Automatic Abstraction of Timed Components

Ramzi Ben Salah Marius Bozga Oded Maler

VERIMAG, 2, av. de Vignate, 38610 Gieres, France
Ramzi.Salah@imag.fr Marius.Bozga@imag.fr 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, projecting the timing constraints on the input clocks and finally hiding internal transitions and minimizing the automaton. As a result we obtain a timed automaton in which output transitions are conditioned on the time elapsed since the relevant input transitions that triggered them. The abstract model does not allow any qualitative behavior which is infeasible due to timing constraints, and maintains a relaxed form of the timing constraints associated with the feasible behaviors. We have implemented most of the ingredients of this technique and intend to apply it to 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 in the construction of a system without deep knowledge of its intimate internal structure but rather using a more abstract description of its observable input-output behavior. This description should be sufficiently detailed to prove the correct interaction of the component with the rest of the world, and sufficiently small to keep the analysis tractable. In the context of discrete systems this means replacing an automaton $\mathcal{A}$ with and automaton $\mathcal{A}'$ with less states and more behaviors.

In this work we apply this methodology to a more refined level of description, namely that of *timed systems* where models represent not only the ordering relation between input and output events, but also constraints on the *temporal distance* between them. We view a timed component as a reactive device that responds to input events within a certain amount of time by emitting output events. We are interested in the analysis of large networks constructed from such components. In such networks many processes may be active simultaneously and the qualitative behavior of the overall system often depends on the relative speeds of the processes and is very hard to analyze as the number of components grows. Examples of timed components include digital circuits (when propagation delays are taken into account), communication channels, software modules (when execution times are considered) and, in fact, any other system consisting of processes that consume some time between initiation and termination. Each timed component is modeled as a timed automaton, an automaton augmented

clock variables whose role is to model and impose the timing constraints associated with the input-output behavior of the component.

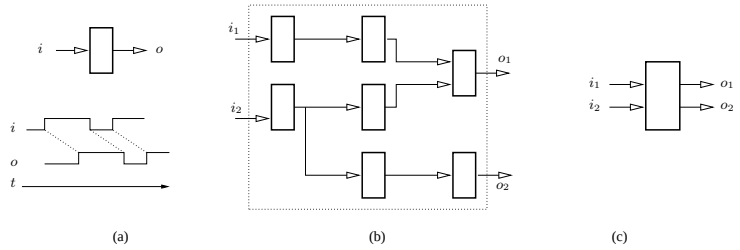


Fig. 1. (a) A timed component and part of its semantics; (b) A network of timed components; (c) An abstraction produced by our technique.

Figure 1 summarizes the contribution of the paper. At the left we see a timed component with an input channel i and output channel o and an illustration of its semantics as a *timed transducer* that maintains some relationship between input and output events (or state changes). We assume uncertainty in timing information so that the transducer is non deterministic and may produce several outputs to a given input. In the middle of the figure we see a *compound component* constructed as an acyclic network of such elements with primary inputs i_1, i_2 and primary outputs o_1, o_2. Our technique analyzes the network and produces a reduced model which gives a conservative approximation of the relation between the timing of input and output events, in the sense that every input-output pair that may be exhibited by the network will be produced as well by the abstract model. Thus, any correctness result or performance guarantee obtained using the abstract model will hold for the concrete model as well.

The major steps in our abstraction procedure, starting from a timed automaton $\mathcal{A}$ given as a product of the timed components, are the following:

1. We transform $\mathcal{A}$ into an equivalent timed automaton $\mathcal{A}^t$, which is $\mathcal{A}$ augmented with additional *input clocks*. Each input clock is created when an input event occurs and is killed when the effect of this event has propagated through the system.[1] The input clocks do not intervene with the dynamics of the automaton as they do not participate in transition guards and are not reset when transitions are taken. Hence the input/output behaviors of $\mathcal{A}$ and $\mathcal{A}^t$ are identical. These event clocks serve to represent the time elapsed since the occurrence of the event.
2. We perform standard timed reachability computation on $\mathcal{A}^t$ which leads to an automaton $\mathcal{A}^r$, based on the simulation/reachability graph, which is *semantically equivalent* to $\mathcal{A}^t$, but in which every path in the underlying transition graph is indeed realizable. Transitions and states of $\mathcal{A}^r$ are conditioned by conjunctions of

[1] We will see later why in the systems that we consider, every event is propagated (or aborted) within a finite amount of time and a finite number of clocks will suffice.

inequalities (zones) on all the clocks of $\mathcal{A}^t$, but the constraints involving the input clocks are practically redundant in these zones.

3. We *project* the timing constraints in $\mathcal{A}^r$ on the input clocks and get rid of all clocks associated with the internal components (after they served us in eliminating impossible behaviors). In the resulting automaton $\mathcal{A}^p$, transitions and states are conditioned by the time elapsed since the occurrence of input events. This projection relaxes some of the timing constraints and leads to more behaviors than possible in $\mathcal{A}^r$, but the set of qualitative behaviors (sequences of transitions) is preserved.
4. We apply a minimization algorithm which merges states in $\mathcal{A}^p$ that are linked by unobservable transitions to obtain the final automaton $\mathcal{A}^m$ which is semantically-equivalent to $\mathcal{A}^p$.[2]

Semantically speaking, we have the following relation between the models:

$$[\![\mathcal{A}]\!] = [\![\mathcal{A}^t]\!] = [\![\mathcal{A}^r]\!] \subseteq [\![\mathcal{A}^p]\!] = [\![\mathcal{A}^m]\!].$$

In terms of complexity, $\mathcal{A}$ has one clock per timed component and an untimed state space exponential in the number of components, while the number of clocks in $\mathcal{A}^m$ is the maximal number of events that may be alive simultaneously, and its state space is much smaller.

We have developed a full tool chain starting from a high-level description of networks of timed components all the way down to the final abstract model and tested its performance on several examples. We believe this technique will have a significant impact on system design.

The rest of the paper is organized as follows. In Section 2 we give some intuition about timed systems and components followed by the unavoidable formalization of the basic notion used in the paper and in the analysis of timed automata. We then move to a step-by-step description of our abstraction technique: adding input clocks (Section 3), timed reachability analysis (Section 4), clock projection (Section 5) and minimization (Section 6).Finally we describe the accompanying implementation effort and report preliminary experimental results.

