Abstractions for Model Checking of Event Timings

Jatindra K. Deka, S. Chaki, Pallab Dasgupta and P.P. Chakrabarti

Abstract— **Verification of timed temporal properties of a circuit is a computationally complex problem both in terms of space and time. In this paper we study different abstractions of timed systems and the temporal logics which are preserved under these abstractions. In particular we show that while known timed logics such as RTCTL and TCTL are preserved by bisimulation equivalence, the timings of events (signal changes) are preserved in a more compact abstraction. Experimental results show that this abstraction requires significantly less space and is competitive in terms of time required for verification.**

I. INTRODUCTION

Temporal logic model-checking [6] is one of the most popular and well studied paradigms for formal verification of hardware and other concurrent systems (see [7] for a survey). One of the main challenges in using model checking techniques effectively is to contain the size of the state-space which is often huge in practice. Some of the most significant approaches in this direction include *symbolic model checking* [7], [10] and *automata theoretic model checking* [3].

An interesting approach towards containing the state explosion problem is to abstract a minimal transition system which preserves properties specified in a given logic. It has been shown that abstractions derived using the *bisimulation* and *stuttering* equivalence relations preserve untimed logics such as CTL, LTL and CTL*. There are known polytime algorithms for determining the abstract transition system having the minimum number of states [9], [11].

Often (and in particular when we analyze the timings of edge triggered circuits), we are interested in the timings of signal changes (such as the posedge or negedge of signals). Since the timings of *signal changes* (which we call *events*) are discrete in nature, it is an interesting objective to investigate whether abstractions which preserve the timings of events are more compact than timing preserving abstractions in general.

In this paper, we show that while *bisimulation equivalence* is timing preserving, *stuttering equivalence* is not. We then introduce an abstraction called *timed stuttering equivalence* which has fewer states than that obtained by *bisimulation equivalence*, and preserves the timings of events. We show that a delay-based partitioning of stuttering equivalent abstractions require less space than bisimulation abstractions, when the transition relation is stored in the form of BDDs. The verification time is roughly the same for both, indicating that the timed stuttering equivalent abstraction does not have much of an overhead in verification time.

II. DEFINITIONS AND PRELIMINARIES

This section presents preliminary notations and definitions used throughout this paper.

In this study, we consider synchronous transition systems only. The unit of time will be called a *cycle*. Timing properties are expressed in terms of the number of cycles. For example, the RTCTL formula, $E[true U_{[4,8]} q]$ is true in a state, s, iff there exists a path starting from s in the transition system where q is true between the 4^{th} and 8^{th} cycle. Thus, we consider only discrete time models. Such a model is quite natural for verification of sequential circuits, where timing is expressed in terms of the number of clock cycles.

Definition 1: [**Untimed Model:**]
An untimed model is a triple, $M = (S, N, L)$, where S is a set of states, $N \subseteq S \times S$ is a transition relation and L is a labeling function mapping each state into a set of atomic propositions that are true in that state. □

A path π is an infinite sequence of states $s_0, s_1, s_2, \ldots$, such that $N(s_i, s_{i+1})$ is true for every i.
We define a model for a timed systems as follows.

Definition 2: [**Timed Model:**]
A timed model is a triple, $M = (S, N, L)$, where S is a set of states, $N \subseteq S \times \mathcal{N} \times S$ is a timed transition relation where $\mathcal{N}$ is the set of positive integers and L is a labeling function mapping each state into a set of atomic propositions that are true in that state. □

In a timed model, we have an integer *delay* associated with each state transition. A *uniform timed model* is a timed model where each transition has unit delay. A sequential circuit is typically modeled by a uniform timed model where the unit delay is the clock delay.

The logic CTL (Computation Tree Logic[1]) is used to specify properties involving qualitative constraints. In CTL, the basic path operator is *until* – for example the path formula pUq (p *until* q) specifies that p holds in each state of the path until we reach a state where q is true, but does not specify any constraints on how early or how late we much reach the state where q is true. The timed logic, RTCTL, extends CTL with the *bounded until* operator which has the form: $\cup_{[a,b]}$, where [a,b] defines the time interval in which the property must be true. We say that $f \cup_{[a,b]} g$ is true in some path if g holds in some future state s on the path, where the total delay to reach s is within the interval [a,b] and f is true in all states preceding s in the path.

III. ABSTRACTIONS FOR UNTIMED SYSTEMS

In this section, we present the definitions of equivalence relations on the states of untimed transition systems which are known to preserve some of the well known temporal logics.

