l_2$), final($l_2$) $\sim_f$ final($l_1$) (see Figure 8). For a signal $w$ and a nested subtrace pair $(l_1, l_2)$ of $f$, $w$ is semi-essential with respect to $(l_1, l_2)$ in $f$, if

- none of before($l_1$), before($l_2$), final($l_2$), and final($l_1$) is a $w$-transition, and
- every $w$-transition is ordered with before($l_1$), before($l_2$), final($l_2$), and final($l_1$).

Similarly, $w$ is essential with respect to $(l_1, l_2)$ in $f$, if $w$ is semi-essential with respect to $(l_1, l_2)$ in $f$, and $l_2$ contains an odd number of $w$-transitions while neither $l_1$ nor $l_2$ contains any $w$-transition, where $l_1$ and $l_2$ are the subtraces of $l_1$ before $l_2$ and after $l_2$ as shown in Figure 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 before($l_1$), before($l_2$), final($l_2$), and final($l_1$). For a subtrace $l$, let $TS(l)$ denote a set of timed state classes in which the transitions in $l$ fire (see Figure 8). $SS(l)$ denotes the corresponding signal state set. The following lemma holds.

**Lemma 6**: 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 $SS(l'_2)$ are distinguished from those in $SS(l'_1)$ by the signal $w$.

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

Consider the first interface transition in $f_1$, and divide $f_1$ into $f_{1h}$ and $f_{1t}$ with it, i.e., $f_1 = < f_{1h}, f_{1t} >$ and $f_{1h}$ ends with this first interface transition in $f_1$. Figure 7 shows the relation among $f_0 \cdots f_2$ as well as $f_{1h}$ and $f_{1t}$. Let $l_1 = < f_1, f_2 >$ and $l_2 = f_{1t}$. Then, $l_{12} = f_{1h}$ and $l_{21} = f_{2h}$ hold. From the definitions of $f_0$, $f_{1h}$, $f_{1t}$, and $f_2$, they end with interface transitions. Thus, each of them are different, and they are caused in this order. Hence, $(l_1, l_2) = (f_1, f_2, ) f_{1t}$ is a nested subtrace pair of $f$. The following theorem holds.

**Theorem 2**: The CSC violation with respect to $f' \in ||f||_G$ with $f = < f_0, f_1, f_2 >$ is resolved by adding a noninterface signal $w$ to the input set, if $w$ is essential with respect to $(l_1, l_2) = (f_1, f_2, ) f_{1t}$ 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 $TS(l'_2)$ and those in $TS(l'_1)$. Those signal states are distinguished by $w$ from Lemma 6.

For example, consider an STG shown in Figure 9 (a). The output $x$ has the possible trigger signal $a$, and so, the initial $V = \{a, x\}$. Then, the reduced STG with interface signals $a$ and $x$ has one CSC violation trace, and its corresponding concrete trace $f$ is shown in Figure 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 $f$. $(f_1, f_2, f_{1t})$ is a nested subtrace pair of $f$, and the noninterface signal $c$ is essential with respect to it. The noninterface signal $b$ is not semi-essential, because it is not ordered with final($l_1$) = $x+$, $TS(l_{12}) = \{ \alpha_1, \alpha'_1 \}$ and $TS(l_{21}) = \{ \alpha_2, \alpha'_2 \}$ 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 $((f_1, f_2), f_{1t})$ in $f$, the CSC violation cannot be resolved by adding a single noninterface signal. However, CSC violations can be resolved by adding two or more noninterface signals to the input set. There are two cases depending on the existence of semi-essential noninterface signals.

