On Timing Analysis of Combinational Circuits*

Ramzi Ben Salah, Marius Bozga and Oded Maler

VERIMAG, 2, av. de Vignate, 38610 Gieres, France
Ramzi.Salah@imag.fr Marius.Bozga@imag.fr Oded.Maler@imag.fr

Abstract. In this paper we report some progress in applying timed automata technology to large-scale problems. We focus on the problem of finding maximal stabilization time for combinational circuits whose inputs change only once and hence they can be modeled using acyclic timed automata. We develop a "divide-and-conquer" methodology based on decomposing the circuit into sub-circuits and using timed automata analysis tools to build conservative low-complexity approximations of the sub-circuits to be used as inputs for the rest of the system. Some preliminary results of this methodology are reported.

1 Introduction

It is well known that timed automata (TA) [AD94] are well suited for modeling delays in digital circuits [D89,L89,MP95]. Although some applications of TA technology for solving timing-related problems for such circuits have been reported [MY96,BMPY97,TKB97,TKY+98,BMT99,BJMY02], the state- and clock-explosion associated with such models, restricted the applicability of TA to small circuits. In this work we try to treat larger combinational circuits by using the old-fashioned recipe of abstraction and approximation. When viewed from a purely-functional point of view, combinational circuits realize instantaneous Boolean functions. However, when gate delays are taken into account, the computation of that function is not considered anymore as an atomic action but rather as a process where changes in the inputs are gradually propagated to the outputs. The question of finding the worst-case propagation delay of the circuit, that is, the maximal time that may elapse between a change in the inputs and the last change in the outputs, is of extreme practical importance as it determines, for example, the frequency of the clock with which a circuit can operate. Static techniques currently practiced in industry are based on finding the longest (in terms of accumulated delays) path from inputs to outputs in the circuit. While these bounds are easy to compute (polynomial in the size of the circuit), they can be over pessimistic because they abstract from the particular logic of the circuit

* This work was partially supported by a grant from Intel and by the European Community Projects IST-2001-35304 AMETIST (Advanced Methods for Timed Systems), http://ametist.cs.utwente.nl

which may prevent such longest paths from being exercised.[1] On the other hand, models based on timed automata do express the interaction between logic and timing and hence can lead to more accurate results. Alas, TA-based techniques are still very far from being applicable to industrial-size circuits.

The present paper attempt to find a better trade-off between accuracy and tractability by using timed automata as an underlying semantic model and by applying abstraction techniques to parts of the circuit in order to build for them small over-approximating timed automata that can be plugged as inputs to other parts of the circuit. Our abstraction technique takes advantage of the acyclic nature of the circuits and their corresponding automata, which implies, among other things, that every variable changes finitely many times before stabilization in every run of the automaton.

The rest of the paper is organized as follows: in Section 2 we give a formal definition of circuits, their "languages" and the maximal stabilization time problem. In section 3 we explain the modeling of such circuits as timed automata. Section 4 is devoted to our abstraction technique, its properties and the way it is implemented using the tools IF/Kronos and Aldebaran. Preliminary experimental results are reported in Section 5 followed by a discussion of related work and future directions.

2 Timed Boolean Circuits

Throughout this paper we restrict ourselves to acyclic circuits.

Definition 1 (Boolean Circuits). *A Boolean circuit is $C = (V, \rightsquigarrow, F)$ where V is a set of nodes, $\rightsquigarrow$ is an irreflexive and anti-symmetric binary relation and F is a function that assigns to every non-input node v a Boolean function F_v*

Here $v \rightsquigarrow v'$ means that v influences v' directly. The transitive closure of $\rightsquigarrow$, $\overset{*}{\rightsquigarrow}$, induces a strict partial order $(V, \overset{*}{\rightsquigarrow})$ where the minimal elements are called *input* nodes and are denoted by V_x. The rest of the nodes are called non-input nodes and denoted by V_y. A subset V_z of V consists of output nodes, those that are observable from the outside. An example appears in Figure 1-(a). The set of immediate *predecessors* of a node is $\pi(v) = \{v' : v' \rightsquigarrow v\}$ and the set of its predecessors (backward cone) is $\pi^*(v) = \{v' : v' \overset{*}{\rightsquigarrow} v\}$.

By substitution we define for every node v a function G_v defined on the inputs in its backward cone, for example, $G_{y_3}(x_1, x_2) = f_3(x_1, f_2(x_1, x_2))$. We will use $X = \mathbb{B}^{|V_x|}$, $Y = \mathbb{B}^{|V_y|}$ and $Z = \mathbb{B}^{|V_z|}$ to denote the sets of possible

[1] A lot of effort has been invested in the problem of detecting such "false paths".

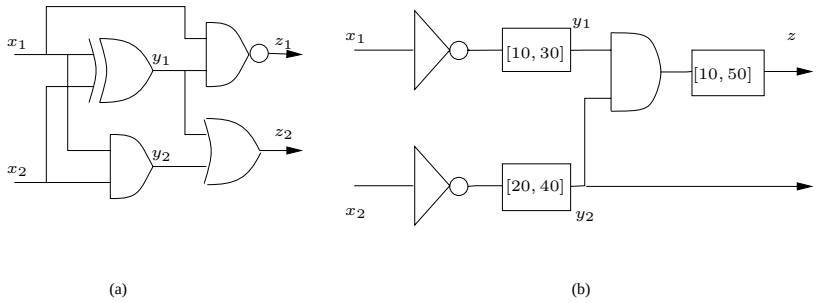

Fig. 1. (a) A Boolean circuit; (b) A timed Boolean circuit.

assignments to input, non-input and output nodes, respectively. The whole circuit can be viewed as computing a function $G : X \to Y$. The *stable state* of the circuit associated with an input vector $\mathbf{x} \in X$ is $\mathbf{y} = G(\mathbf{x})$.

