On Timed Components and their Abstraction

Ramzi Ben Salah
VERIMAG
2, av. de Vignate
38610 Gieres, France
Ramzi.Salah@imag.fr

Marius Bozga
VERIMAG
2, av. de Vignate
38610 Gieres, France
Marius.Bozga@imag.fr

Oded Maler
VERIMAG
2, av. de Vignate
38610 Gieres, France
Oded.Maler@imag.fr

ABSTRACT

We develop a new technique for generating *small-complexity abstractions* of timed automata that provide an approximation of their *timed input-output behavior*. This abstraction is obtained by first augmenting the automaton with additional *input clocks*, computing the "reachable" timed automaton that corresponds to the augmented model and finally "hiding" the internal variables and clocks of the system. As a result we obtain a timed automaton that does not allow any qualitative behavior which is infeasible due to timing constraints, and which maintains a relaxed form of the timing constraints associated with the feasible behaviors. We have implemented this technique and applied it to several examples from different application domains.

1. INTRODUCTION

The basic premise of a component-based design methodology is that a component (a hardware IP block, a software module, a network router) can be used during the construction of a system without deep knowledge of its intimate internal structure but rather using a more abstract (and conservative) description of its observable input-output behavior. This description should be sufficiently detailed to prove the correct interaction of the component with the entire system, and sufficiently small to avoid state explosion. In this work we extend this methodology to *timed systems* models that reflect also *quantitative performance* information. Phenomena such as delays in circuits and communication networks, as well as execution and response times of software are the natural application domains for such models. Using the new abstraction technique presented in the paper, we can *automatically* build a conservative approximation of the *timed input-output behavior* of the component such that any performance guarantees obtained using the abstract model, hold also for the concrete model.

This technique, which has been implemented into a tool, transforms a high-level description of the timed systems[1] into a product of timed automata that captures all the possible behaviors of the system under *all admissible inputs* and *choices of delay parameters*. From this automaton which has one state variable and one clock variablefor every timed element[2] we generate an abstract model with fewer states and clocks which provides an *over-approximation of the time-dependent input-output behavior* of the system. This simplified model can replace the original model within a hierarchical/compositional reasoning methodology. Our technique allows the user to select the appropriate level of aggressiveness in the abstraction, that is, the level of relaxation of the timing and ordering constraints in the abstract model, to achieve a good trade off between the complexity of the model and its faithfulness to the concrete behavior of the system. The major steps in our procedure are:

1. Introduction of additional *input clocks*, each of which measures the time elapsed since the occurrence of a particular input event. When the effect of this event is propagated through the system, its associated clock is deactivated and can be reused by future events.[3] These "dynamic clocks" constitute a novel and non-trivial feature in the theory and practice of timed automata and their number is always bounded, depending on the variability of the input and the structure of the system.
2. Full-fledged reachability analysis of the automaton, resulting in a modified automaton from which all behaviors that violate timing constraints are eliminated.
3. Generation of an abstract model by *hiding* all internal clocks and variables and *projecting* the timing constraints on the input clocks.
4. Minimization of the automaton by merging states which are equivalent (or approximately-equivalent) with respect to observable input-output behavior.

The rest of the paper is organized as follows. Section 2 gives some background on abstraction in general while Section 3 offers a quick survey of timed automata and the computational difficulty inherent in their analysis. Section 4 illustrates our modeling approach and describes the various stages of our abstraction procedure. Preliminary experimental results are described and discussed in Section 5 followed by suggestions for future work.

[1]For circuits this description consists of a network of logical gates with bi-bounded delay elements, for embedded software it consists of descriptions of tasks, resources, durations and scheduling policies.

[2]A timed element is something that measures the time since the occurrence of some event and uses this value to guard a transition.

[3]We restrict ourselves to systems with an *acyclic* structure, systems in which every cycle in the transition graph has at least one transition labelled with an input event. Such systems do not generate "autonomous" cycles and hence every input event generates a "wave" of reactions that propagate through the system within a finite time.

Sixth International Workshop on Specification and Verification of Component-Based Systems (SAVCBS 2007), September 3-4, 2007, Cavtat near Dubrovnik, Croatia.

2. ABSTRACTION IN GENERAL

In verification and other system design activities we have often to deal with a system model S which is too complex to analyze due to its large or even infinite state space. In this case we can try to replace S with a more abstract model S' with the following properties: 1) The complexity of S' is smaller than that of S, where complexity is viewed operationally, that is, S' is easier to analyze than S using some verification tool; 2) Every observable behavior of S is also a behavior of S', but not vice versa (conservative approximation).

