

# Modular Synthesis of Timed Circuits using Partial Orders on LPNs

Eric G Mercer and Chris J. Myers<sup>1,2,3</sup>

*Department of Electrical and Computer Engineering  
University of Utah  
Salt Lake City, USA*

Tomohiro Yoneda<sup>4</sup>

*Department of Computer Science  
Tokyo Institute of Technology  
Tokyo, Japan*

Hao Zheng<sup>5</sup>

*IBM Microelectronics  
Essex Junction, USA*

---

## Abstract

This paper develops a modular synthesis algorithm for timed circuits that is dramatically accelerated by partial order reduction. Each timed circuit module is specified using a *level-ruled Petri net* (LPN), a new type of Petri net that allows timing constraints and Boolean level expressions to be annotated between each place and transition. The algorithm synthesizes each module in a hierarchical design individually. It utilizes partial order reduction to reduce the state space explored for the other modules by considering a single order on independent enabled transitions. This approach better manages the state explosion problem resulting in more than an order of magnitude reduction in synthesis time. The improved synthesis time enables the synthesis of a larger class of timed circuits than previously possible.

---

<sup>1</sup> This research is supported by NSF CAREER award MIP-9625014, NSF Japan Program award INT-0087281, SRC contract 97-DJ-487 and 99-TJ-694, a grant from Intel Corporation, and JSPS Joint Research Projects.

<sup>2</sup> Email: [eemercer@ece.utah.edu](mailto:eemercer@ece.utah.edu)

<sup>3</sup> Email: [myers@ece.utah.edu](mailto:myers@ece.utah.edu)

<sup>4</sup> Email: [yoneda@cs.titech.ac.jp](mailto:yoneda@cs.titech.ac.jp)

<sup>5</sup> Email: [haoz@us.ibm.com](mailto:haoz@us.ibm.com)

## 1 Introduction

In order to achieve high performance, designers are experimenting with aggressive *timed circuits* [26,27,24,12]. Designing these circuits, however, in a fully manual style is very difficult; thus, CAD tools are essential for synthesis and verification. An important issue in developing such CAD tools is the avoidance of state explosion. This paper presents two approaches to managing state explosion: first, a syntactic abstraction in the timed circuit specification; and second, an exact modular synthesis approach using partial order reduction.

Prior work has utilized a timed automaton to provide a low level model for a circuit. It is a state based specification language where transitions between states are governed not only by Boolean functions defined over inputs, but clock valuations too. Its expressiveness lends itself to verification [1,6,11,7,16,4,19]. The expressiveness, however, is not always required for synthesis; thus, it needlessly complicates the analysis problem. A Petri net based representation is an alternative circuit model to a timed automaton. It is a transition based specification language where transitions are governed by time and the marked state of the net. Although it is less expressive than timed automata, it is sufficient for not only synthesis but also verification [5,13,25,29,30,23]. Its structure, however, becomes very complicated when modeling even simple logic functions [30]. This increases the size of the reachable state space and makes specification difficult [2].

This paper uses *level-ruled Petri nets* (LPNs) which are a hybrid of Petri nets and timed automata. They are transition based but employ Boolean functions. Unlike timed automata, the functions are only defined over inputs, not inputs and clock valuations. Boolean functions are a syntactic abstraction in the model structure. Connectivity is no longer explicit in the edges, but implied through the Boolean state of the inputs. This reduces the reachable state space and simplifies specification. LPNs are a refinement of TEL structures described in [2,3]. LPNs restrict conflict to the Petri net formalism and facilitate a partial order reduction on the reachable state space.

Partial order reduction is an important tool in mitigating state explosion in verification [9,28,15,17,4,19]. Partial order reduction is applied to synthesis in [25,31]. The approach in [25] is an unfolding technique that is applied to untimed specifications. Not only is it not clear if the technique can be efficiently applied to a timed model, the technique ignores hierarchy in the specification; thus, it is limited in the size of systems it can be applied to. The approach in [31] exploits hierarchy in the specification by applying a partial order reduction to signals not on the interface of the target subcircuit. It modifies the partial order reduction method in [28,30] to always include all allowed orders of signals on the interface and in the target subcircuit. It then uses the state space based synthesis approach in [21] to produce an exact circuit. The work demonstrates a significant reduction in running time for state space exploration in the synthesis problem and greatly increases the size

of systems that can be analyzed. The approach, however, is tied to the time Petri net. This negatively impacts the size of the reduced state space due to the structural complexity of the model.

The work in this paper extends the modular synthesis approach in [31] to LPNs to further reduce the size of the reachable state space through syntactic abstraction. It first presents a new timing analysis algorithm for LPNs that can be applied to systems that are beyond the capacity of existing algorithms. The new algorithm is required because existing algorithms in [22,3] only support a partial order in timing information and not a partial order in the state space exploration. The partial order in the timing information in [22,3] is similar to that found in [4,19], and like the approach in [19] does not require extra reference clocks for synchronization. The basis for the new algorithm is actually presented in [19] and is based solely on the time separation of transitions, but an initial implementation on timed Petri nets in [18] shows it to be incorrect for Boolean expressions and incomplete for partial order reduction; thus, this work corrects the algorithm and completely derives the conditions necessary to preserve correctness in the reduction. The partial order reduction is restricted to safe nets and works for any type of choice structure. The partial order reduction uses untimed methods from [28] and timed methods [30,4,19] to determine independence between transitions. It augments these definitions to incorporate the notion of independence in the presence of Boolean functions. The definitions are not only tied to the structure of the net, but also consider the timing of transitions. This paper proves the modular synthesis approach on LPNs to produce exact circuits when the net is free of time dependent choice. The approach results in an order of magnitude reduction in the time cost of synthesis and enables the synthesis of a larger class of timed circuits than previously possible.

Interface abstraction is a common approach to reduce the cost of state based synthesis by exploiting hierarchy in the specification. As performing the abstraction by hand is error prone, work in [32] automates the abstraction process. It alters the actual system model by removing from it transitions that are not on the interface of the target subcircuit. Although the simplified model structure reduces the reachable state space, the approach is limited in the transitions it can remove, is not efficient on specifications with Boolean functions, and can produce nonexact circuits due to conservative timing behavior from the abstraction [32]. The work described in this paper does not alter the specification. It reduces the state space by exploring a single firing order on independent transitions not in the target subcircuit.

This paper is organized as follows. Section 2 introduces LPNs. Section 3 presents the new timed state space exploration algorithm. Section 4 formalizes modular synthesis. Section 5 describes a partial order reduction for synthesis, and proves that it is exact. The performance of the proposed method is evaluated in Section 6 using several examples including an experimental pipeline from IBM.

## 2 Level-Ruled Petri Nets

Each timed circuit module is specified using a level-ruled Petri net (LPN). An LPN is a pair  $M = (N, E)$  where  $N$  is an ordinary Petri net and  $E$  is the level-ruled extension. A Petri net is a four-tuple  $N = (T, P, F, \mu_o)$ .  $T$  is a set of transitions.  $P$  is a set of places.  $F \subseteq (T \times P) \cup (P \times T)$  is the flow relation. A *marking* is any subset of places. The initial marking of  $N$  is specified by  $\mu_o$ . For any transition  $t$ ,  $\bullet t = \{p \in P \mid (p, t) \in F\}$  and  $t\bullet = \{p \in P \mid (t, p) \in F\}$  denote the *source places* and the *destination places* of  $t$  respectively. *Source transitions* and *destination transitions* are defined similarly.

A level-ruled extension is a seven-tuple  $E = (I, O, \nu_o, \text{wire}, \text{Eft}, \text{Lft}, \text{Lsat})$ .  $I$  and  $O$  are finite sets of input and output wires, respectively.  $\nu_o$  is the initial Boolean state of the wires in  $O$ . Note that the initial value of a wire in  $I$  must be defined by some other module that produces transitions on that wire. The function  $\text{wire} : T \rightarrow (O \times \{+, -\}) \cup T$  is a mapping from transitions in  $N$  to transitions on wires in  $O$ . A transition on a wire  $u \in O$  can be rising ( $u+$ ) or falling ( $u-$ ) to change its Boolean state. If a transition is not a rising or falling transition on a wire, then the function returns the transition itself. Although such a transition does not affect the state of the output wires, it does affect the state of the net. Again wires in  $I$  are not mapped to transitions as input behavior must be defined by some other module. At times, the function  $\text{wirename}(t)$  may be used instead which returns the name of the wire associated with the event  $t$  or it returns the transition itself (i.e., if  $\text{wire}(t) = u+$ , then  $\text{wirename}(t) = u$ ; and if  $\text{wire}(t) = t$ , then  $\text{wirename}(t) = t$ ).  $R = F \cap (P \times T)$  is a set of rules. For any rule  $r = (p, t) \in R$ ,  $\bullet r = p$  and  $r\bullet = t$  are the *source place* and *destination transition* of  $r$  respectively. Let  $\mathbb{Q}^+$  be the set of nonnegative rational numbers.  $\text{Eft} : R \rightarrow \mathbb{Q}^+$  and  $\text{Lft} : R \rightarrow \mathbb{Q}^+ \cup \{\infty\}$  are functions that return the *earliest* and *latest* firing times of rules such that  $\text{Eft}(r) \leq \text{Lft}(r)$  for all  $r \in R$ .  $\text{Lsat} : R \rightarrow (\{0, 1\}^{|I \cup O|} \rightarrow \{\text{true}, \text{false}\})$  assigns a Boolean function to each rule. This paper restricts the Boolean functions to be simple conjunctions or disjunctions of signal values.