This concludes the formalization of Boolean circuits and their functions. These functions are *instantaneous* with no notion of time. The next step is to lift them to functions (operators) on signals, that is, on functions that specify the evolution of a value over (continuous) time.

Definition 2 (Signals). *Let A be a set and let $\mathcal{T} = \mathbb{R}_+$ be a time domain. An A-valued signal over $\mathcal{T}$ is a partial function $\alpha : \mathcal{T} \to A$ whose domain of definition is an interval $[0, r)$ for some $r \in \mathcal{T}$.*

We use $\alpha[t]$ to denote the value of α at t, and $\alpha[t] = \perp$ to denote the fact that a is not defined at t. When A is finite, signals are piecewise-constant and make discontinuous jumps at certain points in time. This is formalized as follows. The *left limit* of a signal α at time t is defined as $\alpha[t^-] = \lim_{t' \to t} \alpha[t']$. For every piecewise-constant signal α we define:

- The ordered set of *jump points*, $\mathcal{J}(\alpha) = \{t : \alpha[t^-] \neq \alpha[t]\} = \{t_0, t_1, \ldots\}$.
- The set of *maximally-uniform intervals* $\mathcal{I}(\alpha) = \{I_1, I_2, \ldots\}$ where $I_i = [t_{i-1}, t_i)$ for $t_{i-1}, t_i \in \mathcal{J}(\alpha)$.

Clearly, the value of α is uniform over any subset of a maximally-uniform interval. We restrict our attention to well-behaving signals i.e. those for which $\mathcal{J}(\alpha)$ has finitely-many elements in any finite interval. We denote the set of A-valued signals by $\mathcal{S}(A)$.

When a gate or any other I/O device gets a signal as an input, it transforms it into an output signal. This is captured mathematically by what is called a *transducer*, or a *signal operator*, a function that maps signals to signals. We restrict such functions to be *causal*, that is, the value of the output at time t can

depend only on the value of the input in times $[0, t]$ and not on later values. The simplest type of operators are memoryless (instantaneous) operators defined as follows.

Definition 3 (Memoryless Operators). *A memoryless signal operator is a function $f : \mathcal{S}(A) \to \mathcal{S}(B)$ obtained as a pointwise extension of a function $f : A \to B$, that is, $\beta = f(\alpha)$ if $\beta[t] = f(\alpha[t])$ for every t in the domain of α.*

In reality, since gates are realized by continuous physical processes, it takes some time to propagate changes from input to output ports. To define this phenomenon mathematically we need the basic operator with memory for discrete-valued signals, the delay, which takes a signal and "shifts" it in time. One can define a variety of delay operators differing from each other in complexity and in physical faithfulness. The class of models that we consider is called *bi-bounded* inertial delays [BS94] and is characterized by an interval $I = [l, u]$ which gives lower and upper bounds on the propagation delay. For the purpose of this paper we will use the model introduced in [MP95] but since the choice of the delay model is orthogonal to the rest of the methodology we will defer the exact definition of the operator to Section 3 where it will be defined in terms of its corresponding timed automaton and use meanwhile a general semi-formal definition.

Definition 4 (Delay Operators). *A delay operator is a non-deterministic function of the form $D_I : A \times \mathcal{S}(A) \to 2^{\mathcal{S}(A)}$ where $I = [l, u]$ is a parameter of the operator with $l > 0$. A signal β is in $\Delta_I(b, \alpha)$ if*

1. *The value of β is b at the initial interval $[0, t)$;*
2. *Changes in α are not propagated to β before l time elapses;*
3. *Changes in α must be propagated to β if they persist for u time;*
4. *Changes in α that persist for less then l time are not propagated at all to β.*

Figure 2 illustrates such an operator which, typically, will have uncountably-many output signals for an input signal. All signal operators can be lifted naturally into operators on sets of signals.

A timed circuit model is obtained from a Boolean circuit by connecting the output of every non-input node to a delay operator which models the delay associated with the computation of that node (see Figure 1-(b). In other words, a gate with a propagation delay is modeled as a composition of a memoryless Boolean operator and a delay operator (see [MP95]).

Definition 5 (Timed Boolean Circuits). *A timed Boolean circuit is $C = (V, \rightsquigarrow, F, I)$ where $(V, \rightsquigarrow, F)$ is a Boolean circuit and I is a function assigning to every non-input node v a delay interval $I_v = [l_v, u_v]$ such that $0 < l_v \leq u_v < \infty$.*

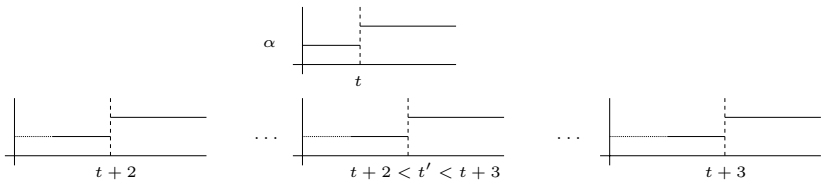

Fig. 2. An input signal α and few of the elements of $D_{[2,3]}(0, \alpha)$.

The semantics of a timed circuit is given in terms of a non-deterministic transducer $F_C : Y \times \mathcal{S}(X) \to 2^{\mathcal{S}(Y)}$ such that $\beta \in F_C(\mathbf{y}, \alpha)$ if α and β satisfy the set of signal inclusions associated naturally with the circuit [MP95] and $\mathbf{y}$ is the initial state of the non-input part of the circuit.

