

Verification of Timed Systems Using POSETs ^{*}

Wendy Belluomini¹ and Chris J. Myers²

¹ Computer Science Department

² Electrical Engineering Department
University of Utah
Salt Lake City, UT 84112

Abstract. This paper presents a new algorithm for efficiently verifying timed systems. The new algorithm represents timing information using geometric regions and explores the timed state space by considering partially ordered sets of events rather than linear sequences. This approach avoids the explosion of timed states typical of highly concurrent systems by dramatically reducing the ratio of timed states to untimed states in a system. A general class of timed systems which include both event and level causality can be specified and verified. This algorithm is applied to several recent timed benchmarks showing orders of magnitude improvement in runtime and memory usage.

1 Introduction

The fundamental difficulty in verification is controlling the state explosion problem. The state spaces involved in verifying reasonably sized systems are large even if the timing behavior of the system is not considered. The problem gets even more complex when verification is done on timed systems. However, verification with timing is crucial to applications such as asynchronous circuits and real-time systems.

A number of techniques have been proposed to deal with state explosion. Approaches have been proposed that use stubborn sets [1], partial orders [2], or unfolding [3]. These techniques reduce the number of states explored by considering only a subset of the possible interleavings between events. These approaches have been successful, but they only deal with untimed verification.

The state space of timed systems is 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.

^{*} This research is supported by a grant from Intel Corporation, NSF CAREER award MIP-9625014, SRC grant 97-DJ-487, and a DARPA AASERT fellowship.

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 can explode if they are large.

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* or DBMs). 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 [10–12] 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 this may limit the timing properties that can be verified. While reducing the number of interleavings is useful, in [10, 11] one region is still required for every firing sequence explored to reach a state. If most interleavings need to be explored, these techniques could still result in state explosion.

The algorithm presented in [13, 14] significantly reduces the number of regions per untimed state by using *partially ordered sets* (or POSETs) 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 [14] to result in very few geometric regions per untimed state. The entire timed state space is explored, so it can be used to verify a wide range of timing properties. However, it is limited to specifications where the firing time of an event can only be controlled by a single predecessor event (known as the *single behavioral place (or rule) restriction*). This restriction can be worked around with graph transformations, but the graph transformations add $n!$ new rules for each event with n behavioral rules [15, 16]. In [17], we presented an approximate algorithm for exploring the entire state space with POSETs on a general class of specifications, lifting the single behavioral rule restriction. However, it may generate regions that are larger than necessary.

This paper presents a new algorithm for timed state space exploration based on geometric regions and POSETs. This algorithm operates on a very general class of specifications, *timed event/level (TEL) structures* [18], which are capable of directly express-

ing both event and level causality. Through a straightforward construction (omitted due to space constraints), it can be shown that TEL structures are at least as expressive as 1-safe time Petri nets [19]. TEL structures can also represent some behavior more concisely due to their ability to specify levels which are not directly supported in time Petri nets. While they are not as expressive as timed automata [6], TEL structures represent an interesting class of timed automata sufficient to accurately model timed circuit behavior. 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 verification extremely competitive with discrete-time when the delay ranges are small and far superior when the ranges are large. The performance of POSET timing is demonstrated by orders of magnitude improvement in runtime and memory usage on several recent timing verification benchmarks.

2 Timed systems and exploration of their timed states

The process of timing verification begins with a specification of a timed system and properties that it must satisfy. To check if these properties are satisfied, the verification algorithm explores the timed state space allowed by the specification. This section presents our formalism for modeling timed systems and exploring their state spaces.

2.1 Timed event/level structures

The algorithm presented in this paper is applied to specifications in the form of TEL structures [18], an extension of timed event-rule structures [15]. TEL structures are very well suited to describing asynchronous circuits since they allow both event causality to specify sequencing and level causality to specify bit value sampling. This section gives a brief overview of TEL structures. See [18] for a more complete description of their semantics. A TEL structure is a tuple $T = \langle N, s_0, A, E, R, \# \rangle$ where:

1. N is the set of signals;
2. $s_0 = \{0, 1\}^N$ is the initial state;
3. $A \subseteq N \times \{+, -\} \cup \$$ is the set of atomic actions;
4. $E \subseteq A \times (\mathcal{N} = \{0, 1, 2, \dots\})$ is the set of events;
5. $R \subseteq E \times E \times \mathcal{N} \times (\mathcal{N} \cup \{\infty\}) \times (b : \{0, 1\}^N \rightarrow \{0, 1\})$ is the set of rules;
6. $\# \subseteq E \times E$ is the conflict relation.

