

Improved POSET Timing Analysis in Timed Petri Nets ^{*}

Eric G Mercer

Chris J. Myers

Tomohiro Yoneda

University of Utah
Salt Lake City, UT 84112
eemercer@ece.utah.edu

University of Utah
Salt Lake City, UT 84112
myers@ece.utah.edu

Tokyo Institute of Technology
Tokyo, Japan
yoneda@cs.titech.ac.jp

Abstract—This paper presents an improved timing algorithm for the analysis of timed Petri nets that is based on the *POSET* algorithm. The new algorithm reduces the number of redundant concurrent orderings the *POSET* algorithm explores by directly considering causal assignments. This paper shows that the new algorithm, when compared to the original *POSET* algorithm, results in an average 2.25 times improvement in runtime and a 57% reduction in stored zones when applied to a suite of example circuits. Although the new algorithm can suffer an exponential increase in the number of causal assignments it must consider, this paper shows it to be a property of the *POSET* algorithm itself that does not happen often in practice.

I. INTRODUCTION

To increase performance, circuit designers are experimenting with *timed circuits*—a class of circuits that rely on a complex set of timing constraints for correct functionality [1, 2]. This is evidenced by industrial scale experimental designs such as the Intel RAPPID instruction length decoder [3], SUN’s GASP pipelines [4], IBM’s IPCMOS pipelines [5], the IBM guTS microprocessor [6], and the University of Washington output prediction logic (OPL) 64-bit adder [7]. Although RAPPID, GASP, and IPCMOS are asynchronous and guTS and OPL are synchronous, all of these designs use formal timing constraints to achieve better performance. Algorithms to check timing constraints are required to make these types of designs practical. Due to the complexity of the timing constraints, however, traditional static timing analysis is not adequate. Timed state space exploration is required; thus, improved timed state space exploration is paramount to producing efficient timed circuits.

The addition of time to any method of state space exploration, in general, exacerbates state explosion. This is because any approach must define a *timed state* that not only describes the *untimed state* of the system (i.e., the

value of all signals in the circuit) but information indicating the occurrence of that untimed state in time. A set of timed states may have a common untimed state but be unique in their occurrence in time; thus, the time representation in the timed state now affects the size of the timed state space. A discrete model of time divides time into a minimum discrete quantum. This bounds the size of the timed state space because time is no longer a continuous quantity [8]. This approach lends itself to symbolic methods to compactly represent the timed state space [9, 10]. Practical results using discrete methods are promising as recently shown by [11]. Choosing an adequate time quantum, however, is critical to discrete timing analysis. If the quantum is too large, then behaviors of the system are masked and lost. If the quantum is too small, then the timed state space is too large to manage [12]. The dependence on the size of the time quantum in discrete models erodes confidence in the analysis of complex timing constraints. A masked timing behavior due to the size of the time quantum can lead to an actual timing failure in an aggressively timed circuit. A more precise time model is required for circuit analysis.

A continuous time model does not restrict when things occur. In a continuous time model, timing information is often represented by *time separations*—the amount of time that elapses between any two transitions in the circuit. A set of seemingly concurrent transitions in a circuit may actually be ordered in time due to various combinations of delays. A timed state space can be derived by ordering these seemingly concurrent transitions according to their time separations. The minimum and maximum time separations of transitions in the circuit can be calculated from the structure of the circuit model. These minimum and maximum separations are used to approximate the actual timed state space [13]. This approach to timing analysis is limited in its scope of application. Although most circuits have deterministic behavior for a given set of inputs, their testing environments are often modeled by random processes. Algorithms to calculate the minimum and maximum time separations do not adequately address non-determinism in environment models, as well as classes of circuits that include arbiters.

Non-determinism can be analyzed in a continuous time

^{*}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.

model by deriving time separations from execution traces in the circuit. Each explored trace can potentially generate a unique set of time separations at each untimed state of the trace. A set of time separations at a given untimed state can be grouped into equivalence classes called regions [14, 15]. A timed state is created for each unique region generated at an untimed state during the trace evolution. A region naturally address the continuous nature of the timed state space but is too small to significantly reduce the number timed states. Zones extend regions by representing larger equivalence classes using convex hulls [16]. These can be represented in *difference bound matrices* (DBMs) [16] since they represent allowed separations in a system, and many efficient algorithms have been developed to manipulate DBMs [17]. DBMs can also be implicitly represented [18, 19].

A zone implicitly represents an allowed execution trace in the circuit. A different ordering on a set of concurrent transitions can lead to a different zone. This creates an exponential branching in the timed state space. As the zone is a partially ordered set relating various transition times, it is possible to reduce branching in the timed state space by adding fewer relations to the set. *Local time semantics* and *partially ordered sets* (POSETs) remove orderings in the zone on sets of independent concurrent signal transitions reducing the representation size of the timed state space [20, 21, 22, 23, 24, 25].

POSET reduction in the timing analysis of *Timed Petri nets* (TPN)—Petri nets with timers on the places—yields a significant reduction in the number of zones associated with each marking [24]. The algorithm in [24] implements the POSET reduction on concurrent independent signal transitions. It explicitly models, however, timers on the places; thus, it needlessly considers many redundant orderings of firings of these timers [21].

An optimization to the POSET algorithm removes some redundant orderings of concurrent place timer firings, but this optimization is limited to specific timing conditions [24]. The work in this paper addresses this issue in a general framework by realizing an optimization suggested in [21], but not formalized or implemented, to remove redundant orderings from the POSET timing analysis of TPNs regardless of the timing conditions. This paper shows that the new algorithm, when compared to the original POSET algorithm, results in an average 2.25 times improvement in runtime and a 57% reduction in stored zones when applied to a suite of example circuits. Although the new algorithm can suffer an exponential increase in the number of causal assignments it must consider, examples with this property appear to be rare. This is a similar issue that is a property of the original POSET algorithm. The new algorithm, however, is limited to TPNs that do not contain independent loops. If a TPN contains an independent loop, then the new algorithm is divergent in the timed state space representation. Fortunately, this property seems to appear in only

contrived examples.