2 Timed Systems

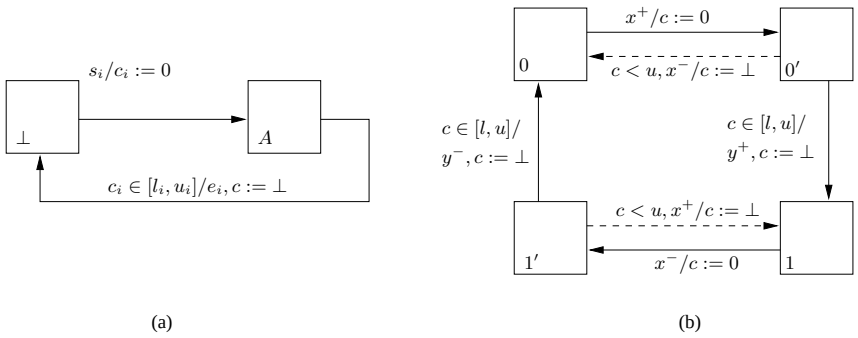
2.1 Intuition

The theory of timed behaviors and timed automata can be formulated using either of two semantic domains, one is event-based (timed-event sequences, punctual discrete events scattered along the real time axis) and one is state-based (signals, functions from the positive reals to a discrete domain). To avoid repetition, we will use signals in the formal definitions but will use both domains for motivating examples and benchmarks. We describe below a class of event-based and a class of state-based timed components that can be substituted into the blocks of Figure 1-(b).

As a first class of examples consider a system consisting of jobs, each job decomposed into a partially-ordered set of tasks $\{T_1, \ldots, T_n\}$ where $T_i \prec T_j$ indicates that

[2] Not yet sure. In the worst case it is an over approximation.

T_i must terminate before T_j starts executing. The computation time of task T_i can vary inside the interval $[l_i, u_i]$. Each task T_i can be modeled as the simple timed automaton depicted in Figure 2-(a). The automaton remains in an idle state $\perp$ until it receives an s_i message, resets its clock c_i to zero and moves to the active state A in which it stays for some $t \in [l_i, u_i]$ until it terminates and returns to state $\perp$, while emitting the e_i message. Networks of such automata can be composed by unifying the output of a task with the inputs of the tasks that it precedes. Additional automata can be introduced to model schedulers, buffers, admission controllers, and more to yield complex systems exhibiting pipelining and concurrent processing. For such systems, the reduced model generated by our technique provides an approximation of the input-output behavior restricted to the arrival and termination of jobs. Such models can be used to approximate the system throughput, find conditions on the inter-arrival times of jobs that cause buffer overflow, and can be plugged as a single component into a model of a larger network.



(a) (b)

Fig. 2. (a) A timed automaton for a task; (b) a timed automaton for a circuit delay element. The $\perp$ symbol indicates that the clock is inactive as its corresponding state variable is stable.

The second class of models is that of digital circuits with delays. We decompose every gate which realizes a Boolean function $x = f(x_1, x_2)$ into two parts, an *instantaneous* part whose output signal x is always the value of f applied to the current values of x_1 and x_2, and a *delay element* which takes x as input and outputs a signal y whose value follows x within $t \in [l, u]$ time. The automaton for modeling this delay element is similar to the task automaton, but has two "stable" states, 0 and 1 where the input and output agree and two "unstable" states $0'$ and $1'$ where the input has changed but the process of propagation to the output has not yet completed, as can be seen if Figure 2-(b). When in a stable state like 0, the automaton is inactive, until the value of the input changes (denoted in the figure by x^+) and the automaton moves to the excited state $0'$ in which the output is still 0 but the clock c is reset. Then, after some $t \in [l, u]$ time, the output catches up with the input and the automaton moves to the stable state 1 and so on.

These two automata have similar features. Both have stable states that cannot be exited from without an input event (s in the task automaton) or a change in the input

value (x^+ to leave 0 and x^- to leave 1 in the delay automaton). On the other hand the duration of their sojourn in the active/excited states is bounded by u. More generally they have the property that all their cycles are labeled by at least one input event. This means that for every input which becomes stable at some point, all the behaviors it induces in the automaton become stable as well within some bounded delay. This is an important property for this class of systems and we leave it as an exercise to the reader to prove that it is preserved by (acyclic, loop-free) composition.

An important issue that we have ignored in the task model but have taken care of in the gate delay model is the following: what happens when a new event arrives while the automaton is in an active/excited state, still digesting the previous event? The transition indicated by dashed arrows in the delay automaton provide one possible modeling solution (which is not electrically correct): if x goes down again before its rising has propagated to the output, the automaton returns to the stable state 0 and forgets the whole episode. Another solution would be make an error transition when this happens. In the task automaton, we can make an error transition when a new s arrives at state A, or simply ignore it. Whatever the solution taken, it is clear that no system can digest an unbounded number of events in finite time, and this topic is closely related to various bounded-variability assumptions concerning the input. In the rest of the paper we use the circuit delay model with "regret" transition, but the whole methodology can be easily adapted to event-based semantics and to other approaches for treating inputs that exhibit excessive variability.

Before proceeding let us mention a special property of the timed automata that we use which is preserved by our transformations. These automata are non-blocking in the sense that there is always a guard satisfied at the right boundary of an invariant.

2.2 Basic Definitions

We now move to the formalizing of our abstraction technique. As a semantic domain we use Boolean signals defined as follows.

Definition 1 (Signals). *Let $\mathbb{R}_+$ be the set of non-negative real numbers. An n-dimensional Boolean signal is a function $\xi : \mathbb{R}_+ \to \mathbb{B}^n$ such that for every $\sigma \in \mathbb{B}^n$, $\xi^{-1}(\sigma)$ is a countable union of* left-closed right-open *intervals.*

We use the notation $\xi = \sigma_1^{t_1} \cdot \sigma_2^{t_2} \cdots$ with $\sigma_i \neq \sigma_{i+1}$ to denote a signal whose value is σ_1 in $[0, t_1)$, σ_2 in the interval $[t_1, t_1 + t_2)$ etc. The *untiming* of the signal, $\mu(\xi) = \sigma_1 \cdot \sigma_2 \cdots$ is the *sequence* of values the signal goes through without the duration information. In the context of automata such an untiming will be called a *qualitative behavior*. A partial signal is a restriction of a signal to some interval $[a, b)$, in such a case we denote the duration of the signal by $|\xi| = b - a$. We will typically use sets of Boolean variables whose valuations constitute the elements of the alphabet. We denote the sets of signals and partial signals over these valuations $S^\omega(X)$ and $S^*(X)$, respectively.