The signal set, N , contains the wires in the specification. The state s_0 contains the initial value of each signal in N . The action set, A , contains for each signal, x , in N , a rising transition, $x+$, and a falling transition, $x-$. The set A also includes a dummy event, $\$$, which is used to indicate an action that does not result in a signal transition. The event set, E , contains actions paired with occurrence indices (i.e., $\langle a, i \rangle$). Rules represent causality between events. Each rule, r , is of the form $\langle e, f, l, u, b \rangle$ where:

1. e = enabling event,
2. f = enabled event,
3. $\langle l, u \rangle$ = bounded timing constraint, and
4. b = a sum-of-products boolean function over the signals in N .

A rule is *enabled* if its enabling event has occurred and its boolean function is true in the current state. A rule is *satisfied* if it has been enabled at least l time units. A rule becomes *expired* when it has been enabled u time units. Excluding conflicts, an event cannot occur until every rule enabling it is satisfied, and it must occur before every rule enabling it has expired. If a rule's boolean function becomes false after the rule has become enabled, but before its enabled event has occurred, this indicates that the enabled event has a hazard which is considered a failure during verification.

The conflict relation, $\#$, is used to model disjunctive behavior and choice. When two events e and e' are in conflict (denoted $e\#e'$), this specifies that either e or e' can occur but not both. Taking the conflict relation into account, if two rules have the same enabled event and conflicting enabling events, then only one of the two mutually exclusive enabling events needs to occur to cause the enabled event. This models a form of disjunctive causality. Choice is modeled when two rules have the same enabling event and conflicting enabled events. In this case, only one of the enabled events can occur.

If a specification is cyclic, then the TEL structure representing it is infinite. However, due to its repetitive nature, this infinite behavior can be described with a finite model by adding an additional set of rules and conflicts which recursively defines the infinite structure [15].

2.2 Timed state space exploration

The *untimed state* of a TEL structure is composed of two parts: the set of rules whose enabling events have occurred, R_m , and the state, s_c , of all the signals in the system. From this untimed state, the set of enabled rules, R_{en} , can be constructed by including only those members of R_m whose boolean expressions are satisfied by s_c . In order to determine the set of satisfied rules R_s , timing information is needed. It is referred to as TI and is included in the *timed state* ($R_m \times s_c \times TI$).

The state space of a TEL structure is explored using a depth first search. In each state, the algorithm chooses a rule from R_s to fire, and places onto the stack the current state and the remainder of R_s . It then fires the chosen rule, adds it to a set of fired rules, R_f , which is part of the timing information, and determines the new timed state. If R_f contains a set of rules sufficient to fire an event e , the new timed state has a marking in which e has fired. If this timed state has not been seen before, it is added to the state space, and a new R_s is calculated. If a timed state is reached that has been seen before, the algorithm pops off the stack a timed state and the list of rules that have not yet been explored for that state. When a state that has been seen before is reached and the stack is empty, the entire timed state space has been found.

The timing information must be updated at every rule firing during state space exploration. Therefore, it is very important that the procedure for updating it is efficient. The timing analysis algorithm presented here uses geometric regions to represent the timing information within a timed state. Whenever a rule r_i becomes enabled, a clock c_i is created to be used in timing analysis. The minimum and maximum age differences of all the clocks associated with rules in R_{en} are stored in a constraint matrix M . Each entry m_{ij} in the matrix M 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 uniquely 0 is used to allow the inclusion of the minimum and maximum ages of the clocks in M . In other words, the

maximum age of c_i is in the entry m_{0i} , and the negative of the minimum age of c_i is in the entry m_{i0} . Note that M only needs to contain information on the timing of the rules that are currently in R_{en} , not on the whole set of rules. This particular way of representing timed regions was first introduced in [7]. This constraint matrix represents a convex $|R_{en}|$ dimensional region. Each dimension corresponds to a rule and the firing times of the rule can be anywhere within the space.