Analyzing S' is computationally easier than the verification of S but due to over approximation, it may happen that the verification of S' may fail although S is correct. The navigation in the space of possible abstractions of S in order to find one which is sufficiently simple to avoid explosion yet sufficiently detailed to prove the property in question, is a major research topic, especially for infinite-state systems such as those used to model software. The current paper is concerned with adapting this methodology to *timed systems* defined using the *timed automaton* formalism, but before moving to those, let us contemplate briefly on the nature of abstraction.

A discrete component S, such as a digital circuit or a reactive program, is a device that maintains some relationship between the sequence of inputs it observes and the sequence of outputs it emits. Mathematically speaking, it can be viewed as a *transducer*, an input-output transition system $S = (X, Y, Z, \delta, \gamma)$ that reads inputs ranging over X, makes transitions in its state space Y, according to the transition relation δ, and outputs elements of Z according to the output function γ. If we view S as a "white box" and observe also the sequence of states visited while producing the output (see Figure 1-(a)), we can view S as realizing some sequential function f from X^* to $Y^* \times Z^*$ However, we do not really care about the internal states of S, it is only the input-output function (or relation) from X^* to Z^* which determines whether S interacts correctly with its environment and meets its specification. So the most natural simplification is to hide Y and consider the sequential function $f : X^* \to Z^*$ as the essence of S.

However, contrary to what one may prematurely think, hiding Y and projecting onto the output does not imply that we gain anything in complexity neither lose anything in accuracy. The reason is that every sequential function has its *inherent state space structure* (minimal realization, Myhill-Nerode congruence relation), regardless of whether the states themselves are observable. In other words, hiding internal states from outside observation does not change the state space nor the transition function, which remains of the form $y' = \delta(y, x)$. The only thing it does is to "remove" the states from the output function (see Figure 1-(a)).

Real abstractions, do reduce complexity and lose information by simplifying the transition relation. The most common way to do so is to define an equivalence relation $\sim$ on Y and replace $S = (X, Y, Z, \delta, \gamma)$ with $S' = (X, Y', Z, \delta', \gamma')$ where $Y' = Y/\sim$, the set of partition blocks of $\sim$. In other words we merge together states that are $\sim$-equivalent (see Figure 1-(b)). A transition (y'_1, x, z, y'_2) exists in S' if a transition (y_1, x, z, y_2) exists for *some* $y_1 \in y'_1$ and $y_2 \in y'_2$. Such an abstraction may lose information and generate more behaviors then are really possible in S. For example, the behavior x_1/z_1z_4 is possible in S' while in S we have either x_1/z_1z_3 or x_2/z_2z_4.

Our goal is to export these ideas to timed automata by hiding some clocks variables, but the explanation is more complicated because in timed automata, like in any other automata with auxiliary variables, the visible transition graph does not convey all the information on the system dynamics but rather a projection of it.

3. TIMED MODELS

Timed extensions of discrete transition systems, such as timed automata or Petri nets, allow one to reason about systems in an *extremely important level of abstraction*. At this level, the process of switching between two discrete states is refined into two transitions, *initiation* and *conclusion*, separated by some real-valued *delay*, which is often not known exactly but bounded. Among the numerous phenomena that can benefit from this style of modeling we mention the execution time of a block of code in a real-time program, communication delays in a network, and the time it takes for a digital electronic gate (or a more complex block) to stabilize to a new value after its input has changed. In this paper we demonstrate our approach using models based on networks of Boolean gates due to their notational economy and because it is easy to generate large examples in a uniform way, but the techniques developed can be adapted to other description levels and application domains.

Timed automata model *duration* of actions using auxiliary *clock variables*. To express a timing constraint between two transitions (such as the initiation and termination of a process) a clock is reset to zero while taking the first transition, and its value is tested to satisfy the constraint as a pre-condition ("guard") for the second transition. Between transitions, when the automaton stays in a state, the value of all active clocks progress in the same pace, representing at each moment the time elapsed since the occurrence of their respective events.