In the same manner that automata realize sequential functions (transductions) on sequences, a timed automaton realizes a (possibly non deterministic) length-preserving sequential function from signals over its input variables to signals over its output variables.

Definition 2 (Timed Transduction). *A function* $f : S^\omega(X) \to 2^{S^\omega(Z)}$ *is a timed transduction if for every* $u \in S^*(X)$, $v, v' \in S^*(Z)$ *such that* $|u| = |v| = |v'|$ *and every* $\alpha \in S^\omega(X)$ *and* $\beta \in S^\omega(Z)$

$$f(u \cdot \alpha) = v \cdot \beta \text{ and } f(u \cdot \alpha') = v' \cdot \beta' \text{ implies } v = v'.$$

In other words, the value of $f(\alpha)$ at t may depend only on the values $\{\alpha[t'] : t' \leq t\}$. Composition of two transductions can be defined naturally where the output of the first becomes the input of the second. A signal is *ultimately-constant* if it is constant from some t onwards. A transduction is *stable* if it maps every ultimately-constant signal to a set of ultimately-constant signals.

Definition 3 (Abstraction). *Let* f *and* f' *be two signal transductions over the same domains. We say that* f' *is an abstraction of* f *if* $f(\xi) \subseteq f'(\xi)$ *for every signal* ξ*. We denote this fact by* $f \preceq f'$*.*

As the mechanical device for realizing timed transduction we use a variant of timed automata with input and output which differs slightly from the classical definitions [AD94,HNSY94] as it reads multi-dimensional *dense-time* Boolean signals, and outputs Boolean signals. Hence the input and output alphabet letters are associated with *states* rather than with *transitions*. We also extend the domain of clock values to include the special symbol $\perp$ indicating that the clock is currently *inactive*.

The set of valuations of a set $\mathcal{C} = \{c_1, \ldots, c_n\}$ of clock variables, each denoted as $v = (v_1, \ldots, v_n)$, defines the clock space $\mathcal{H} = (\mathbb{R}_+ \cup \{\perp\})^n$. A *configuration* of a timed automaton is a pair (q, v) consisting of a discrete state and a clock valuation. For a valuation $v = (v_1, \ldots, v_n)$, $v + t$ is the valuation $(v'_1, \ldots, v'_n)$ such that $v'_i = v_i$ if $v_i = \perp$ and $v'_i = v_i + t$ otherwise. A *clock inequality* is any conditions of the form $\theta \leq d$, $\theta < d$, $\theta \geq d$ or $\theta > d$ for some integer d and θ which is either a clock variable c or a difference $c - c'$ between two clock variables. A *clock condition* is a Boolean combination of clock constraints. The set of all valuations satisfying a clock condition is called a *timed polyhedron*. A timed polyhedron is *convex* if it can be expressed using a *conjunction* of clock conditions and in this case it is called a *zone*. A timed polyhedron Z is *time-convex* if it satisfies the following weaker property: if $v \in Z$ and $v + t \in Z$ for some $t > 0$ then $v + t \in Z$ for every $t' \in [0, t)$. These notions are illustrated in Figure 3.

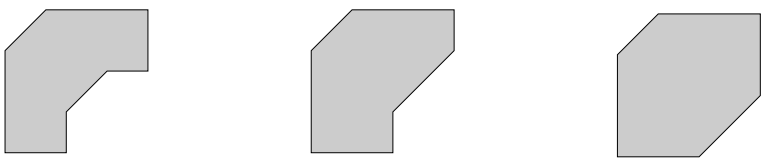


Fig. 3. (a) A timed polyhedron; (b) time-convex timed polyhedron; (c) a convex polyhedron (zone).

Definition 4 (Timed Signal Transducer).
A timed signal transducer is a tuple $\mathcal{A} = (\Sigma, Q, \Gamma, \mathcal{C}, \lambda, \gamma, q_0, I, \Delta)$ where Σ is the input alphabet, ($\mathbb{B}^n$ in this paper), Q is a finite set of discrete states, Γ is the output alphabet and $\mathcal{C}$ is a set of clock variables. The input labeling function $\lambda : Q \to \Sigma$ associates an input letter to every state while the output function $\gamma : Q \to \Gamma$ assigns output letters. The initial state is q_0 and the staying condition (invariant) I assigns to every state q a time-convex subset I_q of $\mathcal{H}$. The transition relation Δ consists of elements of the form (q, g, ρ, q') where q and q' are discrete states, the transition guard g is a zone in $\mathcal{H}$ and ρ is the update function, a transformation of $\mathcal{H}$ defined by a set of assignments of the form $c := 0$, $c := c'$ or $c := \perp$.

Although input-output labels are associated with states, they induce an implicit labeling on the transition and we will consider a transition from q to q' such that $\lambda(q) \neq \lambda(q')$ as labelled by the changes in the input variables whose values in $\lambda(q')$ that are different from their values in $\lambda(q)$, for example $x_1\uparrow$ or $x_2\downarrow$. We use the same convention for output events. The underlying *transition graph* of $\mathcal{A}$ is the result of removing the clocks, the invariants, the transition guards and the clock updates to obtain an ordinary automaton.

The behavior of a timed transducer as it reads a signal ξ consists of an alternation between *time progress* periods where the automaton stays in a state q as long as $\xi[t] = \lambda(q)$ and I_q holds, and *discrete instantaneous transitions* guarded by clock conditions. Formally, a *step* of the automaton is one of the following:

- A time step: $(q, v) \overset{\sigma^t/\tau^t}{\longrightarrow} (q, v + t)$, $t \in \mathbb{R}_+$ such that $\sigma = \lambda(q)$, $\tau = \gamma(q)$, and[3] $v + t \in cl(I_q)$. We will write time steps as $(q, v) \overset{t}{\longrightarrow} (q, v + t)$ when we do not care about the labeling.
- A discrete step: $(q, v) \overset{\delta}{\longrightarrow} (q', v')$, for some transition $\delta = (q, g, \rho, q') \in \Delta$, such that $v \in g$ and $v' = \rho(v)$

A *run* of the automaton starting from a configuration (q_0, v_0) is a finite or infinite sequence of alternating time steps and discrete steps of the form

$$\xi : \quad (q_0, v_0) \overset{\sigma_1^{t_1}/\tau_1^{t_1}}{\longrightarrow} (q_0, v_0 + t_1) \overset{\delta_1}{\longrightarrow} (q_1, v_1) \overset{\sigma_2^{t_2}/\tau_2^{t_2}}{\longrightarrow} (q_1, v_1 + t_2) \overset{\delta_2}{\longrightarrow} \cdots,$$

such that $\sum t_i$ diverges. The input signal carried by the run is $\sigma_1^{t_1} \cdot \sigma_2^{t_2} \cdots$ and the output is $\tau_1^{t_1} \cdot \tau_2^{t_2} \cdots$. The automaton realizes a timed transduction $f_{\mathcal{A}}$ over its input and output alphabets defined by $\xi' \in f_{\mathcal{A}}(\xi)$ iff one of the runs induced by the input signal ξ outputs the signal ξ'.