The stabilization time problem is motivated by the use of Boolean circuits in synchronous sequential machines (the hardware name for automata). At the beginning of every clock cycle new input values together with the values of memory elements (computed in the previous cycle) are fed into the circuit and the changes are propagated until the circuit stabilizes and the clock falls. The "width" of the clock needs to be large enough to cover the longest possible stabilization time of the circuit over all admissible inputs. In our modeling approach we will consider primary inputs that change at most once and within a bounded amount of time and hence, due to acyclicity and the finite upper-bounds associated with the delays, they induce finitely many changes throughout the circuit.

Definition 6 (Ultimately-Constant Signals). *A signal α is* ultimately-constant *(u.c.) if it has a finite number of jump points (i.e. there is some time t such that the signal remains constant after t). The minimal such t for α is called its stabilization time and is denoted by $\theta(\alpha)$. This definition extends to sets of signals by letting $\theta(L) = \max\{\theta(\alpha) : \alpha \in L\}$.*

The following properties hold for every $u.c.$ signal α:

1. The signal $f(\alpha)$ is also u.c. for every Boolean function f.
2. For a delay operator D_I with $I = [l, u]$ and for every $\beta \in D_I(\alpha)$, $\theta(\beta) \leq \theta(\alpha) + u$.

Consequently, u.c. inputs to acyclic timed circuits produce u.c. outputs. Constant signals constitute a special class of u.c. signals and we will use $\alpha_{\mathbf{x}}$ to denote a signal whose value is constantly $\mathbf{x}$.

We can now define the problem of maximal stabilization time of a circuit with respect to a pair of input vectors $\mathbf{x}$ and $\mathbf{x}'$ where $\mathbf{x}$ is the input presented in the preceding cycle, and which determines the initial (stable) state, and $\mathbf{x}'$ is the

value of a new constant signal. We denote by $L(C, \mathbf{x}, \mathbf{x}')$ the set of Y-signals $\beta \in F_C(\mathbf{y}, \alpha_{\mathbf{x}'})$ when the circuit is initialized with the stable state $\mathbf{y} = G(\mathbf{x})$.

Definition 7 (Stabilization Time of a Circuit). *Given a timed Boolean circuit $C = (V, \rightsquigarrow, F, I)$ and two input vectors $\mathbf{x}, \mathbf{x}' \in X$ the stabilization time associated with $(\mathbf{x}, \mathbf{x}')$ is $\theta(C, \mathbf{x}, \mathbf{x}') = \max\{\theta(\beta) : \beta \in L(C, \mathbf{x}, \mathbf{x}')\}$ and the maximal stabilization time of the circuit is $\theta(C) = \max\{\theta(C, \mathbf{x}, \mathbf{x}') : \mathbf{x}, \mathbf{x}' \in X\}$.*

3 Modeling with Timed Automata

Timed automata are automata augmented with continuous clock variables whose values grow uniformly at every state. Clocks can be reset to zero at certain transitions and tests on their values can be used in conditions for enabling transitions.

Definition 8 (Timed Automaton). *A timed automaton is $\mathcal{A} = (Q, C, I, \Delta)$ where Q is a finite set of states, C is a finite set of clocks, I is the staying condition (invariant), assigning to every $q \in Q$ a conjunction I_q of inequalities of the form $c \leq u$, for some clock c and integer u, and Δ is a transition relation consisting of elements of the form (q, ϕ, ρ, q') where q and q' are states, $\rho \subseteq C$ and ϕ (the transition guard) is a conjunction of formulae of the form $(c \geq l)$ for some clock c and integer l.*

A *clock valuation* is a function $\mathbf{v} : C \to \mathbb{R}_+ \cup \{0\}$ and a *configuration* of the automaton is a pair $(q, \mathbf{v})$ consisting of a discrete state (location) and a clock valuation. Every subset $\rho \subseteq C$ induces a reset function Reset_ρ on valuations which resets to zero all the clocks in ρ and leaves the other clocks unchanged. We use **1** to denote the unit vector $(1, \ldots, 1)$ and **0** for the zero vector. We will use the term *constraints* to refer to both guards and staying conditions. A *step* of the automaton is one of the following:

- A discrete step: $(q, \mathbf{v}) \xrightarrow{\delta} (q', \mathbf{v}')$, for some transition $\delta = (q, \phi, \rho, q') \in \Delta$, such that $\mathbf{v}$ satisfies ϕ and $\mathbf{v}' = \mathrm{Reset}_\rho(\mathbf{v})$.
- A time step: $(q, \mathbf{v}) \xrightarrow{t} (q, \mathbf{v} + t\mathbf{1})$, $t \in \mathbb{R}_+$ such that $\mathbf{v} + t\mathbf{1}$ satisfies I_q.

A *run* of the automaton starting from a configuration $(q_0, \mathbf{v}_0)$ is a finite sequence of steps

$$\xi : \quad (q_0, \mathbf{v}_0) \xrightarrow{t_1} (q_1, \mathbf{v}_1) \xrightarrow{t_2} \cdots \xrightarrow{t_n} (q_n, \mathbf{v}_n).$$

We model timed circuits as a composition of timed automata such that each automaton may observe the states of other automata and refer to them in its transition guards and staying conditions.[2] The automaton for a Boolean gate of

[2] To avoid over-formalization we do not define "open" interacting automata. Such definitions can be found in [MP95].

the form $y = f(x_1, x_2)$ is just a trivial one-state automaton that has self-looping transitions for all tuples (x_1, x_2, y) that satisfy the equation. In fact, this is not really an automaton but an instantaneous logical constraint that must always be satisfied. The automaton for the delay operator $D_{[l,u]}$ (Figure 3) has four states, 0, $0'$, 1, $1'$. The 0 and 1 states are stable, that is, the values of the output of the delay is consistent with its input x. When at state 0, if the input changes to 1, the automaton moves to an unstable state $0'$ and resets a clock C to zero. It can stay at $0'$ as long as $C < u$ and can switch to stable state 1 as soon as $C \geq l$. If the input changes back to 0 before the transition to 1 the automaton returns to 0. We call these three types of transitions *excite*, *stabilize* and *regret*, respectively. Note that states 0 and $0'$ are indistinguishable from the outside and another automaton will see a change from 0 to 1 only after the "stabilize" transition.