If there exist in $f_{1h}$ or $f_2$, noninterface signals that are semi-essential with respect to $((f_1, f_2), f_{1t})$, the problem can be reduced to several sub-problems that can be solved by the above approach. Suppose that such a semi-essential noninterface signal $w$ changes as shown in Figure 10. Then, by adding $w$ to the input set, the timed state classes that cause CSC violation (i.e., those in $TS(f_{1h})$ and $TS(f_2)$) are divided into two groups: one is $TS(h_1)$ and $TS(h_4)$, and the other is $TS(h_2)$ and $TS(h_3)$. The first group can be handled by considering a nested subtrace pair $((f_1, f_2), (h_2, f_{1t}, h_3))$. Even if there exists no essential noninterface signal for $((f_1, f_2), f_{1t})$, this $((f_1, f_2), (h_2, f_{1t}, h_3))$ may have it, because $(h_2, f_{1t}, h_3)$ is longer than $f_{1t}$. Similarly, the second group can be handled by a nested subtrace pair $((h_2, f_{1t}, h_3), f_{1t})$, and it may have an essential noninterface signal, because $(h_2, f_{1t}, h_3)$ is shorter than $(f_1, f_2)$. We say that $((f_1, f_2), (h_2, f_{1t}, h_3))$ or $(h_2, f_{1t}, h_3), f_{1t})$ is reduced from $((f_1, f_2), f_{1t})$ with respect to a semi-essential signal $w$. As mentioned above, 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 the signals in $\{a, c, x\}$ to resolve the CSC violation using more noninterface signals.\[\]

Second, if there exists in $f_{1h}$ or $f_2$ no noninterface signal that is semi-essential with respect to $((f_1, f_2), f_{1t})$, a more complicated process is necessary to resolve the CSC violation in $\|f\|_G$. For such a nested subtrace pair $(l_1, l_2)$, our algorithm chooses a noninterface transition $t$ fired in $f$, such that $t$ is concurrent with a transition in a set $\{\text{before}(l_1), \text{before}(l_2), \text{final}(l_2), \text{final}(l_1)\}$. Let $u$ denote such a transition in the above set. It then generates two traces $f'$ and $f''$ from $f$ by interleaving $t$ and $u$. The idea is that our algorithm tries to predict from those interleavings the situation where the noninterface signal $w$ related to $t$ is added to the input set and its every transition is ordered with the other interface transitions, before actually doing it. Interleaving $t$ and $u$ can be done as follows. Suppose that $t$ and $u$ fire in this order in $f$. The new trace $f'$ is obtained simply by adding $[t \sim f u]$ to the ancestor relation of $f$. Similarly, $f''$ is obtained by adding $[u \sim f t]$ 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 $t$ and the noninterface transitions caused by $t$ and fired before $u$, next to $u$.

For example, consider the STG shown in Figure 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 $g_0 = c + a + c$, $g_1 = b + b -$ and $g_2 = c -$. The guided simulation generates $f_0 = c + x_1 + x_2 + a +$, $f_1 = b + x_1 - b -$, and $f_2 = c -$ as shown in Figure 12 (a). Our algorithm first looks for a semi-essential noninterface signal for $((f_1, f_2), f_{1t})$, but both noninterface transitions $x_1+$ and $x_2+$ are concurrent with before, so $((f_1, f_2)) = a +$. Thus, this STG has no semi-essential signals. Here, choose $x_1$ and before $((f_1, f_2)) = a +$, and generate $f'$ and $f''$ by interleaving them. It is not easy to illustrate these traces in a figure like Figure 12, because it does not show the ancestor relation precisely, but assume that in Figure 12, $x_1+$ causes $a+$ in $f'$ while $a+$ causes $x_1+$ in $f''$. Note that in $f''$, $x_1 +$ and $x_2+$
(which is caused by \( x_{1+} \) and fired before \( a+ \)) are moved next to \( a+ \). In \( f' \), \( x_{1} \) is now essential with respect to \( \langle f_{1}, f_{2} \rangle \), because every \( x_{1} \)-transition is ordered with \( a+ \), \( b+ \), \( b- \), \( c- \), and only one \( x_{1}- \) exists in \( f_{1i} \). Thus, this CSC violation can be resolved in \( || f ||_G \) by adding \( x_{1} \) in the input set. In \( f'' \), \( x_{1} \) is semi-essential, but not essential, because \( x_{1} \) 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 Figure 12 (b), because only \( a_{1} \) and \( a_{2} \) cause the CSC violation after adding \( x_{1} \). Then, our algorithm looks for another noninterface transition that is essential with respect to \( (l_{1}, l_{2}) \) in \( f'' \), which is \( x_{2} \) in this case. In other words, \( x_{2} \) can resolve the CSC violation between \( TS(l_{1}, l_{2}) = \{a_{1}\} \) and \( TS(l_{21}) = \{a_{2}\} \). Therefore, the CSC violation with respect to \( f'' \) can be resolved in \( || f'' ||_G \) by adding both \( x_{1} \) and \( x_{2} \).