One can define parallel and cascade *composition* of transducers where in the former the same input is read by two automata in parallel while in the latter the output of the first automaton is the input of the second. We will avoid the tedious but not surprising formal definitions. Clearly the effect of cascade composition of two transducers amounts to the composition of their transductions.

We say that a state q of a transducer is *stable* if all the clock are inactive in this state and the only outgoing transitions are due to change in the input, that is, if there is

[3] We use cl to denote topological closure. This is needed when the invariant is open, e.g. $c < d$.

a transition from q to q' then $\lambda(q) \neq \lambda(q')$. A transducer is *input-dependent* if all the cycles in its transition graph contain at least one transition labeled by an input event and all stable states have bounded invariants. The following are simple observations: 1) The basic automata of Figure 2 are input-dependent; 2) If $\mathcal{A}_1$ and $\mathcal{A}_2$ are input dependent, so is their composition (a cycle in a composition of two automata must amounts to a cycle in each of the components); 3) an input-dependent transducer realizes an ultimately-stable transduction. From now on we restrict ourselves to input-dependent transducers.

We say that a timed transducer $\mathcal{A}'$ which differs from $\mathcal{A}$ only in its invariants and guards is more permissive than $\mathcal{A}$ if for every q, $I_q \subseteq I'_q$ and for every transition $(q, g, \rho, q') \in \Delta$ there is a transition $(q, g', \rho, q') \in \Delta'$ such that $g \subseteq g'$. We denote this fact by $\mathcal{A} \preceq \mathcal{A}'$. Clearly $\mathcal{A} \preceq \mathcal{A}'$ implies $f_{\mathcal{A}} \preceq f_{\mathcal{A}'}$.

3 Adding Input Clocks

Before describing the first transformation, that is, from $\mathcal{A}$ to $\mathcal{A}^t$, let us contemplate on the life and death of an input event in an acyclic network of timed components such as the one depicted in Figure 1-(b). To facilitate the discussion we assume that all components have identical lower and upper bounds l and u on their reaction time. When such an event occurs it may excite one or more of the components to which it is a direct input. In the absence of additional events, each of those components, stabilizes within some $t \in [l, u]$ and emits a change in its output, which may trigger reactions in some further components, and so on. Due to acyclicity, a component which has reacted to an input event cannot be influenced anymore by the same event. Consequently all input events leave the system within a finite amount of time, bounded by $d \cdot u$, where d is the *depth* of the network, the maximal number of sequentially-connected components.

A second observation is that in the circuit model (Figure 2-(b)), every two changes in the value of the same variable must be separated by at least l time in order for both to be alive in the system, otherwise the second change aborts the first via a "regret" transition. Hence the maximal number of live events for each input variable is bounded by $m = d \cdot u/l$. Similar bounds will hold for other approaches for treating excessive input variability and, in fact, one may see that the number of living events in a network is always bounded by the number of its components. These are upper bounds and in practice the maximal number of live events can be much smaller due to logical interference and stronger bounded-variability assumptions.

Automaton $\mathcal{A}^t$ is obtained by augmenting $\mathcal{A}$ with additional state variables that keep track of live events and additional clocks that measure the age of these events. The discrete state variables are counters $i_{x_1}, \ldots i_{x_n}$, one for each input variable, ranging over $M = \{0, 1, \ldots, m\}$ where m is the previously-mentioned upper bound, and a dedicated structure that we call *event-recording table*.

Definition 5 (Event Recording Table). *An event recording table for a timed transducer $\mathcal{A}$ having X as input variables and $\mathcal{C}$ as clocks is $\mathcal{E} : X \times M \rightarrow 2^{\mathcal{C}}$ whose value at a given state has the following intended meaning: $c \in \mathcal{E}(x, i)$ if clock c is active and its value is causally related to the time elapsed since the occurrence of the i^{th} oldest x-event in the system.*

We denote the (finite) set of valuations of the event recording table by $E_{\mathcal{A}}$. We denote by $\hat{\mathcal{C}}$ the set of additional clocks of the form $c_{x_j}[i]$, each measuring the time since the i^{th} oldest change in the value of x_j, for $1 \leq j \leq n$ and $1 \leq i \leq m$.

Definition 6 (Transducer $\mathcal{A}^t$). *Let $\mathcal{A} = (\Sigma, Q, \Gamma, \mathcal{C}, \lambda, \gamma, q_0, I, \Delta)$ be a timed signal transducer. Its extension with input clocks is the transducer $\mathcal{A}^t = (\Sigma, Q^t, \Gamma, \mathcal{C}^t, \lambda^t, \gamma^t, q_0^t, I^t, \Delta^t)$ where $Q^t = Q \times M^n \times E_{\mathcal{A}}$, $\mathcal{C}^t = \mathcal{C} \cup \hat{\mathcal{C}}$ and λ^t, γ^t and I^t are simple extensions of λ, γ and I, respectively, for example, $\lambda^t((q, u, \mathcal{E}) = \lambda(q)$. The initial state is $q_0^t = (q_0, \mathbf{0}, \mathcal{E}_0)$ with $\mathbf{0}$ being a vector of zeros for the counters and $\mathcal{E}_0$ satisfying $\mathcal{E}_0(x, i) = \emptyset$ for every x and i. Then for every $(q, u, \mathcal{E}) \in Q^t$ and every transition $\delta = (q, g, \rho, q') \in \Delta$ we create a transition $((q, u, \mathcal{E}), g, \rho^t, (q', u', \mathcal{E}'))$ in Δ^t where u', $\mathcal{E}'$ and ρ^t are the result of applying the following transition-dependent transformation to u, $\mathcal{E}$ and ρ:*