Fig. 1(a) shows an LPN that describes the behavior of an OR gate with inputs  $a$  and  $b$  and output  $c$ . This LPN includes two transitions on the output signal  $c$ , namely  $t_{c+}$  and  $t_{c-}$ . In the initial state, the signal  $c$  is low. There are two rules in this LPN. The rule from  $p_{c-}$  to  $t_{c+}$  is annotated with an earliest firing time of 6, latest firing time of 9, and a level expression of  $a \vee b$ . The rule from  $p_{c+}$  to  $t_{c-}$ , similarly, is annotated with a earliest firing time of 6 and latest firing time of 9, and an expression  $\neg a \wedge \neg b$ . Figs 1(b) and (c) show LPNs that describe the behavior of the two inverting gates that take  $c$  as an input and produce outputs  $a$  and  $b$ .

A timed circuit specification is defined by a collection of modules  $M = \{M_1, M_2, \dots, M_n\}$  where each module  $M_k$  is defined by an LPN,  $(N_k, E_k)$ . A collection of modules,  $M$ , must be *disjoint*, which means that for any  $i$  and  $j$  such that  $i \neq j$ ,  $T_i \cap T_j = \emptyset$ ,  $P_i \cap P_j = \emptyset$ , and  $O_i \cap O_j = \emptyset$ . A disjoint

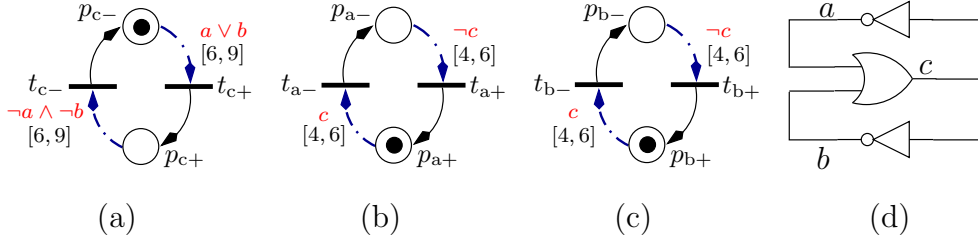


Fig. 1. (a) An OR gate with inputs  $a$  and  $b$  and output  $c$ . (b) An inverter with input  $c$  and output  $a$ . (c) An inverter with input  $c$  and output  $b$ . (d) The circuit synthesized (a), (b), and (c).

collection of modules can be represented using a single LPN of the following form:  $(\bigcup_{i=1}^n N_k, E')$  where each element of  $E'$  is simply the union over all the modules except  $I'$ , which equals  $\bigcup_{i=1}^n I_k - \bigcup_{i=1}^n O_k$ . In other words, a signal is no longer considered an input if it is an output in a constituent module. A collection of modules is *closed* when  $I' = \emptyset$ . The behavior of every wire is defined by some LPN in a closed set of modules. The remainder of this paper assumes that all circuits of interest are defined by a disjoint and closed collection of modules. This is a requirement of the analysis procedures in this paper. The implications of a closed system is that the output to input causality is known for each signal in the module. The modules shown in Figs 1(a), (b), and (c) form a disjoint and closed collection of LPNs. The synthesized circuit from this collection of modules is shown in Fig. 1(d).

A *timed state*  $\sigma$  of an LPN is a three-tuple  $(\mu, \nu, \text{clock})$  where  $\mu$  is a marking,  $\nu$  is a Boolean state, and  $\text{clock} : R \rightarrow \mathbb{Q}^+$  is a function that records the time a rule has been enabled. The *initial state*  $\sigma_o$  is  $(\mu_o, \nu_o, \text{clock}_o)$  where  $\mu_o$  is the initial marking,  $\nu_o$  is the initial Boolean state, and  $\text{clock}_o(r) = 0$  for all  $r \in R$ .

A rule  $r \in R$  is said to be *level-satisfied* in the state  $\nu$ , defined over  $O$ , if  $\text{Lsat}(r)(\nu) = \text{true}$ . A rule  $r$  is said to be *enabled* if its source place is in the marking ( $\bullet r \in \mu$ ), and it is level-satisfied in the Boolean state ( $\text{Lsat}(r)(\nu) = \text{true}$ ). Let  $R_{\text{en}}(\mu, \nu)$  return the set of enabled rules given  $\mu$  and  $\nu$ , and  $R(t) = \{r \in R \mid r \bullet = t\}$  return the set of rules for transition  $t$ . A transition is said to be enabled in the marking  $\mu$  at the Boolean state  $\nu$  if all of its rules are enabled ( $R(t) \subseteq R_{\text{en}}(\mu, \nu)$ ). Let  $\text{enabled}(\mu, \nu)$  be an ordered set of enabled transitions given  $\mu$  and  $\nu$ . For the initial state shown in Fig. 1, with  $a$  and  $b$  initially high and  $c$  low, the only enabled transition is  $t_{c+}$ . Transitions  $t_{b-}$  and  $t_{a-}$  are not enabled because  $c$  is low.

The state of an LPN can change if time passes or a transition fires. Time  $\tau \in \mathbb{Q}^+$  can pass in  $\sigma = (\mu, \nu, \text{clock})$  if for all  $t \in \text{enabled}(\mu, \nu)$  there exists a rule  $r \in R(t)$  such that  $\text{clock}(r) + \tau \leq \text{Lft}(r)$ . Note that a rule is said to *expire* if it exceeds its latest firing time. This formalism allows rules for enabled transitions to expire as long as a single rule exists for each transition that can accept  $\tau$  without expiring. The new timed state is derived from  $\sigma$  as  $\sigma' = (\mu, \nu, \text{clock}')$  where  $\mu$  and  $\nu$  are the marking and Boolean state from  $\sigma$ ;

and  $\text{clock}'(r) = \text{clock}(r) + \tau$  for all  $r \in R_{\text{en}}(\mu, \nu)$ . In the initial timed state for the example, time can be allowed to advance up to 9 time units. After 9 time units, the rule enabling  $t_{c+}$  expires; thus,  $t_{c+}$  is forced to fire.

The transition  $t_f \in T$  can fire in  $\sigma = (\mu, \nu, \text{clock})$  if two conditions hold: first,  $t_f \in \text{enabled}(\mu, \nu)$ ; and second, every  $r \in R(t_f)$  is *satisfied* (i.e.,  $\text{clock}(r) \geq \text{Eft}(r)$ ). The new state is derived from  $\sigma$  as  $\sigma' = (\mu', \nu', \text{clock}')$  where  $\mu' = (\mu - \bullet t_f) \cup t_f \bullet$ , for all  $u \in O$ ,

$$\nu'(u) = \text{update}(t_f, \nu)(u) = \begin{cases} 1 & \text{if } \text{wire}(t_f) = u+ \\ 0 & \text{if } \text{wire}(t_f) = u- \\ \nu(u) & \text{otherwise .} \end{cases}$$

and for all  $r \in R$ ,

$$\text{clock}'(r) = \begin{cases} \text{clock}(r) & \text{if } r \in (R_{\text{en}}(\mu, \nu) \cap R_{\text{en}}(\mu', \nu')) \\ 0 & \text{otherwise .} \end{cases}$$

In this formalism, the firing of a transition consumes no time. It only updates the marking, the Boolean state, and the clock function. In the case of firing  $t_f$  from  $\sigma$  above,  $\mu$  is updated to remove source and to add the destination places of  $t_f$  to get  $\mu'$ ;  $\nu'$  is the new Boolean state of the system after firing  $t_f$ , which is unchanged if  $\text{wire}(t_f) = t_f$ ; and rules that are still enabled in the new state keep their same clock values while all other rules are reset. A Petri net is *safe* if for each reachable marking from the initial state and for each transition  $t$  enabled in each reachable marking the following condition holds:  $(\mu - \bullet t) \cap t \bullet = \emptyset$ . A safe net does not have a marking in its reachable marking set that enables a transition that when fired, the marking update adds to the new marking a place that already exists in the current marking. In practice, this restriction has not impacted the type of systems that can be analyzed using this approach. Firing  $t_{c+}$  from the initial marking shown in Fig. 1, creates the new timed state where the marking is  $\{p_{c+}, p_{a+}, p_{b+}\}$ , the Boolean state is  $(1, 1, 1)$ , and the clock function remains unchanged except that  $\text{clock}((p_{c-}, t_{c+})) = 0$ .

### 3 Timed State Class Construction

A *timed state class* (TSC) is a finite representation of an infinite number of timed states. It is used to capture the infinite behaviors of an LPN. A TSC for an LPN  $M = (N, E)$  is a three-tuple  $s = (\mu, \nu, Z)$  where  $\mu$  is the marking,  $\nu$  is the current Boolean state of the wires, and  $Z$  is a set of relations called a zone that represents a set of clock functions [8]. A *relation* in  $Z$  has the form  $t_a - t_b \leq c$  where  $t_a$  and  $t_b$  are transitions from  $N$ , and  $c$  is a rational number. The relation is understood to mean that the time at which  $t_a$  fired minus the time at which  $t_b$  fired is less than or equal to  $c$ ; thus, it represents the time separation between  $t_a$  and  $t_b$ . A *timed state class transition* is the three-tuple

$(s, t, s')$  denoted as  $s \xrightarrow{t} s'$ , and it means that from the TSC  $s$ , transition  $t$  fires to move the system to the TSC  $s'$ . The TSCs are constructed such that for a timed state  $\sigma$  of  $M$ , if a transition  $t$  can fire from  $\sigma$  after passing some time  $\tau$  to obtain a new state  $\sigma'$ , then there exists a TSC transition  $s \xrightarrow{t} s'$  such that  $\sigma \in s$  and  $\sigma' \in s'$ . A *timed state class sequence* (TSCS) is a connected set of TSC transitions starting from the initial TSC of the system. This is written as  $\rho = s_o \xrightarrow{t_1} s_1 \xrightarrow{t_2} \dots s_{i-1} \xrightarrow{t_i} s_i \dots \xrightarrow{t_n} s_n$ . The notation  $s_{i-1} \xrightarrow{t_i} s_i \in \rho$  means that the timed state transition  $s_{i-1} \xrightarrow{t_i} s_i$  is found in the TSCS  $\rho$  as shown above. This notation is also used for a set of TSCSs.