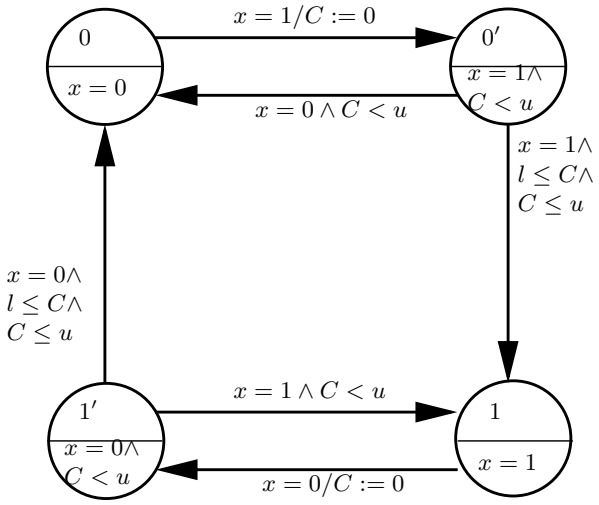

Fig. 3. The timed automaton for a delay element. The x variable refers to the observable state of the input automaton which is 0 at $\{0, 0'\}$ and 1 at $\{1, 1'\}$.

Composing all the automata, together with the model of their inputs we obtain a closed automaton as in Definition 8 whose semantics is identical to that of the timed circuit [MP95]. To be more precise, an automaton whose semantics is $L(C, \mathbf{x}, \mathbf{x}')$ is obtained by letting the initial state be the stable state corresponding to $G(\mathbf{x})$ and composing it with a static automaton for the input $\mathbf{x}'$. The obtained automaton is acyclic and all paths converge in finite time to the only stable state that corresponds to $G(\mathbf{x}')$. The maximal stabilization time is hence the maximal time that the automaton can stay in any unstable state. Note that

in such a state at least one of the components is in a $0'$ or $1'$ state and hence its staying condition forces it to leave the state.

We recall some definitions commonly-used in the verification of timed automata [HNSY94,Y97,LPY97,BDM+98,A99]. A *zone* is a set of clock valuations consisting of points satisfying a conjunction of inequalities of the form $c_i - c_j \geq d$ or $c_i \geq d$. A *symbolic state* is a pair (q, Z) where q is a discrete state and Z is a zone. It denotes the set of configurations $\{(q, \mathbf{z}) : \mathbf{z} \in Z\}$. Symbolic states are closed under the following operations:

- The *time successor* of (q, Z) is the set of configurations which are reachable from (q, Z) by letting time progress without violating the staying condition of q:

$$Post^t(q, Z) = \{(q, \mathbf{z} + r\mathbf{1}) : \mathbf{z} \in Z, r \geq 0, \mathbf{z} + r\mathbf{1} \in I_q\}.$$

  We say that (q, Z) is *time-closed* if $(q, Z) = Post^t(q, Z)$.
- The *δ-transition successor* of (q, Z) is the set of configurations reachable from (q, Z) by taking the transition $\delta = (q, \phi, \rho, q') \in \Delta$:

$$Post^\delta(q, Z) = \{(q', \mathrm{Reset}_\rho(\mathbf{z})) : \mathbf{z} \in Z \cap \phi\}.$$

- The *δ-successor* of a time-closed symbolic state (q, Z) is the set of configurations reachable by a δ-transition followed by passage of time:

$$Succ^\delta(q, Z) = Post^t(Post^\delta(q, Z)).$$

The forward reachability algorithm for TA starts with an initial zone and generates all successors until termination, while doing so it generates the *reachability graph* (also known as the simulation graph).

Definition 9 (Reachability Graph). *The reachability graph associated with a timed automaton starting from a state s is a directed graph $S = (N, \rightarrow)$ such that N is the smallest set of symbolic states containing $Post^t(s, \{\mathbf{0}\})$ and closed under $Succ^\delta$. The edges are all pairs of symbolic states related by $Succ^\delta$.*

The fundamental property of the reachability graph is that it admits a path from (q, Z) to (q', Z') if and only if for every $\mathbf{v}' \in Z'$ there exists $\mathbf{v} \in Z$ and a run of the automaton from $(q, \mathbf{v})$ to $(q', \mathbf{v}')$. Hence the union of all reachable symbolic states gives exactly the reachable configurations.

To compute the maximal stabilization time we add an auxiliary clock T which is never reset to zero and hence in every reachable configuration its value represents the total time elapsed since the beginning of the run. The maximal value of T over all reachable symbolic states (q, Z) with q unstable is the maximal stabilization time (note that due to acyclicity the value of T is bounded in all unstable states). Hence, the problem of maximal stabilization time can, in principle, be solved using standard TA verification tools.

4 The Abstraction Technique

Given the complexity of TA verification we move to an abstraction methodology based on the following simple idea. We decompose the circuit into sub-circuits small enough to be handled completely by TA verification tools. We take the automaton $\mathcal{A}$ which corresponds to such a sub-circuit and use its reachability graph to construct an automaton $\hat{\mathcal{A}}$ having two important properties:

1. The set $L(\hat{\mathcal{A}})$ of signals that it generates is a reasonable over-approximation of the projection of $L(\mathcal{A})$ on the output variables of the circuit.
2. It is much smaller than $\mathcal{A}$ in terms of states and clocks.