3 Timed state space exploration using POSET timing

While geometric regions are an effective way to represent dense-time state spaces, the number of geometric regions can explode for highly concurrent timed systems [14, 5]. In [14], an algorithm is described that uses *partially ordered sets* (POSETs) of events rather than linear sequences to mitigate this state explosion problem. POSET timing techniques take advantage of the inherent concurrency in the TEL structure and prevent additional regions from being added for different sequences of event firings that lead to the same untimed state. 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. Therefore, all properties of the system that can be verified with the standard geometric technique can be verified with the POSET algorithm. This combination of regions could also be done as each region is generated during state space exploration. However the check to see if the combination of two regions is convex takes $O(n^4)$ time in the number of constraints in the matrix. This check must be done between each new region and all the regions that have been generated previously, making this approach prohibitively expensive [13].

The POSET algorithm maintains a *POSET matrix* (also known as a process matrix in [13, 14, 17]), in addition to the constraint matrix. A POSET is a partially ordered set of events created from a TEL structure and a firing sequence. It is constructed from a TEL structure as follows: The POSET is initially empty. Events are added in the same order as they occur in the firing sequence. For an event in the firing sequence, a correspondingly labeled event is added to the POSET. Rules are added to connect the newly enabled event to the events in the POSET that enabled it.

The POSET matrix stores the minimum and maximum possible separations between the firing times of all the events in the POSET that are allowed by the firing sequence currently being explored. At each iteration, the time separations in the POSET matrix are copied into the entries of the constraint matrix that restrict the differences in the enabling times of the rules. Events 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.

3.1 Partially ordered sets without levels

When a new event 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 events in the matrix must be determined. This set of separations must be consistent with the rule firing sequence that resulted in the current state. The rule firing sequence often limits

the separations between events that are possible. There may be separations between events that are possible over all firing sequences but are not possible given the current one. Therefore, the separations in the POSET matrix must be restricted so that they are reachable given the current rule firing sequence.

The POSET matrix is kept consistent with the current rule firing sequence by ensuring that the time separations in the matrix reflect the causality implied by the current rule firing sequence. An event that is enabled by multiple rules does not fire until all of these rules have fired. The last rule to fire actually causes the event to fire, and is referred to as the *causal* rule. More formally, a rule $r_m = \langle e_c, e, l, u \rangle$ is causal to event e given a rule firing sequence $r_0 \dots r_n$, if the firing sequence $r_0 \dots r_{m-1}$ does not enable e and the firing sequence $r_0 \dots r_m$ does enable e . A set of rules R_f enables event e if $\forall r_i = \langle e_i, e, l, u \rangle \in R : (r_i \in R_f) \vee (\exists r_j = \langle e_j, e, l_j, u_j \rangle \in R_f : e_i \# e_j)$ [15, 17].

The significant difference between the POSET technique described here and the work presented in [13, 14] is the method used to compute the POSET matrix. In [13, 14], it is not necessary to use explicit causality information since the causal rule is always the behavioral rule. With multiple behavioral rules, causality must be considered in order to compute a correct POSET matrix. Assume that I_n is a correct, maximally constraining set of inequalities that relate the firing times of a set of events E_n . When a new event e fires with causal rule $r = \langle e_c, e, l, u \rangle$, a new new set of correct, maximally constraining inequalities I_{n+1} can be computed from I_n . Initially, I_{n+1} is set to equal I_n . Then, I_{n+1} is updated with the inequalities $t(e) - t(e_c) \leq u$ and $t(e_c) - t(e) \leq -l$. These inequalities are always true since no rule can delay the firing of e once its causal rule has fired. Next, for each rule $r_i = \langle e_i, e, l_i, u_i \rangle$ in R_f , the inequality $t(e_i) - t(e) \leq -l_i$ needs to be added because each rule enabling e must be satisfied. These new inequalities may cause other inequalities in I_{n+1} to no longer be maximally constraining. This occurs when there exists a subset of I_{n+1} of the form $\{t(e) - t(e_c) \leq u, t(e_i) - t(e) \leq -l_i, t(e_i) - t(e_c) \leq n\}$ where $u - l_i < n$. All of the inequalities in I_{n+1} can be made maximally constraining by running Floyd's all pairs shortest path algorithm[7].

