Timed State Space Exploration Using POSET's

Wendy Belluomini, *Member, IEEE,* and Chris J. Myers, *Member, IEEE*

***Abstract*—This paper presents a new timing analysis algorithm for efficient state space exploration during the synthesis of timed circuits or the verification of timed systems. The source of the computational complexity in the synthesis or verification of a timed system is in finding the reachable timed state space. We introduce a new algorithm which utilizes geometric regions to represent the timed state space and partially ordered sets (POSET's) to minimize the number of regions necessary. This algorithm operates on specifications sufficiently general to describe practical circuits, as well as other timed systems. The algorithm is applied to several examples showing significant improvement in runtime and memory usage.**

***Index Terms*—Formal verification, geometric regions, partial orders, POSET timing, timed asynchronous circuits.**

I. Introduction

THE FUNDAMENTAL difficulty in circuit synthesis and verification is controlling the state explosion problem. The state spaces representing reasonably sized systems are large even if the timing behavior of the system is not considered. The problem gets even more complex when state space exploration is done on timed systems. However, timed state space exploration is crucial to applications such as the synthesis and verification of timed asynchronous circuits as well as the verification of any system that involves real-time constraints.

A number of techniques have been proposed to deal with state explosion. One approach is to minimize the number of interleavings due to concurrency that are explored. These techniques include stubborn sets [1], partial orders [2], or unfoldings [3]. While they have been successful, they only deal with untimed systems. Additionally, approaches that do not consider all interleavings cannot be used for synthesis, since they do not generate a complete state space. Logic synthesis algorithms for timed asynchronous circuits require that all of the boolean states allowed by the state space are found in order to create a correct logic implementation. If the synthesis algorithm is given an incomplete state space, it cannot be guaranteed to generate logic that correctly responds to all inputs to the circuit since there may be reachable states that it is not aware of.

The state space of timed systems is often even larger than the state space of untimed systems and has been more difficult to reduce. The representation of the timing information has a huge impact on the growth of the state space. Timing behavior can either be modeled continuously (i.e., dense-time), where the timers in the system can take on any value between their lower and upper bounds, or discretely, where timers can only take on values that are multiples of a discretization constant. Discrete time has the advantage that the timing analysis technique is simpler and implicit techniques can be easily applied to improve performance [4], [5]. However, the state space explodes if the delay ranges are large and the discretization constant is set small enough to ensure exact exploration of the state space.

Continuous time techniques eliminate the need for a discretization constant by breaking the infinite continuous timed state space into equivalence classes. All timing assignments within an equivalence class lead to the same behavior and do not need to be explored separately. In order to reduce the size of the state space, the size of the equivalence classes should be as large as possible. In the *unit-cube* (or region) approach [6], timed states with the same integral clock values and a particular linear ordering of the fractional values of the clocks are considered equivalent. Although this approach eliminates the need to discretize time, the number of timed states is dependent on the size of the delay ranges and the number of concurrently enabled clocks which can quickly explode for even relatively small systems.

Another approach to continuous time is to represent the equivalence classes as convex *geometric regions* (or zones) [7]–[9]. These geometric regions can be represented by sets of linear inequalities [also known as *difference bound matrices* (DBM's)]. These larger equivalence classes can often result in smaller state spaces than those generated by the unit-cube approach.

While geometric methods are efficient for some problems, their complexity can be worse than either discrete or unit-cube methods when analyzing highly concurrent systems. The number of geometric regions can explode with these approaches since each untimed state has at least one geometric region associated with it for every firing sequence that can result in that state. In highly concurrent systems where many interleavings are possible, the number of geometric regions per untimed state can be huge. Some researchers [10]–[13] have attacked this problem by reducing the number of interleavings explored using the partial-order techniques developed for untimed systems. These algorithms reduce verification time by exploring only part of the timed state space, but the improvement is dependent on the property to be verified. The reduction in interleavings also prevents these techniques from being used for synthesis. Finally, even though the number of interleavings is reduced, in [10], [11] one region is still required for every firing sequence explored to reach a state. If most interleavings

Manuscript received December 31, 1998; revised January 5, 2000. This work was supported by a grant from Intel Corporation, by the National Science Foundation (NSF) under CAREER award MIP-9625014 and an an NSF Traineeship award, by the Semiconductor Research Corporation (SRC) under Contract 97-DJ-487, and by a Defense Advanced Research Projects Agency (DARPA) ASSERT Fellowship. This paper was recommended by Associate Editor T. Szymanski.

W. Belluomini is with the IBM Austin Research Laboratory, Austin, TX 78758-3493 USA (e-mail: wendy@austin.ibm.com).

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

Publisher Item Identifier S 0278-0070(00)04723-0.

need to be explored, these techniques could still result in state explosion.

The algorithm presented in [14]–[16], significantly reduces the number of regions per untimed state by using *partially ordered sets* (or POSET's) of events rather than linear sequences to construct the geometric regions. Using this technique, untimed states do not have an associated region for every firing sequence. Instead, the algorithm generates only one geometric region for any set of firing sequences that differ only in the firing order of concurrent events. This algorithm is shown in [15] to result in very few geometric regions per untimed state. The entire timed state space is explored, so it can be used for both verification [15], [16] and synthesis [17]. However, it is limited to specifications where the firing time of an event can only be controlled by a single predecessor event. This is known as the *single behavioral place restriction.*

In [18], we presented an approximate algorithm for exploring the entire state space with POSET's on a general class of specifications, lifting the single behavioral place restriction. Although it performs better than the algorithm in [15] and [16], in some cases it generates geometric regions which are larger than those actually allowed by the specification, which may lead to the addition of unreachable states to the state space.

This paper presents a new algorithm for timed state space exploration based on geometric regions and POSET's and describes the application of the algorithm to timed Petri nets. The algorithm can also operate on a more general class of specifications, *timed event-rule(ER) structures* [19], but for clarity, the algorithm is presented in the context of the better known timed Petri net model. Unlike the partial-order techniques discussed earlier, the POSET timing algorithm does explore every interleaving between event firings and, therefore, explores all states of the system. This new algorithm dramatically improves the performance of geometric region based techniques on highly concurrent systems, making dense-time state space exploration competitive with discrete-time when the delay ranges are small and far superior when the ranges are large. The performance of the POSET timing algorithm is demonstrated by significant improvement in runtime and memory usage on several examples. These examples include two specifications which are used in [5] to show the disadvantages of continuous time and parameterized versions of a FIFO, counter, selection circuit, and synchronization circuit to show how the POSET method compares with the algorithm presented in [14] and [15].

II. Timed State Space Exploration

The objective of timed state space exploration is to take a specification of the system to be analyzed and produce its reachable state space. This section presents a brief overview of timed Petri nets and the generic algorithm that is used to analyze them.

A. Timed Petri Nets

A one-safe timed Petri net is modeled by the tuple $\langle P, T, F, M_0, \Delta \rangle$ where P is the set of places, T is the set of transitions, $F \subseteq (P \times T) \cup (T \times P)$ is the set of edges, $M_0 \subseteq P$ is the initial marking, and Δ is an assignment of timing requirements to the places. A *marking* is a subset of the places. For a place $p \in P$, the *preset* of p (denoted $\bullet p$) is the set of transitions connected to p (i.e., $\bullet p = \{t \in T | (t,p) \in F\}$), and the *postset* of p (denoted $p\bullet$) is the set of transitions to which p is connected (i.e., $p\bullet = \{t \in T | (p,t) \in F\}$). For a transition $t \in T$ the presets and postsets are similarly defined (i.e, $\bullet t = \{p \in P | (p,t) \in F\}$ and $t\bullet = \{p \in P | (t,p) \in F\}$). Timing is associated with a place p as a timing bound consisting of a lower bound and an upper bound (i.e., $\Delta(p) = \langle l(p), u(p) \rangle$). The lower bound is a nonnegative integer and the upper bound is either an integer greater than or equal to the lower bound or ∞. A transition can fire when all of the places in its preset have had tokens long enough to meet their lower bounds. A transition must fire when all of the tokens in its preset have had tokens long enough to meet their upper bounds. In order for a Petri net to be considered one-safe, the structure of the net must prevent a token from being added to a place which already contains a token.

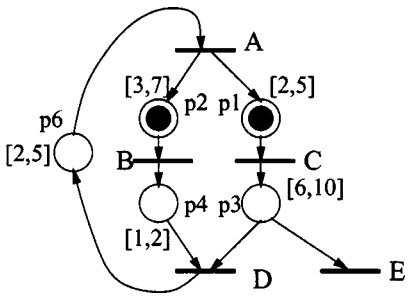


Fig. 1. A timed Petri net.

The behavior specified by a timed Petri net can be defined with a semantics composed of three types of operations: advancement of time, firing of tokens, and firing of transitions. A time-valued clock, c_i, is associated with each marked place p_i. Each clock advances with time and denotes how long the place has been marked. Time is advanced by uniformly increasing the clocks by an amount δ which is less than or equal to *max_advance* of a given marking M. The function *max_advance*, which is defined formally later, is the minimum difference over all marked places between the upper bound of the timing requirement on the place, p_i, and its clock, c_i. A token in place p fires when its clock is between the lower and upper bounds on p, and when it fires, it is colored red (unfired tokens are black). Although token firings are not something usually associated with a Petri net, token firings ensure that a token's age never exceeds the upper bound on its place. A transition fires simultaneously with the last token in its preset firing. When a Petri net has choice, multiple transitions may have their presets become completely marked with red tokens simultaneously. In this case, one of the transitions must fire simultaneously with the last token firing, and the others lose their chance to fire. A transition firing causes all tokens in its preset to be removed and new (black) tokens to be added to all places in its postset.

Fig. 1 shows an example of a timed Petri net. Assume that transition A fires at time zero, creating clocks c_2 and c_1 which are associated with places p_2 and p_1 and initialized to an age of zero. These tokens can fire in either order. The token in place p_2

fires when c_2 is between ages 3 and 7. It is colored red, which immediately causes transition B to fire. The firing of B creates a black token in place p_4 and its clock c_4. The token in place p_1 fires when its clock, c_1 reaches an age between 2 and 5. When the token in place p_1 is colored red, C immediately fires, creating a black token in place p_3 and a clock c_3. The token in p_4 then fires when c_4 is between ages 1 and 2, causing the token in p_4 to be colored red. No transition fires, since no transition has a complete set of red tokens. Next, the token in p_3 fires when its clock is between the ages 6 and 10. Now both D and E have a complete set of red tokens in their presets and either transition can fire. Once a choice is made, the other transition loses its chance to fire.

It is necessary to note at this point the key difference between the standard timed Petri net semantics used in this paper and the *orbital net* specification method used in [15]–[17]. Although orbital nets are similar to timed Petri nets, they have some important differences. The difference that is relevant to timing analysis is that the places of an orbital net are labeled as either *behavioral* or *constraint* and only a single behavioral place can be in the preset of any transition. The timing bounds associated with a behavioral place are used to specify guaranteed timing behavior. The timing requirements associated with a constraint place are used to specify *desired* timing behavior, and they do not affect the actual timing behavior. The *single behavioral place restriction* that is required in orbital nets ensures that the delay between the firing of a transition in the preset of a behavioral place and the firing of a transition in the postset of the same place must always fall between the lower and upper bound of the timing requirement of this place. In other words, no clock can ever exceed the upper bound on its place. In a timed Petri net, however, every place is essentially a behavioral place since any place has the ability to control a transition firing time. When multiple behavioral places are allowed in the preset of a transition, some clocks may exceed their upper bounds. The algorithm presented in this paper eliminates the single behavioral place restriction, allowing an arbitrary number of behavioral places in the preset of any transition, and thus it can analyze any timed Petri net.

B. Timed Firing Sequences

The set of behaviors of a timed Petri net is defined by a set of sequences $S \in ((P^*)(T^*))^*$ where each event (token or transition firing) is numbered sequentially. In order to simplify the notation, a few shorthand operations for dealing with firing sequences need to be defined. When Petri net operations such as postset or preset are used on firing instances, they are assumed to apply to the place containing the token, or the transition that is fired. For example, when considering a sequence σ, $\bullet\sigma_i$ indicates the preset of the transition or place that fires in the ith position in the sequence, σ. When necessary, the function L is used to map an instance of a transition or place in the firing sequence back to the corresponding transition or place in the original net. Finally, the $\in$ operator is used to specify whether a type of firing occurs in the sequence.

The structure of the Petri net defines the set of sequences that are reachable if timing is not considered. The formal definition of the set of reachable sequences requires the definition of the set *firable* which contains the set of transitions which have presets where every place contains a red token.

Definition II.1: The set of firable transitions of a firing sequence $\sigma_{0\cdots n}$ is defined as follows:

$$\begin{aligned}\text{firable}(\sigma_{0\cdots n}) = \{t_i \in T | L(\sigma_n) \in \bullet t_i \wedge \\ \forall p \in \bullet t_i\ \exists \sigma_j \in \sigma_{0\cdots n}\colon (L(\sigma_j) = p) \wedge \\ (\neg\exists \sigma_k \in \sigma_{j\cdots n}\colon (L(\sigma_k) \in \sigma_j\bullet)\}.\end{aligned}$$

The firable set contains all transitions in the postset of σ_n which have red tokens in all of the places in their presets. The definition determines whether a transition, t_i, has a red token in every place in its preset by checking that there is at least one token firing for each place in the preset of t_i which has not been consumed by another transition firing. If the last firing in the sequence is a transition, the firable set is empty since a transition firing at the end of the sequence is not in the preset of any transition t_i. This definition allows us to define the set of sequences which are allowed by the Petri net, $S \in ((P^*)(T^*))^*$, as follows.

Definition II.2: A sequence $\sigma_{0\cdots n} \in S$ if and only if $\forall \sigma_i \in \sigma_{0\cdots n}$

1) $L(\sigma_i) \in P \Rightarrow$
   $(L(\sigma_i) \in M_0 \wedge (\neg\exists \sigma_j \in \sigma_{0\cdots i-1}\colon L(\sigma_j) = L(\sigma_i))) \vee$
   $\exists \sigma_j \in \sigma_{0\cdots i-1}\colon (L(\sigma_j) \in \bullet\sigma_i) \wedge$
   $(\neg\exists \sigma_k \in \sigma_{j\cdots i-1}\colon L(\sigma_k) = L(\sigma_i))$;
2) $L(\sigma_i) \in T \Rightarrow L(\sigma_i) \in \text{firable}(\sigma_{0\cdots i-1})$;
3) $L(\sigma_i) \in P \wedge \text{firable}(\sigma_{0\cdots i}) \neq \emptyset \Rightarrow$
   $L(\sigma_{i+1}) \in \text{firable}(\sigma_{0\cdots i})$.