At each moment along the real-time axis, the state of the automaton is characterized by a configuration (q, v) with q being a discrete state and v a vector of clock valuations ranging over some bounded subset of $\mathbb{R}^n$. Albeit the infinite state space, the basic verification questions for timed automata are decidable [1]. Existing decision techniques suffer, however, from the usual state-explosion problem, aggravated by the clock-explosion problem: during reachability analysis we need to store "symbolic states" of the form (q, P) where q is a discrete state and P is a *set* of clock valuations. These sets are expressed by a conjunction of constraints of forms like $x < d$ or $x - y < d$, and constitute a special class of convex polyhedra that we call *timed polyhedra*. In a state where n clocks are active, timed polyhedra can be n-dimensional and admit up to two constraints for each pair of variables. Consequently the analysis of a system consisting of n timed components may generate in the worst case $O(2^n \cdot n!)$ symbolic states, each with an $O(n^2)$ representation size. Although a lot of effort has been invested during the last decade in finding more efficient ways to analyze timed automata, scalability toward the size requirements of circuit analysis has not been achieved. In this work we start exploring *compositional reasoning* via *abstraction* as an alternative road toward scaling-up timed automata technology.

4. TIMED ABSTRACTION

Timed automata are quite intuitive but their formal definition can be rather irritating outside formal verification circles. To address potential users of the proposed technology we avoid formalization and illustrate our technique using a running example.

4.1 Modeling

Figure 2 shows a *timed Boolean circuit* and one of its possible behaviors. The circuit has an input signal x which may switch arbitrarily, but with bounded variability, that is, it has to wait at least 5 time units between subsequent switchings. Changes in x are propagated through a *bi-bounded delay element* whose output y follows the value of x within some $t \in [1, 2]$ time units and is fed into a similar delay element with output z. Mathematically speaking the

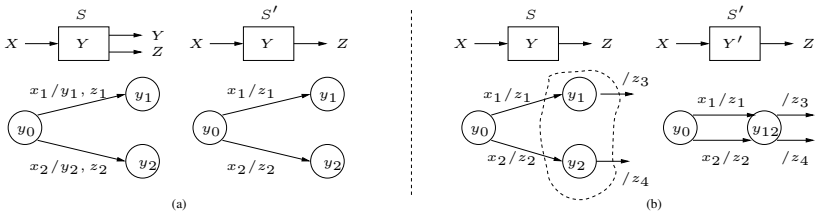

Figure 1: (a) Hiding internal states from the outside does not necessarily reduce complexity; (b) An abstraction S' of S obtained by merging y_1 and y_2 into y_{12}.

relation between signals maintained by the circuit can be expressed by the *delay inclusions* $y \in D_{[1,2]}(x)$ and $z \in D_{[1,2]}(y)$

Following the principles laid out in [7] we model the input and the components using the automata $\mathcal{A}_x$, $\mathcal{A}_y$ and $\mathcal{A}_z$ of Figure 3. We label transitions by *input events, guards, clock resets* and *output events*, for instance, a transition labeled by $x^+, c_y < 2/\{c_z\}, y^-$ can be taken upon the rising of x, provided that $c_y < 2$, and its effect is to reset c_z and lower y. The input automaton $\mathcal{A}_x$ guarantees bounded variability by guarding its transitions with the condition $c_x > 5$ and by resetting clock c_x to zero at every transition.

The modeling of *delay elements* by *timed automata* is a crucial ingredient of our methodology. The automaton $\mathcal{A}_y$ starts at a stable state 0 where its value coincides with the value of its input x. Upon a change in x it moves to an *excited state* $0'$ while resetting its clock c_y. The "stabilize" transition from $0'$ to 1 through which y "catches up" with x, may happen when $c_y \in [1, 2]$, that is, inside the time window $[t + 1, t + 2]$ with t being the time when x has changed.[4] Note that the "excite" transition from 0 to $0'$ is always triggered by an external input but is not visible from the outside, while the stabilization transition from $0'$ to 1 is generated autonomously without an input event (unless one considers the passage of time as such) and is visible to the outside world. Composing the three automata we get the global automaton $\mathcal{A}$ of Figure 4. Note also that each clock is *active* only in global states in which its corresponding gate is excited.

There are different approaches for treating the case where x changes its value again *before propagating to* y. For the purpose of this work, we assume that the automaton returns from $0'$ to 0 (a "regret" transition) and thus it "forgets" the whole episode. Other approaches may treat this phenomenon as an error ("glitch"), or model it in a manner more faithful to the physical realization of logical gates. Either way, this guarantees that the number of events that may be "alive" in the systems is bounded, regardless of the input frequency. In other domains this effect can be achieved by admission controllers or bounded buffers.