The remainder of this paper is organized as follows: Section II presents the semantics of timed Petri nets; the formal model used to represent systems in this work. Section III describes the timing information stored in the timed state space; it shows that a finite representation of the timed state space exists for any valid timed Petri net. Section IV presents the new POSET algorithm; it demonstrates that the new algorithm creates a finite representation of the exact timed state space in the absence of independent loops in the TPN. Finally, Section V concludes by presenting results, strengths and weaknesses of the algorithm, and suggestions for future work.

II. TIMED PETRI NETS

A timed Petri net is a five-tuple $N = \langle P, T, F, \Delta, \mu_o \rangle$. $P = \{p_1, p_2, \dots, p_m\}$ is a finite nonempty set of *places*. $T = \{t_1, t_2, \dots, t_n\}$ is a finite set of *transitions*. $F \subseteq (P \times T) \cup (T \times P)$ is the *flow relation*. $\Delta : P \rightarrow \mathbf{Q}^+ \times (\mathbf{Q}^+ \cup \{\infty\})$ is a function mapping each $p \in P$ to a possibly unbounded delay, where \mathbf{Q}^+ is the set of non-negative rational numbers. For convenience, $\Delta(p) = (l, u)$ where $l \leq u$; $\Delta_l(p) = l$ and $\Delta_u(p) = u$ return the lower and upper delay bounds for place p . $\mu_o \subseteq P$ is the *initial marking* of the net. For any transition t , $\bullet t = \{p \in P \mid (p, t) \in F\}$ and $t \bullet = \{p \in P \mid (t, p) \in F\}$ are the *preset* and *postset* places of transition t . A marking μ is any subset of P .

A *timed state* for a net N is a pair $\sigma = (\mu, C)$, where μ is a marking of N and $C : P \rightarrow \mathbf{Q}^+$ is a timer function. The initial state σ_o is (μ_o, C_o) where $C_o(p) = 0$ for all $p \in \mu_o$. For a timed state $\sigma = (\mu, C)$: let **enabled** $(\mu) = \{t \in T \mid \forall p \in \bullet t, p \in \mu\}$ be the set of enabled transitions in a given marking (i.e., transitions that have a place in the marking for each member of their preset); and let **satisfied** $(\sigma) = \{p \in \mu \mid C(p) \geq \Delta_l(p)\}$ be the set of places in the marking whose timers have achieved their lower bound. A timed state change can either be the firing of a transition, or it can be the passage of time. A transition t_f can fire in $\sigma = (\mu, C)$ if it is enabled ($t_f \in \mathbf{enabled}(\mu)$) and satisfied ($\forall p \in \bullet t_f, p \in \mathbf{satisfied}(\sigma)$) in the timed state. The new timed state $\sigma' = (\mu', C')$ derived from firing t_f in $\sigma = (\mu, C)$ is computed as: $\mu' = \mu'' \cup t_f \bullet$ where $\mu'' = (\mu - \bullet t_f)$; and $\forall p \in \mu'$, if $p \in \mu''$, then $C'(p) = C(p)$, else $C'(p) = 0$. Note that transition t_f happens instantly. A quantity of time $\tau \in \mathbf{Q}^+$ can pass in state $\sigma = (\mu, C)$ iff $\forall t \in \mathbf{enabled}(\mu), \exists p \in \bullet t. C(p) + \tau \leq \Delta_u(p)$, meaning that every enabled transition must be able to accept τ without being forced to fire. Note that a transition can allow place timers in its preset to expire before it fires, but it must fire before they all expire. The new timed state $\sigma' = (\mu', C')$ derived from advancing time by τ in $\sigma = (\mu, C)$ is computed as: $\mu' = \mu$; and $\forall p \in \mu', C'(p) = C(p) + \tau$.

A run $\rho = \sigma_o \xrightarrow{x_1} \sigma_1 \xrightarrow{x_2} \sigma_2 \dots \sigma_{n-1} \xrightarrow{x_n} \sigma_n$ is a possibly infinite sequence of timed states (σ_i) separated by *timed*

transitions—a pair $x_i = (\tau_i, t_i)$ denoting that from the timed state σ_{i-1} , τ_i units of time are passed, and then transition t_i is fired leading to the next timed state σ_i in the current run. Let $\rho(j) = \sigma_j$ for $0 \leq j \leq n$. Let $\Sigma = \{\rho_1, \rho_2, \rho_3, \dots\}$ be the set of all possible runs of N . A state σ is *reachable* iff $\exists(i, j) . (\rho_i \in \Sigma) \wedge (\rho_i(j) = \sigma)$. Let Φ be the set of all reachable timed states in N . A TPN is said to be *one-safe* if for any reachable state $\sigma_{i-1} = (\mu_{i-1}, C_{i-1})$ and timed transition $x_i = (\tau_i, t_i)$ in any possible run $\rho \in \Sigma$, $(\mu_{i-1} - \bullet t_j) \cap t_j \bullet = \emptyset$. The remainder of this work is restricted to one-safe nets.

III. TIMING INFORMATION

The sets of reachable timed states Φ and possible runs Σ are infinite because each timer function associates rational values with the ages of places. To obtain a finite representation of the infinite timed state space, the timed states are grouped into equivalence classes represented by a set of linear inequalities called *zones*. Let x and y be variables and q be a constant rational number, then $x - y \leq q$ relates the difference between x and y to q . An example of a zone from the net fragment shown in Fig.1 is:

$$I = \{3 \leq x_B - x_A \leq 7, 2 \leq x_C - x_A \leq 5\}.$$

This zone restricts the separation between x_A and x_B to the range $[3, 7]$ and the separation between x_A and x_C to the range $[2, 5]$. Any separations that are not defined in the zone are unbounded. If x_A , x_B , and x_C correspond to the transitions in Fig.1, then the example zone I defines when **B** and **C** can occur in time relative to transition **A**.