The first requirement states that if the firing is a token firing in place p, then p is either in the initial marking and this is the first time a token in p fires, or a transition in the preset of p fires earlier in the sequence and the token it generated has not already fired. The second requirement of this definition states that all transitions must be in the firable set when they fire. The last requirement is that if the firable set of a token firing is not empty, the following event in the sequence must be a transition in the firable set of that token.

Each token firing, σ_i, can be associated with the transition firing that created the token by the causal transition function, T_c

Definition II.3: $T_c(\sigma_i, \sigma)$ returns the $\sigma_j \in \sigma_{0\cdots i-1}$ where $(L(\sigma_j) \in \bullet\sigma_i \wedge \neg\exists \sigma_k \in \sigma_{j+1\cdots i-1}\colon L(\sigma_k) \in \bullet\sigma_i)$.

In other words, the function finds the last instance of a transition in the preset of σ_i that occurs before σ_i does. Since the Petri nets are one-safe, this is always the transition firing that created the firing token.

Any sequence of events can be given a *timing assignment* τ which maps an event to the time at which it occurs. For each sequence, $\sigma_{0\cdots n} \in S$, the set of *valid* timing assignments can be defined as follows.

Definition II.4: A timing assignment τ is valid for a sequence $\sigma_{0\cdots n} \in S$ if and only if $\forall \sigma_i \in \sigma_{0\cdots n}$

1) $\tau(\sigma_i) \leq \tau(\sigma_{i+1})$;
2) $L(\sigma_i) \in T \Rightarrow \tau(\sigma_i) = \tau(\sigma_{i-1})$;
3) $L(\sigma_i) \in P \Rightarrow \tau(T_c(\sigma_i, \sigma_{0\cdots n})) + l(\sigma_i) \leq \tau(\sigma_i) \leq$
   $\tau(T_c(\sigma_i, \sigma_{0\cdots n})) + u(\sigma_i)$.

This means that a timing assignment is valid if it corresponds to the order of the firing sequence, all transitions fire simultaneously with the last token in their preset firing, and tokens fire between their lower and upper bounds. A firing sequence $\sigma_{0\cdots n} \in S$ is reachable in a timed Petri net if and only if it can be given a valid timing assignment.

As an example of a timed firing sequence, consider the Petri net in Fig. 1, and assume that both of the tokens shown are created at time zero. Initially either the token in place p_2 or the token in place p_1 can fire, since either can reach its lower bound without the other exceeding its upper bound. Suppose that p_1 fires first. This results in a firing sequence p_1, C, p_2, B. This is an untimed firing sequence and it needs a valid timing assignment. Each place must be given a timing assignment between its upper and lower bounds and transitions must fire simultaneously with their causal places. This firing sequence can be given a valid timing assignment as follows: $(p_1, 4), (C, 4), (p_2, 6), (B, 6)$. The timing assignment would be invalid if the timestamps were not monotonically increasing. For example, $(p_1, 4), (C, 4), (p_2, 3), (B, 3)$ is an invalid timing assignment, as is $(p_1, 4), (C, 4), (p_2, 6), (B, 7)$ since B does not fire simultaneously with its causal place. After the execution of this firing sequence there are tokens in places p_4 and p_3. Assuming the valid timing assignment shown above, the only token that can fire at this point is p_4. The token in p_3 cannot fire, since C fires at four. This means that p_3 cannot fire until ten, and p_4 must fire by eight in order to avoid exceeding its upper bound. The firing of p_4 produces the firing sequence: $(p_1, 4), (C, 4), (p_2, 6), (B, 6), (p_4, 8)$. Now suppose p_3 fires at 12 and both D and E get a complete set of red tokens in their presets. During state space exploration, both possible transition firings are explored. For this example, assume that D fires. This produces a firing sequence $(p_1, 4), (C, 4), (p_2, 6), (B, 6), (p_4, 8), (p_3, 12), (D, 12)$. In this firing sequence transition C is causal to transition D through the token firing in p_3. When D fires, E loses its chance to fire. The firing of D places a token in p_6, whose firing then causes A to fire. The firing of A returns the Petri net to its original state.

C. Exploring the State Space

Cyclic Petri nets have an infinite number of infinitely long firing sequences. Each individual sequence can also have an infinite number of valid timing assignments. State space exploration requires that this infinite set of sequence, timing assignment pairs be divided into a finite set of equivalence classes. The obvious way to do this in the untimed case is to say that two sequences σ and σ' represent equivalent states if the markings that result from executing them are the same $(M(\sigma) = M(\sigma'))$. Therefore, for state space exploration, the *untimed state* of the system is simply the marking. The timed state of the system is represented by the ages of all the currently active clocks. A clock is active in a firing sequence σ if the place it is associated with contains a token when σ has been executed on the Petri net. We define a function T_m that returns the transition firing that created the token that is contained by a place p_i after executing $\sigma_{0\cdots n}$ as follows.

Definition II.5: $T_m(p_i, \sigma)$ returns the $\sigma_j \in \sigma_{0\cdots n}$ where $(L(\sigma_j) \in \bullet p_i \wedge \neg\exists \sigma_k \in \sigma_{j+1\cdots i}\colon (L(\sigma_k) \in \bullet p_i))$.

This function simply returns the latest transition firing in the preset of p_i that occurs in the sequence. If place p_i is not marked after executing the sequence, the function is not defined. This definition can be used to formally define *max_advance*, the function that determines how much time can advance without forcing a token to fire for a firing sequence $\sigma_{0\cdots n}$.

The function *max_advance* returns the minimum difference over all of the unfired (black) tokens between the upper bound on the place containing the token and the current age of the token in the firing sequence. This is the maximum amount of time that can pass before some token must fire or exceed its upper bound.

Definition II.6: The max advance function is defined as follows:

$$\text{max\_advance}(\sigma_{0\cdots n}, \tau) = \min_{p_i\colon M(\sigma_{0\cdots n})[p_i]=black} (u(p_i) - (\tau(\sigma_n) - \tau(T_m(p_i, \sigma_{0\cdots n})))).$$

The *max_advance* function is used to determine all of the possible clock ages that are allowed by a timing assignment, τ, for a sequence $\sigma_{0\cdots n}$.

Definition II.7: For each place $p_i \in M(\sigma_{0\cdots n})$, the age of c_i must satisfy the inequality

$$\tau(\sigma_n) - \tau(T_m(p_i, \sigma_{0\cdots n})) \le c_i \le (\tau(\sigma_n) - \tau(T_m(p_i, \sigma_{0\cdots n})) + max\_advance(\sigma_{0\cdots n}, \tau).$$

This means that a clock is no younger than the time difference between the firing time of the transition that created it and the firing time of last event to fire in the sequence, and must not exceed an age that would force another token to fire. The set of values for a clock c_i that are allowed by a timing assignment τ are referred to as $\tau(c_i)$. Since the ages of the clocks determine which future states are possible, two sequences σ and σ' can be said to have the same *timed state* if $M(\sigma) = M(\sigma')$ and τ is a valid timing assignment to σ if and only if there is a valid timing assignment τ' to σ' such that $\forall p_i \in M(\sigma)\colon \tau(c_i) = \tau'(c_i)$. This definition means that if the clock ages that can result from firing the two sequences are the same, the two sequences result in the same futures and are therefore considered equivalent.

Suppose that there exists a representation R which gives the ages of the clocks allowed by a firing sequence. A timed state, TS, then consists of $M \times R$. Using this representation, the timed state space of the timed Petri net can be explored using the algorithm in Fig. 2. The algorithm does a depth-first search of the timed state space, finding all the timed states that are reachable. The *find_enabled* function uses timing information to determine which actions should be included in the *action_list*, *AL*. An action is a place, transition pair. At least one pair is added to the list for every token firing that is possible given the timing information in R. If firing the token in place p would result in a marking where a transition t has all red tokens in its preset, then the pair (p, t) is added to the list. If firing p does not cause any transition to have all red tokens in its preset, then the pair $(p, t_\emptyset)$ is added to the list. The transition $t_\emptyset$ is used to indicate that this token firing cannot cause any transitions to fire. Note that multiple pairs for the same token firing can be placed in the action list if that place is part of a choice. This ensures that all

```
set_of_states find_timed_states(timed Petri net N){
    R =find_initial(M0);
    timed_state TS = M0 × R;
    M = M0;
    set_of_states S = {TS};
    action_list AL = find_enabled(TS, N);
    bool done=false;
    while (¬done){
        action (p, t)=head(AL);
        push(⟨TS, tail(AL)⟩);
        M(p)=red;
        if (t ≠ t∅) then{
            for each pi ∈ •t, M(pi) = empty;
            for each pi ∈ t•, M(pi) = black;
        }
        R = update(N, R, p, t);
        TS = M × R;
        if (TS ∉ S)then
            S = S ∪ {TS};
            AL=find_enabled(TS, N);
        else
            if (stack is not empty) then (TS, AL)=pop();
            else done = true;
    }
    return S;
}
```

Fig. 2. Timed state space exploration.

possible transition choices are eventually explored by the algorithm. Once the algorithm has computed the action list, it selects the first element of the list (i.e. $head(AL)$) as the action to execute. Since the algorithm must explore the execution of the remaining actions later, it uses the "push" operation to add the remainder of the list ($tail(AL)$) to the stack. The algorithm then executes the action by coloring the fired token red, and, if necessary, updating the marking to reflect that a transition has fired. It then updates the representation of the timing information, R, and checks whether the resulting timed state has been seen before. If it has not been seen before, a new list of actions to execute is computed and the algorithm continues. Otherwise, the algorithm pops a timed state and the list of events that have not yet been explored for that state off the stack. When a state that has been seen before is reached and there are no unexplored actions on the stack, the entire timed state space has been found.

Untimed states are only explored if they can be reached given the timing information in the specification. This can eliminate large portions of the untimed state space for some designs. Many states that are reachable without timing information are not reachable given the timing constraints in the specification. However, the algorithm explores the entire *timed* state space, and the size of the timed state space depends on the representation chosen for the timing information. The algorithm presented in this paper discusses how to represent the timing information with geometric regions and POSET's so that the cost is minimized.

III. Geometric Algorithm

The timing analysis algorithm presented here uses geometric regions (also known as zones) to represent the timing information within a timed state. The minimum and maximum age differences of all the clocks are stored in a constraint matrix R. Each entry r_{ij} in the matrix R has the value $max(c_j - c_i)$, which is the maximum age difference of the clocks. A dummy clock c_0 whose age is always zero is also included. The maximum age difference between c_i and c_0 (r_{0i}) is the maximum age of c_i and the maximum age difference between c_0 and c_i (r_{i0}) is the negation of the minimum age of c_i. Note that R only needs to contain information on the timing of currently marked places, not on every place in the net. This particular way of representing timed regions was first introduced in [7]. This constraint matrix (also known as a difference bound matrix) represents a convex $|\{p: M(p) = black\}|$ dimensional region. Each dimension corresponds to an unfired token, and the age at which it fires can be anywhere within the space.

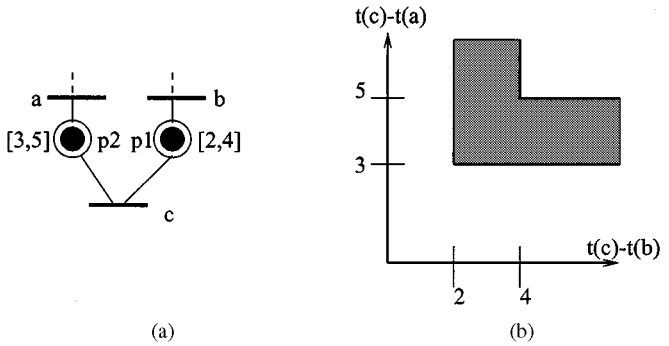


Fig. 3. (a) Net with multiple behavior places. (b) Nonconvex region that represents its timing behavior.

Many matrices can be used to represent the same region in space since some entries may be underconstrained. However, there is a canonical representation where every constraint is *maximally constraining*. A set of constraints is maximally constraining if each constraint can reach its maximum value for some timing assignment without violating any of the other constraints. In the algorithm, the matrix is made maximally constraining through a process called *recanonicalization*. Recanonicalization takes a matrix R where some of the r_{ij}'s are greater than $max(c_j - c_i)$ and produces a matrix where all the r_{ij}'s have their maximum allowed value. The assignment of the r_{ij}'s so that they all have their maximum value is always unique, so the algorithm can determine when a given region is equivalent to or contained in a region that has been seen before. Recanonicalization is essentially the all pairs shortest path problem and can be done in $O(n^3)$ time with Floyd's algorithm [7].

Geometric regions are used in `Orbits` [15], [16] to do timed state space exploration on specifications with the single behavioral place restriction. This restriction is made in `Orbits` to ensure that the geometric regions that represent the time behavior of the system are always convex. If the values of clocks can exceed their upper bounds, the regions representing the time behavior may not be convex. Fig. 3 shows an example of this. In this specification, either the separation between a and c must not exceed five, or the separation between b and c must not exceed four. Since only one of the upper bound constraints needs to be met, the resulting region is nonconvex. Since Floyd's algorithm only works on convex regions, this must be avoided. However, when tokens are allowed to fire independently of transitions, as

```
void update(time Petri net N, geometric region R,
            marking M, place p, transition t) {
    if (R[index(p)][0] > -l(p))
        R[index(p)][0] = -l(p);
    recanonicalize(R);
    project(R, index(p));
    if (t ≠ t∅)
        forall (pi ∈ t•)
            R[0][index(pi)]=0;
            R[index(pi)][0]=0;
            forall(pj : R(pj) = black)
                R[index(pj)][index(pi)]=R[index(pj)][0];
                R[index(pi)][index(pj)]=R[0][index(pj)];
        }
    }
    forall(pj : R(pj) = black)
        R[0][index(pj)]=u(pj);
    recanonicalize(R);
}
```

Fig. 4. Procedure for updating the geometric region.

discussed in Section II, clocks can no longer exceed their upper bounds, and the regions can be guaranteed to be convex. In this example, two regions would be generated to cover the space shown in Fig. 3.