Definition 3: [**Bisimulation Equivalence:**]
Consider two untimed models $M = (S, N, L)$ and $M' = (S', N', L')$ with the same set of atomic propositions. A binary relation $R \subseteq S \times S'$ is called a bisimulation relation if for any $s \in S$ and $s' \in S'$, $R(s, s')$ implies $L(s) = L'(s')$

The authors are with the Dept. of Computer Science & Engg, Indian Institute of Technology Kharagpur, India 721302.

---

[1] Due to lack of space, we refer the reader to [6] for the syntax and semantics of CTL

and

$$(i) \quad (\forall r \in S, N(s,r) \Rightarrow \exists r' \in S' : N'(s',r') \wedge R(r,r'))$$

$$(ii) \quad (\forall r' \in S', N'(s',r') \Rightarrow \exists r \in S : N(s,r) \wedge R(r,r'))$$

A bisimulation equivalence is the maximum bisimulation relation in the subset inclusion preorder. □

The bisimulation equivalence can be computed in time quadratic in the sum of the sizes of the transition relations using a BDD-based algorithm [2]. More time efficient algorithms are also known [9], [11], but they do not use a compact BDD based representation. To obtain the bisimulation equivalent abstraction of a model, M, we first determine the bisimulation equivalence of M with M. Once this bisimulation equivalence is computed, the bisimulation equivalent abstraction of M can be obtained by simply combining states which are bisimulation related.

Definition 4: [**Stuttering Equivalence:**] Consider two untimed models $M = (S, N, L)$ and $M' = (S', N', L')$ with the same set of atomic propositions. A binary relation $R \subseteq S \times S'$ is called a stuttering relation if for any $s \in S$ and $s' \in S'$, $R(s, s')$ implies $L(s) = L'(s')$ and

(i) $(\forall r, N(s,r) \Rightarrow$
$\exists s'_0, \ldots, s'_n (n \geq 0)$: $s'_0 = s'$ and $R(r, s'_n)$ and
$\forall i, 0 \leq i < n$, $N'(s'_i, s'_{i+1})$ and $R(s, s'_i))$

(ii) $(\forall r', N'(s', r') \Rightarrow$
$\exists s_0, \ldots, s_m (m \geq 0)$: $s_0 = s$ and $R(s_m, r')$ and
$\forall i, 0 \leq i < m$, $N(s_i, s_{i+1})$ and $R(s_i, s'))$

A stuttering equivalence is the maximum stuttering relation in the subset inclusion preorder. □

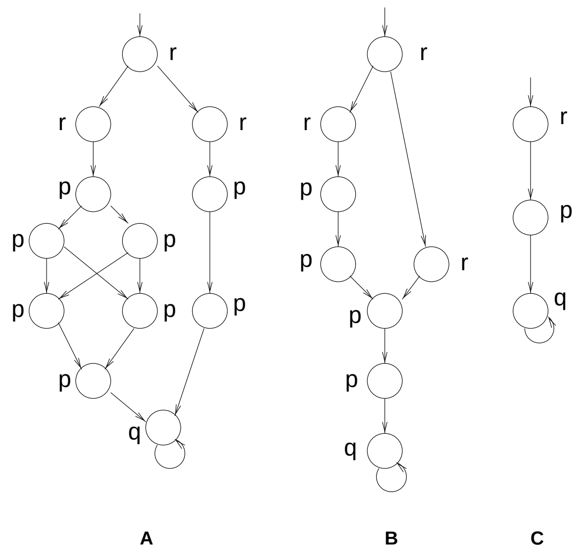

Fig. 1. Bisimulation and Stuttering Equivalence

Example 1: We illustrate the bisimulation and stuttering equivalent abstractions with the help of an example. Consider the uniform timed model shown in part A of Figure 1. The bisimulation abstraction of this model is presented in part B of Figure 1. The stuttering abstraction is shown is part C of Figure 1. □

Intuitively, stuttering equivalence does not distinguish between two paths that differ only in the number of idle cycles (that is, cycles which are equivalent in terms of the set of atomic propositions). It has been shown that stuttering equivalence preserves the truth of CTL* formulas that do not involve the next time operator X. Polynomial time algorithms are known for determining stuttering equivalence. The stuttering equivalent abstraction is obtained by merging states which belong to the stuttering equivalence of model M with respect to itself.