If I is a set of defined relations for a given zone, then $\mathbf{var}(I) = \{x_1, x_2, \dots, x_n\}$ returns the ordered set of variables the relations in I are defined on. The function $\mathbf{var}(I, t)$ returns the variable in I that corresponds to transition t , otherwise it returns NULL. A vector (a_1, a_2, \dots, a_n) of rational constants is *feasible* for zone I iff every relation in the zone obtained by replacing $x_i \in \mathbf{var}(I)$ by a_i for $1 \leq i \leq n$ holds. A feasible vector for the example zone is $(5, 9, 8)$: $x_B - x_A = 9 - 5 = 4$ and $3 \leq 4 \leq 7$; $x_C - x_A = 8 - 5 = 3$ and $2 \leq 3 \leq 5$. A zone is *consistent* if there exists a feasible vector for it. A zone has a canonical form that is obtained by applying a shortest-path algorithm to its set of linear inequalities. The canonical form of the example zone is

$$I = \left\{ \begin{array}{l} 3 \leq x_B - x_A \leq 7, \\ 2 \leq x_C - x_A \leq 5, \\ -2 \leq x_B - x_C \leq 5 \end{array} \right\}.$$

The canonical form of the example zone restricts the separation between **C** and **B** to be in the $[-2, 5]$ range. This means that **B** cannot occur more than 5 time units after **C**; and **C** cannot occur more than 2 time units after **B**. To check that a zone is consistent, it is first put in a canonical form. If a negative weight cycle exists in the canonical form, then the zone is inconsistent.

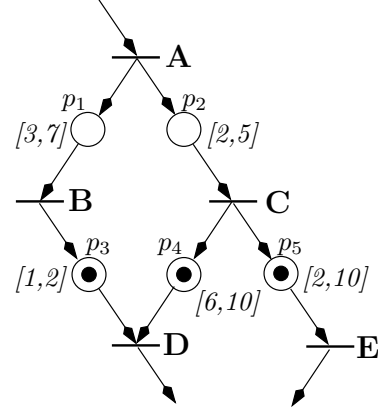


Fig. 1. A fragment of a simple TPN.

IV. IMPROVED POSET ALGORITHM

The POSET algorithm can best be understood by example. In a traditional total order zone based method, the timed state can be represented using a set of inequalities showing the time separations between the place timers. For example, the timed state of the net in Fig.1 after transition **A** has fired is

$$\begin{aligned} \mu &= \{ p_1, p_2 \} \\ I &= \left\{ \begin{array}{l} 0 \leq p_1 - p_0 \leq 5, \\ 0 \leq p_2 - p_0 \leq 5, \\ 0 \leq p_1 - p_2 \leq 0 \end{array} \right\} \end{aligned}$$

The dummy variable p_0 is used as a reference point. All place timers are defined relative to the reference point p_0 .

From this point, the total order algorithm considers the firing of either place p_1 or p_2 . This is because I allows these timers to fire in either order. Consider the firing of p_1 . This causes transition **B** to fire. The firing of **B** removes p_1 from the marking, and adds the new place p_3 . The relations defined for p_1 must also be removed from I_p while new relations are added for p_3 to I_p . The relations added to I_p for p_3 are as follows: first, the relative time of p_2 and p_0 is adjusted to show that at least 3 time units have passed to allow **B** to fire (i.e., $\{3 \leq p_2 - p_0 \leq 5\}$); second, $\{0 \leq p_3 - p_0 \leq 2\}$ is added to set the bounds for the new place; and third, the separation between p_2 and p_3 can be determined by copying the separation between p_2 and p_0 (i.e., $\{3 \leq p_2 - p_3 \leq 5\}$). The resulting timed state is shown below:

$$\begin{aligned} \mu &= \{ p_2, p_3 \} \\ I &= \left\{ \begin{array}{l} 3 \leq p_2 - p_0 \leq 5, \\ 0 \leq p_3 - p_0 \leq 2, \\ 3 \leq p_2 - p_3 \leq 5 \end{array} \right\} \end{aligned}$$

In this state, either p_2 can fire or p_3 can fire since they can both be satisfied in this timed state. If p_2 fires, p_4 and p_5 are added to the marking and to I is updated to

obtain the new timed state of the system:

$$\begin{aligned} \mu &= \{ p_3, p_4, p_5 \} \\ I &= \left\{ \begin{array}{l} 0 \leq p_3 - p_0 \leq 2, \\ 0 \leq p_4 - p_0 \leq 2, \\ 0 \leq t_5 - p_0 \leq 2, \\ -2 \leq p_4 - p_3 \leq 0 \\ -2 \leq p_5 - p_3 \leq 0 \\ 0 \leq p_5 - p_4 \leq 0 \end{array} \right\} \end{aligned}$$

When the algorithm goes back and fires p_2 first followed by p_1 , the following timed state is derived for the same marking.

$$\begin{aligned} \mu &= \{ p_3, p_4, p_5 \} \\ I &= \left\{ \begin{array}{l} 0 \leq p_3 - p_0 \leq 2, \\ 0 \leq p_4 - p_0 \leq 5, \\ 0 \leq t_5 - p_0 \leq 5, \\ 0 \leq p_4 - p_3 \leq 5 \\ 0 \leq p_5 - p_3 \leq 5 \\ 0 \leq p_5 - p_4 \leq 0 \end{array} \right\} \end{aligned}$$

Let us now consider the POSET algorithm described in [24]. This algorithm operates on two zones to determine the state of the TPN: one contains place timer separations (I_p) and the other contains transition separations (I_t). The initial timed state of the net in Fig.1 is nearly the same except $I_p = I$ and $I_t = \emptyset$. The zone I_t is empty, because there are no transitions to define relative to transition **A**.