The algorithm in Fig. 4 shows how the function for updating timing information used in Fig. 2 is implemented with geometric regions. The function takes as input the Petri net specification, the constraint matrix, the place containing the token chosen to fire, and the transition it causes. The *index* function used in the algorithm takes a place, and returns the index in the constraint matrix that corresponds to its token. The first step of the function is to check if the minimum age of the firing token's clock allowed by the matrix is greater than or equal to the lower bound on the age of the token. If it is not, the lower bound on the age of the token in the matrix is set to the minimum age of the token. This ensures that the minimum age of each clock is no less than the difference between the time it is created and the time that the last event in the sequence fires. The row and column corresponding to the fired token is then removed from the matrix by the *project* operation. Next, the algorithm adds clocks for newly created tokens if a transition fires(i.e if the firing transition is not $t_\emptyset$). All of the places in its postset have new tokens, and new entries in the matrix must be created for them. When a token is initially created, its age is zero, so the entries in the matrix for its minimum and maximum age are set to zero. Age relationships between the new tokens and the previously existing ones must also be entered in the matrix. The maximum age difference between a new token and any previously existing token is the maximum age of the previously existing token. Therefore, the new maximum age difference entries are copied from row zero of the matrix which contains the maximum ages of existing tokens. The minimum age difference between the new token and a previously existing token is the minimum age of the previously existing token, and this minimum age is copied from column zero of the matrix. Finally, the algorithm sets the maximum age of each token to the maximum age on its place and recanonicalizes. This allows time to advance as far as possible without causing any token to exceed its maximum age. The new region

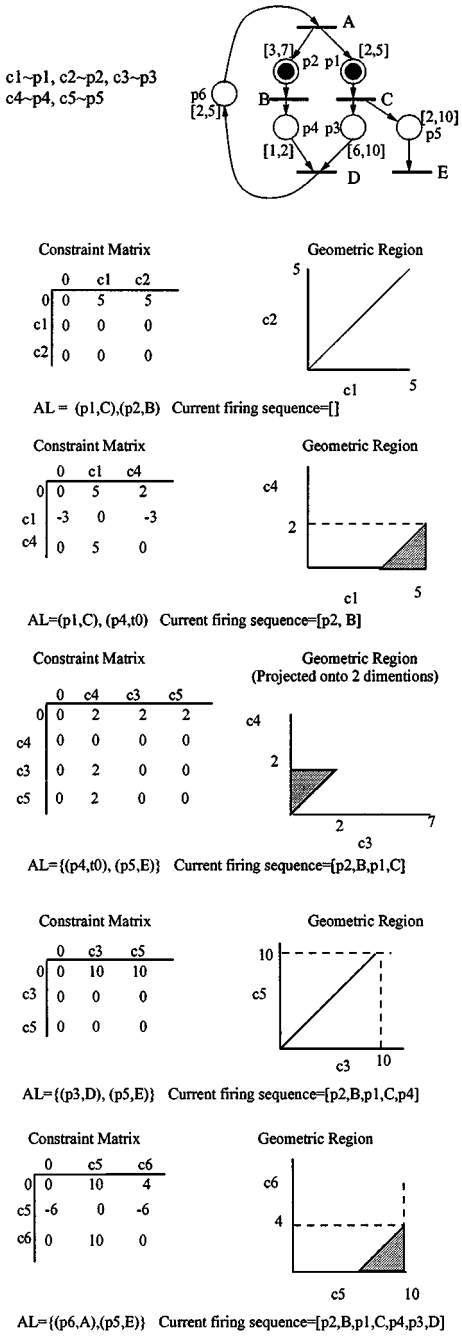


Fig. 5. Firing rules.

now represents all clock ages that are possible given the firing sequence that is currently being explored.

Fig. 5 shows an example of how the geometric algorithm would be applied to the simple timed Petri net shown at the top of the figure. The first column shows the constraint matrix at each step and the second column shows the region in space represented by the matrix. The recanonicalization procedure that is applied after each step is not shown here, but is described in detail in [14]. Initially, places p_1 and p_2 are marked with black tokens, which are given clocks c_1 and c_2, respectively. The initial constraint matrix indicates that the maximum age for both clocks is five. Since the lower timing bounds on both p_1 and p_2 are less than five, they are both added to the action list. The place p_1 is paired with transi-

tion C since its firing allows C to fire, and p_2 is paired with B since its firing allows B to fire. The pair (p_2, B) is chosen to fire. The clock for p_2 is projected out of the constraint matrix, and the matrix is constrained so that that all clocks that existed when p_2 fired must have a minimum age of three. A new clock is added for the new token, p_4. It must be between three and five time units younger than the clock for p_1 since the clock for p_1 has an age between three and five time units when it is added. The action list now contains (p_1, C) and $(p_4, t_\emptyset)$. The firing of p_4 is paired with $t_\emptyset$ since when p_4 fires the other place in the preset of transition D does not contain a red token. The pair (p_1, C) is chosen to fire next, causing tokens to be placed in p_3 and p_5. The new action list contains firings for p_4 and p_5 but not p_3 since the lower bound on p_3 is six, and the maximum age for p_3 allowed by the matrix is two. Next, p_4 is chosen to fire. It does not cause a transition to fire, so no new clocks are added to the constraint matrix. After p_4 fires, the maximum age of the token in p_3 can advance to ten, allowing it to be placed on the new action list, paired with the transition D. The token in p_3 can then fire, producing the last matrix and region in the figure.

This algorithm allows us to analyze any timed Petri net including those with multiple behavioral places. It can, however, generate a large number of regions since at least one region is generated for each firing sequence explored. Section IV introduces the POSET algorithm, which dramatically reduces the number of regions needed to represent the timed state space.

IV. Partially Ordered Sets

While the geometric algorithm described above eliminates the single behavioral place restriction, the number of geometric regions the algorithm generates can explode for highly concurrent timed systems [15], [5]. In [15], an algorithm is described that uses POSET's instead of linear sequences during state space exploration to mitigate this state explosion problem. POSET timing techniques take advantage of the inherent concurrency in the Petri net and prevent additional regions from being added for different sequences of firings that allow the same set of future behaviors. This results in a compression of the state space into fewer, larger geometric regions that, taken together, contain the same region in space as the set of regions generated by the standard geometric technique.

The semantics described in Section II require two sequences to be in different equivalence classes if they result in the same marking but allow different sets of values to be assigned to the active clocks. This is based on the observation that if two sequences σ and σ' result in the same marking, and allow the same set of values to be assigned to the active clocks, a timed state is reachable from σ if and only if it is reachable from σ'. However, in some cases the requirement that the allowable clock values for both sequences must be the same is too restrictive. With additional analysis, it is possible to derive a set of clock values for a marking $M(\sigma)$, which are guaranteed to be allowed by some firing sequence σ' where $M(\sigma) = M(\sigma')$. In other words, given a firing sequence, σ, it is possible to determine not only which clock values are allowed for σ, but also a set of clock values that are guaranteed to be allowed for some other reachable firing sequence, σ', which fired concurrent events in a different order. This allows the POSET algorithm to preemptively construct a larger region for σ, knowing that eventually a firing sequence, σ' for which the clock values are allowed, will be found during the depth first search. When σ' is found, the clock values that it allows are already represented in the region that is constructed for σ, and an additional region is not generated. This effectively combines the regions for σ and σ' and reduces the number of regions in the state space.

The computation necessary to determine this larger set of clock values is based on the concept of *causality*.

Definition IV.1: The function $causal(\sigma, \sigma_i, \sigma_j)$ returns true when $\sigma_i = T_c(\sigma_{j-1}, \sigma)$.

Intuitively, this means σ_i is causal to σ_j if the firing σ_i created the token whose firing is the last in the preset of σ_j to fire, and thus controls the firing time of σ_j.

When σ_i is causal to σ_j, the time separation between σ_j and σ_i is always less than the upper bound on the place in the preset of σ_j that is marked by the firing of σ_i. This is formalized in the following lemma.

Lemma IV.1: If σ_i is causal to σ_j in σ then the inequality: $\tau(\sigma_j) \leq \tau(\sigma_i) + u(\sigma_{j-1})$ is true for all valid timing assignments to σ.

The proof of this lemma (as well as all following lemmas and theorems) is given in the Appendix. There is also a more general property that holds between any two transition firings σ_i and σ_j. If the firing σ_i creates a token that is used in firing σ_j, then the minimum time separation between the firings σ_i and σ_j is at least the lower bound on the place containing that token.

Lemma IV.2: If $L(\sigma_k) \in \bullet\sigma_j \wedge T_c(\sigma_k, \sigma) = \sigma_i$ in σ then the inequality $\tau(\sigma_j) \geq \tau(\sigma_i) + l(\sigma_k)$ is true for all valid timing assignments, τ, to σ.

If the transition fired by σ_i has no choice places in its preset, the lower and upper bounds on these inequalities can always be met by some reordering of the firing sequence that is in S. In order to prove this, a few more definitions and lemmas are required. The first is the definition of the *required* set, which contains the set of events in σ that must fire in order for the firing of an event σ_i to meet the requirements specified by Definition II.2(1) and (2). If σ_i is the first firing of a token that is in the initial marking, then the required set of σ_i is empty. If σ_i is a token firing, and is not the first firing of an initially marked place, then its required set contains its causal transition. If σ_i is a transition firing, then firings of all of the places in its preset are required for it to fire. The last condition is the transitive closure of these requirements, if an event is required for the firing of σ_i then all events required to fire it are also included in the required set for σ_i. These requirements are defined formally as follows.

Definition IV.2: The required set of an event σ_i in $\sigma_{0\cdots n}$ $(required(\sigma_i, \sigma_{0\cdots n}))$ is defined recursively as follows:

1) $L(\sigma_i) \in M_0 \wedge \neg\exists\sigma_j \in \sigma_{0\cdots i-1}\colon L(\sigma_j) = L(\sigma_i) \Rightarrow$ $required(\sigma_i, \sigma_{0\cdots n}) = \emptyset$;
2) $L(\sigma_i) \in P \wedge$ $\neg(L(\sigma_i) \in M_0 \wedge \neg\exists\sigma_j \in \sigma_{0\cdots i-1}\colon L(\sigma_j) = L(\sigma_i)) \Rightarrow$ $T_c(\sigma_i, \sigma_{0\cdots n}) \in required(\sigma_i, \sigma_{0\cdots n})$.;
3) $L(\sigma_i) \in T \wedge$ $L(\sigma_j) \in \bullet\sigma_i \wedge (\neg\exists\sigma_k \in \sigma_{j+1\cdots i}\colon L(\sigma_k) = L(\sigma_j)) \Rightarrow$ $\sigma_j \in required(\sigma_i, \sigma_{0\cdots n})$.;

4) $\sigma_j \in required(\sigma_k, \sigma_{0\cdots n}) \wedge$
$\sigma_i \in required(\sigma_j, \sigma_{0\cdots n}) \Rightarrow$
$\sigma_i \in required(\sigma_k, \sigma_{0\cdots n})$ (Transitive closure.).

A sequence σ' which is created from σ by changing the firing order of the events is referred to as a *reordering* of σ. The reordering is described using a reordering function ρ which returns the firing number of each event in the reordered firing sequence. A sequence σ' is the result of a reordering $\rho(\sigma)$ if and only if $\forall \sigma_i \in \sigma$: $(\rho(\sigma_i) = x \Rightarrow \sigma'_x = \sigma_i)$. A firing $\sigma_i \in \sigma$ is equal to a firing $\sigma'_x \in \sigma'$ if $L(\sigma_i) = L(\sigma'_x)$, and they are both the nth firing of $L(\sigma_i)$ in their respective sequences. It can be shown that if ρ meets the following conditions, then $\rho(\sigma) \in S$ if σ in S. The first requirement is that if σ_j is in the required set of σ_i, then σ_j cannot be made to fire after σ_i in the new sequence. The second requirement is that if a place firing σ_i is followed by a transition firing σ_{i+1}, then σ_i and σ_{i+1} are also consecutive in the reordering. The third requirement deals with choice places. If σ_i is the firing of a choice place, then all of the firings that occur between σ_i and the next firing of a transition in its postset (denoted $next\_t(\sigma_i, \sigma)$) cannot be reordered arbitrarily. A transition which is in the postset of σ_i, but is not $next\_t(\sigma_i, \sigma)$) is referred to as a *conflicting transition*. Once the choice place, σ_i fires, no token firing, σ_k, which is in the preset of a transition that conflicts with $next\_t(\sigma_i, \sigma)$ can be reordered to occur before a token firing σ_j which is in the preset of $next\_t(\sigma_i, \sigma)$. This restriction is necessary to make sure that choices are not resolved differently in the reordered firing sequence and the original firing sequence These conditions are defined formally as follows.

Definition IV.3: A reordering ρ of $\sigma_{0\cdots n}$ is valid if:

1) $\sigma_j \in required(\sigma_i, \sigma) \Rightarrow \rho(\sigma_j) < \rho(\sigma_i)$;
2) $L(\sigma_i) \in T \Rightarrow \rho(\sigma_i) = \rho(\sigma_{i-1}) + 1$;
3) $L(\sigma_i) \in P \wedge |\sigma_i \bullet| > 1 \wedge \sigma_m = next\_t(\sigma_i, \sigma) \Rightarrow$
$\forall \sigma_j \in \sigma_{i+1\cdots n}, \forall \sigma_k \in \sigma_{j+1\cdots n}$: $(L(\sigma_j) \in \bullet\sigma_m) \wedge$
$(L(\sigma_k) \notin \bullet\sigma_m) \wedge (\exists t \in \sigma_k\bullet : t \in \sigma_i\bullet) \Rightarrow$
$\rho(\sigma_j) < \rho(\sigma_k)$.

If a sequence σ is in S, then any reordering of σ, $\rho(S)$, is also in S.

Lemma IV.3: Given $\sigma \in S$ and ρ is a valid reordering of σ, if $\sigma' = \rho(\sigma)$ then $\sigma' \in S$.

Lemma IV.3 can be used to redefine what it means for two sequences to have the same timed state. Previously two sequences are defined to result in the same timed state if every set of clock ages that could result from a valid timing assignment to one of the sequences could also result from a valid timing assignment to the other sequence. The definition of a valid timing assignment is based on the concept of assigning firing times to events that fire in sequence. Therefore, a valid timing assignment must assign firing times that are consistent with the order that events fire in the sequence. Timing assignments that allow events to fire out of order can be made if it is guaranteed that a sequence that can fire in order with that timing assignment exists. The set of valid reorderings of a sequence σ defines when such a reordering exists by creating a partial order that all of the sequences that can result from reordering σ must conform to.

More formally, a sequence σ is used to define a partial order as follows.

Definition IV.4: A partial order consists of a set (S) and an ordering relationship (>). The partial order defined by a sequence σ is as follows:

1) $S = \{\sigma_i \in \sigma\}$;
2) $>= \sigma_i > \sigma_j$ if and only if $\forall \rho(\sigma)$:(ρ is valid $\Rightarrow$
$\rho(\sigma_i) > \rho(\sigma_j)$).

The set of firing sequences that can be derived by reordering the firings in σ in a way that conforms to the partial order defined by σ is referred to as $PO(\sigma)$. This set can be used to define a new set of valid timing assignments for σ.