After the all pairs shortest path algorithm is run, I_{n+1} contains a maximally constraining set of inequalities that includes all the constraints that result from firing e . However, minimum and maximum constraint between e and all of the events in E_n must also be included in I_{n+1} . These additional constraints are immediately derivable from the constraints already in I_{n+1} . The maximum constraints are as follows: $t(e) - t(e_j) \leq u + \max(t(e_c) - t(e_j))$. This inequality holds since the maximum separation between e_j and e occurs when e_j happens as much before e 's causal event as possible. If there is a rule $r_j = \langle e_j, e, l_j, u_j \rangle$ that relates e_j and e , then the minimum constraint is $t(e_j) - t(e) \leq \min(-l_j, -(l - \max(t(e_j) - t(e_c))))$. This inequality holds because the minimum separation between e_j and e occurs when e_j happens as much after e 's causal event as possible, but must be no less than the minimum on the rule relating e_j and e . The inequalities $t(e_j) - t(e) \leq -l_j$ are added to I_{n+1} before the all pairs shortest path step, and are constrained further to $t(e_j) - t(f) \leq -(l - \max(t(e_j) - t(e)))$, if necessary. If there is not a rule $r_j = \langle e_j, e, l_j, u_j \rangle$ that relates e_j and e , the minimum constraint is simply $t(e_j) - t(e) \leq -(l - \max(t(e_j) - t(e_c)))$. I_{n+1} now contains a correct, maximally constraining set of inequalities that represent the minimum and maximum separations between all the events in $E_{n+1} = E_n \cup e$. Note that as an op-

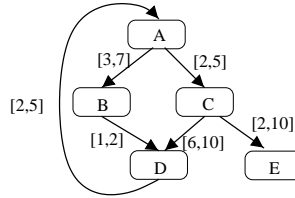
timization, inequalities that are no longer needed to compute future inequalities are removed from I_n . Since the base case is simply $I_0 = \emptyset$, this procedure can be used to construct correct sets of inequalities for an arbitrary rule firing sequence.

A geometric region representing the differences in the ages of a set of clocks associated with a set of enabled rules R_{en} can easily be computed given a POSET matrix using a method similar to the one described in [13, 14]. The maximum difference in the ages of the two clocks c_i and c_j associated with rules $r_i = \langle e_i, f_i, l_i, u_i \rangle$ and $r_j = \langle e_j, f_j, l_j, u_j \rangle$ is simply the maximum difference in the firing times of e_i and e_j which is in the POSET matrix as $t(e_i) - t(e_j) \leq \max$. The minimum likewise exists in the POSET matrix as $t(e_j) - t(e_i) \leq -\min$. These constraints are simply copied into the matrix representing the geometric region. The minimum and maximum bounds of the rules are used to set the minimum and maximum age differences between c_i and c_0 . Floyd's algorithm is then run on the constraint matrix resulting in a maximally constraining set of inequalities. This may further constrain some of the inequalities since the POSET inequalities do not take into account the fact that a clock associated with a rule may not be older than the maximum bound on the rule. Additionally, the normalization algorithm described in [13] to ensure the state space remains finite.

Figure 1 shows timing analysis based on POSETs applied to the small TEL structure shown at the top of the figure. This example shows how our algorithm solves two of the problems that occur when using geometric regions for timed state space exploration: region splitting and multiple behavioral rules. In this example, initially the R_{en} set is $\{\langle A, B \rangle, \langle A, C \rangle\}$, indicating that event A has just fired. The POSET matrix contains a single event, A . The constraint matrix shows that the maximum time since A has fired is 5. If more than 5 time units had passed, the rule $\langle A, C \rangle$ would have been forced to fire. Since both the rules in R_{en} are enabled by A , the difference in their enabling times must be 0, and the region in space that shows this is a 45 degree line.

From this timed state, either event B or event C can fire. In this example, B fires next. 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. After the all pairs shortest path algorithm is run, the separation of 7 that is possible between the firing of A and the firing of B in the POSET matrix is reduced to 5 in the constraint matrix since rule $\langle A, C \rangle$ has a maximum bound of 5 and therefore its clock cannot be more than 5 time units older than another clock.