Let  $\mathcal{T}(M)$  denote the set of all TSCSs allowed by  $M = (N, E)$ , a closed, disjoint LPN. This set is constructed starting from the initial TSC of  $M$ . Recall that  $N = (T, P, F, \mu_o)$  and  $E = (I, O, \nu_o, \text{wire}, \text{Eft}, \text{Lft}, \text{Lsat})$ . The initial TSC of  $M$  is  $s_o = (\mu_o, \nu_o, Z_o)$ .  $\mu_o$  and  $\nu_o$  are taken directly from  $M$ . The initial zone  $Z_o$  is a set of inequalities relating all the transitions that must have fired to create the initial marking. Let  $T(\mu_o)$  be the set of these transitions such that  $t \in T(\mu_o)$  if there exists  $p \in \mu_o$  such that  $t \in \bullet p$ . If this set contains multiple transitions, than one is randomly chosen to create the initial zone. The transitions in  $T(\mu_o)$  are assumed to have fired at the same time to create the initial marking; thus,  $Z_o$  contains a relation  $t_a - t_b \leq 0$  for all pairs of  $t_a$  and  $t_b$  where  $t_a \in T(\mu_o)$  and  $t_b \in T(\mu_o)$ .

The function  $\text{find}(M, s, \mathcal{T})$  in Algorithm 1 creates  $\mathcal{T}(M)$  using *bourne again POSET* (BAP) timing analysis. BAP implements a partial order reduction in the zones of each TSC. This reduces the number of TSCs needed to capture all the behaviors of the LPN [4,19,22,3]. The BAP algorithm, however, only supports simple conjunctive or disjunctive expressions. This means that any level expression can be a single conjunctive expression of any size, or it can be a disjunctive expression with a single wire in each conjunctive member. If an LPN contains a disjunctive expression with multiple wires in a conjunctive member of that expression, then the BAP analysis below is not exact. It can miss reachable TSCs. This paper is restricted to LPNs with this property.

The  $\text{find}(M, s, \mathcal{T})$  function is called with an LPN  $M$ ,  $s = s_o$ , and  $\mathcal{T} = \emptyset$  to start at the initial TSC, and it returns the set of all possible state transitions from which it is possible to construct  $\mathcal{T}(M)$ . The function  $\text{find}(M, s, \mathcal{T})$  and its various support functions are best understood by example. Consider again the system in Fig. 1. Assume that the function  $\text{find}(M, s, \mathcal{T})$  is started from the initial TSC of the system in Fig. 1; thus,  $s_0 = (\mu_0, \nu_0, Z_0)$  with  $\mu_0 = \{p_{a+}, p_{b+}, p_{c-}\}$ ,  $\nu_0 = (1, 1, 0)$  where the order is  $(a, b, c)$ ;  $Z_0 = \{0 \leq t_{a+} - t_{b+} \leq 0, 0 \leq t_{c-} - t_{a+} \leq 0, 0 \leq t_{c-} - t_{b+} \leq 0\}$ ;<sup>6</sup> and  $\mathcal{T} = \emptyset$ .

The function  $\text{find}(M, s, \mathcal{T})$  first calls the function  $\text{fireable}(M, s)$  shown in Algorithm 2. This function returns the set of transitions that can fire concurrently from  $s$ . This means that every transition  $t_f \in \text{fireable}(M, s)$  can fire

<sup>6</sup> The two inequalities  $t_a - t_b \leq c_1$  and  $t_b - t_a \leq c_2$  can be expressed as  $-c_1 \leq t_b - t_a \leq c_2$  where  $c_1$  and  $c_2$  are rational numbers. This transformation is used for all zones.

---

**Algorithm 1**  $\text{find}(M, s, \mathcal{T})$ 

---

```

for all  $t_f \in \text{fireable}(M, s)$  do
  for all  $s' \in \text{successor}(M, t_f, s)$  do
     $s' = \text{remove\_non\_causal}(M, s')$ 
    if  $((s \xrightarrow{t_f} s') \notin \mathcal{T})$  then
       $\mathcal{T} = \mathcal{T} \cup \{(s \xrightarrow{t_f} s')\}$ 
       $\mathcal{T} = \text{find}(M, s', \mathcal{T})$ 
    end if
  end for
end for
return  $\mathcal{T}$ 

```

---



---

**Algorithm 2**  $\text{fireable}(M, s)$ 

---

```

 $T_{\text{en}} = \text{enabled}(\mu(s), \nu(s))$ 
if  $(|T_{\text{en}}| = 1)$  then
  return  $T_{\text{en}}$ 
end if
 $T_f = \emptyset$ 
for all  $C_g \in \text{causal\_groupings}(T_{\text{en}})$  do
  for all  $C_a \in \text{causal\_assignments}(C_g, Z(s))$  do
     $Z = \text{set\_order}(C_g, \text{causal\_groupings}(T_{\text{en}}), Z(s))$ 
     $Z = \text{set\_min\_separations}(C_g, T_{\text{en}}, Z)$ 
     $Z = \text{set\_max\_separations}(C_a, T_{\text{en}}, Z)$ 
    if  $(\text{valid}(Z) = \text{true})$  then
       $T_f = T_f \cup \text{can\_fire\_first}(T_{\text{en}}, Z)$ 
      if  $(T_f = T_{\text{en}})$  then
        return  $T_f$ 
      end if
    end if
  end for
end for
return  $T_f$ 

```

---

before all other transitions in  $\text{fireable}(M, s)$ . The function begins by deriving the set of enabled transitions in  $s$ . The helper functions  $\mu(s)$  and  $\nu(s)$  return the marking and Boolean state respectively from the TSC  $s = (\mu, \nu, Z)$  that is passed in on the parameter list.<sup>7</sup> As  $\text{fireable}(M, s)$  is called with  $s = s_o$  in this example,  $T_{\text{en}} = \{t_{c+}\}$ . At this point the function returns the enabled set  $T_{\text{en}}$  to  $\text{find}(M, s, \mathcal{T})$  as it contains a single enabled transition  $t_{c+}$ .

The function  $\text{find}(M, s, \mathcal{T})$  then computes all successor TSCs that result from firing  $t_{c+}$  by calling the function  $\text{successor}(M, t_{c+}, s)$  shown in Algo-

---

<sup>7</sup> This notation is a type for how members are referenced in a tuple when only a symbol is presented and is used throughout the rest of this presentation.

**Algorithm 3**  $\text{successor}(M, t_f, s)$ 


---

```

 $\mathcal{S} = \emptyset$ 
 $\mu = (\mu(s) - \bullet t_f) \cup t_f \bullet$ 
 $\nu = \text{update}(t_f, \nu(s));$ 
for all  $c_g \in \text{causal\_groupings}(t_f)$  do
  for all  $t_c \in \text{causal\_assignments}(c_g, Z(s))$  do
     $Z = \text{set\_order}(c_g, \text{causal\_groupings}(t_f), Z(s))$ 
     $Z = \text{set\_min\_separations}(c_g, t_f, Z)$ 
     $Z = \text{set\_max\_separations}(t_c, t_f, Z)$ 
    if  $(\text{valid}(Z) = \text{true})$  then
       $\mathcal{S} = \mathcal{S} \cup \{(\mu, \nu, Z)\}$ 
    end if
  end for
end for
return  $\mathcal{S}$ 

```

---

Algorithm 3. The function begins by creating an empty set  $\mathcal{S}$  to hold the successor TSCs of  $s$  and by updating the marking and Boolean state to reflect the firing of  $t_{c+}$ . The function then begins to consider successor TSCs from  $s$  by looking at the causal assignments to  $t_{c+}$  in each of its causal groupings. Each of the causal assignments in each of the causal grouping can potentially create a unique successor TSC to  $s$ .

A *causal grouping* for a transition  $t$  is a set of transitions that must have fired for  $t$  to be enabled. For a transition  $t_c$  to be in a causal grouping for  $t$ , it must be either place or level causal to  $t$ . A transition  $t_c$  is said to be *place-causal* to  $t$  if  $t_c$  is directly connected to  $t$ , which means that there exists a place  $p \in \bullet t$  such that  $t_c \in \bullet p$ . A transition  $t_c$  is said to be *level-causal* to  $t$ , a transition that is enabled in the TSC  $s$ , if  $t_c$  is a transition that causes a rule for  $t$  to be level-satisfied, which means that there exists a rule  $r \in R(t)$  and a transition  $t' \in T$  such that  $\text{wirename}(t') = \text{wirename}(t_c)$ ,  $\text{wire}(t') \neq \text{wire}(t_c)$ ,  $\text{Lsat}(r)(\nu(s)) = \text{true}$ , and  $\text{Lsat}(r)(\nu') = \text{false}$  where  $\nu' = \text{update}(t', \nu(s))$ . Let  $\text{causal\_groupings}(t)$  be the set of all causal groupings for transitions  $t$ . A transition can have multiple causal groupings if it is connected to any place that has two source transitions or if it has a disjunctive level expression on any of its rules. The function  $\text{causal\_groupings}(t_{c+})$  in this example returns two causal groupings for  $t_{c+}$ :  $\{t_{c-}, t_{a+}\}$  and  $\{t_{c-}, t_{b+}\}$ . The disjunctive level expression  $a \vee b$  on its rule creates the two groupings. The causal groupings imply that either  $t_{c-}$  and  $t_{a+}$  or  $t_{c-}$  and  $t_{b+}$  are required to fire  $t_{c+}$ .