Hence if we replace $\mathcal{A}$ by $\hat{\mathcal{A}}$ as a model of the sub-circuit we are guaranteed to over-approximate the semantics of the circuit and hence to over-approximate the stabilization time.

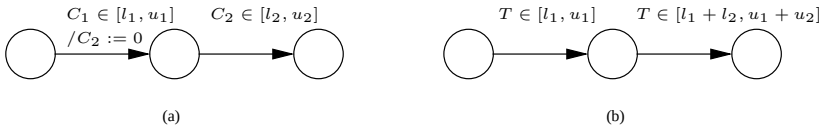

Fig. 4. Projection on the absolute time introduces spurious runs.

To better understand the technique it is worth looking at the reachability graph from a different angle. In timed automata, as in any other automata augmented with auxiliary variables, the transition graph is misleading because a discrete state stands for many possible clock valuation which may differ in the constraints they satisfy and hence in the behaviors that can be generated from them. It might be the case that a state q will never be reached with a clock valuation satisfying some transition guard and hence the corresponding transition will never be taken. By performing the reachability algorithm for $\mathcal{A}$ starting from an initial state we obtain a graph which represents the "feasible part" of $\mathcal{A}$, excluding behaviors that violate timing constraints. Figure 5-(a) shows the reachability graph for the circuit of Figure 1-(b) where the inputs change from $(0, 1)$ to $(1, 0)$. In fact the reachability graph can serve as a skeleton of another timed automaton $\mathcal{A}'$ whose semantics in terms of runs is equivalent to that of $\mathcal{A}$. To see that, one just has to associate with each symbolic state (q, Z) the staying condition Z and label each transition $(q, Z) \xrightarrow{\delta} (q', Z')$ by the guard and reset of δ. The resulting automaton $\mathcal{A}'$ differs from $\mathcal{A}$ in two aspects: certain states of $\mathcal{A}$ are split into several copies according to clock values, and all transitions that are not possible in $\mathcal{A}$ due to timing constraints do not appear in $\mathcal{A}'$ at all.

Now if we relax some timing constraints in $\mathcal{A}'$ we may introduce spurious behaviors that violate these constraints, however we will *not* add any new *qualitative* behavior (sequence of events) that was not possible in $\mathcal{A}$ because such behaviors have already been eliminated while computing the reachability graph. The most straightforward way to relax timing constraints is to project the constraints on a subset of the clocks and discard the rest. In particular if we throw away all clocks except T which measures the absolute time, the relaxed guard for any transition will be of the form $T \in [t_1, t_2]$. Clearly, a transition can be taken in the new automaton iff there is a run of the original automaton in which the corresponding transition could be taken at some time $t \in [t_1, t_2]$. However, this abstraction can add additional runs which are impossible in the original automaton as the following example shows. Consider the automaton of Figure 4-(a) where the first transition could take place in $[l_1, u_1]$ while the second can take place between l_2 and u_2 *after* the occurrence of the first. Applying the above procedure we obtain the automaton of Figure 4-(b) where the second transition could be taken anywhere in $[l_1 + l_2, u_1 + u_2]$ regardless of the time of the first.

The next step is to hide transitions which are not observable from the outside, i.e. all transitions of non-output variable and all non-visible transitions ("excite" and "regret") of the output variables y_2 and z. The one-clock automaton thus obtained for our example appears in Figure 5-(b). We then apply a minimization algorithm which merges states that are indistinguishable with respect to the remaining visible transitions. More formally we consider the congruence relation $\sim$ on the nodes of the labeled reachability graph defined as the largest relation satisfying:

$$q_1 \sim q_2 \text{ iff } \forall \delta, I \;\; q_1 \stackrel{\tau^* \cdot (\delta, I)}{\longrightarrow} q_1' \Rightarrow (\exists q_2' \text{ s.t. } q_2 \stackrel{\tau^* \cdot (\delta, I)}{\longrightarrow} q_2' \wedge q_1' \sim q_2'). \quad (1)$$

Here (δ, I) stands for a transition-interval pair and τ^* to an arbitrary sequence of unobservable transition. This relation is the "safety bisimulation" of [BFG+91]. The minimized automaton, whose states are congruence classes of $\sim$, can be seen in Figure 6-(a).

Relation (1) looks at transition labels in a purely-syntactic manner, that is, the label $-y_2[20, 30]$ in Figure 6-(a) is considered distinct from $-y_2[20, 40]$ and hence the transitions are not merged. To obtain a more aggressive abstraction we define a weaker equivalence $\sim'$ that ignores differences in intervals:

$$q_1 \sim' q_2 \text{ iff } \forall \delta, I \;\; q_1 \stackrel{\tau^* \cdot (\delta, I)}{\longrightarrow} q_1' \Rightarrow (\exists q_2', I' \text{ s.t. } q_2 \stackrel{\tau^* \cdot (\delta, I')}{\longrightarrow} q_2' \wedge q_1' \sim' q_2'). \quad (2)$$

The states of the minimized automaton are equivalence classes of $\sim'$ and the transitions between these classes are labeled by $(\delta, \bar{I})$ where $\bar{I}$ is the join (convex hull) of all the intervals I_i such that there are transition labeled by (δ, I_i)

between elements of the corresponding classes (see Figure 7).[3] The result of minimization with respect to $\sim'$ appears in Figure 6-(b) and one can see that it gives a succinct over-approximation of the behavior of y_2 and z.