The *semantics* of this automaton consists of all xyz signals it can generate, that is, the signals carried by all runs of the automaton. These runs are sequences of configurations separated by transitions or by time-passage periods. The behavior where x rises at 6, y follows after 1 time unit and z follows 1.9 time units after y, is captured by the following run where configurations are presented as tuples of the form

$$\begin{pmatrix} x, c_x \\ y, c_y \\ z, c_z \end{pmatrix}$$

where $\perp$ denotes inactive clocks:

$$\begin{pmatrix} 0', 0 \\ 0, \perp \\ 0, \perp \end{pmatrix} \xrightarrow{6} \begin{pmatrix} 0', 6 \\ 0, \perp \\ 0, \perp \end{pmatrix} \xrightarrow{x^+} \begin{pmatrix} 1', 0 \\ 0', 0 \\ 0, \perp \end{pmatrix} \xrightarrow{1} \begin{pmatrix} 1', 1 \\ 0', 1 \\ 0, \perp \end{pmatrix}$$

$$\xrightarrow{y^+} \begin{pmatrix} 1', 1 \\ 1, \perp \\ 0', 0 \end{pmatrix} \xrightarrow{1.9} \begin{pmatrix} 1', 2.9 \\ 1, \perp \\ 0', 1.9 \end{pmatrix} \xrightarrow{z^+} \begin{pmatrix} 1', 2.9 \\ 1, \perp \\ 1, \perp \end{pmatrix}$$

The circuit behavior carried by this run can be represented either in a state-based manner (as a signal) or in an event-based manner (as a time-event sequence, see [2]) as follows:

$$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}^6 \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}^1 \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}^{1.9} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \quad 6 \cdot x^+ \cdot 1 \cdot y^+ \cdot 1.9 \cdot z^+$$

We use the term *qualitative behavior* to denote the sequence of signal values without reference to timing. For this example the qualitative behavior is $x^+y^+z^+$ and can be viewed as an equivalence class of all signals of the form $t_1 \cdot x^+ \cdot t_2 \cdot y^+ \cdot t_3 \cdot z^+ \cdot t_4$ for any $t_1, t_2, t_3, t_4 \geq 0$.

If we ignore timing constraints, remove all references to clocks from transition guards and leave only the rising and falling labels, we obtain a timed automaton which is practically equivalent to an untimed automaton. This can be viewed as a very aggressive form of abstraction whose set of qualitative behaviors is the set of all sequences of labels carried by *all paths* in the transition graph, for example $x^+y^+z^+x^-y^-z^-$ or $x^+y^+x^-y^-$. However, taking timing into account one can see that given the variability constraint on x, the second behavior is impossible because state $110'$ is never reached with a combination of clock values that satisfies the guard $c_x > 5 \wedge c_z < 2$.

4.2 Reachability Analysis

The analysis of the timed automaton itself, rather than its untimed abstraction, is typically performed by constructing the *reachability graph*, also known as the *simulation graph* [6], which gives a (somewhat non-intuitive) representation of that part of the timed automaton which is reachable from some initial state or set of states. To illustrate the idea let us compute the reachability graph for $\mathcal{A}$ starting from an initial configuration $(0'00, c_x = 0)$. In this state we can let time progress indefinitely and can reach all clock valuations satisfying $c_x \geq 0$. This is represented as a "symbolic state" $(0'00, c_x \geq 0)$. The next step is to intersect this set with the transition guard $c_x > 5$ to obtain all the configurations from which the transition labeled by x^+ can be taken, represented by the symbolic state $(000, c_x \geq 5)$. Finally, by applying to this set the resetting of c_x and c_y, we obtain the symbolic state $(10'0, c_x = c_y = 0)$.

The process is then repeated from the new symbolic state where time passage is limited by 2 which is the upper bound on the rising

[4]The fact that the automaton *must* leave state $0'$ when c_x reaches 2 can be expressed either using staying conditions ("invariants") associated with states, or "deadlines" and "urgencies" associated with transitions, [9]. Using the latter terminology, stabilization transitions are *delayable*.

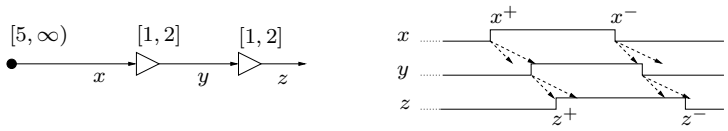

Figure 2: A simple timed circuit and a typical behavior.