In this state event C or rule $\langle B, D \rangle$ can fire next and C is chosen. When C fires, the POSET matrix no longer needs to contain A since all events it has enabled have fired. The POSET matrix shows that B could have fired at most 5 time units after C and C could have fired at most 2 time units after B . Now there are three rules enabled and the region is 3-dimensional. In the figure, a two dimensional projection of the region into the $\langle C, D \rangle, \langle B, D \rangle$ plane is shown. This region shows the advantage of the POSET technique. Even though 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 allowed by the rule firing sequence that produced this state. The dashed line in the middle of the region shows the two regions that would be generated by standard geometric techniques. The upper region



Ren={[A, B], [A,C]} Current firing sequence={A}

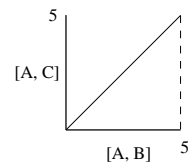
POSET Matrix

$$\begin{array}{c|c} \underline{A} & \\ \hline A & 0 \end{array}$$

Constraint Matrix

$$\begin{array}{c|cc} 0 & [A, B] & [A, C] \\ \hline 0 & 5 & 5 \\ [A,B] & 0 & 0 \\ [A,C] & 0 & 0 \end{array}$$

Geometric Region



Ren={[A,C], [B,D]} Current firing sequence={A,B}

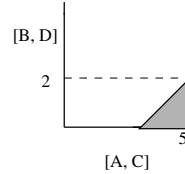
POSET Matrix

$$\begin{array}{c|cc} \underline{A \ B} & & \\ \hline A & 0 & -3 \\ B & 7 & 0 \end{array}$$

Constraint Matrix

$$\begin{array}{c|cc} 0 & [A, C] & [B, D] \\ \hline 0 & 5 & 2 \\ [A,C] & -3 & 0 \\ [B,D] & 0 & 5 \end{array}$$

Geometric Region



Ren={[B,D], [C,D], [C,E]} Current firing sequence={A,B,C}

POSET Matrix

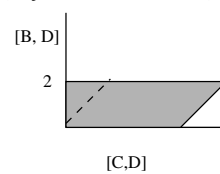
$$\begin{array}{c|cc} \underline{B \ C} & & \\ \hline B & 0 & 5 \\ C & 2 & 0 \end{array}$$

Constraint Matrix

$$\begin{array}{c|ccc} 0 & [B, D] & [C, D] & [C, E] \\ \hline 0 & 2 & 7 & 7 \\ [B,D] & 0 & 5 & 5 \\ [C,D] & 0 & 2 & 0 \\ [C,E] & 0 & 2 & 0 \end{array}$$

Geometric Region

(Projected onto 2 dimensions)



Ren={[D,A], [C,E]} Current firing sequence={A,B,C,D}

POSET Matrix

(C is causal)

$$\begin{array}{c|cc} \underline{C \ D} & & \\ \hline C & 0 & -6 \\ D & 10 & 0 \end{array}$$

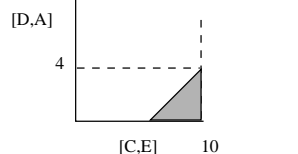
Constraint Matrix

(C is causal)

$$\begin{array}{c|cc} 0 & [C, E] & [D, A] \\ \hline 0 & 10 & 4 \\ [C, E] & 0 & 0 \\ [D, A] & 0 & 10 \end{array}$$

Geometric Region

(C is causal)



(B is causal)

$$\begin{array}{c|cc} \underline{C \ D} & & \\ \hline C & 0 & -6 \\ D & 7 & 0 \end{array}$$

(B is causal)

$$\begin{array}{c|cc} 0 & [C, E] & [D, A] \\ \hline 0 & 10 & 4 \\ [C, E] & 0 & 0 \\ [D, A] & 0 & 7 \end{array}$$

Geometric Region

(B is causal)

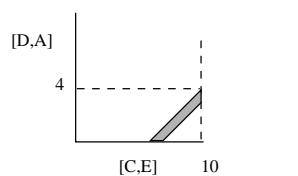


Fig. 1. Example of timing with partially ordered sets.

contains timing assignments where B fired first, and the lower region contains timing assignments where C fired first.

In this timed state, rules $\langle B, D \rangle$, $\langle C, D \rangle$, and $\langle C, E \rangle$ are enabled. Once both of the rules that enable D fire, event D can fire. When D fires, information on event B can be removed from the POSET matrix, but since C enables another event, E , it remains. Two different maximum separations between C and D are possible depending on whether event C or event B was causal to D , and two different geometric regions result. In this example, one region is a subset of the other, but this is not always the case.