We have implemented the above mentioned technique. Our tool chain starts with a circuit description as Boolean equations with delays and generates from it automatically a network of interacting timed automata written in the IF format [BGM02]. After generating the reachability graph with the interval labels we apply the Aldebaran tool set ([BFKM97]), slightly modified to implement minimization with respect to $\sim'$ to obtain the abstract model.

5 Experimental Results

We have conducted some preliminary experiments with our approach on some sample circuits that we have constructed. First, to demonstrate the semantic advantage of timed automata we analyzed the circuit of Figure 8 which has a false path. We use delays of $[83, 85]$ for all gates (except the inverters that have zero delay) and compare our results with static timing analysis which gives stabilization time of $7 \times 85 = 594$. Since our method works for the moment for one pair of input vectors, we repeat the analysis for all 12 pairs and obtain the results of Table 1. As one can see, the TA-based analysis discovers that the maximal stabilization time is only $6 \times 85 = 510$.

x	00			01			10			11		
x′	10	01	11	11	00	10	00	11	01	01	10	00
stab-time	510	340	340	170	510	425	510	0	255	255	0	510

Table 1. Maximal stabilization time for all input pairs for the circuit of Figure 8.

The major set of experiments was conducted on circuits consisting of a sequential concatenation of an increasing number of copies of the circuit of Figure 1-(a) (the y_3 and y_4 of stage n are the x_1 and x_2 of stage $n+1$). We assume that input x_1 may rise anywhere in $[10, 35]$ and x_2 in $[15, 63]$. In general, the complexity of the reachability graph is sensitive to the choice of delay bounds: for an interval $[l, u]$, the larger is the ratio $(u - l)/l$, more "scenarios" are possible and transitions at "deep" gates can precede transitions in gates closer to the input.[4] Table 2 shows the performance of our technique (computation time and size of the reachability graph) as a function of the number of stages for three

[3] Another choice might be to join only intervals that have a non-empty intersection.

[4] In fact, if we assume no lower-bound on the delay (the "up-bounded" model of [BS94]), events can happen in any order.

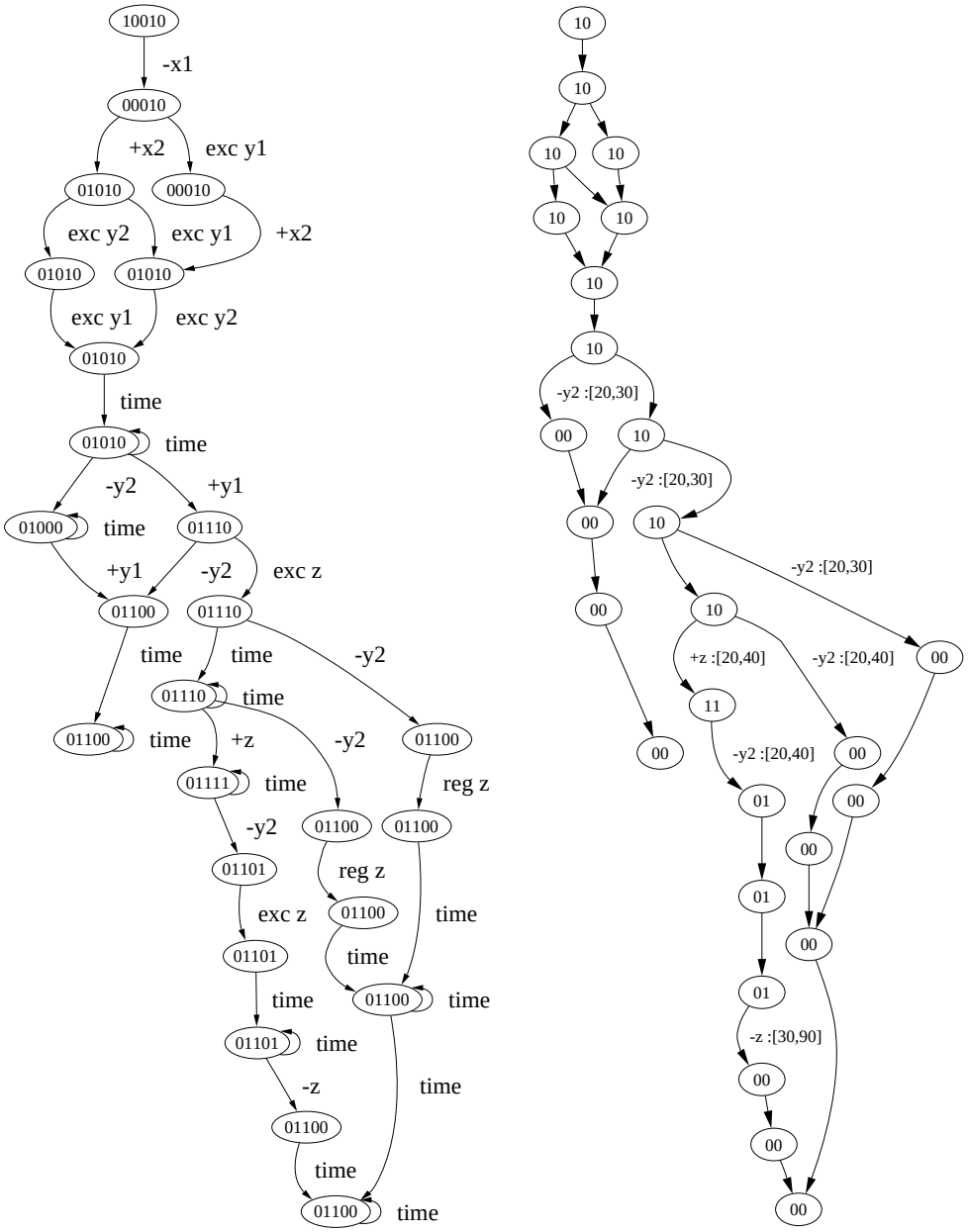

Fig. 5. (a) The reachability graph for the circuit of Figure 1-(b). The transition labels `exc z`, `reg z`, `+z` and `-z` correspond, respectively, to excitation, regret, rising and falling of the variable z. (b) The corresponding one-clock automaton after hiding internal transitions. The label `+z[20,30]` means that z may change from 0 to 1 anytime inside the interval $[20, 30]$.