The firing of p_1 causes transition **B** to fire. **B** is added to I_t by defining its firing time relative to **A**: $\{3 \leq \mathbf{x}_B - \mathbf{x}_A \leq 7\}$. Again, new relations are added for p_3 to I_p . The first two are the same, but the last one is now computed from I_t to be $\{3 \leq p_2 - p_3 \leq 7\}$ which denotes that p_2 is at least 3 time units ahead of p_3 and no more than 7 time units ahead. Now, the two zones are put into canonical form to tighten down all relations resulting in the following timed state:

$$\begin{aligned} \mu &= \{ p_2, p_3 \} \\ I_p &= \left\{ \begin{array}{l} 3 \leq p_2 - p_0 \leq 5, \\ 0 \leq p_3 - p_0 \leq 2, \\ 3 \leq p_2 - p_3 \leq 5 \end{array} \right\} \\ I_t &= \{ 3 \leq \mathbf{x}_B - \mathbf{x}_A \leq 7 \}. \end{aligned}$$

Note that so far I_p is still equal to I .

When p_2 fires, the transition **C** is added to I_t . This is done by defining its separations relative to **A**. I_t is then put into canonical form and all relations relating it to **A** can now be removed from I_t because **A** is not required by the new marking. p_4 and p_5 are also added to the marking

and to I_p to create the new timed state of the system:

$$\begin{aligned} \mu &= \{ p_3, p_4, p_5 \} \\ I_p &= \left\{ \begin{array}{l} 0 \leq p_3 - p_0 \leq 7, \\ 0 \leq p_4 - p_0 \leq 7, \\ 0 \leq t_5 - p_0 \leq 2, \\ -2 \leq p_4 - p_3 \leq 5 \\ -2 \leq p_5 - p_3 \leq 5 \\ 0 \leq p_5 - p_4 \leq 0 \end{array} \right\} \\ I_t &= \{ -2 \leq \mathbf{x}_B - \mathbf{x}_C \leq 5 \}. \end{aligned}$$

In this case, I_p is not equal to I . Instead, it is equal to the union of the two I 's found by two possible interleavings to obtain this marking. The reason is this equivalence class does not include any explicit relations ordering transitions **B** and **C**. Although the algorithm fired **B** before it fired **C**, this information has been discarded when the timed state is derived for the marking shown in Fig.1. Note that this does not allow any new timed states that do not already exist in the allowable runs of the net; thus, the larger equivalence class contains more timed states from the set of reachable states reducing the number of equivalences classes needed to represent the timed state space. This is the essence of the *POSET* algorithm in [22, 23, 24, 25].

This algorithm though has a couple sources of inefficiency. The first is the need to maintain two separate zones. The second is that operating on place transition firings can substantially increase the number of zones needed to represent the timed state space. This second problem can be illustrated using the example shown in Fig.2. In this example, the transition **D** is enabled by places p_1 , p_2 , and p_3 . Each place has a timer that is reset when it receives a token and expires when it reaches the upper time bound on its place. Assume that the places p_1 , p_2 , and p_3 all receive their tokens at the same time and have their timers reset. In this case, the algorithm explores in a total order the expiration of timers on the places p_1 , p_2 , and p_3 to determine the time separations between **D** and its enabling transitions **A**, **B**, and **C** [24]. Although the algorithm does not order **A**, **B**, and **C** in any timing information, it does explore the following orders of timer firings to potentially create separate timed states for each firing order: $\{(p_2, p_3, p_1), (p_3, p_2, p_1), (p_1, p_3, p_2), (p_3, p_1, p_2), (p_1, p_2, p_3), (p_2, p_1, p_3)\}$. The (p_2, p_3, p_1) order is redundant with the (p_3, p_2, p_1) order because both orders determine the separation of **D** from its enabling transitions based on the *causal assignment* of the p_1 timer (i.e., the firing p_1 sets the upper bound on when **D** fires). In this case, half of the explored orders are redundant; thus, the performance of the algorithm degrades as the number of active concurrent place timers increases.

The goal of the POSET algorithm is to avoid relating independent concurrent transitions in the net. The equivalence classes that it stores in the finite representation of the timed state space, however, are constructed from orderings of place timer firings. The POSET algorithm uses place timer firings to find causal assignments; thus,

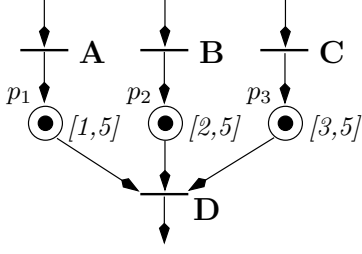


Fig. 2. A simple TPN to show redundant interleavings.

it explores timer firings in a total order. In the example in Fig.2, the POSET algorithm explores six orderings of timer firings with half of those orderings being redundant because they do not lead to new causality assignments. This causes an increase in the size of the timed state space representation because more equivalence classes are required to cover all reachable states. Recall that the POSET algorithm uses a place timer zone to represent the timed state space, and it uses a transition zone to remove orders on timers in the place timer zone construction. When a transition fires, both zones are updated. Between transition firings, however, new place timer zones are generated and stored in the timed state space; thus, the algorithm's performance is degraded. The timed state space is larger than it needs to be, and runtime is impacted by exploring the redundant orderings.

An optimization to the POSET algorithm only stores clock regions when transitions fire. The clock regions generated in exploring the total orders between transition firings are discarded. They are only used to determine causality. This optimization does improve the timed state space representation, but our experiments have shown that it does not improve the runtime performance. Another optimization to the POSET algorithm removes some redundant orderings of concurrent place timer firings, but this optimization is limited to specific timing conditions [24]. The algorithm presented here—*bourne again POSET* timing analysis (BAP)—addresses this issue in a general framework by realizing an optimization suggested in [21] to remove redundant orderings from the POSET timing analysis of TPNs regardless of the timing conditions. This optimization reduces the representation size of the timed state space by covering the reachable timed states with fewer equivalence classes. Another advantage of this algorithm is that it represents the timed state space using a single zone.