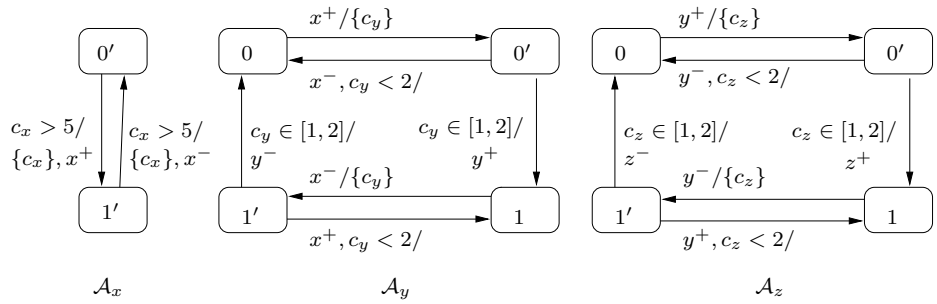

Figure 3: Modeling the circuit of Figure 2 with timed automata.

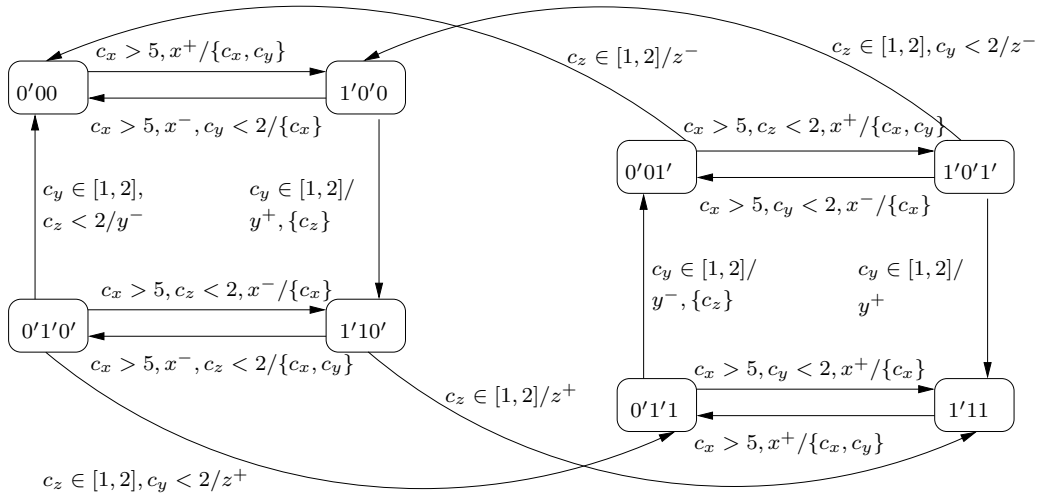

Figure 4: The global automaton $\mathcal{A} = \mathcal{A}_x \circ \mathcal{A}_x \circ \mathcal{A}_z$ for the circuit.

of y, hence the symbolic state is $(10'0, c_x = c_y \leq 2)$. The transition back to $0'00$ cannot be taken due to empty intersection with the guard $c_x > 5$ and this transition is eliminated. The intersection with the guard $c_y \in [1, 2]$ gives $(10'0, 1 \leq c_x = c_y \leq 2)$ and the result of the transition after resetting c_z is $(1'10', c_x \leq 2 \wedge c_z = 0)$. In this state time can progress until $c_z = 2$ resulting in the symbolic state $(1'10', 1 \leq c_x - c_z \leq 2 \wedge c_z \leq 2)$ and so on and so forth until we obtain the reachability graph of Figure 5. The procedure is guaranteed to terminate due to the finite number of bounded timed polyhedra [1, 6].

We interpret the reachability graph as a timed automaton $\mathcal{A}'$ as follows: for each symbolic state (q, P) we define a copy of state q whose staying condition (and its outgoing transition guards) are restricted to their intersections with P. Transitions whose guards become empty in the process, as well as states that become unreachable, are removed. On the other hand it may happen that the reachability graph contains two or more symbolic states (q, P) and (q, P') that correspond to alternative paths to q, and hence the state will be split in the resulting timed automaton. For example state $0'00$ as an initial state can have all clock valuations with $c_x \geq 0$, but when reached again through the path $x^+y^+z^+x^-y^-z^-$, the value of c_x must always exceed 2. Such state splitting will occur very often in systems such as circuits where there are many "diamonds", that is, two competing events e_1 and e_2 that may happen in both $e_1 \prec e_2$ and $e_2 \prec e_1$ orders and converge to the same state q. If these events reset clocks c_1 and c_2, respectively, the reachability graph will contain two symbolic states, $(q, c_1 \leq c_2)$ and $(q, c_2 \leq c_1)$.