Definition IV.5: A timing assignment τ is PO valid for σ if $\exists \sigma' \in PO(\sigma)$: τ is valid for σ'.

Two firing sequences σ and σ' can now be considered partial-order equivalent if $M(\sigma) = M(\sigma')$ and τ is a PO valid timing assignment to σ if and only if there is a PO valid timing assignment τ' to σ' such that $\forall p_i \in M(\sigma)$: $\tau(c_i) = \tau'(c_i)$. This definition eliminates the ordering of concurrent events from consideration in creating the equivalence class and, therefore, allows the equivalence classes to be larger. When a sequence σ is explored, a geometric region can be created that includes all of the timing assignments that are PO valid for σ. Since a timing assignment is only PO valid for σ if there is some untimed reachable firing sequence for which it is valid, even though it may violate the ordering of σ, it is guaranteed that the search eventually finds a firing sequence for which it is valid. When this sequence is explored, the search can immediately backtrack, thus eliminating timed states.

In order to be able to build this larger region based on the partial order implied by a firing sequence, the algorithm must know what timing assignments are PO valid for σ while σ is being explored. Lemmas IV.1 and IV.2 show that there are upper and lower bounds on the separation between transition firing times that depend only on causality. If causality is preserved in a reordering of a firing sequence, these upper and lower bounds are preserved as well. Therefore, if for all sequences σ' in $PO(\sigma)$, $causal(\sigma, \sigma_i, \sigma_j) \Rightarrow causal(\sigma', \sigma'_{(\rho(\sigma_i))}, \sigma'_{(\rho(\sigma_j))})$, then all valid timing assignments to sequences in $PO(\sigma)$ satisfy the inequalities in the lemmas. The next lemma states that causality is preserved by reordering.

Lemma IV.4: If $causal(\sigma, \sigma_i, \sigma_j)$ and ρ is a valid reordering used to map σ to σ', then $causal(\sigma', \sigma'_{(\rho(\sigma_i))}, \sigma'_{(\rho(\sigma_j))})$

If the geometric regions representing valid timing assignments are created based on Lemmas IV.1 and IV.2, then the entire state space is found, but it may contain invalid timing assignments since the lemmas do not guarantee that there are valid timing assignments that fall in the entire range allowed by the inequalities. This means that although all states in the state space are found, some extra states may be found as well. This may result in false negative verification results or suboptimal synthesized circuits. In order to explore the state space exactly, we need to be able to determine from the sequence σ, the minimum value of x and the maximum value of y, for which if σ_i is causal to σ_j, there exists a valid reordering of σ, such that $x < \tau(\rho(\sigma_i)) - \tau(\rho(\sigma_j)) \leq y$. Lemmas IV.1 and IV.2 provide bounds for these values and if σ_j does not have any choice places in its preset, x and y are exactly the bounds from Lemmas IV.1 and IV.2.

Theorem IV.1: For any firing sequence $\sigma \in S$ that has a valid timing assignment, if σ_i is causal to σ_j, and σ_j does not have a choice place in its preset ($\neg\exists\sigma_k$: $\bullet\,\sigma_j \cap \bullet\sigma_k \neq \emptyset$), there exists a firing sequence $\sigma' \in S$ created from a reordering ρ for which there is a valid timing assignment τ' where $\tau'(\sigma'_{(\rho(\sigma_i))}) + u(\sigma_{j-1}) = \tau'(\sigma'_{(\rho(\sigma_j))})$.

Intuitively, this theorem means that, if transition firing σ_i is causal to transition firing σ_j and the transition fired by σ_j does not have a choice place in its preset, then the maximum separation between firings σ_i and σ_j over all valid reorderings of the sequence is defined. There is always a reordering with a valid timing assignment where the age of the last place to fire in the preset of σ_j reaches its upper bound. Therefore, there is always a reordering where the maximum separation between σ_i and σ_j is $u(\sigma_{j-1})$. This means it is possible to determine the maximum separation between σ_i and σ_j over all valid firing sequences where σ_i is causal to σ_j by examining a single firing sequence σ.

Theorem IV.2: For any firing sequence $\sigma \in S$ that has a valid timing assignment, if σ_i is a transition firing in σ, there exists at least one place firing σ_j: $L(\sigma_j) \in \bullet\sigma_i$ for which in some firing sequence $\sigma' \in S$ constructed from ρ there exists a valid timing assignment τ' in which $\tau'(\sigma'_{(\rho(T_c(\sigma_j,\sigma))})) + l(\sigma_j) = \tau'(\sigma'_{(\rho(\sigma_i))})$.

This theorem deals with minimum separations between transition firings. Unlike Theorem IV.1 it does not have the restriction that the transition firing in question has no choice places in its preset. Intuitively, the theorem states that for every transition firing σ_i, there exists a reordering with a valid timing assignment where σ_i fires at the minimum time allowed by the places in its preset. This minimum time is the earliest time at which all of the places in the preset of σ_i have tokens whose ages meet the lower bounds on their places. The theorem shows that there is always a sequence where σ_i fires at this minimum time. Therefore, it is possible to determine the minimum firing time of σ_i over all valid reorderings of σ, by examining a single sequence.

These theorems are sufficient to construct a geometric region based state space representation for the set of timing assignments that are possible in a specification if it contains no choice places. When there are choice places, the analysis becomes more complex. Although Theorem IV.2 still applies, Theorem IV.1 only applies to transitions that do not have choice places in their presets. When a transition firing σ_i has a choice place in its preset, the maximum time separation between a firing of σ_i and its causal transition σ_j may not be able to reach $u(\sigma_{j-1})$ for any valid reordering of σ. This is illustrated in Fig. 6. Assume that t_0, t_1, and t_2 all fire at time zero. If t_2 is causal to t_4, p_3 must fire after p_2 and before p_1. If p_3 fires before p_2, then p_2 is causal to t_4. If p_3 fires after p_1, then t_3 fires instead of t_4. The transition t_2 can only be causal to t_4 if p_3 fires between one and two time units after it becomes marked, and it cannot reach its upper bound, 100. It is possible to compute the upper bound for transitions with choice places in their presets, but the computation is complex, and in the worst case can involve examining the entire firing sequence. Therefore, when a transition t_i with a choice place in its preset fires, the maximum separation between t_i and its causal transition is set to the maximum allowed by the *current* firing sequence. This means that all timing assignments to the firing of t_i that are in the region are

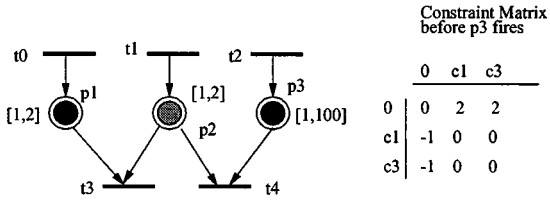


Fig. 6. A choice computation.

valid for the current firing sequence. Therefore, no reordering of the token firings in the preset of t_i is needed for t_i to fire at the computed upper bound. This ensures that the resulting region is exact, but the restriction results in more regions being generated than may be necessary.

The result of the restriction on reorderings imposed by choice places is that the worst case complexity of the POSET algorithm when applied to Petri nets with choice is no better than the geometric algorithm presented in Section III. However, in practice most circuit specifications are dominated by concurrent behavior rather than choice behavior. The POSET algorithm still shows significant benefit over the geometric algorithm in such a specification. In a specification consisting mostly of choice behavior, concurrency is limited and, therefore, state explosion is less of a problem. In this kind of specification the POSET algorithm essentially reduces to the geometric algorithm with some additional overhead. Alternatively, the geometric algorithm can be used directly on such a specification. Finally, we have found that for most circuit specifications, the additional restriction imposed by the choice places has little impact on the generated state space. If the restriction is eliminated, larger regions are generated, which are supersets of the actual regions, but new markings are rarely found. Therefore, eliminating the restriction produces a conservative and faster solution. If this is acceptable, transitions with choice places in their presets can be treated the same as other transitions.

V. POSET Algorithm

The POSET algorithm creates the larger equivalence classes discussed in Section IV by maintaining a *POSET* matrix in addition to the constraint matrix discussed in Section II. The POSET matrix stores the minimum and maximum possible separations between transition firing times that can still effect future behavior. These separations represent the set of possible timing assignments to the partial order that is created by the firing sequence currently being explored. At each iteration, the separations in the POSET matrix are copied into the entries of the constraint matrix that restrict the differences in the ages of the tokens. Transitions are projected out of the POSET matrix when their timing information is no longer needed, so the algorithm only needs to retain and operate on local timing information.

When a new transition fires and is added to the POSET matrix, the minimum and maximum time separations between its firing time and the firing times of all other transitions in the matrix is determined. They must only allow timing assignments to the partial order that are valid. This means that the separations must be consistent with the causality in the firing sequence being explored. This is the major difference between

```
void update_POSET(time Petri net N, POSET matrix PM, constraint matrix R, causal place p,
                  causal transition t_c, firing transition t_f, marking M) {
  forall(t_i ≠ t_f : t_i is represented in PM)
    if(t_i = t_c){
      PM[index(t_i)][index(t_f)] = -l(p);
      if (∃p_c ∈ •t_f : |p_c •| > 1) PM[index(t_f)][index(t_i)] = R[0][index(p)];
      else PM[index(t_f)][index(t_i)] = u(p);
    } elseif(t_i ∈ •p_i ∧ p_i ∈ •t_f){
      PM[index(t_i)][index(t_f)] = -l(p_i);
      PM[index(t_f)][index(t_i)] = ∞;
    } else{
      PM[index(t_i)][index(t_f)] = ∞;
      PM[index(t_f)][index(t_i)] = ∞;
    }
  recanonicalize(PM);
  forall (t_i : is represented in PM)
    if (¬∃p_i ∈ t_i• : p_i is marked) project(PM, index(t_i));
}
```

Fig. 7. Procedure for updating the geometric region.

the POSET technique described here and the work presented in [14] and [15]. In [14] and [15], it is not necessary to use explicit causality information since the causal place is always the behavioral place. With multiple behavioral places, causality must be considered in order to compute a correct POSET matrix.

Fig. 7 shows the algorithm which is called by the function which updates the region. The algorithm first examines all of the transitions currently in the POSET matrix (PM) and determines what relationship each transition has to the firing transition. This is quite simple since all of the information necessary to do this is present in the firing sequence being explored. If a transition in PM is the causal transition for the firing transition t_f, then the minimum separation in PM is set to the lower bound on the causal place, p. If there is a choice place in the preset of t_f then the maximum separation is set to the maximum age of p that is allowed by the constraint matrix. This sets the separation to the maximum allowed by the current firing sequence instead of the maximum allowed over all valid reorderings of the current sequence. With this restriction, when a transition with a choice place in its preset fires, the maximum timing assignment that it can have is limited by the maximum amount time can advance before another place must fire. For example, consider the choice place in Fig. 6 and assume that transitions t_0, t_1, and t_2 all fire at the same time. The constraint matrix that results after the token in p_2 fires is shown in the figure. If the token in p_3 fires next, the transition t_4 fires. Transition t_2 is causal to t_4 through p_3. The maximum bound on p_3 is 100, but this is not the value placed into the POSET matrix by the algorithm. Since t_4 has a choice place in its preset, the value two, which is the maximum age of p_3 in the *current* constraint matrix, is used instead.

If there is no choice place in the preset of t_f, then the only limitation is the upper bound on the causal place, and the separation between t_f and t_c is set to the upper bound on p. If a transition is not causal, but does create one of the tokens used in the firing of t_f, then a constraint is added indicating that the lower bound on the place containing that token must be met, but no upper bound is set. If a transition is unrelated to the firing transition, then no constraints are set. Once all of the constraints have been added to the POSET matrix, it is recanonicalized, causing all of the unconstrained entries to be set to the maximum value allowed by the constraints. Finally, any transitions that are no longer in the preset of marked places are removed from the matrix.

The constraints computed in the POSET matrix can then be used to compute a new constraint matrix when a transition fires. The constraint matrix contains the possible differences in the ages of marked tokens. Since the difference in these ages depends on when the tokens are created, if the minimum and maximum differences between the firing times of all transitions in the presets of marked places is known, the differences in token ages are known as well. When the POSET algorithm is used, the differences in transition firing times that are stored in the POSET matrix are used to generate all of the constraints on the differences in token ages in the constraint matrix. After these constraints are copied to the constraint matrix, time is allowed to advance by setting the maximum age of all the tokens to the upper bounds on their respective places. The constraint matrix is then recanonicalized, resulting in a new geometric region. The recanonicalization process may further constrain some of the inequalities that are copied from the POSET matrix since the POSET inequalities do not take into account the fact that no token may exceed the upper bound on the place holding it.