IV. ABSTRACTION FOR PRESERVING EVENT TIMES

In bisimulation equivalence we may have sequences of states having identical values of the signals (that is, identical set of labels). It is possible to replace these sequences by lump delays. We define *timed stuttering equivalence* on the basis of this notion.

Definition 5: [**Timed Stuttering Equivalence:**] Consider two timed models $M = (S, N, L)$ and $M' = (S', N', L')$ with the same set of atomic propositions. A binary relation $R \subseteq S \times S'$ is called a timed stuttering relation if for any $s \in S$ and $s' \in S'$, $R(s, s')$ implies $L(s) = L'(s')$ and

(i) $(\forall r, N(s,t,r) \Rightarrow$
$\exists s'_0, \ldots, s'_n (n \geq 0)$: $s'_0 = s'$ and $R(r, s'_n)$ and
$\forall i, 0 \leq i < n$, $N'(s'_i, \delta_i, s'_{i+1})$ and $R(s, s'_i)$
and $\sum_{0 \leq i < n} \delta_i = t)$

(ii) $(\forall r', N'(s', t', r') \Rightarrow$
$\exists s_0, \ldots, s_m (m \geq 0)$: $s_0 = s$ and $R(s_m, r')$ and
$\forall i, 0 \leq i < m$, $N(s_i, \delta_i, s_{i+1})$ and $R(s_i, s')$
and $\sum_{0 \leq i < m} \delta_i = t')$

A timed stuttering equivalence is the maximum stuttering relation in the subset inclusion preorder. □

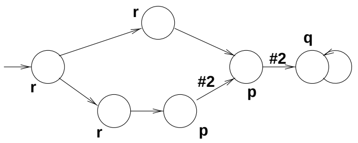

Fig. 2. Timed Stuttering Equivalence

Example 2: Fig 1 shows the bisimulation and stuttering abstractions of a timed model. The timed stuttering abstraction of the same model is shown in Figure 2. □

Given a uniform timed model, we can compute it's timed stuttering equivalent abstraction as follows. We first compute the bisimulation equivalent abstraction of the model. Let $Q \subseteq S$ denote the set of states in the bisimulation equivalent abstraction, $M = (S, N, L)$, which have only one parent state and one child state. Let $s \in Q$, $N(p, \tau, s)$, and $N(s, \delta, c)$, where p and c are respectively the unique parent state and the unique child state of s. If $L(p) \neq L(s)$, we simply remove s from Q. If $L(p) = L(s)$, then we remove s from S (and Q) and replace the transitions (p, τ, s) and (s, δ, c) by the transition $(p, \tau + \delta, c)$. We continue this process until Q is empty. It is easy to see that the total time complexity of finding the abstraction is polynomial in the number of states of the original model.

We call the states in the timed stuttering equivalent abstraction as *event states*. The states which have been replaced by lump delays are called *non-event* states. For example, Fig 2 shows only the event states of the uniform timed model of Fig 1 B. The remaining states in Fig 1 B are non-event states. It should be noted that by definition of timed stuttering equivalence, the atomic propositions true at a non-event state is the same as that of its preceding event state. The lump delays on a transition account for the number of non-event states compacted into that transition.

V. TIMED LOGICS PRESERVED UNDER ABSTRACTION

In this section, we illustrate that the untimed bisimulation equivalent abstraction of a uniform timed model preserves the truth of RTCTL properties, where as stuttering equivalent abstractions (both timed and untimed) fail to preserve RTCTL properties. We also describe a timed logic, ETCTL, and show that timed stuttering equivalent abstractions preserve ETCTL properties.

Theorem 1: The untimed bisimulation equivalent abstraction of a uniform timed model preserves the truth of RTCTL properties.

Proof: It has been shown that bisimulation equivalence preserves CTL (including the X operator)[4]. RTCTL differs from CTL only in the bounded-until operator. The bounded-until operator can be expressed recursively in terms of the X operator[5]. For example, the formula $E[fU_{[a,b]}g]$ can be expressed in the form:

$$E[fU_{[a,b]}g] = \begin{cases} f \wedge EXE[fU_{[a-1,b-1]}g] & \text{if } a > 0 \text{ and } b > 0 \\ g \vee (f \wedge EXE[fU_{[0,b-1]}g] & \text{if } a = 0 \text{ and } b > 0 \\ g & \text{if } a = 0 \text{ and } b = 0 \end{cases}$$

Other bounded-until properties can be similarly expressed in terms of the X operator, and hence the bisimulation equivalence of a uniform timed model preserves RTCTL properties. □

It has been shown[4] that properties specified with the X operator are not always preserved under stuttering equivalence. For uniform timed models we can write formulas of the form EXf and AXf as $E[trueU_{[1,1]}f]$ and $A[trueU_{[1,1]}f]$ respectively. Thus, untimed stuttering equivalence does not preserve RTCTL. In timed stuttering equivalence, we retain the timing information in terms of lump delays. Since the number of bits required to store a delay value is logarithmic in the magnitude of the delay value, the saving is significant. However, RTCTL properties are not preserved in general, unless we break down the lump delays.