It is not hard to see that the new timed automaton $\mathcal{A}'$ admits exactly the same set of behaviors as the original automaton $\mathcal{A}$, together with an additional evident property that any configurations that satisfies the staying condition of a state is indeed reachable. Every finite or infinite path (a qualitative behavior) in the transition graph of $\mathcal{A}'$ is an untimed abstraction of a *feasible behavior* of $\mathcal{A}$, a behavior that satisfies the timing constraints. If our goal is to verify some untimed property of the system, we can remove the clocks from $\mathcal{A}'$ (after having used them to eliminate infeasible paths) and apply standard untimed verification algorithms. However if we want to compose the system with other components it might not be a good idea to get rid of *all* timing information. The untimed abstraction does not constrain in any way the time between x^+, y^+ and z^+, which can be arbitrarily small or large, and will make it difficult (if not impossible) to prove the correctness of the interaction of the circuit with its environment. An abstraction which maintains *some* of the timing constraints but which has less states and clocks, would be very useful in this context.

4.3 Abstraction by Clock Projection

We want the abstract model to approximate the *timed input-output relationship* maintained by the system. Clock c_x measures the time since the last change in the external input x while clocks c_y and c_z measure time elapsed since the occurrence of *internal events*, the excitation of the two gates, events that are of no interest to the general public. We can thus "hide" these clocks and project the guards and staying conditions on clock c_x to obtain the automaton $\mathcal{A}''$ of Figure 6. Note that a projection of a polyhedron P into a lower dimensional polyhedron P' makes some of the constraints which are implicit (redundant) in P, explicit in P'. For example the polyhedron defined by $1 \leq c_z \leq 2 \wedge 1 \leq c_x - c_z \leq 2$ is projected onto $2 \leq c_x \leq 4$.

When y is not observable outside the system, the set of all xz behaviors of $\mathcal{A}''$ is exactly that of $\mathcal{A}$ and $\mathcal{A}'$ and no information is lost. Unfortunately, in the general case, the projection of clocks does lose information. Consider the same circuit but with y visible to the external world. In this case $\mathcal{A}''$ is an over approximation because it allows a behavior like $5 \cdot x^+ \cdot 1 \cdot y^+ \cdot 3 \cdot z^+$, where y "chooses" to change in the earliest time $t \in [1, 2]$ after x while z is allowed to chooses the largest element in $[2, 4] = [1 + 1, 2 + 2]$ while in $\mathcal{A}$ and $\mathcal{A}'$ its choices were restricted to the interval $[t + 1, t + 2]$. This is the type of accuracy we are ready to sacrifice for the purpose of complexity reduction.

The outcome of our abstraction technique is a timed automaton over the inputs and outputs of the system, where output transition guards involve clocks that measure the time elapsed since the occurrence of input events. In the previous example we used input clock c_x which was reset at *every* change in x. This construction was correct because the *variability constraint* prevented the arrival of an x-event while the circuit is still busy "digesting" the previous event. When this constraint is relaxed, an x-labeled transition may be taken in a state where one or more gates excited by the previous x-transition have not yet stabilized. In our example, if we change the variability constraint from $c_x \geq 5$ to $c_x \geq 3$, x may change at state $1'10'$ where y has already stabilized but z is still excited by the previous change. If we reset c_x we lose the time of that previous event, and when we project transitions guards on c_x we *do not* express the temporal distance between the rising of z and its *triggering event*.[5]

To guarantee correct abstraction each input event should reset *its proper clock* which will stay active as long as the "wave" of reactions it triggered has not propagated through the system. Within our modeling methodology, the number of input events that may be active simultaneously in an acyclic system is bounded and hence a finite number of clocks will suffice to retain the information necessary for relating the timing of input and output events. To implement these input clocks we modify the timed automaton model to include a pool of *dynamic clocks* which are activated by input events and killed when the effect of these events propagates to the output. The attachment of these clocks to input events is not fixed and the same clock can, for example, denote at some point the time elapsed since the oldest x_1 event still in the system, and at some other point, the time since the most recent x_2 event. Technically speaking, we replace the input generator by one which creates a new clock at every transition, and keeps track of the input events that are still alive in the system and the clocks that represent them. It is worth mentioning that such dynamic clocks are useful in other, more theoretical, contexts [8].