1. *If δ is labeled by an input event x and resets clock c, then $i_x := i_x + 1$, $\mathcal{E}(x, i_x) := \{c\}$ and $\rho := \rho \cup \{c_x[i_x] := 0\}$;*
2. *If a clock $c \in \mathcal{E}(x, i)$ appears in g and the transition resets a clock c', then $\mathcal{E}(x, i) := \mathcal{E}(x, i) \cup \{c'\}$;*
3. *If δ is labeled by $c := \perp$ then $\mathcal{E}(x, i) := \mathcal{E}(x, i) - \{c\}$ for every x and i such that $c \in \mathcal{E}(x, i)$. If this operation renders some $\mathcal{E}(x, i)$ empty, we perform event and clock* shifting*, that is, for every $k \geq i$ we let $\mathcal{E}(x, k) := \mathcal{E}(x, k + 1)$, add to ρ^t the assignments $c_x[k] := c_x[k + 1]$ and $c_x[i_x] = \perp$ and finally let $i_x := i_x - 1$.*

Rule 1 corresponds to the arrival of a new event and the generation and resetting of its clock. Rule 2 follows the propagation of the event to a subsequent component. Rule 3 represent the stabilization of a component after reacting to an event. If this was the last component in the circuit excited by the event, this signifies death of that event in the system. Clock shifting is just a technical means to utilize the same finite pool of clocks.

Proposition 1. *Automaton $\mathcal{A}^t$ is an ordinary timed transducer which is semantically equivalent to $\mathcal{A}$.*

The set of states is finite due to finiteness of M^n and $E_{\mathcal{A}}$. The extra event-recording machinery does not interfere with the original dynamics of $\mathcal{A}$, neither in the guards, in resetting the clocks of $\mathcal{A}$, nor in the assignment of input-output symbols to states. In other words, the projection of Q^t on Q defines a behavior-preserving homomorphism from $\mathcal{A}^t$ to $\mathcal{A}$.

4 The Reachable Automaton

To determine the possible behaviors of a timed automaton (and verify whether all of them satisfy certain properties) one must analyze the infinite transition system which the automaton represents. The problem is more complicated than in ordinary finite-state automata because of the clock variables. We recall some definitions commonly used in the verification of timed automata [HNSY94,Y97,LPY97,BDM$^+$98]. 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 $\{(q, v) : v \in Z\}$. 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:

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

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 $\delta = (q, g, \rho, q') \in \Delta$:

$$Post^\delta(q, Z) = \{(q', \rho(v)) : v \in Z \cap g\}.$$

– 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 7 (Reachability Graph). *The reachability graph associated with a timed automaton $\mathcal{A}$ is a directed graph $G = (N, \rightarrow, n_0)$ such that N is the smallest set of symbolic states containing $n_0 = Post^t(q_0, \{\mathbf{0}\})$ 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 $v' \in Z'$ there exists a clock valuation $v \in Z$ and a run of the automaton from (q, v) to (q', v'). Hence the union of all symbolic states in N gives exactly the configurations of $\mathcal{A}$ reachable from $(q_0, \mathbf{0})$. From the reachability graph G of a timed transducer $\mathcal{A}$, one can build a timed transducer $\mathcal{A}^r$ with an equivalent behavior, that is, $f_\mathcal{A} = f_{\mathcal{A}^r}$.

Definition 8 (Interpreted Timed Transducer). *Let $\mathcal{A} = (\Sigma, Q, \Gamma, \mathcal{C}, \lambda, \gamma, q_0, I, \Delta)$ be a timed transducer and let $G = (N, \rightarrow, n_0)$ be its reachability graph. The interpreted timed transducer of $\mathcal{A}$ is $\mathcal{A}^r = (\Sigma, N, \Gamma, \mathcal{C}, \lambda^r, \gamma^r, n_0, I^r, \Delta^r)$ where for every $(q, Z) \in N$, $\lambda^r(q, Z) = \lambda(q)$, $\gamma^r(q, Z) = \gamma(q)$, $I^r(q, Z) = Z$ and for every edge $(q, Z) \rightarrow (q', Z')$ in G, associated with transition $\delta = (q, g, \rho, q') \in \Delta$, we fix a transition $((q, Z), g \cap Z, \rho, (q', Z')) \in \Delta^r$.*

An additional important property of the interpreted timed transducer is that if we relax completely its timing constraints, still each of its input-output behaviors is an untiming of some behavior of the original transducer $\mathcal{A}$. More concisely, with abuse of notation, this can be phrased as:

$$\mu(f_\mathcal{A}) = f_{\mu(\mathcal{A})}.$$

Applying the construction of interpreted timed transducer to $\mathcal{A}^t$ we obtain the automaton $\mathcal{A}^r$ which is equivalent to $\mathcal{A}$.

5 Clock Projection

The next step is to project all the timing constraints in $\mathcal{A}^r$ on the input clocks $\hat{\mathcal{C}}$ to obtain $\mathcal{A}^p$ which is an abstraction of $\mathcal{A}$ and satisfies $f_{\mathcal{A}^r} \preceq f_{\mathcal{A}^p}$. Moreover, due to the property of the reachability graph the untiming of $f_{\mathcal{A}^p}$ is equivalent to that of $f_{\mathcal{A}^r}$. To gain some intuition on the type of information which is lost and that which is preserved in the procedure, let us look at the example depicted in Figure 4, which consists of a network with two components $\mathcal{A}_1, \mathcal{A}_2$ which simply propagate the value of x after some delay to y_1 and later to y_2. Clocks c_1 and c_2 are the clocks of the components and clock c_x is the added input clock.[4]

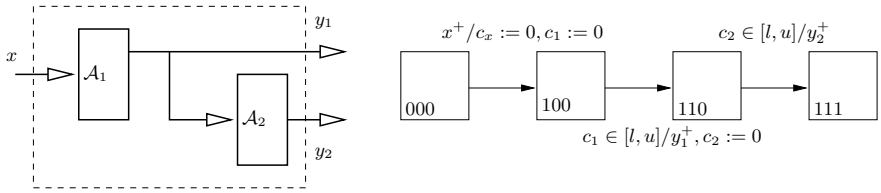


Fig. 4. (a) A network consisting of two timed components; (b) its timed automaton $\mathcal{A}^t$ augmented with one input clock.