3.2 Partially ordered sets with levels

In [18], we extended a geometric region based timed state space exploration algorithm to TEL structures which include arbitrary level annotations. The POSET algorithm presented in the previous section can also be extended to TEL structures with a limited class of level annotations. The algorithm is based on the ability to determine which previous event firing is causal to each new event firing. Recall that in our algorithm, rules fire independently of events, and an event fires when a set of rules sufficient to enable it have fired. When there are no level expressions, the causal event is simply the enabling event of the causal rule. However, if there are level expressions, this is not necessarily the case. With levels, a rule does not always become enabled when its enabling event fires. A rule only becomes enabled when its enabling event has fired *and* its level expression evaluates to true. Therefore, an event e is causal to event f if the firing of event e enables f 's causal rule either because it is its enabling event or because it changes the value of the state such that f 's causal rule becomes enabled.

Determining this causality is straightforward during state space exploration. Whenever a rule fires, its causal event is recorded. Then when an event fires, a procedure similar to the one described in the previous subsection is used to determine the new set of inequalities that belong in the POSET matrix. The major difference is that now any event in the TEL structure may be causal to the firing event and all events need to be checked for causality. Additionally, the causality relationship may imply other time relationships between event firings. Due to space constraints, they are not described here. However, all of the constraints can be easily computed as long as the boolean expressions are restricted to pure **and** and pure **or** expressions. This limited class of TEL structures is expressive enough to model all TEL structures since more complex expressions can be modeled through graph transformations.

4 Results and conclusions

The POSET algorithm drastically reduces the number of geometric regions generated during state space exploration of highly concurrent systems. We have also made additional optimizations to the state space exploration process such as eliminating timed states to be explored from the stack if a region that is a superset of previous regions is found, and reducing the number of interleavings between rule firings. This new POSET timing algorithm along with these optimizations has been implemented within the CAD

tool ATACS and produce very good results as illustrated with the parameterized timing verification benchmarks in this section.

The first two, the Alpha and Beta examples, are from [5]. 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 DBMs (i.e., geometric regions) could only handle 5 stages of this highly concurrent example while their symbolic discrete-time technique using numerical decision diagrams (NDDs) could handle 18 stages in 12 hours 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 are shown in Figure 2. These results indicate that POSET timing is orders of magnitude faster and more memory efficient. In fact, our techniques found the reachable states space for 512 stages in about 73 minutes 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.

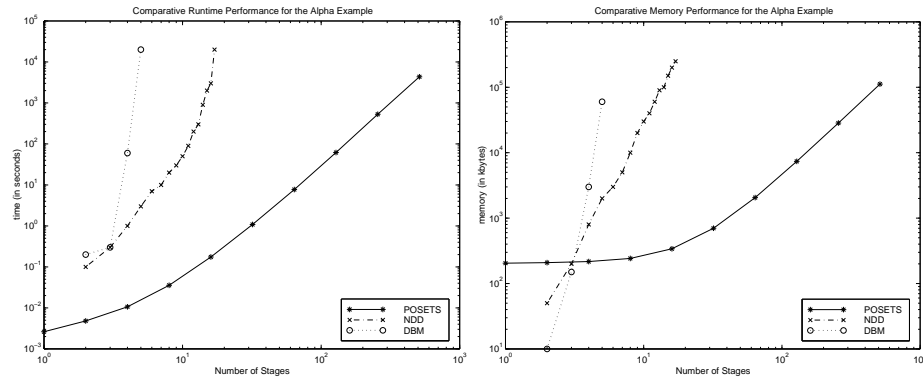


Fig. 2. Comparative performance for the Alpha example.

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 DBMs could only handle 4 stages while their technique could handle 9 stages. A semilog plot of their results and ours are shown in Figure 3. POSET timing can handle 14 stages in 108 MB of memory in just 16 minutes. 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.