A *causal assignment* to a causal grouping for a transition  $t$  is the selection of a transition in the causal grouping that is used to determine the separation between  $t$  and the other transitions in the causal grouping. In the LPN semantics, an enabled transition  $t$  can fire as soon as each of its rules is satisfied, which means that each of the rules  $r \in R(t)$  has been enabled at least  $\text{Eft}(r)$  time units. When this condition is met, then the latest time that transition  $t$

can fire is at the latest firing time of one of its rules because it must fire before all of them expire. This means that for any rule  $r \in R(t)$ , transition  $t$  could fire up to  $\text{Lft}(r)$  time units from becoming enabled; thus, a causal assignment  $t_c$  to  $t$  implies:

- (i) the firing of  $t_c$  caused a rule in  $r \in R(t)$  to become enabled either through a level or a place; and
- (ii) the firing of  $t$  happens up to  $\text{Lft}(r)$  time units after  $t_c$ .

Let  $\text{causal\_assignments}(c_g, Z)$  be the set of transitions from  $c_g$  such that each transition exists in a relation in the zone  $Z$ . If a transition for a potential causal assignment is not found in any relation  $Z$ , then it is assumed to not be causal in  $Z$ ; thus, it is not considered. Given that the initial zone  $Z$  contains relations for all transitions, there are two causal assignments for each grouping  $\{t_{c-}, t_{a+}\}$  and  $\{t_{c-}, t_{b+}\}$  in the example. All of these causal assignments must be considered since each valid one can create a unique successor state from  $s$ .

A causal grouping and assignment determine the orders and separations amongst transitions in the zone, as well as the transitions being added to the zone. Let  $c_g$  be a causal grouping for an enabled transition  $t$ , and let  $Z$  be a zone. The  $\text{set\_order}(c_g, \text{causal\_groupings}(t_f), Z)$  function changes relations in  $Z$  to order the level-causal transitions of  $c_g$  to occur before the level-causal transitions in the other causal groupings of  $t_f$ . This ordering ensures that each grouping is actually causal in  $Z$ . This ordering is not required for place-causal transitions because of the one-safe property of the LPNs. If there exists a place in an LPN that has multiple source transitions creating multiple causal groupings, then only one of those source transitions exists in the zone for the LPN to be one-safe.

For the example of  $t_{c+}$  firing from the initial state, let the causal grouping  $c_g$  be  $\{t_{c-}, t_{a+}\}$ . In this case, the  $\text{set\_order}$  function does not need to change any relations in  $Z$  to order the level-causal transitions  $t_{a+}$  and  $t_{b+}$  because all transitions fire at the same time to create the initial zone; thus, both groupings enable  $t_{c+}$  at the same time. If the zone  $Z$ , however, were given as  $\{-2 \leq t_{b+} - t_{a+} \leq 2\}$ , then the call to  $\text{set\_order}$  would change the zone  $Z$  to  $\{0 \leq t_{b+} - t_{a+} \leq 2\}$ , which means that  $t_{b+}$  always fires between zero and two time units after  $t_{a+}$ ; thus, the  $\{t_{c-}, t_{a+}\}$  grouping always causes  $t_{c+}$  to become enabled from this zone. The  $\text{set\_order}$  function only changes relations in the zone. It does not add any new relations. If a transition does not exist in any relation in the zone, then it is not ordered in the zone by  $\text{set\_order}$ .

The earliest that a transition can fire from a zone is when all of its rules are satisfied. In the example, the earliest that  $t_{c+}$  can fire is six time units after  $t_{c-}$  has fired and its expression  $a \vee b$  evaluates to true; thus, there is at least a six time unit separation between the firing of  $t_{c+}$  and  $t_{c-}$ ,  $t_{a+}$ , and  $t_{b+}$ . If there had been another rule feeding into the  $t_{c+}$  transition, then any transition associated with that rule would also set a minimal separation on  $t_{c+}$ . In the LPN semantics, a transition cannot fire unless its rules are

all either satisfied or expired. Let  $c_g$  be a causal grouping for an enabled transition  $t$ , and let  $Z$  be a zone. The function `set_min_separations`( $c_g, t, Z$ ) adds the necessary relations to  $Z$  to force the firing of  $t$  to be delayed until each of its rules are satisfied. Consider again the example of firing  $t_{c+}$  from the initial zone  $Z = \{0 \leq t_{a+} - t_{b+} \leq 0, 0 \leq t_{c-} - t_{a+} \leq 0, 0 \leq t_{c-} - t_{b+} \leq 0\}$ . Let the causal grouping  $c_g$  be  $\{t_{c-}, t_{a+}\}$ . The `set_min_separations`( $c_g, t_{c+}, Z$ ) function adds two relations to  $Z$ :  $6 \leq t_{c+} - t_{c-}$  and  $6 \leq t_{c+} - t_{a+}$ . These relations set the separation between  $t_{c+}$  and the two transitions  $t_{c-}$  and  $t_{a+}$  to be at least six time units according to the rule for  $t_{c+}$ ; thus, regardless of the causal assignment to  $t_{c+}$  in the causal grouping, the zone contains relations to satisfy its rule. If the zone  $Z$ , however, is given as  $\{-2 \leq t_{b+} - t_{a+} \leq 2\}$ , then `set_min_separations`( $c_g, t_{c+}, Z$ ) will only add the  $6 \leq t_{c+} - t_{a+}$  relation to  $Z$  because the  $t_{c-}$  transition is not in  $Z$ . When a transition from a causal grouping is missing in the zone, then no relations are added for that transition.

The latest time that a transition can fire from a zone is before all of its rules expire. This means that a transition can potentially fire at the latest firing time of any of its rules. Let  $t_c$  be the causal assignment for an enabled transition  $t$ , and let  $Z$  be a zone. The `set_max_separations`( $t_c, t, Z$ ) function adds a relation to  $Z$  to set the latest firing of  $t$  relative to  $t_c$  according to the latest firing time on the rule associated with  $t$  and  $t_c$ . Consider the example of firing  $t_{c+}$  from the initial zone  $Z = \{0 \leq t_{a+} - t_{b+} \leq 0, 0 \leq t_{c-} - t_{a+} \leq 0, 0 \leq t_{c-} - t_{b+} \leq 0\}$  in the `successor`( $M, t_{c+}, s$ ) function. Let the causal grouping  $c_g$  be  $\{t_{c-}, t_{a+}\}$  and the causal assignment be  $t_{a+}$ . The `set_max_separations`( $t_{a+}, t_{c+}, Z$ ) function adds the relation  $t_{c+} - t_{a+} \leq 9$  to  $Z$ . The final state of  $Z$  in `successor`( $M, t_{c+}, s$ ) for the given causal group and assignment after `set_order`, `set_min_separations`, and `set_max_separations` is

$$\left\{ \begin{array}{l} 0 \leq t_{a+} - t_{b+} \leq 0, 0 \leq t_{c-} - t_{a+} \leq 0, \\ 0 \leq t_{c-} - t_{b+} \leq 0, 6 \leq t_{c+} - t_{c-} \leq \infty, \\ 6 \leq t_{c+} - t_{a+} \leq 9, \end{array} \right\}.$$

If the function `successor`( $M, t_{c+}, s$ ), however, is called from the initial marking and Boolean state, but with the zone  $Z = \{-2 \leq t_{b+} - t_{a+} \leq 2\}$ , then the final state of  $Z$  after `set_order`, `set_min_separations`, and `set_max_separations` for the same causal group and assignment is  $\{0 \leq t_{b+} - t_{a+} \leq 2, 6 \leq t_{c+} - t_{a+} \leq 9\}$ .

The `set_max_separations`( $t_c, t_f, Z$ ) function is conservative under timed dependent choice semantics. *Time dependent choice* (TDC) semantics resolve conflict through time. Consider the LPN shown in Fig. 2(a) where  $t_1$  conflicts with  $t_3$ . The event  $t_3$  never fires in this LPN. The state reached after  $t_2$  is fired cannot time transition more than 9 time units without being forced to fire  $t_1$ ; thus, the LPN semantics implement TDC resolution by default. Consider now the choice construct in Fig. 2(b). In this construct, both  $t_1$  and  $t_3$  are again enabled, only this time the firing of  $t_3$  can never happen at the latest firing time of its rule. This is because  $t_1$  is forced to fire before its rule expires, and

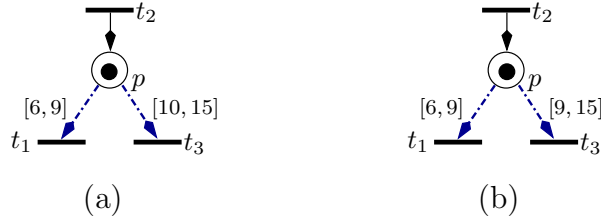


Fig. 2. A timing that allows (a) only  $t_1$  to fire and (b) both  $t_1$  and  $t_3$  to fire.

the  $\text{Lft}(t_1) \leq \text{Lft}(t_3)$ . In this case,  $\text{set\_max\_separations}(t_2, t_3, Z)$  will allow  $t_3$  to fire at its latest firing time; thus, it is conservative.