Fig. 8 shows timing analysis based on POSET's applied to the small timed Petri net shown at the top of the figure. This example shows how the algorithm solves two of the problems that occur when using geometric regions for timed state space exploration: region splitting and multiple behavioral places. In this example, initially the action list consists of $(p_1, C), (p_2, B)$ indicating that both p_1 and p_2 have tokens old enough to fire, and both token firings result in the firing of a transition. The POSET matrix contains a single transition, A. The constraint matrix shows that the maximum age of both p_1 and p_2 is five. From this timed state, either the token in place p_1 can fire, causing C to fire, or the token in place p_2 can fire, causing B to fire. In this example, p_2 and B are chosen. The POSET matrix now contains the minimum and maximum separations between the firing times of A and B. The values are copied into the constraint matrix, since they correspond to the age difference between tokens created by A and tokens created by B. After recanonicalization,

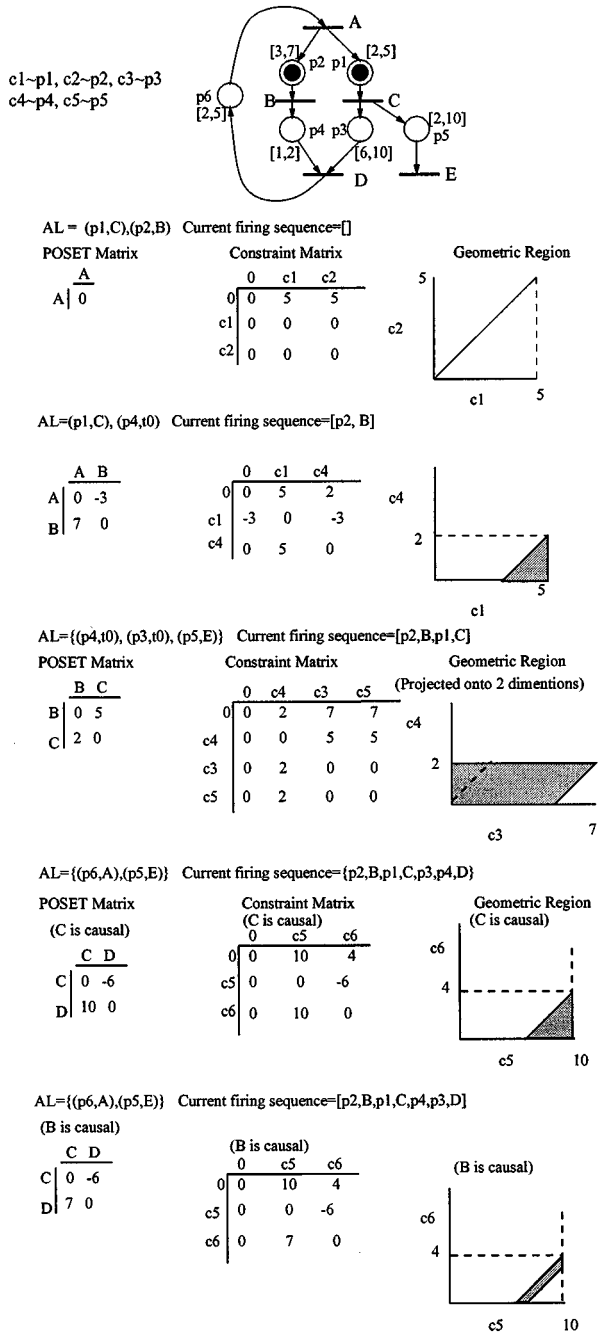


Fig. 8. Example of timing with partially ordered sets.

the separation of seven that is possible between the firing of A and the firing of B is reduced to five since the token in place p_1 has a maximum bound of five. In this state, the token in p_1 can fire causing transition C, or the token in place p_4 can fire. The firing of p_4 does not cause a transition since p_3 has not fired yet. Token p_1 and C are chosen. When C fires, the POSET matrix no longer needs to contain A since all of the places in its postset are unmarked. The POSET matrix shows that B could have fired at most five time units after C and C could have fired at most two time units after B. Now there are three marked places and the region is three-dimensional. In the figure, a two-dimensional projection of the region onto the (c_3, c_4) plane is shown. This region shows the advantage of the POSET technique. Even though

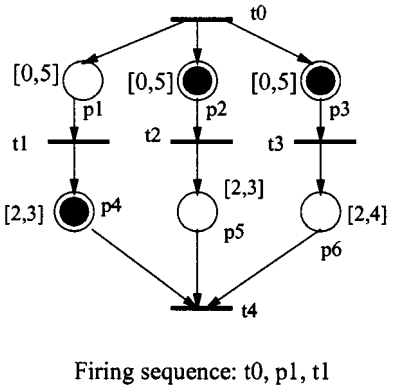


Fig. 9. Example of interleaving optimization.

in this particular firing sequence B fires before C, the region produced here contains timing assignments where C fires before B. Since B and C occur in parallel, all of these timing assignments are valid for the partial order created by the firing sequence $[p_2, B, p_1, C]$. The dashed line in the middle of the region shows the two regions that would be generated by the standard geometric technique. The upper region contains timing assignments where B fires first, and the lower region contains timing assignments where C fires first. In this timed state, tokens in places p_3, p_4, and p_5 can fire. Once the tokens in p_3 and p_4 have fired, D fires. When D fires, information on event B can be removed from the POSET matrix, but since C still has a marked token in its postset, p_5, C remains. Two different maximum separations between C and D are possible depending on whether transition C or B is causal to D. This is determined by whether the token in place p_4 or p_3 fires last. The figure shows the two different geometric regions that result from the two different firing sequences. In this example, one region is a subset of the other, but this is not always the case.

VI. Optimizations

There are a number of optimizations to this algorithm that can be made to reduce the number of geometric regions generated and decrease state space size. The simplest is to check for *subsets* when checking to see if a region has been explored already. If a region is a subset of a region that has been explored, then all of its possible future behaviors are explored by the exploration of the larger region. Any exploration starting from the smaller region generates redundant regions. Checking for a subset can be done simply by checking to see if all of the entries in one matrix are less than their counterparts in the other matrix. A similar optimization can be made by removing a region from the list to be explored in the future when a superset of that region is added to the list. The smaller region can also be removed from the representation of the state space in order to save memory.

The previous optimizations can provide substantial run-time improvement, but the most significant improvement results from the removal of certain interleavings between token firings from consideration. The purpose of exploring different interleavings between token firings is to ensure that all possible causal places for each transition firing are explored. If two different token firing interleavings result in the same causal

POSET matrix: p4 is causal

	t1	t2	t3	t4
t1	0	5	5	-2
t2	5	0	5	-2
t3	5	5	0	-2
t4	3	8	8	0

POSET matrix: p5 is causal

	t1	t2	t3	t4
t1	0	5	5	-2
t2	5	0	5	-2
t3	5	5	0	-2
t4	8	3	8	0

POSET matrix: p6 is causal

	t1	t2	t3	t4
t1	0	5	5	-2
t2	5	0	5	-2
t3	5	5	0	-2
t4	9	9	4	0

Fig. 10. POSET matrices with various causal places.

place for a given transition firing, no additional information is generated by exploring both of them, due to the way the POSET algorithm generates POSET matrices. When information on a new transition, t, is added to the POSET matrix, the causal place determines the upper bound on the time separation between the firing of t and its causal transition. Two firing sequences with the same causal place for t always result in the same time separations between the firing of t and the other transitions in the matrix.

Consider for example the Petri net in Fig. 9. Initially the firing sequence p_1, t_1 has been explored. Since there are many possible interleavings between the firing of p_4 and the firing of the other tokens in the Petri net, it reduces execution time if only one interleaving where p_4 is causal to t_4 is explored. Fig. 10 shows the POSET matrices generated as t_4 fires when each of the places in the preset of t_4 is causal. Looking at the POSET matrices, we can see that when p_4 is causal to t_4 it generates a unique matrix that is not a subset of the matrices generated when other places are causal. The matrix in Fig. 10(a) is also the matrix generated whenever p_4 *fires* last, regardless of whether the token in p_4 is created first or last. For example, the firing sequences $p_1, t_1, p_2, t_2, p_3, t_3, p_6, p_5, p_4, t_4$, and $p_3, t_3, p_2, t_2, p_1, t_1, p_5, p_6, p_4, t_4$ result in generation of the same POSET matrix. Since there are multiple firing sequences where p_4 fires last, this POSET matrix is generated multiple times when all token firing interleavings are explored. Additionally, since a different geometric region is generated for each token firing interleaving, many additional geometric regions are generated by exploring all of the token firing interleavings which are going to create the same POSET matrix. In order to reduce the number of interleavings explored, the algorithm should only generate the POSET matrix in Fig. 10(a) once, and not explore the other token firing interleavings that lead to it.

The difficulty is deciding when a token can always be fired as soon as it is old enough, and when it must be interleaved so it has a chance to be causal. In general, solving this problem would require a lot of computation. However, in certain cases, interleavings can be eliminated by a structural examination of the Petri net. The details of this process are explained in [20]. It does not add significant overhead to the POSET algorithm, and in some cases drastically reduces the number of regions explored.

VII. Results

The POSET algorithm drastically reduces the number of geometric regions generated during state space exploration of highly concurrent systems. The new algorithm along with the optimizations discussed in Section VI has been implemented within the CAD tool `ATACS` and produces very good results as illustrated with the parameterized examples in this section.

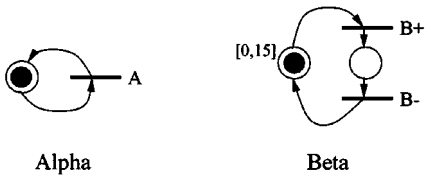


Fig. 11. Alpha and Beta examples.

The first two, the Alpha and Beta examples, are from [5] and one stage of each is shown in Fig. 11. Each stage of the Alpha example is composed of a single event which can fire repeatedly at a given interval and is not effected by any other events in the system. In [5], they showed that techniques based on DBM's (i.e., geometric regions) could only handle five stages of this highly concurrent example while their symbolic discrete-time technique using numerical decision diagrams could handle 18 stages in 12 h on a SUN UltraSparc with 256MB of memory. A *loglog* plot of the results from [5] and our results using POSET timing on a SPARC 20 with 128 MB of memory is shown in Fig. 12. These results indicate that POSET timing can be orders of magnitude faster and more memory efficient. Our techniques found the reachable states space for 512 stages in about 73 min using 112 MB of memory. This simple example clearly has only one untimed state regardless of the number of stages, and POSET timing can represent the timed state space using only one geometric region. Our technique does not find the region in its first iteration, however. It first finds a number of smaller regions before finding the final region that is a superset of all the rest. Therefore, although its performance is very good, it does not analyze the example instantaneously.

One stage of the Beta example is composed of one state bit per stage with two events, one to set and one to reset the bit. In [5], they showed that DBM's could only handle four stages while their technique could handle nine stages. A semilog plot of their results and ours is shown in Fig. 13. POSET timing can handle 14 stages in 108 MB of memory in just 16 min. For the Beta example, the number of states is exactly 2^n where n is the number of stages, so POSET timing could handle an example with 32 times more untimed states than in [5]. Again, POSET timing is able to represent all the timing behavior in this example using one geometric region per state. Clearly, the Alpha and Beta examples are ideally suited to our algorithm, but they are used in [5] to demonstrate the weakness of traditional geometric region based methods. Also, since these examples do not have multiple behavioral places, the performance of our algorithm is no better than the performance of the `Orbits` algorithm [14], [15]. They

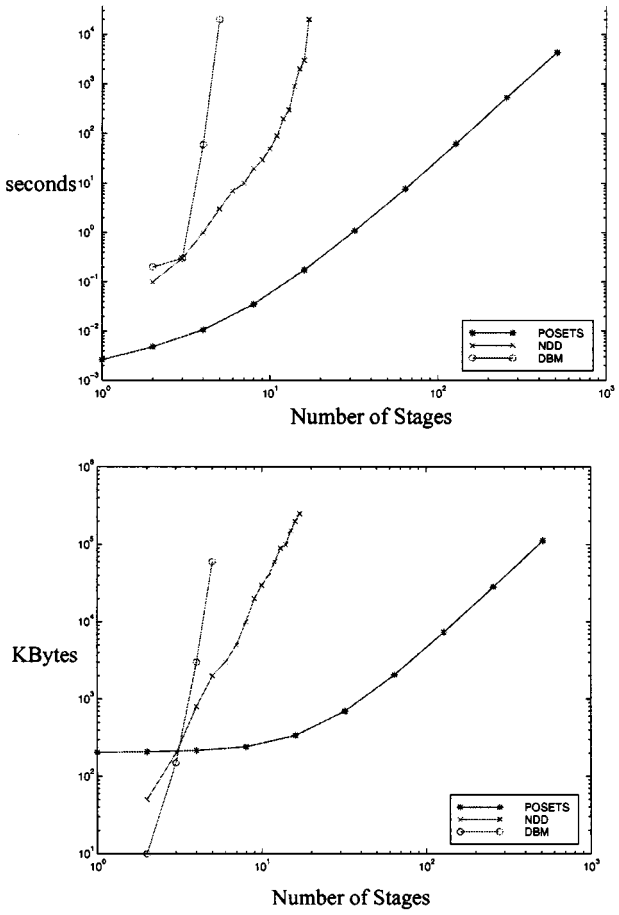


Fig. 12. Comparative performance for the Alpha example.

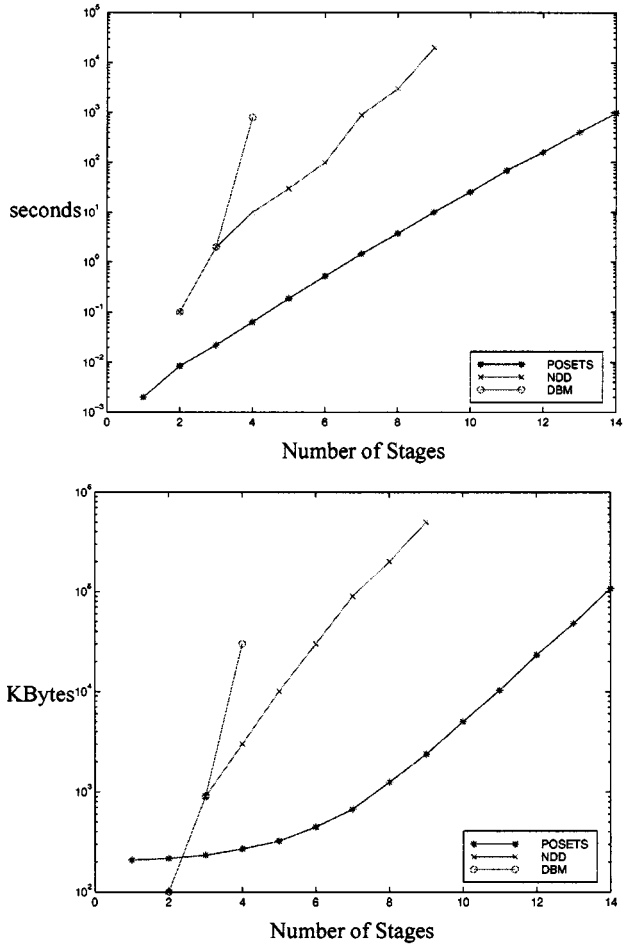


Fig. 13. Comparative performance for the Beta example.

are presented here to show the performance of region based time representation compared to discrete time approaches.

The next example is a n-bit synchronous counter. The basic operation of the counter is that when the clock goes high, the next value of the count is determined from the previous value. When the clock goes low, the new value is latched and fed back to determine the next count. This example has several transitions which contain multiple behavioral places in their presets. As the size of the counter specification is scaled to more bits, the size of the presets of transitions grows. In [19], graph transformations are described that create a new specification which satisfies the single behavioral place restriction allowing verification by `Orbits` [14], [15]. Table I shows runtimes and regions generated using `ATACS` and `Orbits` for counters ranging in size from two bits to seven bits. The results using different combinations of optimizations in `ATACS` are indicated in the tables as follows: "Geometric" indicates the geometric algorithm presented in Section III without any optimizations. "PO" indicates the POSET algorithm without any optimizations. "Sub/sup" indicates the POSET algorithm with the subset and superset optimizations. "Interleaving" indicates that only the interleaving optimization is used, and "all" indicates that subsets, supersets, and interleaving are used. The last column, "Orbits," gives the results of running `Orbits`. `Orbits` also contains many optimizations, all of which are used for this comparison. Entries of "mem" in the table indicate that the machine, a 400-MHz Pentium II with 512MB of memory, runs out of memory. The example size is indicated in the first column, where "T" represents the number of transitions and "P" represents the number of places. Runtime comparisons are difficult between `ATACS` and `Orbits` since `ATACS` is implemented in C and `Orbits` is implemented in Scheme. Although `Orbits` is run on a compiled version of Scheme, which is much faster than interpreted Scheme, its runtimes are still degraded by the difference in implementation language. For this reason, differences in regions generated are useful to compare the algorithms in an implementation independent way.