In general, the above process should be repeated by generating 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 pseudo code shown in Figure 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 1 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 3: The algorithm shown in Figure 13 returns “false”, only when the given STG have no CSC.

Proof: It can return “false”, only when there exists no noninterface transition that is concurrent with some of before\( (l_{1}) \), before\( (l_{2}) \), final\( (l_{2}) \), and final\( (l_{1}) \). It implies that every noninterface signal except for the signals related to before\( (l_{1}) \), before\( (l_{2}) \), final\( (l_{2}) \), and final\( (l_{1}) \) is semi-essential. Since no essential signal exists and no reduced nested subtrace pair works, such every noninterface signal included in \( \langle f_{1}, f_{2} \rangle \) must appear in \( f_{1i} \) even times. Hence, the CSC violation in \( TS(f_{1h}) \) and \( TS(f_{2}) \) cannot be resolved, even if every noninterface signal is used. This implies that the given STG has no CSC.

C. Guided simulation

For a given abstracted trace \( g \), the 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 Figure 14. It consists of two phases. The 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 \( R_{2}^{l} \) and \( R_{1}^{l} \)) 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 the phase 2,
guided_sim(g, G, V, x) {
    h = guided_sim_phase1(g, G, V, x, a0, null);
    f = guided_sim_phase2(h, G, V, x, a0);
    return f;
}

guided_sim_phase1(g, G, V, x, α, h) {
    if (g is empty) return h;
    if (((g, α) is already visited) return “backtrack”;
    g1 = head(g);
    nec = necessary(α, g1);
    dep = ∅;
    for all t ∈ nec
        dep = dep ∪ dependent(α, t);
    for all t ∈ dep {
        α’ = fire(α, t);
        if (t == g1) g’ = tail(g);
        else g’ = g;
        h’ = append(h, t);
        result = guided_sim_phase1(g’, G, V, x, α’, h’);
        if (result ≠ “backtrack”) return result;
    }
    return “backtrack”;
}

dependent(α, t) {  
    E = new = {t};  
    while(true) {  
        new’ = ∅;
        for all x ∈ new
            for all y ∈ conflict(x) ∩ NonIF_Dum
                new’ = new’ ∪ necessary(α, y) – E;
        if (new’ == ∅) break;
        E = E ∪ new’;
        new = new’;
    }
    return E;
}

necessary(α, t) {  
    /* α = (μ, I) */
    if (t is already visited) return ∅;
    if (t ∈ enabled(μ))
        if (t ∈ firable(α)) return {t};
        else return {choose_one(firable(α) ∩ NonIF_Dum)};
    E = ∅;
    p = choose_one(μ – μ);
    for all x ∈ p ∩ NonIF_Dum
        E = E ∪ necessary(α, x);
    return E;
}

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

the concurrent noninterface or dummy transitions are added to
h or moved or order to satisfy the regularity.

In guided_sim_phase1, if g is nonempty, it picks the first
transition of g, denoted by g1, and computes its necessary set,
the set of transitions that should be fired for firing g1,
by 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 non-
interface 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 g1, and so, one of the firable
noninterface or dummy transitions chosen by choose_one is
returned. In the net shown in Figure 15, where t1 · · · t5 are
noninterface transitions, and g1 and g2 are interface transitions,
the necessary set of g1 is {t1}. guided_sim_phase1 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 Figure 15 for an abstracted
trace g1;g2. As shown above, the necessary set of g1 is {t1}.
If t1 is fired from the current marking, g1 becomes enabled,
but, g2 can never be fired. In this case, firing t3, t2, t4, and t1
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 the backtracking mechanism with firing 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 ∈ E,
and for each x ∈ E, the necessary transitions of conflict(x) ∩
NonIF_Dum are also in E. The dependent set can be defined as
a closure, and so, the while loop in Figure 14 computes it. In
our example, the dependent set of t1 is {t1, t3}. Hence, even if
guided_sim_phase1 fires t1 first, it eventually can fire t3
and then t2 after several backtracks. After firing t2, it is
straightforward to fire t4 and t1, because they are the necessary
transitions for g1. When g1 becomes enabled, its necessary set
is {g1} and its dependent set is {g1, t6}. The correct transition
to be fired here is t6, 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.