Before presenting the BAP algorithm, it is necessary to define the functions in the algorithm which are illustrated using the example net shown in Fig.1. For the marking in Fig.1, the latest time of **D** allowed by the net can be defined relative to the time of **B** or **C**. Transition **B** is *causal* to **D** if the latest time of transition **D** is defined relative to **B** in the equivalence class. Thus, any tran-

sition t may have several causality assignments. Let the function $\mathbf{causal}(\mu)$ return the set of causality assignments for the transitions in $\mathbf{enabled}(\mu) = (t_1, t_2, \dots, t_n)$. Each member of the set $\mathbf{causal}(\mu)$ has the form (u_1, u_2, \dots, u_n) where each u_i is the causal assignment for t_i in the set of enabled transitions. Note that there can be an exponential number of causal assignments for any given set of enabled transitions.

Let $\mathbf{enabling}(\mu, I, t) = \{t_e \in T \mid (\exists p \in \bullet t, p \in \mu) \wedge \mathbf{var}(I, t_e) \neq \text{NULL}\}$ be the set of enabling transitions for t . The zone I is used in $\mathbf{enabling}(\mu, I, t)$ to identify those transitions that fired to create the marking μ . If a set of transitions load a single place, then only one of those transitions could have fired for the net to be one-safe. I is used to identify that transition. For the marking in Fig.1 and a zone I where $\mathbf{var}(I) = \{x_B, x_C\}$, $\mathbf{enabling}(\mu, I, \mathbf{D}) = \{\mathbf{B}, \mathbf{C}\}$. Let $\Delta_l(t, t') = \max_{p \in t \bullet \cap \bullet t'} \Delta_l(p)$ if $t \bullet \cap \bullet t' \neq \emptyset$ otherwise it equals ∞ . This function returns the minimum separation allowed between transitions t and t' . This separation is unbounded if t and t' are not connected by a place; $\Delta_u(t, t')$ is defined similarly. For $s_c = (\mu, I)$ where μ is the marking in Fig.1, transition **D** cannot happen before p_3 and p_4 are satisfied; thus, the delays for p_3 and p_4 define minimum time separations between **D** and its enabling transitions **B** and **C**. These time separations are defined as $\Delta_l(\mathbf{B}, \mathbf{D}) = 1$ and $\Delta_l(\mathbf{C}, \mathbf{D}) = 6$. A maximum time separation between **D** and its enabling transitions **B** and **C** is defined by either p_3 or p_4 depending on the causal assignment. This separation can be either $\Delta_u(\mathbf{B}, \mathbf{D}) = 2$ or $\Delta_u(\mathbf{C}, \mathbf{D}) = 10$.

The minimum separations between an enabled transition and its enabling transitions are specified in an equivalence class by the function $\mathbf{lower}(\mu, I, t_f) = I \cup \{(\mathbf{var}(I, t_f) - \mathbf{var}(I, t_e) \geq \Delta_l(t_e, t_f))\}$ for all $t_e \in \mathbf{enabling}(\mu, I, t_f)$. $\mathbf{lower}(\mu, I, t_f)$, intuitively, returns a zone I' that when used in an equivalence class $s'_c = (\mu', I')$, where μ' is the marking generated from firing t_f in μ , accepts timed states from runs that satisfy the minimum time separations between t_f and its enabling transitions. A maximum separation between an enabled transition and one of its enabling transitions—the causal assignment—is defined in I as follows: if t_e is the causal assignment for t_f , then the algorithm must restrict the maximum time separation between t_f and t_e to be the upper bound from the place connecting t_f and t_e (i.e., $I' = I' \cup \{\mathbf{var}(I, t_f) - \mathbf{var}(I, t_e) \leq \Delta_u(t_e, t_f)\}$). An equivalence class $s'_c = (\mu', I')$ created from $s_c = (\mu, I)$ by firing t_f in μ to obtain μ' with $I' = \mathbf{lower}(\mu, I, t_f) \cup \{\mathbf{var}(I, t_f) - \mathbf{var}(I, t_e) \leq \Delta_u(t_e, t_f)\}$ now only covers timed states in runs that satisfy the minimum time separations between t_f and its enabling transitions, as well as the maximum time separation between t_f and t_e .

The sets of allowed traces Σ and reachable states Φ can be generated by firing enabled transitions and evolving consistent equivalence classes from an initial state in a

depth first order. Care must be taken, however, to order transition firings according to their allowed separations in the current equivalence class, otherwise unreachable timed states and runs can be introduced into Σ and Φ . An unreachable run and timed state are introduced into Σ and Φ by an equivalence class when a transition is fired from a set of seemingly concurrent enabled transitions that is actually totally ordered through a combination of delays to always occur after other enabled transitions. Let **can_fire** (μ, I, t) return *true* if there exists a causal assignment $u \in \mathbf{causal}(\mu)$ to create a consistent equivalence class that allows t to fire before all other enabled transitions. This can be checked by first creating a zone I that includes the minimum separations for all enabled transitions: $\forall t_f \in \mathbf{enabled}(\mu), I = I \cup \mathit{lower}(\mu, I, t_f)$. The maximum separations in I are set according to the causal assignment u . If the zone is consistent, then t can fire first in $s_c = (\mu, I)$ iff all paths from other enabled transitions in μ to t have a positive weight in the zone I . The function **fireable** $(\mu, I) = \{t \in \mathbf{enabled}(\mu) \mid \mathbf{can_fire}(\mu, I, t) = \mathit{true}\}$ returns the set of transitions that can concurrently fire from (μ, I) . Finally, **delete** (I, μ) removes from I any variables associated with transitions that are no longer found in **enabling** (μ, I, t) for any t in **enabled** (μ) .