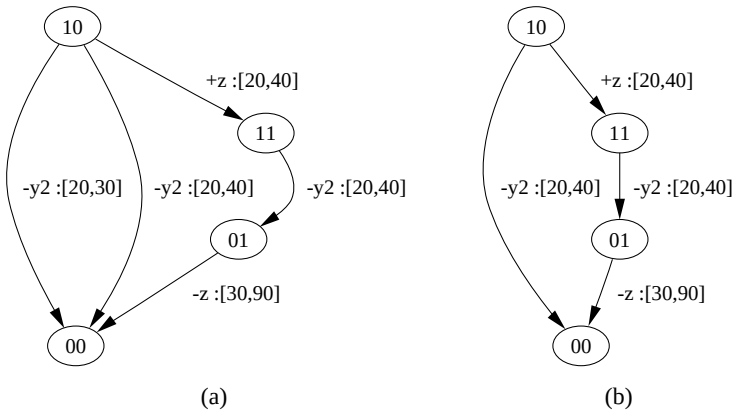

Fig. 6. (a) The results of applying standard minimization. (b) The result of minimization with interval fusion.

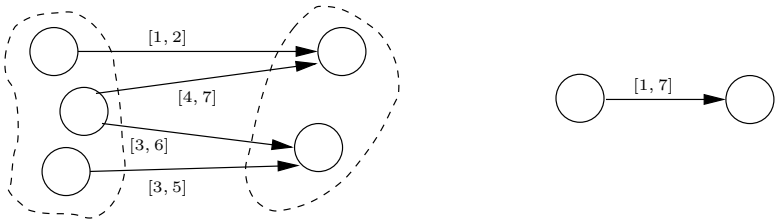

Fig. 7. Minimization by joining intervals.

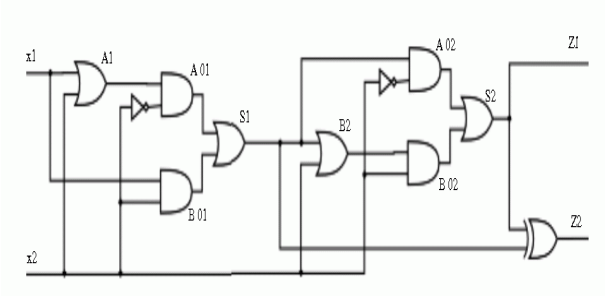

Fig. 8. A circuit with a false path.

choices of gate delay intervals $[1, 2]$, $[10, 12]$ and $[100, 102]$. All the experiments were stopped upon memory overflow (1GB). For the $[100, 102]$ interval we were able to analyze up to 22 stages (88 gates).

$[l, u]$	$[1, 2]$			$[10, 12]$			$[100, 102]$		
no.	states	min	time	states	min	time	states	min	time
1	71	4	0:01	65	3	0:01	65	3	0:01
2	934	12	0:02	270	7	0:02	270	7	0:02
3	–	–	–	2690	11	0:03	2690	11	0:04
4				5397	23	0:05	4080	16	0:06
5				217951	144	9:44	21498	30	0:12
6				–	–	–	50543	39	0:30
7							73502	48	1:01
8							95619	57	1:54
9							117736	66	3:12
10							139853	75	5:08
11							161970	84	7:32
12							184087	93	10:05
13							206204	102	14:42
14							228321	111	20:39
15							250438	120	28:15
16							272555	129	36:46
17							117736	138	49:36
18							316789	147	1:04:04
19							338906	156	1:21:48
20							361023	165	1:42:59
21							383140	174	1:58:56
22							405257	183	2:30:31
23							–	–	–

Table 2. Testing our technique with varying delay bounds. The 'states' column indicates the number of symbolic states in the model of stage n before the last minimization and the 'min' columns indicate the number of states after minimization. The 'time' column indicates the time for computing the abstraction of all stages up to $n - 1$ and the reachability graph for stage n.

As the results show, currently the analysis of circuits with few dozens of gates for one pair of input vectors is feasible using our technique. This is a significant improvement for TA technology but still a small step toward industrial-size circuits. The current bottleneck is the memory consumption while generating the reachability graph and we believe the situation can be improved significantly if we modify the algorithm to take advantage of the acyclic nature of the automata.

6 Discussion

There have been numerous publications on abstraction in general and abstraction of timed systems in particular, e.g. [AIKY95,WD94,B96,PCKP00], some based on relaxing the timing constraints and refining them successively if the abstract system cannot be verified. In [TAKB96] an assume-guarantee framework is defined for timed automata, which is used later to verify a multi-stage asynchronous circuit [TB97] by using small abstractions for each stage. These abstractions are generated manually. The closest work to ours is [ZMM03] which uses timed Petri nets for describing circuits and their desired properties. To abstract a circuit they apply "safe transformations" that consist of hiding of internal actions and clocks, and possibly over-approximating the set of behaviors. This work does is not specialized to acyclic circuits and the formal properties of the abstraction (defined in terms of *trace theory*) seem to be more complicated. Other attempts to solve the maximal stabilization time using TA are reported in [TKB97,TKY$^+$98].