Example 3: Consider the simple timed model shown in Fig 3. Suppose we wish to verify whether *there exists a computation in which the atomic proposition q remains true until at some time $t \leq 9$ the property f holds.* This can be represented in TCTL as $t.E[qU((t \leq 9) \wedge f)]$, and in RTCTL (that is, CTL with the *bounded-until* operator) as $E[q\ U_{[0,9]}\ f]$.

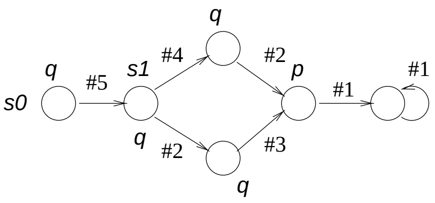

Fig. 3. A simple timed model

Suppose f is a boolean formula with atomic propositions only. Then if q holds in the event state s_0 but f does not, then it suffices to verify whether $E[qU_{[0,4]}f]$ holds in state s_1. In fact, the set of non-event states between the states s_0 and s_1 becomes transparent when f does not involve timed properties.

Now, let us consider the case when f is a property which checks whether *there exists a state exactly at a delay of 7 time units where p holds.* In RTCTL, we represent this as $E[true\ U_{[7,7]}\ p]$. Clearly f does not hold in s_0. Also $E[qU_{[0,4]}f]$ does not hold in state s_1. However, it is easy to see that $E[qU_{[0,9]}f]$ still holds in s_0, since f is true in two non-event states between s_0 and s_1. □

Example 3 illustrates the fact that though all non-event states between two successive events are equivalent in terms of the untimed properties true in them, they differ in terms of the timed properties. This leads to the creation of equivalence subsequences (in terms of properties) between non-event states and, as in other models of timed structures [1], the number of equivalence classes grows exponentially with the number of overlapping (interleaved) timing constraints.

In a previous work [8], we addressed this problem and showed that this problem does not arise when we reason about the timings of events or signal changes. In that work we proposed a logic called ETCTL which is a real-time temporal logic for reasoning about the timings of events. The formal syntax of ETCTL is as follows. B denotes the set of *boolean formulas* over the set of atomic propositions $q \in P$. Z denotes the set of *trigger formulas.* S denotes the set of ETCTL formulas.

- $B = false|true|q|\neg q|B \wedge B|B \vee B|\neg B$
- $Z = posedge(B)|negedge(B)|Z \wedge Z|Z \vee Z$
- $S = B|S \wedge S|S \vee S|E(S\ U\ S)|A(S\ U\ S)|\neg S|Z|$
  $Z \wedge E(S\ U_{[a,b]}\ S)|Z \wedge A(S\ U_{[a,b]}\ S)$

The formal semantics is presented in [8]. Intuitively, the formulas $posedge(B)$ and $negedge(B)$ represent the positive edge and the negative edge of the signal defined by the boolean formula B. These formulas (called *trigger formulas*) can be true only in event states (along certain paths). Since we constrain all *bounded-until* formulas to appear in conjunction with trigger formulas, we effectively verify all timings with respect to event states only. This guarantees (and is formally proved in [8]) that ETCTL formulas satisfy the following property. We illustrate this fact through an example.

Definition 6: **[Interval Independent Property:]** A temporal property is said to be "interval independent" if the truth of the property is identical in all non-event states between any two event states in a timed abstraction. □

Example 4: Figure 4 shows the timed model for a transmitter of digital data. The transmitter requires 16 cycles for its internal set-up, and then waits for 128 cycles to acquire the transmission medium before transmitting the data. The details of internal set-up and acquiring the transmission medium has been abstracted away in the form of lump delays of 16 and 128 respectively.