The BAP timing analysis algorithm is presented in Algorithm 1. The idea of the BAP algorithm is to determine valid causal assignments using the separations in the equivalence class rather than exploring a total order on place timer firings. In this sense, the algorithm no longer considers place timer firings directly. The BAP algorithm simply tests causal assignments using the timing information. If an assignment is valid, then it uses it to evolve a new equivalence class.

The function of the algorithm is best illustrated by example. Suppose that transition **A** just happened in the net shown in Fig.1; thus, the marking is defined as $\mu = \{p_1, p_2\}$, and the zone $I = \emptyset$ with $\mathbf{var}(I) = x_{\mathbf{A}}$. I contains no relations because there is nothing to define relative to **A** in this marking. The set of fireable transitions is computed by adding transitions **B** and **C** to I with there allowed separations. Because **A** is the only causal assignment, a single consistent zone is generated: $I = \{-2 \leq x_{\mathbf{B}} - x_{\mathbf{C}} \leq 5\}$ This zone allows **B** and **C** to happen in any order; thus, **fireable** $(\mu, I) = \{\mathbf{B}, \mathbf{C}\}$. If $\mathit{stack} = (((\mu, I), \mathbf{B}), ((\mu, I), \mathbf{C}))$ at the start of the **while** loop, then the function $\mathit{stack.pop}()$ sets $t_f = \mathbf{B}$. The new marking μ' is computed by letting t_f happen: $\mu' = \{p_2, p_3\}$. The function **lower** (μ, I, t_f) constrains the minimum separation between **A** and **B** to be greater than 3. As **enabling** $(\mu, I, t_f) = \{\mathbf{A}\}$, $I' = \{x_{\mathbf{B}} - x_{\mathbf{A}} \geq 3\}$ after calling the function **lower** (μ, I, t_f) . The transition **A** is the only causal assignment for **B**, so $(x_{\mathbf{B}} - x_{\mathbf{A}} \leq 7)$ is added to I' giving $I' = \{3 \leq x_{\mathbf{B}} - x_{\mathbf{A}} \leq 7\}$. Since all transitions are needed for future firings, $I_n = I'$ after calling the function **delete** (I', μ') . I' is consistent and I_n is not found in the representation of the timed state

Algorithm 1 Find (Φ, Σ) for $N = \{P, T, F, \Delta, \mu_o\}$

```

 $\Phi = \{(\mu_o, I_o)\}$   $\Sigma = \emptyset$   $\mathit{stack} = \emptyset$ 
for all  $t \in \mathit{fireable}(\mu_o, I_o)$  do
   $\mathit{stack.push}((\mu_o, I_o), t)$ 
end for
while  $\mathit{stack.empty}() \neq \mathit{true}$  do
   $((\mu, I), t_f) = \mathit{stack.pop}()$ 
   $\mu' = (\mu - \bullet t_f) \cup t_f \bullet$ 
  for all  $t_e \in \mathit{enabling}(\mu, I, t_f)$  do
     $I' = \mathit{lower}(\mu, I, t_f)$ 
     $I' = I' \cup \{(var(I', t_f) - var(I', t_e) \leq \Delta_u(t_e, t_f))\}$ 
     $I_n = \mathit{delete}(I', \mu')$ 
    if  $\mathit{consistent}(I') = \mathit{true} \wedge (\mu', I_n) \notin \Phi$  then
       $\Phi = \Phi \cup \{(\mu', I_n)\}$ 
       $\Sigma = \Sigma \cup \{((\mu, I), t_f, (\mu', I_n))\}$ 
      for all  $t \in \mathit{fireable}(\mu', I_n)$  do
         $\mathit{stack.push}((\mu', I_n), t)$ 
      end for
    end if
  end for
end while

```

space, so (μ, I_n) is added to Φ with an appropriate edge added to Σ . The function **fireable** $(\mu', I_n) = \{\mathbf{C}\}$ for this zone, so when the function returns back to the **while** loop, $\mathit{stack} = (((\mu', I_n), \mathbf{C}), ((\mu, I), \mathbf{C}))$. At this point, the algorithm loops back and pops an entry of the stack to set $\mu = \{p_2, p_3\}$, $I = \{3 \leq x_{\mathbf{B}} - x_{\mathbf{A}} \leq 7\}$, and $t_f = \mathbf{C}$. From this point, it computes a new zone $\{-2 \leq x_{\mathbf{B}} - x_{\mathbf{C}} \leq 5\}$. Notice that to this point that the algorithm is similar to the original POSET algorithm except that I_p is no longer used. At this point, transitions **D** and **E** are both fireable. Let us consider firing **D**. From this point BAP will generate two equivalence classes:

$$\begin{aligned}
 I_{\mathbf{B}} &= \{6 \leq x_{\mathbf{D}} - x_{\mathbf{C}} \leq 7\} \\
 I_{\mathbf{C}} &= \{6 \leq x_{\mathbf{D}} - x_{\mathbf{C}} \leq 10\},
 \end{aligned}$$

where the subscripts denote the causal assignments used to generate the zones. In this class, only $I_{\mathbf{C}}$ is stored in Φ because it includes more timed states than $I_{\mathbf{D}}$. In this manner, the algorithm continues until the stack is empty.

If this algorithm is applied to the example shown in Fig.2, it would directly generate three zones, one for each potential causal assignment, rather than consider six possible orderings of place firings. The result is that the number of zones needed to represent the timed state space is cut in half. When there is larger numbers of enabling transitions to the transition being fired, the improvement is even more pronounced.

The following theorem can be proven for the BAP algorithm. Proof is omitted due to space limitations.

Theorem 1 For a given one-safe net N : the set of reachable markings found by BAP is equivalent to the set of reachable markings allowed by N .