Invalid causal assignments are identified when the zone resulting from the added ordering relations and separations for that causal grouping and assignment is put into its canonical form. The canonical form for a zone is obtained by applying a shortest-path algorithm to the graph structure implied by its set of relations where each transition appearing in any relation of  $Z$  is a node of the graph, and the nodes are connected by their defined separations as weighted directed edges. If two nodes in the zone are not related by an inequality, then their separation is infinite. A zone is said to be *valid* if no negative weight cycles exist in the graph structure implied by its set of relations. Let the function  $\text{valid}(Z)$  put the zone  $Z$  in its canonical form and return `true` if its canonical form is valid, otherwise it returns `false`. If a causal assignment to  $t$  prevents a rule  $r \in R(t)$  from being enabled for at least  $\text{Eft}(r)$  time units, then  $\text{valid}(Z)$  will return `false` because a negative weight cycle will exist. The  $\text{successor}(M, t_f, s)$  function adds to the set  $\mathcal{S}$  all valid states from all causal groupings and assignments for  $t_f$ .

The function  $\text{successor}(M, s, t_{c+})$  called from the initial state returns a single successor state from firing  $t_{c+}$ . This state is given as  $s' = (\mu', \nu', Z')$  where  $\mu'$  is  $(p_{a+}, p_{b+}, p_{c+})$ ,  $\nu'$  is  $(1, 1, 1)$ , and  $Z$  is

$$\left\{ \begin{array}{l} 0 \leq t_{a+} - t_{b+} \leq 0, 0 \leq t_{c-} - t_{a+} \leq 0, \\ 0 \leq t_{c-} - t_{b+} \leq 0, 6 \leq t_{c+} - t_{c-} \leq 9, \\ 6 \leq t_{c+} - t_{a+} \leq 9, 6 \leq t_{c+} - t_{b+} \leq 9 \end{array} \right\}.$$

All causal groupings and assignments lead to this state because  $t_{a+}$ ,  $t_{b+}$ , and  $t_{c-}$  fired at the same time in the initial state. The zone in this new state, however, contains relations that are not important to successor states derived from firing transitions in this new state. As an example, any relation on the firing time of  $t_{c-}$  is no longer useful as it is not used by any transition enabled in the new state. The number of TSCs required to capture all behaviors of an LPN can be reduced if unnecessary relations are removed from the zone in each successor state. The function  $\text{remove\_non\_causal}(M, s')$  in  $\text{find}(M, s, \mathcal{T})$  deletes relations from the zone in  $s'$  that are no longer required to derive new future states from  $s'$ .

Consider the new state  $s' = (\mu, \nu, Z)$  generated from firing  $t_{c+}$  from the

initial state of the LPN in Fig. 1. The function `remove_non_causal`( $M, s'$ ) in this example removes from  $Z(s')$  the relations for the transitions  $t_{c-}$ ,  $t_{a+}$ , and  $t_{b+}$ . The new zone  $Z'$  in  $s'$  is  $\{0 \leq t_{c+} - t_{c-} \leq 0\}$ . The relations involving  $t_{c-}$  are removed because  $t_{c-}$  is not needed in a causal grouping for any enabled event. The relations involving  $t_{a+}$  and  $t_{b+}$  are removed because the relations in  $Z'$  indicate that  $t_{a+}$  and  $t_{b+}$  can never be causal to  $t_{a-}$  and  $t_{b-}$  respectively. This is because  $t_{c+}$  always fires at least six time units after  $t_{a+}$  and  $t_{b+}$ ; thus, the rules for  $t_{a-}$  and  $t_{b-}$  can never become enabled due to the firing of  $t_{a+}$  or  $t_{b+}$  respectively.

The new TSC from firing  $t_{c+}$  is added to  $\mathcal{T}$  in `find` and the recursive call is made to compute the set of fireable events from this new TSC. The  $T_{\text{en}}$  set in this case is  $\{t_{a-}, t_{b-}\}$ ; thus, the function must explore the causal groupings and assignments for these transitions to find those that can fire concurrently from  $s$ . The functions from Algorithm 3 are extended to sets to derive this set of transitions. For an ordered set of transitions,  $T_{\text{en}} = (t_1, t_2, \dots, t_n)$ , let `causal_groupings`( $T_{\text{en}}$ ) be the set of all possible combinations of causal groupings for transitions in  $T_{\text{en}}$ , which means that `causal_groupings`( $T_{\text{en}}$ ) =  $\prod_{t \in T_{\text{en}}}(\text{causal\_groupings}(t))$  where  $\prod$  is the Cartesian product; therefore, If  $T_{\text{en}} = (t, u)$  and `causal_groupings`( $t$ ) =  $\{\{a_1, a_2\}, \{b_1\}\}$  and `causal_groupings`( $u$ ) =  $\{\{c_1, c_2\}, \{d_1\}\}$ , then

$$\begin{aligned} \text{causal\_groupings}(T_{\text{en}}) = & \{(\{a_1, a_2\}, \{c_1, c_2\}), (\{a_1, a_2\}, \{d_1\}), \\ & (\{b_1\}, \{c_1, c_2\}), (\{b_1\}, \{d_1\})\}; \end{aligned}$$

thus, every combination of causal groupings is considered. For  $T_{\text{en}} = (t_{a-}, t_{b-})$  in this example, `causal_groupings`( $T_{\text{en}}$ ) is  $\{(\{t_{a+}, t_{c-}\}, \{t_{b+}, t_{c-}\})\}$  because there is a single causal grouping for  $t_{a-}$  and  $t_{b-}$ . The  $C_g$  set is one combination in `causal_groupings`( $T_{\text{en}}$ ). The function `causal_assignments`( $C_g, Z(s)$ ) returns all combinations of causal assignments to the combination of causal groupings in  $C_g$ . For  $T_{\text{en}} = (t, u)$ ,  $C_g = (\{a_1, a_2\}, \{c_1, c_2\})$ , and  $Z$  containing relations for all of the transitions in  $C_g$ , `causal_assignments`( $C_g, Z$ ) =  $\{(a_1, c_1), (a_1, c_2), (a_2, c_1), (a_2, c_2)\}$ . For  $T_{\text{en}} = (t_{a-}, t_{b-})$ , its single causal grouping, and the zone containing only  $t_{c+}$ , `causal_assignments`( $C_g, Z(s)$ ) =  $(t_{c+}, t_{c+})$  because  $t_{a+}$  and  $t_{b+}$  are not found in  $Z$ . The `set_order`, `set_min_separations`, and `set_max_separations` functions are extended in a similar manner, only these functions do not consider all combinations. Rather, they can be defined to consider each member of the  $T_{\text{en}}$  set separately; thus, the `set_order` function orders the level-causal signals in its causal grouping in  $C_g$  to occur before the level-causal signals in all of its other causal groupings. The `set_min_separations` and `set_max_separations` functions follow similarly. The resulting valid zone in its canonical form for this example after `set_order`, `set_min_separations`, and `set_max_separations` is  $\{4 \leq t_{a-} - t_{c+} \leq 6, 4 \leq t_{b-} - t_{c+} \leq 6, -2 \leq t_{a-} - t_{b-} \leq 2\}$ .

The function `can_fire_first`( $T_{\text{en}}, Z$ ) in Algorithm 2 returns the set of transitions from  $T_{\text{en}}$  that can fire concurrently from  $Z$ . This set is derived from the relations in  $Z$ . For each transition  $t_f \in T_{\text{en}}$ , the `can_fire_first` function examines all relations  $t - t_f \leq c$  in  $Z$  where  $t \in T_{\text{en}}$ . If  $c \geq 0$  in each of these relations,

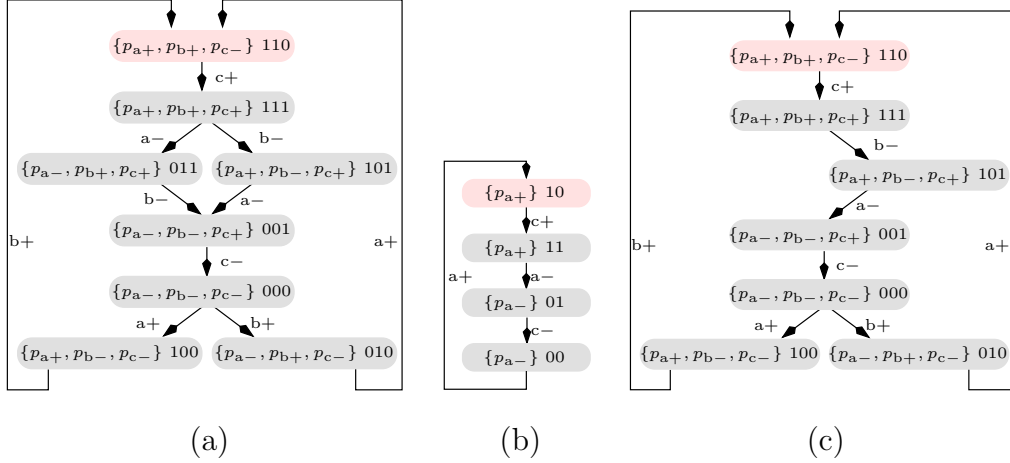


Fig. 3. (a) The set of TSCs found by regular synthesis. (b) The reduced set of TSCs for Fig. 1(b). (c) The set of TSCs found by modular synthesis.

then  $t_f$  can fire before all other enabled transitions; thus,  $t_f$  is added to the return set. For this example,  $\text{can\_fire\_first}(T_{\text{en}}, Z)$  returns  $\{t_{a-}, t_{b-}\}$  because of the  $-2 \leq t_{a-} - t_{b-} \leq 2$  relations denoting that  $t_{a-}$  and  $t_{b-}$  can happen in either order within two time units of each other. The  $\text{fireable}(M, s)$  function adds this set to the set of concurrently fireable events and returns to  $\text{find}(M, s, \mathcal{T})$ . Although this example contains a single combination for its causal grouping and assignment, other examples have many combinations; thus, if  $T_f = T_{\text{en}}$  the algorithm returns  $T_f$  to stop the exploration process as shown above.