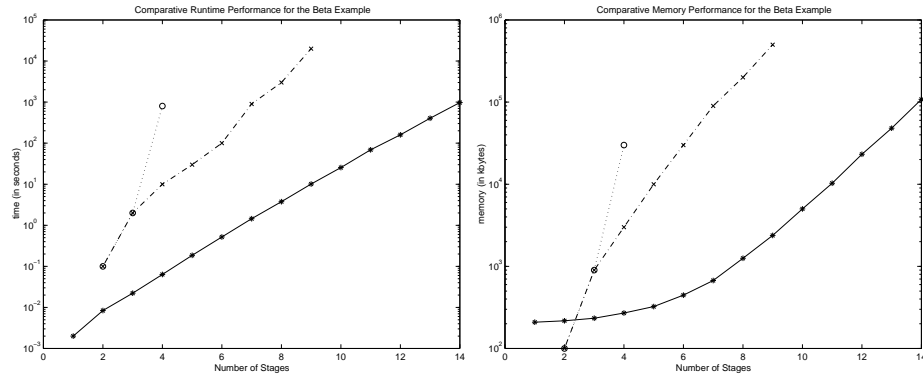


Fig. 3. Comparative performance for the Beta example.

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 events which are enabled by multiple behavioral rules. In [15], graph transformations are described that can create a new specification which satisfies the single behavioral rule restriction allowing verification by `Orbits` [13, 14]. Using these graph transformations, `Orbits` could only analyze a 3-bit counter because it required 10,222 geometric regions to find the 64 untimed states. With our new POSET timing algorithm, it only requires 294 geometric regions to represent the entire timed state space for the 3-bit counter. In fact, our algorithm could analyze up to a 6-bit counter. This drastic difference in region count occurs because the graph transformation adds $n!$ new rules for each event that has n behavioral rules. In the 3-bit counter most of the events had 4 behavioral rules, causing a huge combinatorial explosion in the number of regions.

The last example is a STARI communication circuit described in detail in [20, 21]. The STARI circuit is used to communicate between two synchronous systems that are operating at the same clock frequency, π , but are out-of-phase due to clock skew which can vary from 0 to *skew*. The environment of this circuit is composed of a `clk` process, a transmitter, and a receiver. The STARI circuit is composed of a number of FIFO stages built from 2 C-elements and 1 NOR-gate per stage which each have a delay of l to u . There are two properties that need to be verified: (1) each data value output by the transmitter must be inserted into the FIFO before the next one is output (i.e., $ack(1)-$ precedes $x(0).t-$ and $x(0).f-$) and (2) a new data value must be output by the FIFO before each acknowledgment from the receiver (i.e., $x(n).t+$ or $x(n).f+$ precedes $ack(n+1)-$) [22]. To guarantee the second property, it is necessary to initialize the FIFO to be approximately half-full [21]. In addition to these two properties, we also

verified that every gate is hazard-free (i.e., once a gate is enabled, it cannot be disabled until it has fired).

There have been two nice proofs of STARI's correctness [21, 23], but they have been on abstract models. In [22], the authors state that COSPAN which uses the unit-cube (or region) technique for timing verification [24] ran out of memory attempting to verify a 3 stage gate-level version of STARI on a machine with 1 GB of memory. This paper goes on to describe an abstract model of STARI for which they could verify 8 stages in 92.4 MB of memory and 1.67 hours. We first verified STARI at the gate-level with delays from [22] (i.e., $\pi = 12$, $skew = 1$, $l = 1$, and $u = 2$). Using POSET timing, we can verify a 3 stage STARI in 0.74 MB in only 0.40 seconds. For an 8 stage STARI, the verification took 11 MB and only 55 seconds. In fact, POSET timing could verify 10 stages in 124 MB of memory in less than 20 minutes. This shows a nice improvement over the abstraction method and a dramatic improvement over the gate-level verification in COSPAN. For 10 stages, POSET timing found 14,531 untimed states and only needed 14,859 geometric regions to describe the timed state space. This represents a ratio of only 1.02 geometric regions per untimed state.

Finally, the complexity of POSET timing is relatively independent of the timing bounds used. We also ran our experiments using $l = 97$ and $u = 201$, $skew = 101$, and $\pi = 1193$ which found more untimed states. With $l = 102$, we found less untimed states. Both cases with higher precision delay numbers had comparable performance to the one with lower precision delay numbers. This shows that higher precision timing bounds can be efficiently verified and can lead to different behaviors. It would not be possible to use this level of precision with a discrete-time or unit-cube based technique, since the number of states would explode with such large numbers.