4.4 Minimization

By hiding internal clocks we obtain an abstract model whose number of clocks need not be equal to the number of timed elements but rather depends on the maximal number of input events that may be "alive" simultaneously in the system. The number of such events depends, of course, on the size of the system as well as on other properties such as the number of inputs, their variability as well as structural properties such as width vs. depth (sequentiality vs. parallelism). Under reasonable assumptions concerning these parameters, the reduction in the number of clocks is significant.

The final step in our procedure aims at reducing the number of *discrete states* by merging states that are equivalent or approximately equivalent in terms of the observable behaviors they admit. Candidates for merging are states that differ from each other only by values of internal variables and of clocks, for example states such as $1'0'0$ and $1'10'$ in automaton $\mathcal{A}''$ of Figure 6, after hiding

[5] In our previous work [3] we have applied this abstraction technique to systems whose inputs changes *only once* at time zero, so that one additional clock was sufficient to project on.

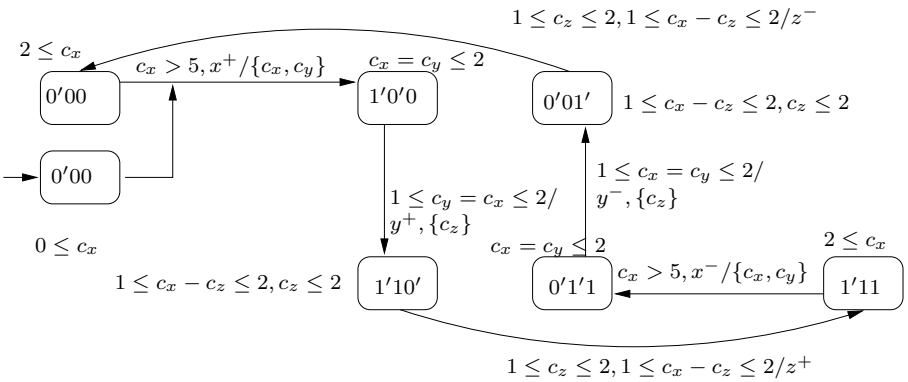

Figure 5: The reachability graph of the automaton in Figure 4, interpreted as a timed automaton $\mathcal{A}'$.

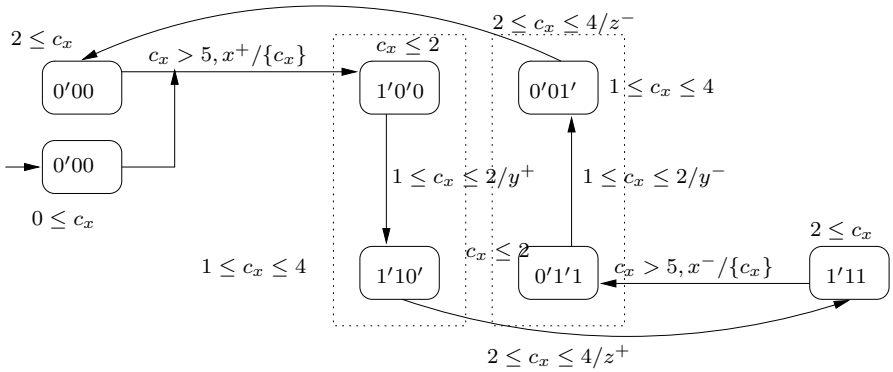

Figure 6: The automaton $\mathcal{A}''$ obtained from $\mathcal{A}'$ by hiding clocks c_y and c_z. The dotted boxes group together states that differ only by the value if the internal variable y.

y. A commonly-used minimization rule (also for untimed systems) is the following.

Let q be a source state for several paths, each consisting of a sequence of unobservable transitions, except the last transition which changes one observable variable and goes to state q'. In this case q and all the intermediate states can be collapsed into one state whose staying condition is the union of those of all states, and which has a transition to q' guarded by the union of all transition guards to q' from the intermediate states. Applying this rule we obtain the automaton of Figure 7 which is nothing but a demonstration of the following equivalence on delay operators: $D_{[1,2]}(D_{[1,2]}(x)) = D_{[2,4]}(x)$. A similar transformation was presented in [11] for timed Petri nets.