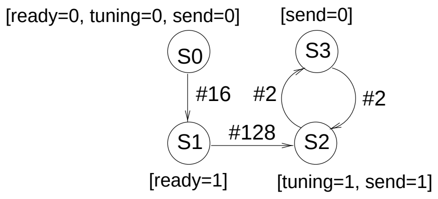

Fig. 4. Transmitter example

Suppose we are interested in verifying the following RTCTL property which says that whenever the transmitter is in a ready state, it sends data exactly after 128 cycles:

$$AG(ready \rightarrow E[true\ U_{[128,128]}\ send])$$

Now consider the sequence of 128 non-event states between the states s_1 and s_2. In order to verify the given property, we need to evaluate the truth of the subformula $E[true\ U_{[128,128]}\ send]$ in each of these states. It is easy to see that this sub-formula holds in every alternate pair of non-event states in this sequence. Thus with respect to the truth of this formula, the sequence of states between s_1 and s_2 gets partitioned into 64 sub-sequences (each having 2 states). The number of sub-sequences (in this case, $64 = 2^{8-1}/2$) can be exponential in the number of bits required to store the delay value of the edge (in this case, 8 bits). The effective number of states required for labeling during model checking is more than 64.

Delay	TSA		BA		2TSA	
	BDD Nodes	Time (ms)	BDD Nodes	Time (ms)	BDD Nodes	Time (ms)
0-0	44246	7.06	106941	28.99	60326	11.03
0-1	44157	7.24	106941	28.95	60326	11.36
0-2	44157	7.52	111967	29.15	60326	11.59
0-3	44157	7.81	117894	29.41	60326	12.01
0-4	44157	8.25	123320	29.57	61189	12.54
0-5	44157	8.72	128132	29.78	61187	13.06
0-6	44157	9.32	132370	30.04	61187	13.82
0-7	44157	10.20	135951	30.27	61187	14.71
0-8	44157	11.07	138651	30.42	61185	15.87
0-9	44157	12.02	140903	30.55	61185	17.21
0-10	44157	13.17	143372	30.68	61185	18.61
0-11	44157	14.44	145490	30.77	61185	20.42
0-12	44157	15.78	147515	30.92	61185	22.37
0-13	44157	17.23	149306	31.17	61185	24.58
0-14	44157	19.03	150897	31.27	61185	26.99
0-15	44157	20.67	152320	31.51	61185	29.95
0-16	44157	22.66	153687	31.87	61185	33.04
0-17	44157	24.65	154978	31.75	61185	36.85
0-18	44157	27.14	156189	31.79	61185	40.28
0-19	44157	29.47	157315	31.98	61185	44.16
0-20	44157	31.93	158404	32.18	61185	48.61

TABLE I

Results with different timing properties

Delay	TSA		BA		2TSA	
	BDD Nodes	Time (ms)	BDD Nodes	Time (ms)	BDD Nodes	Time (ms)
0-1	42461	7.69	42461	7.90	42461	7.75
0-2	40940	15.81	60997	10.25	40940	15.91
0-3	40014	14.73	74610	13.17	48881	14.35
0-4	40894	13.66	88539	16.69	48414	14.88
0-5	41573	12.61	102481	19.91	53013	14.78
0-6	42207	11.27	114595	23.14	55171	14.69
0-7	42773	11.28	127276	27.75	62641	17.12
0-8	43689	10.80	136141	32.07	60588	17.17
0-9	44435	11.72	149489	36.84	62151	15.90
0-10	45674	10.34	160862	38.00	63504	16.03
0-11	46382	9.70	168006	41.60	65528	16.33
0-12	46800	9.98	176606	46.30	65997	16.68
0-13	47814	9.74	185252	51.14	69215	17.58
0-14	48429	9.78	190375	54.07	71902	17.16
0-15	48903	9.49	198674	55.24	75188	19.57
0-16	49266	10.84	203128	66.21	74724	17.77
0-17	50298	9.32	208644	62.43	75224	18.18
0-18	50429	8.98	212624	64.67	73591	17.25
0-19	51310	9.02	221808	69.77	75103	18.18
0-20	51917	9.10	226367	73.73	75382	17.87

TABLE II

Variations in transition delays

Suppose we are interested to verify whether the transmitter sends data 128 cycles after it *became ready.* Note that we now want to verify a timed property with respect to the event of the transmitter *becoming ready.* We use the formula `posedge(ready)` to label states where the signal `ready` changes from 0 to 1. The desired property is now stated in ETCTL as:

$$AG(posedge(ready) \rightarrow E[true\ U_{[128,128]}\ send])$$

Since signal changes can take place only on event states, posedge(ready) can be true only on event states. Consequently, we require to verify the truth of the subformula $E[true\ U_{[128,128]}\ send]$ only on event states. This saves us the effort of determining the truth of the subformula on the exponential number of non-event states between event states. Thus unlike in the first case, the effective number of states remains as 4. □

Theorem 2: The timed stuttering equivalent abstraction of a timed model preserves ETCTL properties.