Due to space limitation we do not discuss here possible variation of the techniques such as different abstraction styles, nor other important ingredients of the methodology such as the partitioning strategy. The adaptation of the technique to cyclic circuits and to open systems in general is a very challenging goal whose achievement can have a big impact on the design of timed systems.

Acknowledgment: Many colleagues have contributed throughout the years to the work described in this paper. In particular, Sergio Yovine and Stavros Tripakis helped us a lot in understanding various aspects of the verification of timed automata. They also participated in previous efforts to apply it to circuit analysis. The development of the abstraction methodology was carried out at certain points in time by Nishant Sinha, Fadhel Graiet, Olfa Ben Sik Ali and Bara Diop. Comments and criticism made by Avi Efrati, Ken Stevens and anonymous referees improved the quality of the paper.

References

[A99] R. Alur, Timed Automata, *Proc. CAV'99* LNCS 1633, 8-22, Springer, 1999.

[AD94] R. Alur and D.L. Dill, A Theory of Timed Automata, *Theoretical Computer Science* 126, 183–235, 1994.

[AIKY95] R. Alur, A. Itai, R.P. Kurshan and M. Yanakakis, Timing Verification by Successive Approximation, *Information and Computation* 118, 142-157, 1995.

[B96] F. Balarin, Approximate Reachability Analysis of Timed Automata, *Proc. RTSS'96*, 52-61, IEEE, 1996.

[BGM02] M. Bozga, S. Graf and L. Mounier, IF-2.0: A Validation Environment for Component-Based Real-Time Systems, In *Proc. of CAV'02*, LNCS 2404, Springer, 2002.

[D89] D. Dill, Timing Assumptions and Verification of Finite-State Concurrent Systems, in *Automatic Verification Methods for Finite State Systems*, LNCS 407, Springer, 1989.

[BFKM97] M. Bozga, J.-C. Fernandez, A. Kerbrat and L. Mounier, Protocol Verification with the Aldebaran Toolset, *Software Tools for Technology Transfer* 1, 166-183, 1997.

[BDM+98] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine, Kronos: a Model-Checking Tool for Real-Time Systems, *Proc. CAV'98*, LNCS 1427, Springer, 1998.

[BJMY02] M. Bozga, H. Jianmin, O. Maler and S. Yovine, Verification of Asynchronous Circuits using Timed Automata, *ENTCS* 65, 2002.

[BFG+91] A. Bouajjani, and J.-C. Fernandez, S. Graf, C. Rodriguez and J. Sifakis, Safety for Branching Time Semantics, *Proc. ICALP'91*, LNCS 510, Springer, 1991.

[BMPY97] M. Bozga, O. Maler, A. Pnueli, S. Yovine, Some Progress in the Symbolic Verification of Timed Automata, in *Proc. CAV'97*, 179-190, LNCS 1254, Springer, 1997.

[BMT99] M. Bozga, O. Maler and S. Tripakis, Efficient Verification of Timed Automata using Dense and Discrete Time Semantics, in Proc. CHARME'99 , 125-141, LNCS 1703, Springer, 1999.

[BS94] J.A. Brzozowski and C-J.H. Seger, *Asynchronous Circuits*, Springer, 1994.

[HNSY94] T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine, Symbolic Model-checking for Real-time Systems, *Information and Computation* 111, 193–244, 1994.

[L89] H.R. Lewis, Finite-state Analysis of Asynchronous Circuits with Bounded Temporal Uncertainty, TR15-89, Harvard University, 1989.

[LPY97] K.G. Larsen, P. Pettersson and W. Yi, UPPAAL in a Nutshell, *Software Tools for Technology Transfer* 1/2, 1997.

[MP95] O. Maler and A. Pnueli, Timing Analysis of Asynchronous Circuits using Timed Automata, in *Proc. CHARME'95*, LNCS 987, 189-205, Springer, 1995.

[MY96] O. Maler and S. Yovine, Hardware Timing Verification using KRONOS, In *Proc. 7th Israeli Conference on Computer Systems and Software Engineering*, 1996.

[PCKP00] M.A. Pena, J. Cortadella, A. Kondratyev and E. Pastor, Formal Verification of Safety Properties in Timed Circuits, *Proc. Async'00*, 2-11, IEEE Press, 2000.

[TAKB96] S. Tasiran R. Alur, R.P. Kurshan and R. Brayton, Verifying Abstractions of Timed Systems, in *Proc. CONCUR'96*, 546-562, Springer, 1996.

[TB97] S. Tasiran and R.K. Brayton, STARI: A Case Study in Compositional and Hierarchical Timing Verification, in *Proc. CAV'97*, 191-201, LNCS 1254, Springer, 1997.

[TKB97] S. Tasiran, Y. Kukimoto and R.K. Brayton, Computing Delay with Coupling using Timed Automata, *Proc. TAU'97*, 1997.

[TKY+98] S. Tasiran, S. P. Khatri, S. Yovine, R.K. Brayton and A. Sangiovanni-Vincentelli, A Timed Automaton-Based Method for Accurate Computation of Circuit Delay in the Presence of Cross-Talk, *FMCAD'98*, 1998.

[WD94] H. Wong-Toi and D.L. Dill, Approximations for Verifying Timing Properties, in *Theories and Experiences for Real-Time System Development*, World Scientific Publishing, 1994.

[Y97] S. Yovine, Kronos: A verification tool for real-time systems, *International Journal of Software Tools for Technology Transfer* 1, 1997.

[ZMM03] H. Zheng, E. Mercer and C. Myers, Modular Verification of Timed Circuits using Automatic Abstraction, *IEEE Trans. on CAD*, to appear, 2003.