Our results clearly show that POSET timing can dramatically improve the efficiency of timing verification allowing larger, more concurrent timed systems to be verified. It does so without eliminating parts of the state space, so it does not limit the properties that can be verified. In the future, we plan to further increase the size and generality of the specifications that can be verified with the POSET method. We believe the abstraction technique from [22] and POSET timing methods are orthogonal, and we are interested in trying to combine them for further improvement. Finally, 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 [25].

Acknowledgments

We would like to thank Mark Greenstreet of the University of British Columbia, Brandon Bachman, Eric Mercer, and Robert Thacker of the University of Utah and Tom Rokicki of Hewlett Packard for their helpful comments.

References

1. A. Valmari. A stubborn attack on state explosion. In *International Conference on Computer-Aided Verification*, pages 176–185, June 1990.

2. P. Godefroid. Using partial orders to improve automatic verification methods. In *International Conference on Computer-Aided Verification*, pages 176–185, June 1990.
3. K. McMillan. Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In G. v. Bochman and D. K. Probst, editors, *Proc. International Workshop on Computer Aided Verification*, volume 663 of *Lecture Notes in Computer Science*, pages 164–177. Springer-Verlag, 1992.
4. J. R. Burch. Modeling timing assumptions with trace theory. In *ICCD*, 1989.
5. M. Bozga, O. Maler, A. Pnueli, and S. Yovine. Some progress in the symbolic verification of timed automata. In *Proc. International Conference on Computer Aided Verification*, 1997.
6. R. Alur. *Techniques for Automatic Verification of Real-Time Systems*. PhD thesis, Stanford University, August 1991.
7. D. L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Proceedings of the 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 Transactions on Software Engineering*, 17(3), March 1991.
9. H. R. Lewis. Finite-state analysis of asynchronous circuits with bounded temporal uncertainty. Technical report, Harvard University, July 1989.
10. T. Yoneda, A. Shibayama, B. Schlingloff, and E. M. Clarke. Efficient verification of parallel real-time systems. In Costas Courcoubetis, editor, *Computer Aided Verification*, pages 321–332. Springer-Verlag, 1993.
11. A. Semenov and A. Yakovlev. Verification of asynchronous circuits using time Petri-net unfolding. In *Proc. ACM/IEEE Design Automation Conference*, pages 59–63, 1996.
12. E. Verlind, G. de Jong, and B. Lin. Efficient partial enumeration for timing analysis of asynchronous systems. In *Proc. ACM/IEEE Design Automation Conference*, 1996.
13. T. G. Rokicki. *Representing and Modeling Circuits*. PhD thesis, Stanford University, 1993.
14. T. G. Rokicki and C. J. Myers. Automatic verification of timed circuits. In *International Conference on Computer-Aided Verification*, pages 468–480. Springer-Verlag, 1994.
15. C. J. Myers. *Computer-Aided Synthesis and Verification of Gate-Level Timed Circuits*. PhD thesis, Stanford University, 1995.
16. C. J. Myers, T. G. Rokicki, and T. H.-Y. Meng. Automatic synthesis of gate-level timed circuits with choice. In *16th Conference on Advanced Research in VLSI*, pages 42–58. IEEE Computer Society Press, 1995.
17. W. Belluomini and C. J. Myers. Efficient timing analysis algorithms for timed state space exploration. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*. IEEE Computer Society Press, April 1997.
18. W. Belluomini and C. J. Myers. Timed event/level structures. In collection of papers from TAU'97.
19. P. Merlin and D. J. Faber. Recoverability of communication protocols. *IEEE Transactions on Communications*, 24(9), 1976.
20. M. R. Greenstreet. *STARI: A Technique for High-Bandwidth Communication*. PhD thesis, Princeton University, 1993.
21. M. R. Greenstreet. Stari: Skew tolerant communication. unpublished manuscript, 1997.
22. S. Tasiran and R. K. Brayton. Stari: A case study in compositional and hierarchical timing verification. In *Proc. International Conference on Computer Aided Verification*, 1997.
23. H. Hulgaard, S.M. Burns, T. Amon, and G. Borriello. Practical applications of an efficient time separation of events algorithm. In *ICCAD*, 1993.
24. R. Alur and R. P. Kurshan. Timing analysis in cospan. In *Hybrid Systems III*. Springer-Verlag, 1996.
25. R. A. Thacker. Implicit methods for timed circuit synthesis. Master's thesis, University of Utah, 1998.