Proof: We abstract timed stuttering equivalence with respect to its bisimulation equivalent abstraction. Bisimulation equivalent abstraction is done by ignoring the delays in the timed model. Stuttering equivalence abstraction preserves CTL, and hence CTL subformulas of ETCTL are preserved in the stuttering equivalent abstraction.

We consider a state which has unique parent and unique child in bisimulation equivalent abstraction. If a state and its parent have the same labeling, we remove that state from the original timed model and update the delay accordingly. Since ETCTL formulas are interval independent (as shown in [8]), the abstraction will preserve ETCTL properties. □

VI. Experimental Results

Table I and Table II show results computed on a 550 MHz Pentium-III with Linux. The tables present results for timed stuttering abstraction (TSA), bisimulation abstraction (BA), and power-of-2 stuttering equivalent abstraction (2TSA). In 2TSA we break up long delays into sum of powers of 2 (say, 13 is broken into $8 + 4 + 1$).

For performing the experimentation we have chosen the most generic ETCTL property $\varphi = posedge(a) \wedge E(b\ U_{[0,y]}\ c)$. Table I shows the peak BDD nodes and the runtimes for verifying φ with the value of y tuned as shown in the first column of Table I. The transition system chosen randomly has delays distributed from 0 to 8. The timed stuttering equivalent abstraction has 5000 reachable states (from the start state).

Table II presents results for verifying the property $posedge(a) \wedge E(b\ U_{[0,8]}\ c)$ on different timed structures. In Table II, the leftmost column indicates the range of delay values distributed over the transition system. The peak BDD nodes and the runtimes (in milliseconds) are shown for TSA, BA, and 2TSA.

The results shown in Table I and Table II illustrate that TSA requires significantly less space than BA. Table II further shows that for transition systems having larger delay ranges, verification on TSA requires less time than on BA. Table I shows that for the same transition system and different timed properties, TSA and BA are competitive in terms of time (and TSA is better in terms of space). 2TSA represents an alternative which has performance roughly between TSA and BA.

References

[1] Alur, R., Timed Automata. Manuscript: www.cis.upenn.edu/ alur/Nato97.ps.gz, 1998.

[2] Berezin, S., Campos, S., and Clarke, E.M., Compositional Reasoning in Model Checking. *Proc. of COMPOS '97 Workshop*, LNCS Vol 1536, 1998.

[3] Bernholtz, O., Vardi, M., and Wolper, P., An automata theoretic approach to branching-time model checking. In *CAV'94: 6th Int. Conf. on Computer Aided Verification*, LNCS Vol 818, 1994.

[4] Browne, M.C., Clarke, E.M., and Grumberg, O., Characterizing finite Kripke structures in propositional temporal logic. *Theoretical Computer Science*, **59** (1-2), 1988.

[5] Campos, S., and Clarke, E.M., Real-time symbolic model checking for discrete time models. In *Theories and experiences for real-time system development*, AMAST series in comput. 1994.

[6] Clarke, E.M., Emerson, E.A., and Sistla, A.P., Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. on Program. Lang. & Systems*, **8**, 2, 244-263, 1986.

[7] Clarke, E.M., and Kurshan, R.P., Computer aided verification. *IEEE Spectrum*, **33**, 6, 61-67, 1996.

[8] Dasgupta, P., Deka, J. K., and Chakrabarti, P. P., Model Checking on Timed Event Structure. *IEEE Trans. on Computer Aided Design*, **19**, 5, 601-611, 2000.

[9] Groote J.F., and Vaandrager, F.W., An efficient algorithm for branching bisimulation and stuttering equivalence. In *Proc. of* 17^{th} *ICALP, Warwick*, LNCS, Vol 443, 626-638, 1990.

[10] McMillan, K.L., *Symbolic Model Checking*, Kluwer, 1993.

[11] Paige, R., and Tarjan, R., Three efficient algorithms based on partition refinement, *SIAM Journal of Computing*, 16(6), 1987.