Performing reachability computation on this automaton we obtain the automaton $\mathcal{A}^r$ with the same structure in which transitions y_1^+ and y_2^+ are guarded, respectively, by the following zones:

$$Z_1 = (l \leq c_1 \leq u \wedge c_x = c_1) \qquad Z_2 = (l \leq c_2 \leq u \wedge l \leq c_2 - c_x \leq u)$$

After projection we obtain the automaton $\mathcal{A}^p$ with zones

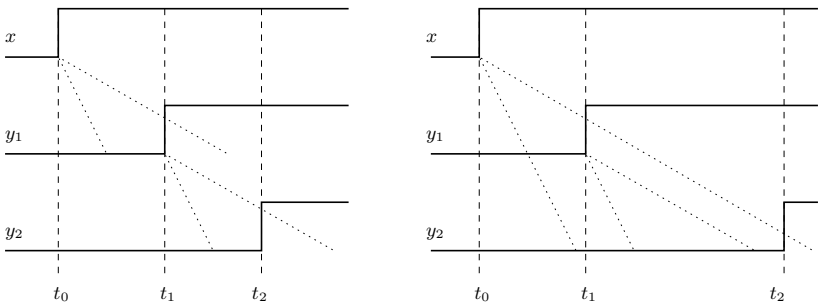
$$\hat{Z}_1 = (l \leq c_x \leq u) \qquad \hat{Z}_2 = (2l \leq c_x \leq 2u)$$

To understand the difference in the semantics of the two automata let us look at the two behaviors depicted in Figure 5. Both behaviors consist of x^+ at some time t_0, then y_1^+ at some t_1 and y_2^+ at time t_2. In the original automaton these should satisfy $l \leq t_1 - t_0 \leq u$ and $l \leq t_2 - t_1 \leq u$. The first condition is reflected by Z_1 because at any time t, $c_1 = t - t_0$, and the second condition is imposed by Z_2 because at time t, $c_2 = t - t_1$. After the projection, the relation between t_1 and t_2 is broken and only the constraint $2l \leq c_x \leq 2u$, which was redundant in Z_2, remains in $\hat{Z}_2$, which allows y_2^+ to occur at $t_2 \notin [t_1 + l, t_1 + u]$. The abstraction technique proposed in [ZMM03] for timed Petri nets has the same effect.

Formally, if Z is a zone consisting of valuations $(v, \hat{v})$ for the clocks $\mathcal{C} \cup \hat{C}$. Its projection on the input clocks is

$$Z{\downarrow} = \{\hat{v} : \exists v (v, \hat{v}) \in Z\}$$

[4] To simplify the discussion we consider only one input event.



(a) (b)

Fig. 5. (a) A behavior of $\mathcal{A}^t$; (b) a behavior of $\mathcal{A}^p$ in which $t_2 \in [t_0 + 2l, t_0 + 2u]$ but $t_2 \notin [t_1 + l, t_1 + u]$.

Syntactically this operation is performed by adding to the description of Z all the minimal constraints of the form $c \leq d$ and $c - c' \leq d$ which are implied by the constraints in Z and then removing all the constraints that mention clocks in C. The remaining constraints define $\hat{Z}$ and this operation is easily implemented using difference-bounds matrices (DBM). The automaton $\mathcal{A}^p$ is obtained from $\mathcal{A}^r$ by simply applying the projection operation to all guards, invariants and clock updates. An important property of $\mathcal{A}^p$ is that all its clocks are input clocks which are *not reset to zero* by any transition in an internal component which is not labeled by an input event.

6 State Minimization

Automaton $\mathcal{A}^p$ has less clocks but its discrete state space is still large as it makes distinctions between states that differ only in the value of internal variables. Our goal is to produce a small-size abstraction for reducing the complexity of verification when using the reduced model of the component inside a larger network. Our last step is to merge such states and reduce the automaton as much as possible without changing significantly its semantics. Contrary to what one may prematurely think, careless merging of states that agree on observable variables may not be faithful to the semantics because such states may admit *different observable futures*. We will sketch below two minimization procedures. The first one is more aggressive and preserves the reachable configurations of $\mathcal{A}^p$ but may admit more behaviors.[5] The other minimization technique restricts the merging to states which are equivalent with respect to a time-abstracting bisimulation and hence it does preserve the semantics. When states are merged, silent transitions between them disappear and their invariants are combined to yield the invariant of the merged state.

[5] At this point we are not sure whether, for the special class of timed automata that we use, this reduction preserves the semantics, so we prefer to consider it as an over-approximation.

We say that a transition of $\mathcal{A}^p$ is *observable* if it is associated with at least one input event, output event or clock shifting (death of an input event). A transition which is not observable is called *silent* and is denoted by τ. A silent run is a run consisting solely of time passage and silent transitions. Note that since our automata are input-dependent there are no silent cycles and every silent run has a finite number of transitions and a finite maximal duration until it reaches a stable state. A state of $\mathcal{A}^p$ is called *rigid* if all its incoming transitions are observable. Given a timed automaton with a set of states Q, we denote by $\Omega(Q)$ the set of its rigid states. The set of silent successors of a rigid state q (including q itself) is denoted by $Succ^\tau(q)$. Note that all elements of $Succ^\tau(q)$ have the same input/output labels.

The first minimization procedure is based on having one state corresponding to each rigid state and all its silent successors.

Definition 9 (Minimized Timed Transducer). *Let $\mathcal{A}^p = (\Sigma, Q, \Gamma, \mathcal{C}, \lambda, \gamma, q_0, I, \Delta)$ be a timed transducer obtained by the sequence of steps previously described. Its minimization is $\mathcal{A}^m = (\Sigma, Q^m, \Gamma, \mathcal{C}, \lambda^m, \gamma^m, q_0^m, I^m, \Delta^m)$ where $Q^m = \Omega(Q)$,*

$$I_q^m = \bigcup_{q' \in Succ^\tau(q)} I_{q'},$$

and Δ^m is constructed from Δ as follows: for every observable transition $\delta = (p, g, \rho, p') \in \Delta$ and every q, q' such that $p \in Succ^\tau(q)$ and $p' \in Succ^\tau(q')$ we define a transition (q, g, ρ, q') in Δ^m. The labeling functions λ^m and γ^m are the restrictions of λ and γ to $\Omega(Q)$.