The maximum counter size that `Orbits` can analyze is 3 bits. `Orbits` requires 1648 s and 10 222 regions to analyze a 3-bit counter, while the POSET algorithm with all optimizations can analyze a 3-bit counter in .07 s and 89 regions. This dramatic difference in region count and runtime occurs because the graph transformation adds $n!$ new places for each event that has n behavioral places. In the 3-bit counter most of the transitions have four behavioral places, causing a huge combinatorial explosion in the number of regions produced by `Orbits`. This example also shows the impact of the interleaving optimization. For a 3-bit counter, the interleaving optimization reduces the region count from 1627 regions to 89 regions, and allows the algorithm to analyze up to a 7-bit counter without running out of memory. Since the number of transitions with many places in their presets is high in this example, eliminating unnecessary token firing

TABLE I
RESULTS FOR COUNTERS

Runtimes for counters (in seconds)						
	geometric	PO	sub/sup	interleaving	all	Orbits
cnt2 (T = 40, P = 77)	.08	.07	.07	.07	.07	5
cnt3 (T = 45, P = 98)	17	2	2	.07	.07	1648
cnt4 (T = 93, P = 215)	mem	mem	mem	.73	.73	mem
cnt5 (T = 189, P = 453)	mem	mem	mem	10	19	mem
cnt6 (T = 381, P = 929)	mem	mem	mem	136	136	mem
cnt7 (T = 765, P = 1886)	mem	mem	mem	1945	1945	mem
Regions generated for counters						
	geometric	PO	sub/sup	interleaving	all	Orbits
cnt2 (T = 40, P = 77)	211	171	168	57	49	240
cnt3 (T = 45, P = 98)	5687	1627	1620	89	89	10222
cnt4 (T = 93, P = 215)	mem	mem	mem	257	257	mem
cnt5 (T = 189, P = 453)	mem	mem	mem	705	705	mem
cnt6 (T = 381, P = 929)	mem	mem	mem	1857	1857	mem
cnt7 (T = 765, P = 1886)	mem	mem	mem	4737	4737	mem

TABLE II
RESULTS FOR FIFO'S

Runtimes for FIFOs (in seconds)						
	geometric	PO	sub/sup	interleaving	all	Orbits
FIFO1 (T = 11, P = 21)	.02	.01	.01	.006	.006	.13
FIFO2 (T = 13, P = 29)	.06	.03	.02	.02	.02	.78
FIFO3 (T = 23, P = 44)	16	1.2	1	.6	.5	32
FIFO4 (T = 29, P = 55)	mem	mem	90	10	7	1346
FIFO5 (T = 35, P = 66)	mem	mem	mem	mem	78	mem
FIFO6 (T = 41 P = 77)	mem	mem	mem	mem	928	mem
Regions generated for FIFOs						
	geometric	PO	sub/sup	interleaving	all	Orbits
FIFO1 (T = 11, P = 21)	120	56	44	36	29	42
FIFO2 (T = 13, P = 29)	341	95	92	65	65	.78
FIFO3 (T = 23, P = 44)	19,872	1814	1092	841	629	2909
FIFO4 (T = 29, P = 55)	mem	mem	34,208	6414	3969	36,758
FIFO5 (T = 35, P = 66)	mem	mem	mem	mem	21,780	mem
FIFO6 (T = 41 P = 77)	mem	mem	mem	mem	121,319	mem

interleavings produces a dramatic reduction in regions and runtime.

The next example is an asynchronous FIFO composed of lazy-active/passive buffers. These buffers perform one communication on their read port to receive a new data value, followed by another communication on their write port to send the value on to the next stage. When many FIFO stages are composed together the resulting specification has many transitions with multiple behavioral places. The results generated for FIFO's ranging in length from one stage to six states are shown in Table II. The longest FIFO that `Orbits` can analyze consists of four buffers and requires 36 758 geometric regions and 1346 s. The analysis of a FIFO with four buffers using the POSET algorithm and all optimizations requires 3969 geometric regions and 7 s. The POSET algorithm can analyze up to six buffers.

The next example is the two level selector circuit shown in Fig. 14. The circuit first receives a request on the $ReqA$ wire. This causes module A to send a request on the $SelA$ wire. It receives a response either on the $SAckB$ wire or the $SAckC$ wire. Module A then sends a request on either the $ReqB$ or the

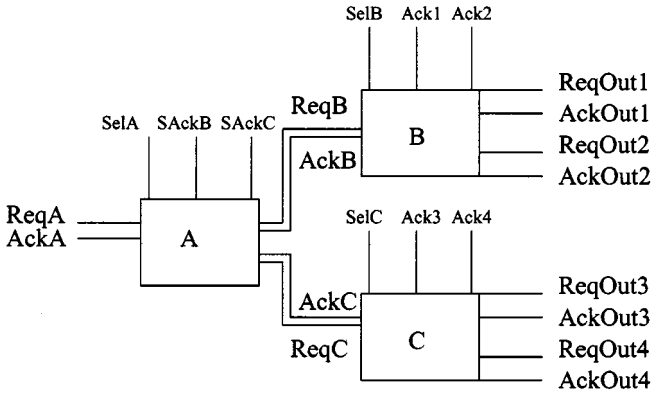


Fig. 14. 2 level selector.

$ReqC$ wires, depending on which response is received for the $SelA$ request. Suppose that $ReqB$ is selected. When module B receives the request on $ReqB$, it sends a request on $SelB$. The response determines whether module B initiates a communication on $ReqOut1$ or $ReqOut2$. When its output communication

TABLE III
RESULTS FOR SELECTOR UNIT

Runtimes for Selectors (in seconds)							
	geometric	PO	sub/sups	interleaving	all	approx	Orbits
sel1 (T = 18, P = 31)	.3	.25	.1	.03	.03	.03	.6
sel2 (T = 37, P = 76)	oom	oom	34	11	5	5	152
sel3 (T = 23, P = 44)	oom	oom	oom	oom	587	309	oom
Regions generated for Selectors							
	geometric	PO	sub/sup	interleaving	all	approx	Orbits
sel1 (T = 18, P = 31)	793	402	187	62	57	58	133
sel2 (T = 37, P = 76)	mem	mem	8536	4432	1732	1706	5417
sel3 (T = 53, P = 110)	mem	mem	mem	mem	51029	40221	mem

is complete, it sends an acknowledge on $AckB$. This allows module A to acknowledge that the selection is complete by sending an acknowledge on $AckA$. This circuit illustrates the behavior of the algorithm on specifications with choice. Three versions of the example are analyzed. In the first, the B and C blocks are replaced with simple handshakes, and only the A block is analyzed. In the second, the B block is removed and replaced with a handshake. The third version contains all three selectors. The results for this example are shown in Table III. Since this example has choice, and additional column, "approx" is added to the table to show the results when the choice restriction in the POSET algorithm is removed. When the "approx" option is used, the algorithm does not check to see of a transition has a choice place in its preset when computing upper bounds in the POSET matrix. All of the other optimizations are also used with the approximation. In this example and the next example, the set of reachable markings found with this approximation is the same as the set of markings found with the exact algorithm. There is an improvement in runtime on the order of 40% when the approximation is used on the largest example. This shows that the conflict restriction is adding extra regions and degrading performance somewhat, but that the effect is not dramatic. If conservative results are acceptable, this approximation can be used to improve performance. If conservative results are not acceptable the runtime penalty to achieve exact results is not prohibitive.

Table III also shows that the POSET algorithm in ATACS compares favorably with Orbits. Orbits requires 152 s and 5417 regions to analyze the two selector version, while the exact POSET algorithm with all optimizations requires only 1732 regions. For the full circuit with both B and C blocks included, the POSET algorithm completes the analysis, using 51 029 regions, and Orbits runs out of memory and does not complete. These results show that even when the algorithm restricts regions when choice places are involved, it still generates many fewer regions than Orbits.

The final example comes from the Intel RAPPID design [21]. The RAPPID design is a fully asynchronous instruction length decoder for the x86 instruction set. This design is shown to be three times faster while using half the power of a corresponding synchronous design from a 400 MHz x86 processor. The key to the performance is a very efficient synchronization mechanism which is called the *tagunit*. One tagunit is shown in Fig. 15. The operation of this circuit is that it can receive a tag from one of seven places ($Tagin_i$). If the instruction is ready (*InstRdy*) and

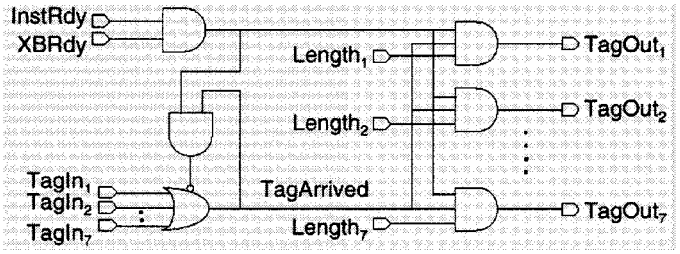


Fig. 15. The tag unit circuit.

the crossbar is ready (*XBRdy*), it tags out to one of seven places ($TagOut_i$) depending on the length of the instruction ($Length_i$). The correctness of the tagunit is verified using ATACS and Orbits, and the results are shown in Table IV. In order to parameterize the example, we verified *tagunits* of various sizes where the size is the number of places from which a tag could be received and then transmitted. The *tagunit* specification contains many choice places, and the impact of the choice restriction is illustrated using the approximation described previously. The result of the approximation in the tagunit is similar to the result in the selector. Removing the choice restriction produces approximately a 40% improvement in runtime for the largest tag unit. Unlike the selector, Orbits completes the largest tag unit specification. Orbits does not fail due to state explosion in this example, but ATACS with all optimizations produces approximately one third the regions that Orbits produces for all sizes of tag unit except size one. This example has fewer transitions with large numbers of places in their presets, which explains the improved performance of Orbits.

In our experience, ATACS with all of the optimizations performs better than Orbits in all specifications that have multiple behavioral places. If a specification does not have multiple behavioral places, the ATACS algorithm and the Orbits algorithm produce identical results.

VIII. CONCLUSION AND FUTURE WORK

Our results clearly show that POSET timing can dramatically improve the efficiency of timing verification allowing larger, more concurrent timed systems to be verified. The results on the Alpha and Beta examples show that the POSET algorithm allows region based timing analysis to scale well on highly concurrent examples. The results on the counter example show that the POSET algorithm is a dramatic improvement over Orbits

TABLE IV
RESULTS FOR TAG UNIT

Runtimes for tag units (in seconds)							
	geometric	PO	sub/supersets	interleaving	all	approx	Orbits
tag1 (T = 17, P = 42)	53	1	.6	.3	.2	.2	3.2
tag2 (T = 25, P = 69)	mem	37	21	2.2	1.7	1.3	35
tag3 (T = 33, P = 98)	mem	101	57	7	5.5	4.1	66
tag4 (T = 41, P = 134)	mem	284	149	30	12	9	107
tag5 (T = 49, P = 188)	mem	mem	278	44	34	22	162
tag6 (T = 57, P = 242)	mem	mem	mem	57	51	37	229
tag7 (T = 65, P = 304)	mem	mem	mem	103	103	69	284
Regions generated for tag units							
	geometric	PO	sub/supersets	interleaving	all	approx	Orbits
tag1 (T = 17, P = 42)	20077	915	629	360	221	221	442
tag2 (T = 25, P = 69)	mem	14799	6020	1277	825	718	2751
tag3 (T = 33, P = 98)	mem	26330	10630	2228	1493	1283	4816
tag4 (T = 41, P = 134)	mem	40937	17481	3412	2352	2007	7409
tag5 (T = 49, P = 188)	mem	mem	24299	4814	3387	2875	10530
tag6 (T = 57, P = 242)	mem	mem	mem	6450	4235	3903	14179
tag7 (T = 65, P = 304)	mem	mem	mem	8304	6017	5075	18356

when there are a large number of behavioral rules, and the analysis of the selectors shows that the penalty incurred by the algorithm to exactly analyze specifications with choice is not overwhelming. Finally the results from the tag unit show that the algorithm can be used to successfully analyze real world circuits, and that it performs significantly better than `Orbits` on such circuits. The POSET algorithm achieves these improvements without eliminating parts of the state space, so it does not limit the properties that can be verified, and the generated state space can be used for synthesis.

In the future, we plan to further increase the size and generality of the specifications that can be verified with the POSET method. We plan on adding support for level based specifications to the algorithm, which will facilitate the representation of gate level circuits. Also, our algorithm currently represents the state space explicitly, and we are working on applying implicit techniques. Our preliminary results show that this can lead to a significant improvement in memory performance [22].

APPENDIX I

Lemma IV.1: If σ_i is causal to σ_j in $\sigma_{0\cdots n}$ then the inequality: $\tau(\sigma_j) \leq \tau(\sigma_i)+u(\sigma_{j-1})$ is true for all valid timing assignments to $\sigma_{0\cdots n}$.

Proof: We know that the firing of transition σ_i created the token whose firing causes transition σ_j to fire. This allows us to prove the desired inequality, $\tau(\sigma_j) \leq \tau(\sigma_i) + u(\sigma_{j-1})$.

$$\begin{aligned}
\tau \text{ is valid} &\Rightarrow \tau(\sigma_{j-1}) \leq \tau(T_c(\sigma_{j-1}, \sigma)) + u(\sigma_{j-1}) \\
&\quad \{\text{Definition II.4}\} \\
&\Rightarrow \tau(\sigma_{j-1}) \leq \tau(\sigma_i) + u(\sigma_{j-1}) \\
&\quad \{\sigma_i = T_c(\sigma_{j-1}, \sigma), \text{ Definition IV.1})\} \\
&\Rightarrow \tau(\sigma_j) \leq \tau(\sigma_i) + u(\sigma_{j-1}) \\
&\quad \{\text{Definition II.4, If } \tau \text{ is valid,} \\
&\quad \tau(\sigma_j) = \tau(\sigma_{j-1})\}.
\end{aligned}$$

■

Lemma IV.2: If $L(\sigma_k) \in \bullet\sigma_j \wedge T_c(\sigma_k, \sigma) = \sigma_i$ in σ then the inequality $\tau(\sigma_j) \geq \tau(\sigma_i) + l(\sigma_k)$ is true for all valid timing assignments, τ, to σ.