Applying  $\text{find}(M, s, \mathcal{T})$  to the collection of modules shown in Fig. 1 finds eight TSCs with ten TSC transitions. The reachable states are shown in Fig. 3(a) with the timing information omitted.

## 4 Synthesis of Timed Circuits

Consider a set  $M = \{E, S_1, \dots, S_i, \dots, S_n\}$  of modules which is closed, where  $E$  is the environment of the whole circuit and  $S_i$  is a specification of the  $i$ -th subcircuit. Let  $S_i = (N_i, E_i)$ . The goal of modular synthesis is to synthesize a subcircuit specified by  $S_i$ . Note that this method expects that the synthesized subcircuit works correctly with respect to  $S_i$  in  $M$ . It may, however, work incorrectly with respect to  $S_i$  in a different module set; thus, the synthesis procedure depends on the behavior of the other modules  $E, S_1, \dots, S_n$ .

For a set  $\mathcal{T}$  of TSCs and a module  $S_i$ , a *reduced state graph* is as follows:

$$\begin{aligned} \text{rsg}(\mathcal{T}, S_i) = & \{(\text{proj}(s, S_i), t, \text{proj}(s', S_i)) \\ & \mid t \in T_i \vee \text{wirename}(t) \in I_i, s \xrightarrow{t} s' \in \mathcal{T}\} \end{aligned}$$

where  $\text{proj}(s, S_i)$  results in an untimed state that includes a marking with only places from transitions in  $S_i$  and a state vector composed of only the values of input and output wires in  $S_i$  (i.e.,  $\text{proj}(s, S_i) = (\mu \cap P_i, \nu|_{I_i \cup O_i})$  for

$s = (\mu, \nu, Z)$ ). This represents the set of untimed state transitions projected to  $S_i$  that occur in  $\mathcal{T}$ , and change the values of wires in  $S_i$ ; thus,  $\text{rsg}(\mathcal{T}, S_i)$  specifies a state graph of  $S_i$  the does not include any timing information. If  $S_i$  is selected to be the module shown in Fig. 1(b), then applying  $\text{rsg}(\mathcal{T}, S_i)$  to the state graph in Fig. 3(a) would result in the state graph in Fig. 3(b).

Let  $\mathcal{T}(M)$  denote the set of TSCSs of  $M$  such that every TSCS starts from the initial TSC  $s_0$  of  $M$ . A modular synthesis with respect to  $S_i$  in  $M$  is done by applying a synthesis algorithm for timed circuits in [21], denoted by *synthesis*, to  $\text{rsg}(\mathcal{T}(M), S_i)$ . Hence,  $C_i = \text{synthesis}(\text{rsg}(\mathcal{T}(M), S_i))$  is our goal.

If we consider  $\{S_1, \dots, S_n\}$  to be one module and assume that there are no internal signals in the environment, it is equivalent to the non-modular synthesis method. In modular synthesis, the state graphs (i.e.,  $\text{rsg}(\mathcal{T}(M), S_i)$ ) are much smaller than those for non-modular synthesis, because many variables are projected out in  $\text{rsg}(\mathcal{T}(M), S_i)$ . In order to obtain  $\mathcal{T}(M)$ , however, this method has to generate the full state space of the system, even if one subcircuit  $S_i$  is being considered at a time; thus, as long as this method uses  $\mathcal{T}(M)$ , there is no advantage to modular synthesis. This paper proposes another approach to generate a reduced state space  $\mathcal{T}_{S_i}(M)$  by means of a partial order reduction. The advantages of the partial order approach are two-fold: first, concurrent orders of invisible behavior are not considered; and second, an optimal circuit is always synthesized. The disadvantage of the partial order approach is that a portion of the invisible behavior still needs to be considered. Note that it is possible to apply the abstraction technique and then to apply the partial order reduction.

## 5 Partial Order Reduction

This section extends the partial order reduction algorithm shown in [4,19,28,31] so that it can handle LPNs for synthesis of timed circuits.

$\mathcal{T}(M)$  is obtained by firing every possible output transition from every reachable timed state class. This *total order exploration* algorithm often suffers from the state explosion problem. A *partial order exploration* algorithm generates a set of reduced TSCSs with respect to  $S_i$  that still has sufficient information to synthesize a correct circuit for  $S_i$ .

In order to explain the proposed idea more formally, this section defines *project*. For a TSCS  $s \xrightarrow{t} s' \xrightarrow{t'} \dots$  with  $s = (\mu, \nu, Z)$  and a specification  $S_i$ ,

$$\text{project}(s \xrightarrow{t} s' \xrightarrow{t'} \dots, S_i) = \begin{cases} \text{proj}(s, S_i) \xrightarrow{t} Y & \text{if } t \in T_i \vee \text{wirename}(t) \in I_i \\ Y & \text{else} \end{cases},$$

where  $Y = \text{project}(s' \xrightarrow{t'} \dots, S_i)$ . This definition can also be extended for a set of TSCSs. The *project* function removes from  $\mathcal{T}(M)$  any transition not

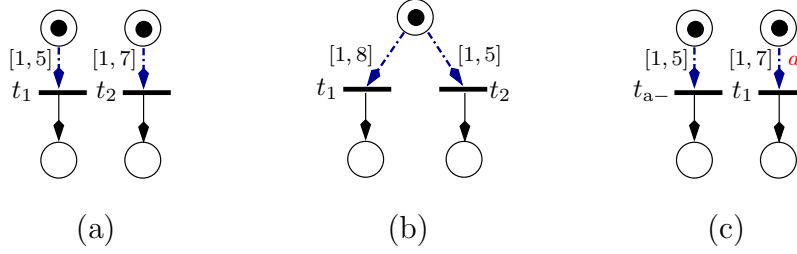


Fig. 4. An LPN fragment where  $t_1$  and  $t_2$  are (a) independent, (b) in place-conflict, and (c) in level-conflict.

related to  $S_i$ . It is subtly different from the  $\text{rsg}(\mathcal{T}(M), S_i)$  function in that it preserves timing information where the other removes timing information to get the untimed subset of  $\mathcal{T}(M)$ .

Now, for a given set  $M$  of modules and a module  $S_i$  for the specification of the target subcircuit, modular synthesis constructs a set  $\mathcal{T}_{S_i}(M)$  of reduced TSCSs such that the following proposition holds.

**Proposition 5.1**  $\text{project}(\mathcal{T}(M), S_i) = \text{project}(\mathcal{T}_{S_i}(M), S_i)$ .

If this proposition holds, then the following theorem can be proven showing the proposed method gives the same solution as the method in Section 3.

**Theorem 5.2**  $\text{rsg}(\mathcal{T}(M), S_i) = \text{rsg}(\mathcal{T}_{S_i}(M), S_i)$ .

**Proof.** Suppose  $(\text{proj}(s, S_i), t, \text{proj}(s', S_i)) \in \text{rsg}(\mathcal{T}(M), S_i)$ ; thus, there exists a TSCS  $\rho = s_0 \xrightarrow{t_1} \dots s \xrightarrow{t} s' \dots$  in  $\mathcal{T}(M)$  such that either  $t \in T_i$  or  $\text{wirename}(t) \in I_i$ . From Property 5.1, there also exists a TSCS  $\rho'$  in  $\mathcal{T}_{S_i}(M)$  such that  $\text{project}(\rho, S_i) = \text{project}(\rho', S_i)$ ; thus,  $(\text{proj}(s, S_i), t, \text{proj}(s', S_i)) \in \text{rsg}(\mathcal{T}_{S_i}(M), S_i)$ . The other direction can be proven similarly.  $\square$

Next, this section shows how to construct  $\mathcal{T}_{S_i}(M)$  such that Property 5.1 holds. The idea of partial order reduction is to prune some successor timed state classes during state space exploration. This means that some firing orders between transitions are omitted. If some firing orders between transitions in  $T_i$  are missed, however, then Property 5.1 does not hold; thus, modular synthesis has to guarantee that every possible firing order between transitions on signals in  $S_i$  is generated. Let function  $\text{visible}(S_i)$  return the set of these transitions.

$$\text{visible}(S_i) = \{t \mid t \in T_i \vee \text{wirename}(t) \in I_i\}$$

Consider the simple example net shown in Fig. 4(a). If  $t_1$  and  $t_2$  are both visible transitions, then both orderings are important and ignoring one of them would result in an incorrect circuit.

Suppose that  $t_1$  and  $t_2$ , which are outside  $S_i$ , are in *conflict* (i.e., only  $t_1$  or  $t_2$  can occur). If firing  $t_1$  is missed in  $\mathcal{T}_{S_i}(M)$ , then the behavior of  $S_i$  that is caused by the descendant of  $t_1$  may be missed too. This results in a wrong

subcircuit. The function  $\text{conflict}_p(t)$  returns the *place-conflicts* for  $t$ .

$$\text{conflict}_p(t) = \{t' \mid \bullet t \cap \bullet t' \neq \emptyset, t' \neq t\}$$

An example of a place conflict is shown in Fig. 4(b). The firings of  $t_1$  and  $t_2$  must be interleaved or behaviors of  $S_i$  may be missed. This can result in an incorrect circuit. For the LPN in Fig. 4(b),  $\text{conflict}_p(t_1) = \{t_2\}$  and  $\text{conflict}_p(t_2) = \{t_1\}$ .