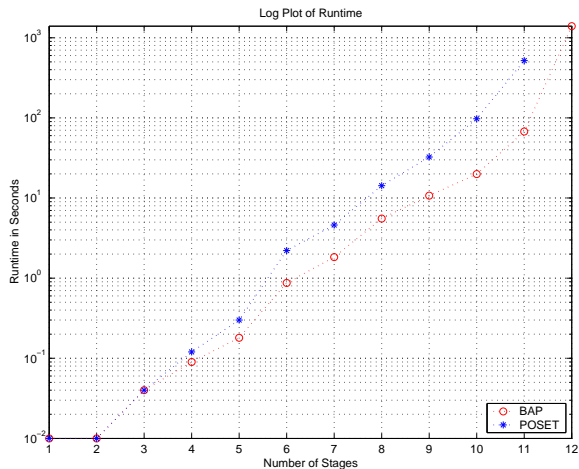


Fig. 3. Plot of runtime versus STARI stages.

V. RESULTS AND CONCLUSION

The improved POSET timing analysis algorithm, BAP, has been implemented in the CAD tool *ATACS* using C++. The algorithm runs correctly on a test suite of 149 examples, 83 of which have measurable runtimes on a Pentium III processor with 256MB of memory. For the 83 examples with measurable runtimes, the BAP algorithm improved runtime 2.25 times on average over the POSET algorithm presented in [24, 25]. In addition, BAP represents an equivalent timed state space using 57% fewer zones resulting in a significant reduction in memory.

Fig.3 presents the runtimes of both the BAP and POSET algorithms for various stages of STARI buffers. A STARI buffer enables communication between 2 synchronous systems operating at the same clock frequency but with each system seeing some different amount of clock skew; thus, the 2 systems are out-of-phase with respect to one another [26]. In timed *COSPAN*, it is reported that a 3 stage gate-level STARI buffer ran out of memory during state space exploration [15]. Work in [27], however, is able to verify an abstract model of an 8 stage STARI in 1.67 hours. Fig.3 shows that a 3 stage gate-level STARI buffer is analyzed in 0.04 seconds for both the BAP and POSET algorithms. The POSET algorithm verifies an 11 stage STARI buffer in 519.12 seconds. On the same buffer, The BAP algorithm runs in 67.5 seconds yielding a 7.7 times improvement over the POSET algorithm. This improvement is largely due to the fact that the BAP algorithm finds the equivalent timed state space using 40% fewer zones. Although the POSET algorithm exhausts memory for a 12 stage STARI buffer, the BAP algorithm completes in 24 minutes.

Fig.4 presents runtimes for the BAP and POSET algorithms run on various sizes of TAG units. The TAG unit is a circuit in the Intel *RAPPID* instruction length decoder. It is instrumental to the decoder's improved per-

formance [3]. The function of the TAG unit is to receive an incoming pulse on a set of inputs indicating that it is at the beginning of the next instruction. When the TAG unit receives an incoming pulse, it looks at the length of its decoded instruction, and then generates a pulse to the TAG unit at the beginning of the next instruction according to the received length. The size of the TAG unit denotes the number of different units that send it a tag, as well as the number of units that it can forward that tag to. The *RAPPID* design is based on a TAG unit of size 7. For performance comparison, the TAG unit is extended to larger sizes. From Fig.4, the POSET algorithm evaluates a size 13 TAG unit in 343 seconds. The BAP algorithm runs in 86 seconds on the same size TAG unit. Although the POSET algorithm exhausts memory on a TAG unit of size 14, the BAP algorithm completes on a TAG unit of size 28 in 59 minutes.

Although the BAP algorithm improves in runtime and zone count over the POSET algorithm for many benchmarks, it does lose in certain examples. If an example contains an independent loop (i.e., a sequence of transitions that are completely independent of the other transitions in the system), then the state space exploration in the BAP algorithm does not converge. This is due to the fact that the time separations between transitions in the independent loop and the remainder of the system grow unbounded in the POSET; thus, an isomorphic zone at any given untimed state is never found in the exploration. Fortunately, this property has only been found in contrived examples. The next issue relates the potential exponential explosion in the number of causal assignments that must be evaluated for a given set of enabled transitions. The POSET algorithm never considers inconsistent causal assignments because it explicitly explores all allowed orderings of place firings. The BAP algorithm, however, does not explicitly explore place firing orders; thus, it must consider all causal assignments for a given set of enabled transitions to discover the transitions that can fire. In certain examples, many of the causal assignments the BAP algorithm considers are found to be inconsistent. This adversely affects the BAP algorithms runtime in 6 of the 149 examples. The degradation is significant in an extreme case example where 99% of the causal assignments are inconsistent. For this example, the POSET algorithm runs 40 times faster than the BAP algorithm. While the BAP algorithm does not improve the complexity of timed state space exploration, it moves the complexity to a location where it impacts performance less often.

Future work for this research is found in several areas. Relating to performance, determining if a causal configuration is inconsistent currently requires the application of a modified shortest-path algorithm with complexity $O(n^2)$. This is costly in certain examples that consider many inconsistent causal assignments. Research to avoid generating inconsistent causal assignments, as

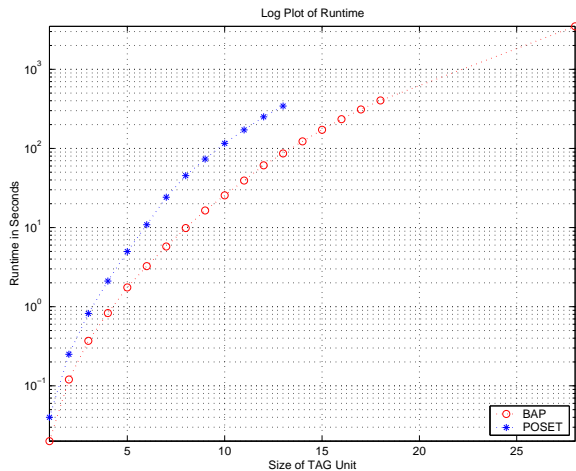


Fig. 4. Plot of runtime versus TAG Units.

well as approximating the fireable set may significantly increase the runtime performance of BAP. The current version of BAP does not directly support gate level models of circuits; thus, it is limited in the scope of its application. *Timed event/level structures* (TELS) are a more natural model for circuits [25]. Extending BAP to support TELS or other circuit models will make it directly applicable to many circuits. Partial order reduction has been successfully applied to further improve timed state space exploration [28, 29, 20, 21]. This technique considers a restricted ordering of concurrent independent transitions in a circuit. The work presented here can be extended to support partial order reduction. Other work of interest includes: adding in correlation to model on-chip delay variations; introducing an inertial delay model into BAP to handle allowable glitching; and creating heuristics that only explore a portion of the timed state space if an error is first found in the untimed state space.