The situation gets more complicated when the system admits more parallelism and input events may appear more frequently. We have developed a variety of minimization algorithms that are similar in spirit to those described in [5]. We employ a variety of progressively more "liberal" criteria that merge states which: 1) Admit exactly the same sequences of observable transitions and guards; 2) Differ in guards but the guards are included in each other; 3) Differ in guards and the guard of the new state is the *convex hull* of the guards of the original states; 4) Differ in the order of some sequence of events that admit.

We have implemented all the abovementioned features into a new experimental version of the verification tool IF [4], including an automatic translation from a circuit description language to timed automata, generation and maintenance of dynamic clocks, projection and minimization. The software implementing this technique consists of more than 15000 lines of C++ code.

5. EXPERIMENTAL RESULTS

To assess our approach we applied it first on some classes of synthetic circuits, the first of which is a family of k-long buffers like the one described in the example, with delays in $[3, 5]$. We performed the experiments with two versions of the buffer, one where only the output of gate k is observable, and the other where the output of gate $k/2$ is visible as well. Table 1 shows the results of applying our technique while assuming input variability bounded by 40. Column w shows the maximal number of input events that may be alive in the buffer, which ranges from 1 to 3 depending on the circuit depth. The first pair of columns shows the number of symbolic states and transition in the computed reachability graph. The rest of the table shows the size of the reduced graph using three minimization criteria: *Hidemin* indicates merging only in the case of identical guards, *TimedMin* merges states when guards are included in each other while *Temporal Min* ignores guards and considers equivalence with respect to transition labels.

The other class of examples is inspired by recent research on performance analysis of embedded software, e.g. [10]. We consider systems that generate different types of tasks with some bounded frequency. Each type of task has to go through a partially-ordered set of treatments. Each type of treatment requires a specific resource (machine) for some duration with the possibility of resource conflicts between tasks. These conflicts are resolved by a scheduler applying a simple policy. Each task type has a dedicated bounded buffer. We have applied our technique to an instance of this problem with 2 task types, 3 machines, a priority-based scheduler and parameters that allow 3 events to be alive simultaneously in the system. An unoptimized version of IF generates a reachability graph with 1282 states and 1975 transitions. The version of IF that we use, with dynamic clocks and various optimization that we do not bother to detail, yields a graph with 127 states and 205 transition. After minimization with zone inclusion we obtain the automaton of Figure 8 with 18 states and 33 transitions. Transitions in the reduced model correspond to arrivals of new tasks and their termination.

6. DISCUSSION

We have developed a new promising technique for automatic generation of abstractions for open timed systems. Timed automata with dynamic input clocks may turn out to be the appropriate formalism for characterizing the timed input-output behaviors of complex systems, whose approximation by nice analytical expressions is too coarse. Our technique can also be part of a divide-and-conquer methodology where abstract models of sub-systems are composed together in order to verify a system too large to be analyzed as a whole. Much more experimentation and fine tuning are needed, however, in order to assess the applicability of our approach.

Our original ambitious aim was to provide a "fully-open" abstraction without assuming *any restriction* on the inputs, and letting this restrictions come from each particular environment with which the abstract model is to be composed. However, we have learned in the process that unrestricted inputs generate too many simultaneous waves that lead to explosion. One should be careful, though, not to confuse what is assumed and what should be guaranteed.

7. REFERENCES

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

[2] E. Asarin, P. Caspi and O. Maler, Timed Regular Expressions, *The Journal of the ACM* 49, 172-206, 2002.

[3] R. Ben Salah, M. Bozga and O. Maler, On Timing Analysis of Combinational Circuits, *FORMATS'03*, 204-219, LNCS 2003.

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

[5] C. Daws and S. Tripakis, Model Checking of Real-Time Reachability Properties using Abstractions, *TACAS'98*, LNCS 1384, 1998.

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

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

[8] O. Maler, D. Nickovic and A. Pnueli, Real Time Temporal Logic: Past, Present, Future, *FORMATS'05*, 2-16, LNCS 3829, Springer, 2005.

[9] J. Sifakis and S. Yovine, Compositional Specification of Timed Systems, *STACS'96*, 347-359, LNCS 1046, Springer, 1996.

[10] E. Wandeler, A. Maxiaguine, L. Thiele: Quantitative Characterization of Event Streams in Analysis of Hard Real-Time Applications, *Real-Time Systems* 29, 205-225, 2005.

[11] H. Zheng, E. Mercer, and C.J. Myers, Modular verification of timed circuits using automatic abstraction, *IEEE Trans. on CAD* 22, 2003.