In guided_sim_phase2, each transition determined by
find_firing_trans is fired until h, which is updated by also
find_firing_trans, becomes empty as shown in Figure 16.
In find_firing_trans, if the first transition t of h is interface,
its enabled in the current marking because all its
necessary transitions are supposed to be fired. In order to
satisfy the 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.
guided_sim_phase2(h, G, V, x, α) {
    f = null;
    fired = null;
    while(true) {
        (t, h) = find_ring(trans(α, h));
        f = append(f, t);
        α = fi_re(α, t);
        if (h is empty) break;
    }
    return f;
}

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

find_concur_trans(α, t, h, h') {
    fi nd x ∈ firable(α) ∩ NonIFDum
    s.t. conflict(x) ∩ (prefix(h, x) - fired) = ∅;
    if (such x exists) {
        fired = fired ∪ {x};
        return (x, h);
    }
    else return (t, h');
}

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

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) ∩ prefix(h, x) ≠ ∅, 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) ∩ prefix(h, x) = ∅. If such a x is found, it is returned with the updated 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. 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’ = tail(h) are returned.

If t is noninterface or dummy, the 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 ∈ fired, it is just removed from fired, and the next transition of h is processed.

Otherwise, t and h’ = tail(h) are returned.

In the previous example, for an abstracted trace g₁g₂,

\[ t_3 t_2 t_4 t_1 t_6 t_5 g_1 t_7 t_9 g_2 \]

is obtained by guided_sim_phase1. This does not satisfy regularity, because t₇ and t₉ fire after g₁. When h = g₁ t₇ t₉ g₂ is first given to find_firing_trans, (t₇, h) is returned with fired = \{t₇\}. After firing t₇, both t₈ and t₉ become firable. t₈ is not chosen, because conflict(t₈) = \{t₉\} and prefix(h, t₈) = \{t₉\}. After firing t₉, however, t₈ is chosen and fired. Finally, g₂ is 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 the guided simulation with only one phase may be possible, but we believe that the above approach minimizes the number of the backtracks.

VII. LIMITATIONS

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 non-zero delay (i.e., its earliest firing time is greater than 0). This restriction is necessary to guarantee the termination of the second phase of the guided simulation. In the untimed case, this termination is not guaranteed, if a loop formed only by noninterface or dummy transitions exists [23]. 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, the 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.

VIII. EXPERIMENTAL RESULTS

The proposed method has been naïvely implemented using the C language. 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 gigabytes of memory.

For speed independent circuit synthesis, tools moebius [21] and csat [22] use the similar idea of the decomposition based synthesis. Since our method is extended for timed circuit synthesis and they run on different machines, the precise comparison with those tools is not very meaningful. According to the rough comparison on the several circuits used in [23] and [21], the performance of the synthesis and the

---

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.
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 [24], and the literal counts of the synthesized circuits are compared. Table I 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 non-optimal input sets. Since these examples are small, the CPU times for both methods are almost the same.

The last example shown in Table I 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 seconds.

Table II shows the results for much larger examples, which are taken from [14]. They are specifications for IIR filters, FIR filters, and portions of the Discrete Cosine Transform (DCT) circuits obtained from SpecC/Balsa high-level specifications (slightly modified versions are used for this experiment due to some improvement of our Balsa compiler). Those with “b” are allowed to use more operational units. Thus, they have more concurrency 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 the partial order reduction and POSET techniques are used for these examples, because otherwise the timed circuit synthesis does not terminate due to many dummy transitions left by the exact timed net contraction (the untimed circuit synthesis is not improved much by these techniques). Although the timed circuit synthesis takes longer time especially for the specifications with more concurrency, 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 seconds, but it runs out of memory for the other specifications. Since there are 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.

### IX. Conclusion

This report presents a decomposition based method for efficient synthesis of large timed circuits. The idea proposed for the speed independent circuits [23] has been extended for the 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.

**ACKNOWLEDGMENT**

We’d like to thank Walter Vogler and Mark Schaefer for their helpful comments. Furthermore, we would like to thank Josep Carmona, Jordi Cortadella, Victor Khomenko, and Alex Yakovlev for giving us their benchmarks and tools.

### REFERENCES