Proof:

$$\begin{aligned}
&\tau \text{ is valid} \wedge (\sigma_i = T_c(\sigma_k, \sigma)) \Rightarrow \tau(\sigma_k) \geq \tau(\sigma_i) + l(\sigma_k) \\
&\qquad \{\text{Definition II.4}\}. \qquad \text{(A.1)}
\end{aligned}$$

Now we need to show that $\tau(\sigma_j) \geq \tau(\sigma_k)$ in order to prove the inequality. There are two cases to consider. The first is if σ_i is causal to σ_j in σ:

$$\begin{aligned}
&causal(\sigma, \sigma_i, \sigma_j) \Rightarrow \sigma_k = \sigma_{j-1} \quad \{\text{Definition IV.1}\} \\
&\quad \sigma_k = \sigma_{j-1} \Rightarrow \tau(\sigma_j) = \tau(\sigma_k) \quad \{\text{Definition II.4}\} \\
&\qquad \tau(\sigma_j) = \tau(\sigma_k) \wedge \text{(A.1)} \Rightarrow \tau(\sigma_j) \geq \tau(\sigma_i) + l(\sigma_k).
\end{aligned}$$

The second case is when σ_i is not causal to σ_j in σ:

$$\begin{aligned}
&\neg causal(\sigma, \sigma_i, \sigma_j) \wedge \sigma_i = T_c(\sigma_k, \sigma) \wedge L(\sigma_k) \in \bullet\sigma_j \Rightarrow \\
&k < j-1 \{\text{Definition IV.1}\} \\
&\qquad k < j-1 \Rightarrow \tau(\sigma_k) \leq \tau(\sigma_j) \\
&\qquad \{\text{Definition II.4 and Definition II.2}\} \\
&\tau(\sigma_k) \leq \tau(\sigma_j) \wedge (A.1) \Rightarrow \tau(\sigma_j) \geq \tau(\sigma_i) + l(\sigma_k).
\end{aligned}$$

■

Before we can prove Lemmas IV.3 and IV.4 from the text, we need to prove two support lemmas concerning reorderings. The first lemma proves that a firing of a place p cannot be reordered to occur after a later firing of p in the sequence. The second lemma proves that a transition cannot be reordered to occur after any future firings of places in its preset.

Lemma X.1: Given that $\sigma \in S$, ρ is a valid reordering of σ, and $L(\sigma_i) \in P$:

$$(L(\sigma_i) = L(\sigma_j) \wedge i < j) \Rightarrow \rho(\sigma_i) < \rho(\sigma_j).$$

Proof: Suppose that σ_j is a place firing such that $L(\sigma_j) = L(\sigma_i)$ and $\neg\exists\sigma_k \in \sigma_{i+1\cdots j-1}$: $L(\sigma_k) = L(\sigma_j)$. In words, σ_j

is the firing of $L(\sigma_i)$ that occurs immediately after σ_i. The required set of σ_j contains all of the firings necessary for σ_j to fire. Since the net is one-safe and σ_i occurs before σ_j, σ_i must be in the required set of σ_j since a token cannot be added to a place until one is removed. Therefore, σ_i is in the required set of σ_j. Since required sets are transitively closed, any firings of $L(\sigma_i)$ that occur before σ_i are also in the required set of σ_j. Therefore, by Definition IV.3, $(L(\sigma_i) = L(\sigma_j) \wedge i < j) \Rightarrow \rho(\sigma_i) < \rho(\sigma_j)$. ■

Lemma X.2: Given that $\sigma \in S$, ρ is a valid reordering of σ, $L(\sigma_i) \in T$:

$$L(\sigma_j) \in \bullet\sigma_i \wedge j > i \Rightarrow \rho(\sigma_j) > \rho(\sigma_i).$$

Proof: First we show that a transition firing is always in the required set of the firings of tokens in its preset that immediately follow it. Formally

$$(L(\sigma_j) \in \bullet\sigma_i \wedge \neg\exists\sigma_k \in \sigma_{i\cdots j-1}\colon L(\sigma_k) = L(\sigma_j)) \Rightarrow \sigma_i \in \mathit{required}(\sigma_j, \sigma).$$

Suppose that p is a place in the preset of $L(\sigma_i)$. In the sequence σ, it is the firing σ_i that removes the token from p before another one is created and later fired by the firing of σ_j. Since the net is one-safe, the token fired by σ_j cannot be placed in p, until the previous one is removed by the firing of σ_i. Therefore, σ_i is in the required set of the firings of places in its preset that immediately follow it. By Lemma X.1, and the fact that required sets are transitively closed, σ_i is also in the required set of all future firings of places in its preset. Therefore, $L(\sigma_j) \in \bullet\sigma_i \wedge j > i \Rightarrow \rho(\sigma_j) > \rho(\sigma_i)$ ■

Lemma IV.3: Given that $\sigma \in S$ and ρ is a valid reordering of σ, if $\sigma' = \rho(\sigma)$ then $\sigma' \in S$.

Proof: We need to show that $\forall\sigma'_x \in \sigma'$, σ'_x meets the requirements for a sequence to be in S (Definition II.2). First we deal with transition firings. The only requirement on transition firings is stated in Definition II.2(2). All transition firings must satisfy the requirement, this is the first case.

Case 1: $L(\sigma'_x) \in T$.

We need to show that $L(\sigma'_x) \in \mathit{firable}(\sigma'_{0\cdots x-1})$. The definition of firable (Definition II.1) for transitions has two requirements. The first is that $L(\sigma'_{x-1}) \in \bullet L(\sigma'_x)$. This is shown as follows: Assume that x is the result of $\rho(\sigma_i)$. We know that $L(\sigma_i) \in \mathit{firable}(\sigma_{0\cdots i-1})$ since $\sigma \in S$. This implies that $L(\sigma_{i-1}) \in \bullet\sigma_i$ by the definition of firable. Now we can show that $L(\sigma'_{x-1}) \in \bullet\sigma'_x$

$$L(\sigma_{i-1}) \in \bullet\sigma_i \Rightarrow L(\sigma'_{x-1}) \in \bullet\sigma'_x$$
$$\{\rho(\sigma_{i-1}) = \rho(\sigma_i) - 1, \text{ Definition IV.3(1)}\}.$$

We next need to show that:

$$\forall p \in \bullet\sigma'_x\colon \exists\sigma'_y \in \sigma'_{0\cdots x-1}\colon (L(\sigma'_y) = p) \wedge (\neg\exists\sigma'_z \in \sigma'_{y+1\cdots x-1}\colon (L(\sigma'_y) \in \bullet\sigma'_z)).$$

This is the second term in the definition of firable. Since this condition is true for σ_i, it is true for σ'_x if no token firing that is needed to fire σ_i in σ, can be moved after σ'_x in σ', and no transition that shares a place, p, in the preset of σ_i can be moved between the firing of p immediately preceding σ_i in $\sigma$and σ'_x in σ'. If a token firing is necessary to fire σ_i then it is in the required set of σ_i, therefore, it fires before σ'_x by the definition of a valid reordering (Definition IV.3). There are two proof obligations to ensure that a transition that shares a place p with $L(\sigma_i)$ is not reordered between the firing of p and σ_i:

1) No transition that shares a place in the preset of σ_i and occurs after σ_i can be reordered to occur before σ_i. This is proved as follows:
   If a transition firing σ_j shares a place, p, in the preset of σ_i and fires after σ_i, then there must be a firing of a token in p, that occurs between σ_i and σ_j. This firing, σ_k, is in the required set of σ_j. The net is one-safe, which implies that σ_i is in the required set of firings of p that occur after σ_i in σ, since no future firing of p can occur until σ_i has fired to remove the existing token. Therefore, σ_i is in the required set of σ_k, and also in the required set σ_j since required sets are transitively closed. This implies that σ_j cannot be reordered to occur before σ_i,.
2) No transition that shares a place in the preset of σ_i and occurs before σ_i, can be reordered to occur between a shared place firing and σ_i. This is proved as follows:
   Suppose that σ_k is the firing of shared place place p, which occurs before σ_i. Lemma X.2 shows that no transition can be reordered to occur after a firing of a place in its preset. Therefore, if σ_k is a firing of the shared place p, any transition firing, σ_j, that has p in its preset cannot be reordered after σ_k. This shows that no transition that shares a place in the preset of σ_i and occurs before σ_i, can be reordered to occur between a shared place firing and σ_i.

This shows that $L(\sigma'_x) \in \mathit{firable}(\sigma'_{0\cdots x-1})$. The next two cases deal with the requirements placed on token firings by Definition II.2.

Case 2: $L(\sigma'_x) \in P$: We need to show that either $L(\sigma'_x)$ is in M_0 and σ'_x is the first firing of $L(\sigma'_x)$ or that there is a transition firing to create the token. The firing σ'_x is created by reordering some firing σ_i from the original sequence σ. Since $\sigma \in S$, σ_i is either the first firing of an initially marked place or has its token created by a transition. We can prove that this condition also holds for all σ'_x by showing the following:

1) If σ_i is the first firing of an initially marked place then σ'_x is the first firing of an initially marked place.
2) If σ_i is not the first firing of an initially marked place then there is a transition firing to create the token fired by σ'_x.

The first condition is proved directly using Lemma X.1 and the definition of reordering. The lemma states that the firing of a place p cannot be reordered to occur after a later firing of p. The definition of reordering requires that $L(\sigma_i) = L(\sigma'_x)$. Therefore, if σ_i is the first firing of an initially marked place and σ'_x is the reordering of σ_i then σ'_x is the first firing of an initially marked place.

The second condition is more complex. We know that σ_i has a transition to create its token since $\sigma \in S$. We need to show that this transition is also there in the reordering to create the token for σ'_x. Formally:

$$L(\sigma_i) \in P \wedge \exists\sigma_j \in \sigma_{0\cdots i-1}\colon (L(\sigma_j) \in \bullet\sigma_i) \wedge (\neg\exists\sigma_k \in \sigma_{j+1\cdots i-1}\colon L(\sigma_k) = L(\sigma_i)) \Rightarrow \rho(\sigma_j) < \rho(\sigma_i).$$

This requires that the transition firing that created the token is not reordered after σ_i and that no place firing that uses the token created by σ_j is reordered to occur between σ_j and σ_i. The reordering restriction on the transition firing is guaranteed by the definition of a valid reordering.

$$L(\sigma_i) \in P \wedge \exists \sigma_j \in \sigma_{0\cdots i-1}\colon (L(\sigma_j) \in {\bullet}\sigma_i) \wedge (\neg\exists \sigma_k \in \sigma_{j+1\cdots i-1}\colon L(\sigma_k) = L(\sigma_i)) \Rightarrow \sigma_j \in \mathit{required}(\sigma_i, \sigma)$$

$$\sigma_j \in \mathit{required}(\sigma_i, \sigma) \Rightarrow \rho(\sigma_j) < \rho(\sigma_i) \ \{\text{Definition IV.3}\}.$$

We now need to show that no place firing which fires σ_i's token can be reordered to occur between σ_i and the transition firing, σ_j, that creates the token firing in σ_i. If a place firing, σ_k is going to use the token, then $L(\sigma_i) = L(\sigma_k)$. As shown in Lemma X.1, if $k > i$, $\sigma_i \in \mathit{required}(\sigma_k, \sigma)$ so σ_k cannot be reordered to occur before σ_i. This means that any firing occurring after σ_i is eliminated. The only token firing that would not have to violate Lemma X.1 to be reordered between σ_j and σ_i is the firing of $L(\sigma_i)$ immediately preceding σ_i, which we call σ_k. This firing could be reordered between σ_j and σ_i without forcing token firings out of order. However, since the net is one-safe, this cannot happen. Since σ_k is the firing of $L(\sigma_i)$ immediately preceding σ_i, σ_j is the only firing of transition $L(\sigma_j)$ that occurs between σ_k and σ_i. Since the net is one-safe, this implies that the firing of σ_k must occur before the firing of σ_j since it is necessary to remove the token from $L(\sigma_k)$ before another one can be created by the firing of $L(\sigma_j)$. Therefore, no token firings which use the token needed by σ_i can be reordered to occur between σ_j and σ_i.

We now need to prove that the remaining condition from Definition II.2 is met.

Case 3: $L(\sigma'_x) \in P$, we need to show that

$$L(\sigma'_x) \in P \wedge \mathit{firable}(\sigma'_{0\cdots x}) \neq \emptyset \Rightarrow L(\sigma'_{x+1}) \in \mathit{firable}(\sigma'_{0\cdots x}).$$

If the firable set of the subsequence ending in σ_i is nonempty in σ, it is followed by a transition firing σ_{i+1}. Since σ_{i+1} always follows σ_i in a valid reordered sequence, any firing which has a nonempty firable set in σ is followed by a transition in σ'. Therefore, if no token firing that has an empty firable set in σ has nonempty one in σ', the requirement is satisfied. Now we need to show that in a valid reordering it is not possible for a token firing to have an empty firable set in σ and a nonempty one in σ'. Since transition firings and their causal token firings are reordered consecutively, if a token firing, σ_i has an empty firable set in σ, and the result of its reordering, σ'_x, has a nonempty firable set in σ', then any transition in the firable set of σ'_x is not the same transition that actually consumes the original token firing in σ. This can only occur if the sequence is reordered in such a way that choices are resolved differently in σ and σ'. The definition of a valid reordering (Definition IV.3(3)) forces all choices to be resolved in the same direction in σ in σ'. Therefore, $L(\sigma'_i) \in P \wedge \mathit{firable}(\sigma'_{0\cdots i}) \neq \emptyset \Rightarrow L(\sigma'_{i+1}) \in \mathit{firable}(\sigma'_{0\cdots i})$ holds.

We have now shown that $\sigma' \in S$ ■

Lemma IV.4: If $\mathit{causal}(\sigma, \sigma_i, \sigma_j)$ and ρ is a valid reordering used to map σ to σ', then $\mathit{causal}(\sigma', \sigma'_{(\rho(\sigma_i))}, \sigma'_{(\rho(\sigma_j))})$

Proof: Definition IV.3(2) states that $L(\sigma_i) \in T \Rightarrow \rho(\sigma_i) = \rho(\sigma_{i-1}) + 1$. Therefore, the last token in the preset of σ_i to fire is in the same place in both sequences. Now we just need to show that $\sigma_j = T_c(\sigma_{i-1}, \sigma) \Rightarrow \sigma'_{\rho(s_j)} = T_c(\sigma'_{\rho(\sigma_{i-1})}, \sigma')$. If $\sigma_j = T_c(\sigma_{i-1}, \sigma)$, then it is in the required set of σ_i and can not be reordered to fire later than σ_i. This satisfies the first constraint of Definition II.3. Now we need to show that no other transition firing in the preset of σ_{i-1} can be mapped between σ_j and σ_{i-1}. Since the net is one-safe, if there are two or more transitions in the preset of a place, p, there must be choice places somewhere in the net to prevent all of them from firing in the same iteration through the net and placing multiple tokens in p. Once a transition in the preset of p has fired, no other transition in the preset of p is in the firable set of any token firing until after the token in p has been removed, otherwise the net would not be one safe. Also, as shown in the previous lemma, no place firing σ_k: $L(\sigma_k) = L(\sigma_{i-1})$ can be reordered to occur between σ_j and σ_{i-1}. Therefore, there is no valid reordering of the firing sequence where $\sigma'_{\rho(s_j)} \neq T_c(\sigma'_{\rho(\sigma_{i-1})}, \sigma')$. Therefore, $\mathit{causal}(\sigma', \sigma'_{(\rho(\sigma_i))}, \sigma'_{(\rho(\sigma_j))})$ is true for all valid reorderings. ■

In order to prove the two theorems, additional definitions are necessary. The first definition specifies the timing assignment τ_{minv} where each event occurs at the earliest possible time.