Proposition 2. *The following two statements are equivalent:*

1. *There is a silent run ξ of $\mathcal{A}^p$ from (q, v) to $(q', v + t)$;*
2. *There is a time step of $\mathcal{A}^m$ from (q, v) to $(q, v + t)$.*

Proof: see appendix.

Unfortunately this property is not sufficient by itself for preserving the behavior as the example in Figure 6 shows. In this example states $\{r_i, p_i\}$ are merged with some rigid state q_i. Suppose that the visible transition a is possible in $\mathcal{A}^p$ at r_1 but not at any of its silent *predecessors*, and that visible transition b is possible at p_2 and not in any of its silent *successors*. After merging we may create the possibility of an observable sequence a, b which is not possible in $\mathcal{A}^p$. Note that the situation here is more subtle then in purely-discrete abstraction because the original transition guards are preserved in $\mathcal{A}^m$ and may prevent this false transitivity. Because of the monotonicity of the guards along silent paths (clocks are not reset), the only reason for b not to be enabled at r_2 is logical, but then due to some confluence properties of our automata, it might be the case that p_2 can always be reached via another a-labeled path. But since we cannot prove this at the moment of writing we are content with the fact that $\mathcal{A}^m$ over-approximates the behavior of $\mathcal{A}^p$ and preserves, by construction, its set of reachable configurations.

The second minimization procedure is based on computing a congruence relation, a variant of time-abstracting bisimulation [LY97,TY01] on the states of $\mathcal{A}^p$ which declares two states as equivalent if they admit the same set of future qualitative behaviors.

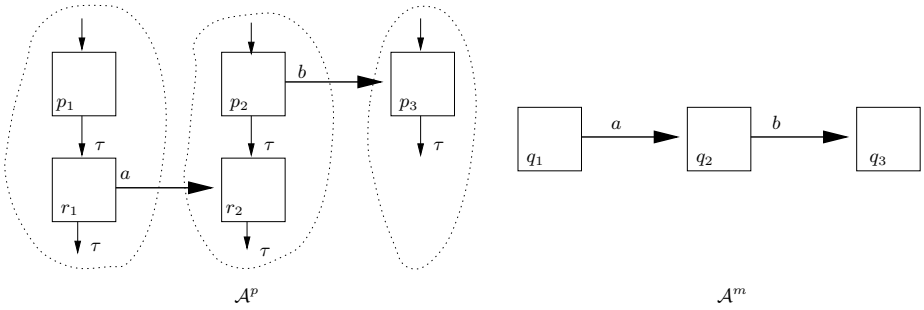


Fig. 6. The relation between abstract and concrete runs.

Let $q \xrightarrow{\tau^*} q'$ denote the fact that there is a silent path from q to q' and let a stand for an observable action. The relation $\sim$ on the states of $\mathcal{A}^p$ is defined as the largest relation satisfying for every q and p

$$q \sim p \text{ iff } \forall a \, \forall q' \, (q \xrightarrow{\tau^*} \xrightarrow{a} \xrightarrow{\tau^*} q') \Rightarrow (\exists p' \; p \xrightarrow{\tau^*} \xrightarrow{a} \xrightarrow{\tau^*} p' \wedge q' \sim p').$$

Then, the minimized automaton is similar to Definition 9 except for the fact that merging is restricted to equivalence classes of $\sim$. The question which minimization to prefer is an empirical one concerning the tradeoff between faithfulness and complexity that we hope to report in the final version of this paper.

7 Implementation Status and an Example

We have implemented a tool chain for performing the various steps in our abstraction technique. We start with a high level description of the system as a network of Boolean gates with delays which is transformed into a network of timed automata. We then perform time reachability computation while generating and maintaining input clocks. Note that this involves the introduction of dynamic clocks whose denotation may change over time, a fact which requires a special procedure for DBM normalization. To avoid explosion due to interleaving we use our new BFS exploration algorithm [BBM06] which merges symbolic states that are reached by confluent paths. We are in the process of completing the adaptation of the minimization procedures. All in all, the implementation effort consists of 43K lines of code, in addition to the standard IF code base.

We demonstrate the effect of our abstraction on a small circuit with on input x which enters two delay elements with different delays whose outputs go into an AND gate (see Figure 7). The delay bounds are $[4, 6]$ and $[10, 12]$ for the delay elements and $[1, 2]$ for the AND gate. With this parameters, two x-events may be alive simultaneously in the system. The automaton $\mathcal{A}^p$ obtained automatically by our tool is shown in Figure 8 before minimization. After minimization the number of states will be reduced from 17 to 13. This saving is, of course, very small because this circuit has very few internal

variables, and will be much more significant when we apply the technique to circuits with 10-20 gates after completing the implementation.

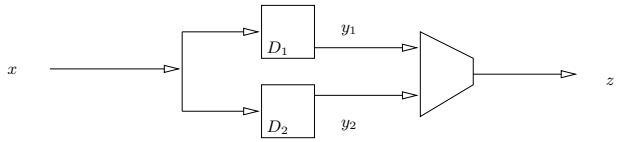


Fig. 7. A circuit with y_1 and y_2 being delays (with different parameters) of x and z the delay of $y_1 \wedge y_2$.

8 Discussion

We have developed a novel methodology for automatic abstraction of timed components and implemented most of its ingredients. In addition to its potential contribution to more efficient verification of timed automata, the new concepts introduced in this work concerning the propagation of event "waves" in an acyclic networks of timed components are of interest by themselves and may be useful for other approaches for analyzing this kind of systems. We are currently completing the implementation of the minimization procedures in order to apply the technique to larger examples.

References

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

[BBM06] R. Ben Salah, M. Bozga and O. Maler, On Interleaving in Timed Automata, *CONCUR'06*, 465-476, LNCS 4137, 2006.

[BBM03] R. Ben Salah, M. Bozga and O. Maler, On Timing Analysis of Combinational Circuits, *FORMATS'03*, 204-219, LNCS 2791, 2003.

[BGM02] 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.

[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.

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

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

[LY97] K.G Larsen and W. Yi: Time-abstracted Bisimulation: Implicit Specifications and Decidability, *Information and Computation* **134**, 75-101, 1997.

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

[TY01] S. Tripakis and S. Yovine, Analysis of Timed Systems Using Time-Abstracting Bisimulations, *Formal Methods in System Design* **18**, 25-68 2001.

[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.J. Myers, Modular verification of timed circuits using automatic abstraction, *IEEE Trans. on CAD* 22, 2003.