ACKNOWLEDGEMENTS

The authors would like to thank Wendy Belluomini of the IBM Austin Research Laboratories for her significant contributions to this manuscript.

REFERENCES

- [1] C. J. Myers and T. H. Y. Meng. Synthesis of timed asynchronous circuits. *IEEE Transactions on VLSI Systems*, 1(2):106–119, June 1993.
- [2] C. J. Myers. *Computer-aided synthesis and verification of gate-level timed circuits*. PhD thesis, Stanford University, October 1995.
- [3] K. S. Stevens, 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(2):217–228, February 2001.
- [4] Ivan Sutherland and Scott Fairbanks. GasP: A minimal FIFO control. In *Proc. of International Symposium on Advanced*

Research in Asynchronous Circuits and Systems, pages 46–53. IEEE Computer Society Press, March 2001.

- [5] S. Schuster, W. Reohr, P. Cook, D. Heidel, M. Immediato, and K. Jenkins. Asynchronous interlocked pipelined CMOS circuits operating at 3.3–4.5 GHz. In *Proc. International Solid State Circuits Conference*, February 2000.
- [6] H. P. Hofstee, 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*, 1(3):66–74, May 1998.
- [7] S. Sun, L. McMurchie, and C. Sechen. A high-performance 64-bit adder implemented in output prediction logic. In *Advanced Research in VLSI*, pages 213–222, March 2001.
- [8] J. R. Burch. *Trace Algebra for Automatic Verification of Real-Time Concurrent Systems*. PhD thesis, Carnegie Mellon University, 1992.
- [9] S. Campos and E. Clarke. Real-time symbolic model checking for discrete time models. *AMAST Series in Computing: Theories and Experiences for Real-Time System Development*, 1995.
- [10] M. Bozga, O. Maler, A. Pnueli, and S. Yovine. Some progress in the symbolic verification of timed automata. In *Proc. International Conf. on Computer Aided Verification*, 1997.
- [11] M. Bozga, O. Maler, and S. Tripakis. Efficient verification of timed automata using dense and discrete time semantics. In *Correct Hardware Design and Verification Methods*, 1999.
- [12] T. Henzinger, Z. Manna, and A. Pnueli. What good are digital clocks. In *Proc. of International Conf. on Automata, Languages, and Programming*, volume 623 of *Lecture Notes in Computer Science*, pages 545–558. Springer-Verlag, 1992.
- [13] H. Hulgaard. *Timing Analysis and Verification of Timed Asynchronous Circuits*. PhD thesis, Department of Computer Science, University of Washington, 1995.
- [14] R. Alur. *Techniques for Automatic Verification of Real-Time Systems*. PhD thesis, Stanford University, August 1991.
- [15] R. Alur and R. P. Kurshan. Timing analysis in cospan. In *Hybrid Systems III*. Springer-Verlag, 1996.
- [16] D. L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Proc. of the Workshop on Automatic Verification Methods for Finite-State Systems*, 1989.
- [17] K. Larsen, F. Larsson, P. Pettersson, and W. Yi. Efficient verification of real-time systems: Compact data structure and state-space reduction. In *IEEE Real-Time Systems Symposium*, December 1997.
- [18] G. Behrmann, K. Larsen, J. Pearson, C. Weise, and W. Yi. Efficient timed reachability analysis using clock difference diagrams. In *International Conf. on Computer Aided Verification*, volume 1633 of *Lecture Notes in Computer Science*. Springer-Verlag, July 1999.
- [19] J. Møller, J. Lichtenberg, H. Andersen, and H. Hulgaard. Fully symbolic model checking of timed systems using difference decision diagrams. In *Workshop on Symbolic Model Checking*, June 1999.
- [20] J. Bengtsson, B. Jonsson, J. Lilius, and W. Yi. Partial order reductions for timed systems. In *Proc. International Conf. on Concurrency Theory*, pages 485–500, 1998.
- [21] M. Minea. *Partial order reduction for verification of timed systems*. PhD thesis, Carnegie Mellon University, 1999.
- [22] T. G. Rokicki. *Representing and Modeling Circuits*. PhD thesis, Stanford University, 1993.
- [23] C. J. Myers, 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 of Integrated Circuits*, 18(6):769–786, June 1999.
- [24] W. Belluomini and C. J. Myers. Timed state space exploration using POSETs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 19(5), May 2000.
- [25] W. Belluomini, C. J. Myers, and H. P. Hofstee. Timed circuit verification using TEL structures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 20(1):129–146, January 2001.

- [26] Mark R. Greenstreet. Implementing a STARI chip. In *Proc. of International Conf. Computer Design (ICCD)*, pages 38–43. IEEE Computer Society Press, 1995.
- [27] S. Tasiran and R.K. Brayton. Stari: A case study in compositional and heirarchical timing verification. In *International Conf. on Computer Aided Verification*, 1997.
- [28] T. Yoneda, A. Shibayama, B. Schlingloff, and E. M. Clarke. Efficient verification of parallel real-time systems. In Costas Courcoubetis, editor, *Computer Aided Verification*, pages 321–332. Springer-Verlag, 1993.
- [29] F. Pagani. Partial orders and verification of real-time systems. In *Proc. of Formal Techniques in RealTime and Fault-Tolerant Systems*, volume 1135, pages 327–346. Springer-Verlag, 1996.