Definition IX.1: Define the *min_valid* timing assignment (τ_{minv}) to a sequence $\sigma_{0\cdots n}$ recursively as follows.

$\forall \sigma_i \in \sigma$:

1) $L(\sigma_i) \in M_0 \wedge (\neg\exists \sigma_j \in \sigma_{0\cdots i-1}\colon L(\sigma_i) = L(\sigma_j)) \Rightarrow \tau_{minv}(\sigma_i) = l(\sigma_i)$;
2) $L(\sigma_i) \in P \wedge \neg(L(\sigma_i) \in M_0 \wedge \neg\exists \sigma_j \in \sigma_{0\cdots i-1}\colon L(\sigma_i) = L(\sigma_j)) \Rightarrow \tau_{minv}(\sigma_i) = \max(\tau_{minv}(T_c(\sigma_i, \sigma)) + l(\sigma_i), \tau_{minv}(\sigma_{i-1}))$;
3) $L(\sigma_i) \in T \Rightarrow \tau_{minv}(\sigma_i) = \tau_{minv}(\sigma_{i-1})$.

This definition follows from the definition of valid timing assignment. Transitions always fire simultaneously with their causal place, so their minimum firing times are determined by the minimum firing times of this place. The minimum firing times of places are determined by when their tokens are created and by the other firings preceding them in the sequence. Since the firing order of the sequence must be reflected by the timing assignment, a place cannot fire before the minimum valid firing time of all places preceding it.

The definition for the maximum valid timing assignment requires examining previous events to determine the maximum timing assignment allowed by the net. It also must examine future events in order to determine the maximum firing time allowed by the sequence. In order to prevent a circular definition, the maximum valid timing assignment is defined in two parts. The first part creates a timing assignment that allows all tokens to fire at the latest possible time after they are created and ignores sequence order. This timing assignment is not valid and is called $\tau_{\max}$. The second part enforces the sequence order and creates a valid timing assignment, τ_{maxv}.

Definition IX.2: Define the maximum timing assignment ($\tau_{\max}$) to a sequence $\sigma_{0\cdots n}$ recursively as follows.

$\forall \sigma_i \in \sigma$:

1) $L(\sigma_i) \in M_0 \wedge (\neg\exists \sigma_j \in \sigma_{0\cdots i-1}: L(\sigma_i) = L(\sigma_j)) \Rightarrow$
   $\tau_{\max}(\sigma_i) = u(\sigma_i)$;
2) $L(\sigma_i) \in P \wedge \neg(L(\sigma_i) \in M_0 \wedge$
   $\neg\exists \sigma_j \in \sigma_{0\cdots i-1}: L(\sigma_i) = L(\sigma_j)) \Rightarrow$
   $\tau_{\max}(\sigma_i) = \tau_{\max}(T_c(\sigma_i, \sigma)) + u(\sigma_i)$;
3) $L(\sigma_i) \in T \Rightarrow \tau_{\max}(\sigma_i) = \tau_{\max}(\sigma_{i-1})$.

Definition IX.3: Define the *max_valid* timing assignment ($\tau_{\max v}$) to a sequence $\sigma_{0\cdots n}$ as follows:

$$\forall \sigma_i \in \sigma: \tau_{\max v}(\sigma_i) = \min_{\sigma_j \in \sigma_{i\cdots n}} \tau_{\max}(\sigma_j).$$

These definitions also follow directly from the definition of a valid timing assignment. Transitions always fire simultaneously with their causal place, so their maximum firing times are determined by the maximum firing times of this place. The maximum firing times of places are determined by when the token in the place is created by its causal transition, and by the other firings occurring after it in the sequence. Since the firing order of the sequence must be reflected by the timing assignment, the maximum valid timing assignment to a place firing is limited by the maximum valid timing assignments of all firings following it. These definitions allow us to prove upper and lower bounds on the times between transition firings that are possible over all valid reorderings of a firing sequence.

Theorem IV.1: For any firing sequence $\sigma \in S$ that has a valid timing assignment, if σ_i is causal to σ_j, and σ_j does not have a choice place in its preset ($\neg\exists \sigma_k$: $\bullet\sigma_j \cap \bullet\sigma_k \neq \emptyset$), there exists a firing sequence $\sigma' \in S$ for which there is a valid timing assignment τ' where $\tau'(\sigma'_{(\rho(\sigma_i))}) + u(\sigma_{j-1}) = \tau'(\sigma'_{(\rho(\sigma_j))})$.

Proof: Definitions IX.3 and IX.2 state that this equation can always be satisfied for any σ where σ_i is causal to σ_j unless there is some σ_k that limits the maximum firing time of σ_j. A firing σ_k limits that maximum firing time of σ_j if it fires after σ_j in σ and has a lower maximum valid firing time than σ_j. Since σ_j is a transition, it must fire at the same time as its causal token firing σ_{j-1}. All firings limiting the firing time of σ_j are actually limiting the firing time of σ_{j-1} and must be moved to fire before σ_{j-1}. We need to show that we can create a reordering ρ which generates a sequence where all such firings are moved before the firing of σ_{j-1}. Since σ_j has no choice places in its preset and σ_{j-1} is in the preset of σ_j, only requirement (1) of Definition IV.3 applies to the order of firings relative to σ_{j-1}. Therefore, we can move all σ_k: $\sigma_j \notin required(\sigma_k)$ before the firing of σ_{j-1}. We create a reordering ρ where

$$\rho(\sigma_k) > \rho(\sigma_{j-1}) \Rightarrow$$
$$(k = j) \vee \sigma_j \in required(\sigma_k) \vee$$
$$\tau_{maxv}(\sigma_k) \geq \tau_{maxv}(T_c(\sigma_{j-1}, \sigma)) + u(\sigma_{j-1})).$$

This implies the following in a sequence $\sigma' = \rho(\sigma)$ where $\rho(\sigma_j) = x$ and $\rho(\sigma_k) = y$

$$y > x \Rightarrow \sigma'_x \in required(\sigma'_y) \vee$$
$$\tau_{maxv}(\sigma'_y) \geq \tau_{maxv}(T_c(\sigma'_{x-1}, \sigma')) + u(\sigma'_{x-1}).$$

Any firing that occurs after σ'_x (which is the reordered σ_j) in the new sequence either did not limit the firing time of σ_j in σ or requires σ_j to fire. All of the events that have σ'_x in their required sets can now be given timing assignments that do not limit the firing time of σ'_x because σ'_x must fire before they can fire, and moving its maximum valid timing assignment later also moves theirs later. Since no firings that limit the firing time of σ'_x occur after σ'_x, this can always be done without violating the ordering constraint. Therefore, there exists a firing sequence $\sigma' \in S$ for which there is a valid timing assignment τ' where $\tau'(\sigma'_{(\rho(\sigma_k))}) + u(\sigma_{j-1}) = \tau'(\sigma'_{(\rho(\sigma_j))})$. ■

Theorem IV.2: For any firing sequence $\sigma \in S$ that has a valid timing assignment, if σ_i is a transition firing in σ, there exists at least one place firing $L(\sigma_j) \in \bullet\sigma_i$ for which in some firing sequence $\sigma' \in S$ constructed from ρ there exists a valid timing assignment τ' in which $\tau'(\sigma'_{(\rho(T_c(\sigma_j,\sigma)))}) + l(\sigma_{j-1}) = \tau'(\sigma'_{(\rho(\sigma_i))})$.

Proof: The proof of this theorem is similar to the proof of the Theorem IV.1. The goal is to move any firings that are limiting the minimum firing time of σ_i to fire after σ_i. Since the firing time of a transition is determined by the preceding token firing, we are again dealing with the firing time of the token firing σ_{i-1}. Since this time we are trying to move firings to occur after σ_i instead of before σ_i, Definition IV.3(3) does not restrict the possible reorderings relative to σ_i. Also any event that fires after σ_i cannot be in the required set of any event firing before σ_i since $\sigma \in S$. Therefore, all events firing before σ_i that are not in the required set of σ_i and limit the minimum firing time of σ_i can be reordered to fire after σ_i. When this is done, only events in the required set of σ_i limit its minimum firing time. Since all of the token firings necessary to fire σ_i are in its required set, there is at least one token for which σ_i can fire at its minimum firing time. ■

ACKNOWLEDGMENT

The authors would like to thank Dr. P. Beerel of the University of Southern California for his advice on the proofs. They would also like to thank the anonymous reviewers for their helpful comments.

REFERENCES

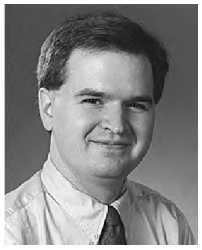
[1] A. Valmari, "A stubborn attack on state explosion," in *Proc Int. Conf. Computer-Aided Verification*, June 1990, pp. 176–185.
[2] P. Godefroid, "Using partial orders to improve automatic verification methods," in *Proc. Int. Conf. Computer-Aided Verification*, June 1990, pp. 176–185.
[3] K. McMillan, "Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits," in *International Workshop on Computer-Aided Verification: Lecture Notes in Computer Science*, G. V. Bochman and D. K. Probst, Eds. Berlin, Germany: Springer-Verlag, 1992, vol. 633, pp. 164–177.
[4] J. R. Burch, "Modeling timing assumptions with trace theory," in *Proc. ICCD*, 1989, pp. 208–211.
[5] M. Bozga, O. Maler, A. Pnueli, and S. Yovine, "Some progress in the symbolic verification of timed automata," in *Proc. Int. Conf. Computer-Aided Verification*, 1997.
[6] R. Alur, "Techniques for automatic verification of real-time systems," Ph.D. dissertation, Stanford Univ., Stanford, CA, Aug. 1991.
[7] D. L. Dill, "Timing assumptions and verification of finite-state concurrent systems," in *Proc. Workshop on Automatic Verification Methods for Finite-State Systems*, 1989.

[8] B. Berthomieu and M. Diaz, "Modeling and verification of time dependent systems using time petri nets," *IEEE Trans. Software Engineering*, vol. 17, Mar. 1991.
[9] H. R. Lewis, "Finite-State Analysis of Asynchronous Circuits with Bounded Temporal Uncertainty," Harvard Univ., Cambridge, MA, Tech. Rep., July 1989.
[10] T. Yoneda, A. Shibayama, B. Schlingloff, and E. M. Clarke, "Efficient verification of parallel real-time systems," in *Computer-Aided Verification*, C. Courcoubetis, Ed. Berlin, Germany: Springer-Verlag, 1993, pp. 321–332.
[11] A. Semenov and A. Yakovlev, "Verification of asynchronous circuits using time Petri-net unfolding," in *Proc. ACM/IEEE Design Automation Conf.*, 1996, pp. 59–63.
[12] E. Verlind, G. de Jong, and B. Lin, "Efficient partial enumeration for timing analysis of asynchronous systems," in *Proc. ACM/IEEE Design Automation Conf.*, 1996.
[13] J. Bengtsson, B. Jonsson, J. Lilius, and W. Yi, "Partial order reductions for timed systems," in *Proc. Int. Conf. Concurrency Theory*, Sept. 1998.
[14] T. G. Rokicki, "Representing and modeling circuits," Ph.D. dissertation, Stanford Univ., Stanford, CA, 1993.
[15] T. G. Rokicki and C. J. Myers, "Automatic verification of timed circuits," in *Int. Conf. Computer-Aided Verification*, 1994, pp. 468–480.
[16] 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 circuit," *IEEE Trans. Computer-Aided Design*, vol. 18, pp. 769–786, June 1999.
[17] ——, "Automatic synthesis of gate-level timed circuits with choice," in *Advanced Research on VLSI*. Los Alamitos, CA: IEEE Computer Society Press, 1995, pp. 42–58.
[18] W. Belluomini and C. J. Myers, "Efficient timing analysis algorithms for timed state space exploration," in *Proc. Int. Symp. Advanced Research in Asynchronous Circuits and Systems*. Los Alamitos, CA: IEEE Computer Society Press, Apr. 1997, pp. 88–100.
[19] C. J. Myers, "Computer-aided synthesis and verification of gate-level timed circuits," Ph.D. dissertation, Stanford Univ., Stanford, CA, 1995.
[20] W. Belluomini, "Algorithms for synthesis and verification of timed circuits and systems," Ph.D. thesis, Univ. Utah, Salt Lake City, 1999.
[21] S. Rotem, K. Stevens, R. Ginosar, P. Beerel, C. Myers, K. Yun, R. Kol, C. Dike, M. Roncken, and B. Agapiev, "RAPPID: An asynchronous instruction length decoder," in *Proc. Int. Symp. Advanced Research in Asynchronous Circuits and Systems*, Apr. 1999, pp. 60–70.
[22] R. A. Thacker, "Implicit methods for timed circuit synthesis," master's thesis, Univ. Utah, Salt Lake City, 1998.

Wendy Belluomini (M'00) received the B.S degree in computer science in 1994 from the California Institute of Technology, Pasadena, the M.S. degree in 1996 from the University of Washington, Seattle, and the Ph.D. degree in 1999 from the University of Utah, Salt Lake City.

Since 1999, she has been working at IBM Austin Research Laboratory. Her current interests include timing verification and asynchronous circuit design.

Dr. Belluomini received a National Science Foundation (NSF) Traineeship in 1996 and a DARPA ASSERT fellowship in 1997.



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

He has been an Assistant Professor in the Department of Electrical Engineering, University of Utah, Salt Lake City, since 1995 where he also serves as Director for the Center for Asynchronous Circuit and System Design. His current research interests are innovative architectures for high-performance and low-power, algorithms for the computer-aided analysis and design of real-time concurrent systems, formal verification, and asynchronous circuit design.

Dr. Myers received a National Science Foundation (NSF) Fellowship in 1991, an NSF CAREER award in 1996, and a Best Paper Award at Async'99.