A level expression may also create conflict. If the firing of a transition  $t_1$  could disable a level expression on another transition  $t_2$ , then the algorithm must consider ordering the firings of  $t_1$  and  $t_2$ . If the firing of a transition  $t_1$  can be disabled by another transition  $t_2$ , then they must also be interleaved. The function  $\text{conflict}_l(t)$  returns the set of *level-conflicts* for transition  $t$ .

$$\begin{aligned} \text{conflict}_l(t) = \{t' \mid \exists r . (t' \in r \bullet \wedge \text{disable}(\text{Lsat}(r), t)) \\ \vee (t \in r \bullet \wedge \text{disable}(\text{Lsat}(r), t')) \vee \text{disjunctive}(\text{Lsat}(r), t, t'))\} \end{aligned}$$

where  $\text{disable}(\text{Lsat}(r), t)$  returns **true** if firing transition  $t$  can potentially disable the Boolean function  $\text{Lsat}(r)$ , and  $\text{disjunctive}$  returns **true** if  $t$  and  $t'$  appear in different product terms of a disjunctive expression. This is necessary in order to consider all possible transitions that determine the latest firing time of  $t$ . An example of a level conflict is shown in Fig. 4(c). The firings of  $t_{a-}$  and  $t_1$  must be interleaved to produce the correct circuit; thus,  $\text{conflict}_l(t_1) = \{t_{a-}\}$  and  $\text{conflict}_l(t_{a-}) = \{t_1\}$ .

Thus, for a TSCS  $\rho$  in  $\mathcal{T}(M)$ , we have to guarantee that  $\mathcal{T}_{S_i}(M)$  includes  $\rho'$  such that the same set of transitions eventually fire in both  $\rho$  and  $\rho'$ . This is guaranteed by the combination of these three requirements summarized in the definition of the following set  $\text{conflict}(t)$  as the set of transitions that are visible or in place or level-conflict with  $t$ .

$$\text{conflict}(t) = \text{visible}(S_i) \cup \text{conflict}_p(t) \cup \text{conflict}_l(t)$$

The partial order algorithm for modular synthesis generates only one firing order for transitions not in the conflict set, and every possible firing order for transitions in the conflict set. Unfortunately, it is not quite this simple. Consider the net in Fig. 5(a). In this case,  $t_3$  is not enabled; thus, only  $t_1$  can fire and not  $t_3$ . The firing sequence starting from  $t_2$ , however, can enable  $t_3$  to fire if the firing of  $t_1$  is postponed; thus, if  $t_1$  is fired alone, then the possibility of the firing of  $t_3$  is missed. Modular synthesis must therefore interleave  $t_1$  and  $t_2$  to find all behaviors of  $S_i$ . This case must also be considered for level-conflict too. Consider the LPN in Fig. 5(b). In this case,  $t_1$  and  $t_{a-}$  are in level-conflict, but  $t_{a-}$  is not enabled. To enable  $t_{a-}$ ,  $t_3$  must first fire, but  $t_3$  is not enabled because  $c$  is low in the current state. The transition  $t_{c+}$  must therefore fire to enable  $t_3$ . To enable  $t_{c+}$ , however, the algorithm must first fire  $t_2$ ; thus  $t_1$  must be interleaved with  $t_2$  to see if it is possible that it can start a chain reaction resulting in the firing of  $t_{a-}$  before  $t_1$ .

In order to handle these cases in general, if the method wants to fire a transition  $t$  at a timed state class  $s$ , it must compute  $\text{dependent}(s, t)$ . It

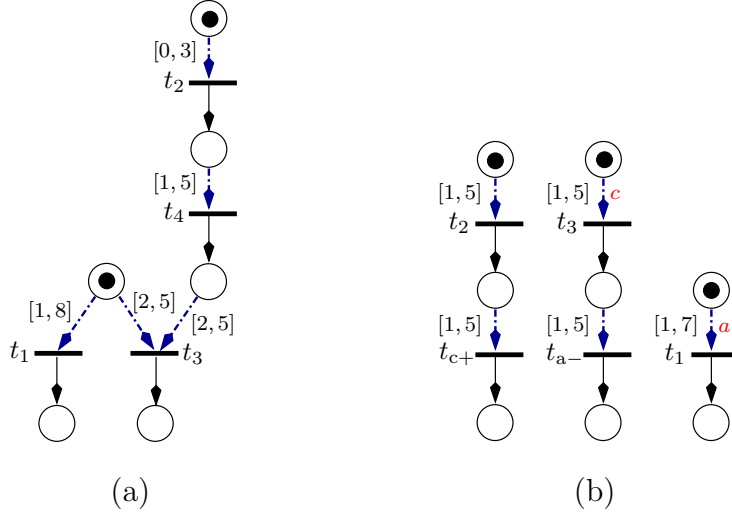


Fig. 5. (a) An LPN fragment where  $t_1$  and  $t_3$  are in place-conflict. (b) An LPN fragment where  $t_{a-}$  and  $t_1$  are in level-conflict.

is a set of enabled transitions such that the interleavings of the firings of those transitions should be generated for the correct results. For example,  $\mathbf{dependent}(s, t_1) = \{t_1, t_2\}$  for Figs 5(a) and (b).

A transition  $t$  is called *firable* in a timed state class  $s$ , if for some timed state class  $s'$ ,  $s \xrightarrow{t} s' \in \mathcal{T}(M)$ . For a firable transition  $t$  and a timed state class  $s$ ,  $\mathbf{dependent}(s, t)$  must include  $t$  and satisfy the following:

if  $t' \in \mathbf{dependent}(s, t)$ , then

$$\text{for every } u \in \mathbf{conflict}(t'), \mathbf{active}(s, u, t') \subseteq \mathbf{dependent}(s, t),$$

where  $\mathbf{active}(s, u, t)$  is the set of transitions whose firings possibly lead the LPN to a timed state class where  $u$  is enabled in time in the sense that  $u$  can fire earlier than  $t$ . If  $u$  cannot become enabled in time, then modular synthesis does not have to interleave the firings of  $t$  and  $u$ . More formally,

$$\mathbf{active}(s, u, t) = \{x \mid (x, d) \in \mathbf{necessary}(s, u, \{t\}), \\ x \text{ can fire } d \text{ time units earlier than } t\},$$

where  $\mathbf{necessary}(s, u, T_D)$  is a set of enabled transitions with some timing information such that in order to fire  $u$ , the firing of at least one transition in this set is necessary. Intuitively, the computation of  $\mathbf{necessary}(s, u, T_D)$  is implemented such that empty source places of transitions are searched upwards from  $u$  until some enabled transition is found. For example, when modular synthesis computes  $\mathbf{necessary}(s, t_3, T_D)$  in Fig. 5(a), it searches upward in the net until  $t_2$  is found. During the search, if a transition is reached in which its source places are marked, but a level expression on one of its incoming rules is not satisfied, the upward search must locate a transition that would contribute toward making the expression satisfied. The upward search is then restarted from this transition. For example, while calculating  $\mathbf{necessary}(s, t_{a-}, T_D)$  in

Fig. 5(b), the search encounters  $t_3$  that has its necessary token but is not level-satisfied. In this case, the search determines that  $t_{c+}$  is needed to cause the rule for  $t_3$  to be level-satisfied. It then locates a transition for  $c+$  and continues the upward search from there until it finds  $t_2$  that is fireable. If the level expression is  $c \wedge d$  instead, nothing would change except that the algorithm would have the option of searching for  $t_{d+}$ . This is because both transitions must fire to satisfy the expression; thus, the transition with the best necessary set is used. If, however, the level expression is  $c \vee d$ , then the algorithm must calculate the necessary set for both the  $t_{c+}$  and  $t_{d+}$  transitions and form the union of the results. This is because only one of the two transitions is required to satisfy the expression for  $t_3$ . As it is not known which one will occur, both must be considered together.

In the calculation of  $\text{necessary}(s, u, T_D)$ ,  $T_D$  is used to terminate loops.  $\text{necessary}(s, t, T_D)$  is actually the set of pairs  $(x, d)$  where  $x$  is a transition found in the above search process, and  $d$  is the sum of the earliest firing times in the shortest path from  $x$  to  $u$ . The value of  $d$  denotes that it takes at least  $d$  time units for  $u$  to become enabled after the firing of  $x$ . If the maximum time separation  $\delta$  between  $x$  and  $t$  is less than  $d$  time units, it means that  $u$  can never fire before  $t$ . In other words, our method does not have to consider the interleavings between  $t$  and  $u$  in this case; hence,  $\text{active}(s, u, t)$  only contains those  $x$  which can fire  $d$  time units earlier than  $t$ . For example,  $\text{necessary}(s, t_{a-}, \{t_1\}) = \{(t_2, 3)\}$  in Fig. 5(b), and so  $t_2$  is in the active set. If the latest firing time of  $t_1$  is 3 instead (i.e.,  $\delta = 2$ ), there is no possibility for  $t_{a-}$  to fire before  $t_1$ , so  $t_2$  would not be included in the active set in this case.

Finally, our method constructs  $\mathcal{T}_{S_i}(M)$  by firing only transitions in  $\text{ready}(s)$  in each  $s$ , where  $\text{ready}(s)$  is  $\text{dependent}(s, t)$  for some transition  $t$  returned by  $\text{fireable}(M, s)$ , such that all transitions in the set are fireable.