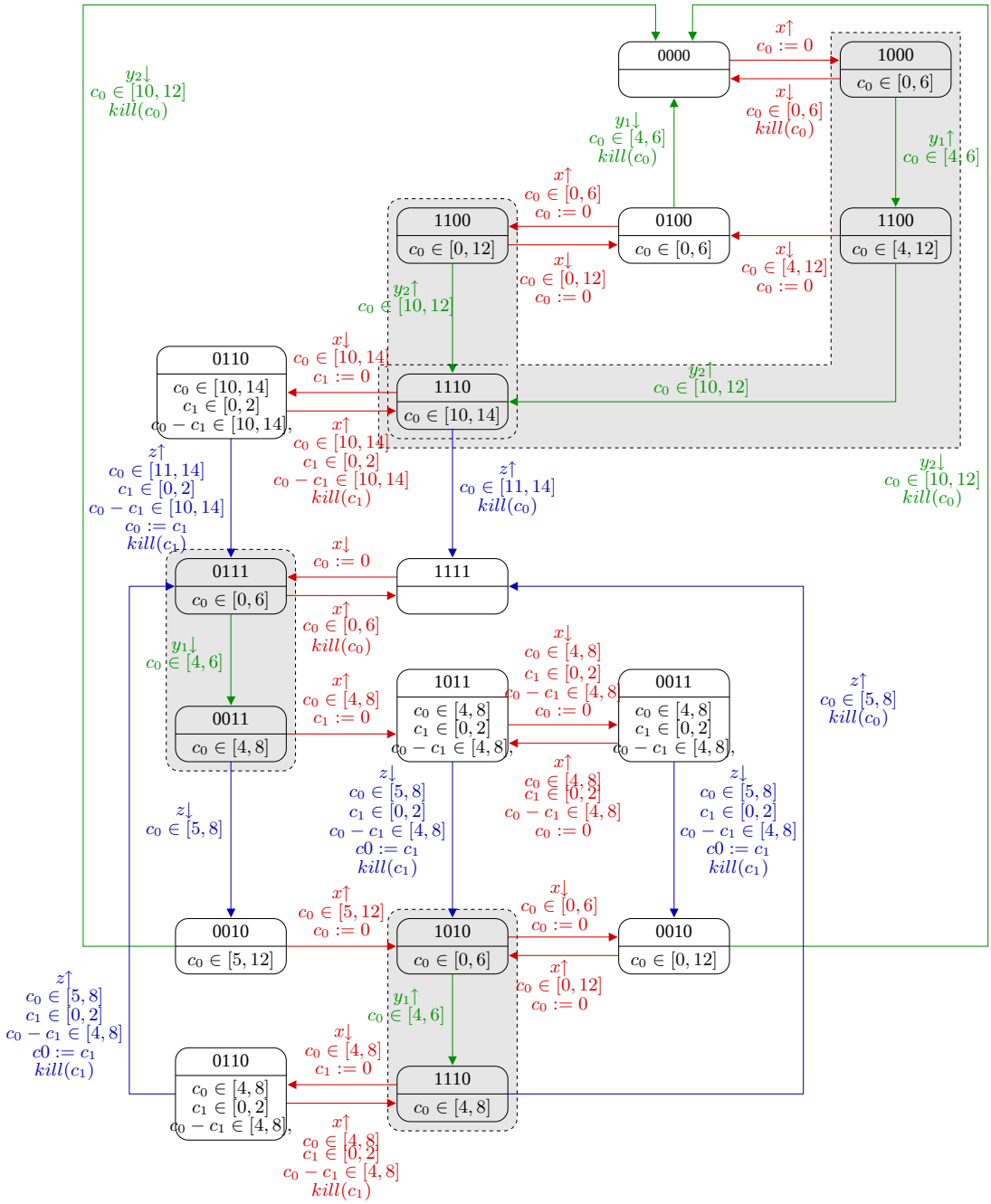


Fig. 8. The automaton $\mathcal{A}^p$ obtained for the circuit of Figure 7. It uses two input clocks, c_0 and c_1. The candidates for merging are shaded.

Appendix

Proposition 2. *The following two statements are equivalent:*

1. *There is a silent run ξ of $\mathcal{A}^p$ from (q, v) to $(q', v + t)$;*
2. *There is a time step of $\mathcal{A}^m$ from (q, v) to $(q, v + t)$.*

Proof: Let us prove that first direction, $1 \Rightarrow 2$, by induction on the number of discrete transitions in ξ. In the base case when there are no discrete transitions, $q = q'$ and the two statements are identical. The proof of the inductive step from runs with k discrete transitions to runs with $k + 1$ transitions goes as follows. Such a run has the following form:

$$(q, v) \xrightarrow{t_1} (q, v+t_1) \xrightarrow{\tau_1} (q_1, v+t_1) \cdots (q_k, t) \xrightarrow{t_{k+1}} (q_k, v+t+t_{k+1}) \xrightarrow{\tau_{k+1}} (q', t+t_{k+1})$$

with $t = t_1 + \ldots + t_k$. According to the inductive hypothesis there is a matching time step $(q, v) \xrightarrow{t} (q, v + t)$ in $\mathcal{A}^m$ and what remains to show is that it can be prolonged by t_{k+1} time. Since $v + t + t_{k+1}$ satisfies I_{q_k} and $I_{q_k} \subseteq I^m_q$ we have indeed the time step $(q, v) \xrightarrow{t+t_{k+1}} (q, v + t + t_{k+1})$.

To prove the direction $2 \Rightarrow 1$ we assume a time step $(q, v) \xrightarrow{t} (q, v')$ in $\mathcal{A}^m$. Clearly, $v' \in I^m_q$ and hence $v' \in I_{q'}$ for some $q' \in Succ^\tau(q)$. If $q = q'$ we are done as $(q, v) \xrightarrow{t} (q, v')$ is also a time step of $\mathcal{A}^p$. Otherwise, due to the properties of the simulation graph, there is some $t' > 0$ and $q'' \in Succ^\tau(q)$ such that $(q'', v' - t') \xrightarrow{\tau} (q', v' - t') \xrightarrow{t'} (q', v)$. If $q'' = q$ we are done otherwise we repeat the same argument for $(q'', v' - t')$. Since all silent paths are acyclic we finally reach (q, v) and establish the existence of the corresponding run. □