Consider again the example shown in Fig. 1 where the module to synthesize is shown in Fig. 1(b). In the initial TSC, there is only one fireable transition,  $t_{c+}$ , so the algorithm certainly must fire that one. After firing  $t_{c+}$ , there are two fireable transitions  $t_{a-}$  and  $t_{b-}$ . First, consider firing  $t_{a-}$ :  $\text{conflict}(t_{a-})$  includes  $t_{a+}$ ,  $t_{a-}$ ,  $t_{c+}$ . The transition  $t_{c-}$  is not included in the dependent set since  $t_{a-}$  is needed to enable  $t_{c-}$  resulting in a circular dependency. Circular dependencies are found for all other transitions as well, so  $\text{dependent}(s, t_{a-}) = \{t_{a-}\}$ .  $\text{conflict}(t_{b-})$  is obtained similarly, and they are all again eliminated due to circular dependencies. This means that  $\text{ready}(s)$  can be set equal to either  $t_{a-}$  or  $t_{b-}$ , and these signals do not need to be interleaved. One possible state space found using this partial order approach is shown in Fig. 3(c). Note that applying  $\text{rsg}(\mathcal{T}, S_i)$  to the state graph shown in Fig. 3(c) again results in the state graph shown in Fig. 3(b) meaning that the synthesized circuit is the same for the total order and this partial order method.

The above algorithm generates  $\mathcal{T}_{S_i}(M)$  such that  $\mathcal{T}_{S_i}(M) \subseteq \mathcal{T}(M)$ ; thus,  $\text{project}(\mathcal{T}_{S_i}(M), S_i) \subseteq \text{project}(\mathcal{T}(M), S_i)$  holds. The other direction—meaning that for any  $\rho \in \mathcal{T}(M)$ , there exists  $\rho'$  in  $\mathcal{T}_{S_i}(M)$  such that  $\text{project}(\rho, S_i) =$

Table 1  
A comparison between flat and modular synthesis.

Example	Flat			Modular		
	States	BAP	Synth	States	BAP	Synth
fifoN4	39317(130130)	1182.94	7.47	95	0.10	0.00
fifoN39				251	0.48	0.00
stari12	41792(159248)	542.50	5.48	139(206)	0.30	0.00
stari18				349(1902)	9.15	0.00
isp_3	38113(38142)	82.25	19.78	14423(14664)	17.32	0.44
isp_branch_v2				62601(62632)	136.06	18.56
isp_join	31622(32082)	54.89	9.2	8678(8688)	14.29	0.83
isp_fork	38589(38590)	67.25	6.7	12306(12307)	18.55	1.02
isp_priority	107481(107589)	181.21	22.59	17902(17904)	29.84	1.39

$\text{project}(\rho', S_i)$ —can be proven by extending the approach used in the proof of Lemma 1 in [31] so that it considers level-rules. Hence, we can prove that Property 5.1 holds under the proposed algorithm.

## 6 Results and Conclusion

The BAP algorithm and modular synthesis approach are implemented by the CAD tool *ATACS*. The performance of modular synthesis is best illustrated on examples that can be dynamically changed to produce larger sets of TSCs. Although this type of example is not representative of all designs, for a preliminary study, it does serve to establish the limits of a flattened synthesis approach and to demonstrate the benefits of this modular synthesis approach. A control circuit for a pipeline is an example of a design that can be easily modified to create larger sets of TSCs. Results from three such examples are shown in Table 1. The *fifoN* example is a *fifo* from SUN [20]. The *stari* example is another *fifo* described in [10]. Finally, the *isp* examples are interlocked synchronous pipelines from IBM [14]. Table 1 compares flat synthesis and modular synthesis times. The *States* column is the number of unique Boolean states found by the BAP algorithm followed by the number of TSCs in parentheses. The *BAP* column is the running time spent finding the TSCs. The *Synth* column is the amount of running time spent in synthesizing a single component. The running time is in seconds reported on a Pentium III 930MHz with 256Mb of memory. The blank entries in the table indicate that the flat synthesis did not complete in a reasonable amount time. These results show that the idea of applying partial order reduction to the modular synthesis of timed circuits can provide substantial improvements in synthesis time. This includes several examples that could not previously be synthesized using a flat

synthesis approach.

## References

- [1] Alur, R., and D. L. Dill, *A theory of timed automata*, Theoretical Computer Science **126** (1994), 183–235.
- [2] Belluomini, W., “Algorithms for synthesis and verification of timed circuits and systems,” Ph.D. thesis, University of Utah, Utah, 1999.
- [3] Belluomini, W., C. J. Myers, and H. P. Hofstee, *Timed circuit verification using TEL structures*, IEEE Transactions on Computer-Aided Design of Integrated Circuits **20** (2001), 129–146.
- [4] Bengtsson, J., B. Jonsson, J. Lilius, and W. Yi, *Partial order reductions for timed systems*, Proc. International Conf. on Concurrency Theory (1998), 485–500.
- [5] Berthomieu, B., and M. Diaz, *Modeling and Verification of Time Dependent Systems Using Time Petri Nets*, IEEE Transactions on Software Engineering **17** (1991).
- [6] Bozga, M., O. Maler, A. Pnueli, and S. Yovine, *Some Progress in the Symbolic Verification of Timed Automata*, International Conf. on Computer Aided Verification (1997).
- [7] Daws, C., A. Olivero, S. Tripakis, and S. Yovine, *The tool KRONOS*, Lecture Notes in Computer Science **1066** (1995).
- [8] Dill, D. L., *Timing assumptions and verification of finite-state concurrent systems*, Proc. of the Workshop on Automatic Verification Methods for Finite-State Systems (1989).
- [9] Godefroid, P., *Using partial orders to improve automatic verification methods*, Proc. of Computer Aided Verification Workshop (1990).
- [10] Greenstreet, M. R., *Implementing a STARI chip*, Proc. International Conf. Computer Design (1995), 38–43.
- [11] Henzinger, T., X. Nicollin, J. Sifakis, S. Yovine, *Symbolic model-checking for real-time systems*, Proc. of the 7th Symposium Logics in Computers Science (1992).
- [12] Hofstee, H. P., S. H. Dhong, D. Meltzer, K. J. Nowka, J. A. Silberman, J. I. Burns, S. D. Posluszny, and O. Takahashi. *Designing for a gigahertz*, IEEE MICRO **3** (1998), 66–74.
- [13] Hulgaard, H., and S. M. Burns, *Efficient timing analysis of a class of Petri nets*, Proc. International Conf. on Computer Aided Verification (1995).
- [14] Jacobson, H., P. Kudva, P. Bose, P. Cook, S. Schuster, and E.G. Mercer, *Synchronous Interlocked Pipelined CMOS*, Proc. of International Symposium on Advanced Research in Asynchronous Circuits and Systems (2002).
- [15] Katz, S., and D. Peled, *Defining conditional independence using collapses*, Semantics for concurrency BCS-FACS Workshop (1990).
- [16] Maler, O., and A. Pnueli, *Timing Analysis of Asynchronous Circuits using Timed Automata*, Proceedings of CHARME’95 (1995).
- [17] McMillan, K. L., “Symbolic model checking : An approach to the state explosion problem,” Ph.D. thesis, Carnegie Mellon University, Pittsburgh, 1992.
- [18] Mercer, E. G., C. J. Myers, and T. Yoneda, *Improved POSET timing analysis in timed Petri nets*, Proc. of Synthesis and System Integration of Mixed Technologies (2001), 127–134.

- [19] Minea, M., “Partial order reduction for verification of timed systems,” Ph.D. thesis, Carnegie Mellon University, Pittsburgh, 1999.
- [20] Molnar, C. E., I. W. Jones, Bill Coates, and Jon Lexau, = *A FIFO Ring Oscillator Performance Experiment*, Proc. of International Symposium on Advanced Research in Asynchronous Circuits and Systems (1997), 279–289.
- [21] Myers, C. J., “Computer-Aided Synthesis and Verification of Gate-Level Timed Circuits,” Ph.D. thesis, Stanford University, California, 1995.
- [22] Myers, C. J., T. G. Rokicki, and T. H. Y. Meng, *POSET timing and its application to the synthesis and verification of gate-level timed circuits*, IEEE Transactions on Computer-Aided Design **18** (1999), 769–786.
- [23] Peña, Marco A., J. Cortadella, A. Kondratyev, and E. Pastor, *Formal verification of safety properties in timed circuits*, Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems (2000), 2–11.
- [24] Schuster, S., W. Reohr, P. Cook, D. Heidel, M. Immediato, and K. Jenkins, *Asynchronous interlocked pipelined CMOS circuits operating at 3.3-4.5 GHz*, Proc. International Solid State Circuits Conf. (2000).
- [25] Semenov, A., A. Yakovlev, E. Pastor, M. Peña, J. Cortadella, and L. Lavagno, *Partial order based approach to synthesis of speed-independent circuits*, Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems (1997), 254–265.
- [26] Stevens, K. S., S. Rotem, R. Ginosar, P. Beerel, C. J. Myers, K. Y. Yun, R. Koi, C. Dike, and M. Roncken, *An asynchronous instruction length decoder*, IEEE Journal of Solid-State Circuits **36** (2001), 217–228.
- [27] Sutherland, I., and S Fairbanks, *GasP: A minimal FIFO control*, Proc. of International Symposium on Advanced Research in Asynchronous Circuits and Systems (2001), pages 46–53.
- [28] Valmari, A., *A stubborn attack on state explosion*, Proc. of Workshop on Computer Aided Verification (1990).
- [29] Yoneda, T., and H. Schlingloff, *Efficient verification of parallel real-time systems*, Formal Method in System Design (1997), 187–215.
- [30] Yoneda, T., and H. Ryu, *Timed trace theoretic verification using partial order reduction*, Proc. of Fifth International Symposium on Advanced Research in Asynchronous Circuits and Systems (1999), 108–121.
- [31] Yoneda, T., E. G. Mercer, and C. J. Myers, *Modular synthesis of timed circuits using partial order reduction*, Proc. of Synthesis and System Integration of Mixed Technologies (2001), 151–158.
- [32] Zheng, H., E. Mercer, and C. Myers, *Automatic abstraction for verification of timed circuits and systems*, International Conf. on Computer Aided Verification (2001).