Compositional Timing Analysis

Ramzi Ben Salah, Marius Bozga, Oded Maler

Compositional Timing Analysis

Ramzi Ben Salah
VERIMAG - Centre Equation
2 Avenue de Vignate
38610 Gières, France
Ramzi.Salah@imag.fr

Marius Bozga
VERIMAG - Centre Equation
2 Avenue de Vignate
38610 Gières, France
Marius.Bozga@imag.fr

Oded Maler
VERIMAG - Centre Equation
2 Avenue de Vignate
38610 Gières, France
Oded.Maler@imag.fr

ABSTRACT

We develop and implement a methodology for automatic abstraction of systems defined as networks of timed components modeled by timed automata. The abstraction technique yields an abstract model with much less clocks and states which over-approximate the timed behavior of the concrete system. Using this technique we can analyze timed system of size beyond the capabilities of contemporary analysis tools for timed automata.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification; B.7.2 [**Integrated Circuits**]: Design Aids; F.1.1 [**Computation by abstract devices**]: Models of Computation

General Terms

Theory, Verification

Keywords

timed automata, abstraction, reachability analysis, compositional generation

1. INTRODUCTION

The only way to master the complexity of large systems is to apply a *hierarchical/compositional* design methodology. The basic principle of such a methodology is that a subsystem treated at one level of abstraction as a model M of some detail, is encapsulated as a component M' when a higher level is considered. The major difference between M and M' is that the latter abstracts away from the internal details appearing in M and focuses on the observable input-output (interface) behavior of the component which determine its high-level functionality. To take an illustrative example, consider the process of submitting a paper to a conference. Viewed from the perspective of the authors, the model M is a complex network or processes involving ideas, model formulation, proving theorems, implementation, experiments, literature survey and typesetting, admitting complex inter dependencies. On the other hand, from the perspective of the program chair all this internal details are irrelevant and the abstract model M' only specifies whether a substantial contribution has been submitted by the deadline, regardless of the respective timing of the sub processes.

For M' to be useful in the analysis at the higher level it should be significantly less complex than M and since there is no free lunch, the price of the abstraction is that M' is less precise than M in terms of the observable behaviors it admits. We work in the context of set-theoretic nondeterminism where the abstract model, by ignoring certain variables, allows more observable behaviors than a detailed model would. To be more concrete, think of M as an automaton whose state space consists of vectors of valuations to variables. Projecting away some variables will make the automaton more nondeterministic and will enlarge the set of observable sequences it can generate. In the verification context, this over-approximation of the semantics guarantees that correctness proofs obtained using the abstract model are valid for the concrete model. For the purpose of performance evaluation, which is one of the central issues in embedded systems, performance guarantees obtained using M' will hold for the detailed model M as well.

In this work we develop a novel methodology, supported by a tool chain, where M is a network of *timed* components, representing processes that respond to input events by emitting the corresponding output events within some time delay which is known to be bounded within a given interval. Such timed components may represent the execution times of software modules in a streaming application, communication delays in a network, the response times of servers or propagation delays in digital gates. Each component is modeled as a timed automaton with variables (a variant of the timed automaton of [3] adapted for the compositional context) and the whole network is equivalent to a global timed automaton with at least one state variable and one clock[1] per component. Although major analysis problems for timed automata are decidable, and despite of a lot investment and engineering innovation in timed automata tools over the last decade, the analysis of such networks remains infeasible beyond a modest amount of components. The essence of our work is the *automatic* generation of a smaller timed automaton M' with less states and less clocks which preserves the observed qualitative semantics of M and over-approximates its timed semantics. The reduced model M' can then replace M as a component in a higher level.

The core idea behind our technique is to use the clocks associated with the internal processes in M in order to analyze the possible time-dependent behaviors of the system. To do that we need to

EMSOFT'09, October 12–16, 2009, Grenoble, France.

[1]To avoid confusion with hardware or real-time operating systems and emphasize that clocks in timed automata are fictitious clocks used to *model* the elapse of time and have nothing to do with the implementation of the system.

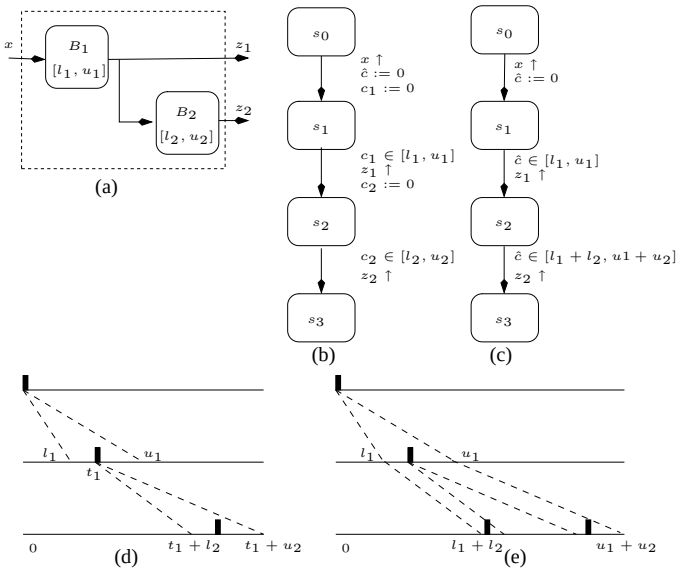

Figure 1: (a) A network of 2 components; (b) The automaton with an auxiliary clock; (c) The automaton after projection on the auxiliary clock; (d) The semantics of the network; (e) The approximate semantics after projection.

restrict the number of components in M to the capabilities of existing verification tools (5-20 components) and treat larger systems in a divide-and-conquer fashion by decomposing them into smaller chunks, applying the abstraction technique and composing the resulting abstract models. Before diving into the formal details of the chain of transformations leading from M to M' let us illustrate informally the nature of the abstraction via an example (see also [8]).

The network in Figure 1 has two components, B_1 which responds to input x by emitting output z_1 within a delay in the interval $[l_1, u_1]$ and B_2 which reacts to z_1 within $t \in [l_2, u_2]$ time. We assume that x occurs at time zero and that both z_1 and z_2 are considered observable. The range of possible behaviors is illustrated in Figure 1-(d) where z_2 is constrained to occur between l_2 and u_2 time after z_1. As a first step we augment the timed automaton representing the composition $B_1||B_2$ with an additional clock $\hat{c}$ which measures the time elapsed since event x, see Figure 1-(b). This clock does not participate in the transition guards and hence does not influence the behavior of the automaton, it only adds time stamps to the runs. We then compute the interpreted timed automaton based on forward reachability[2] and obtain an equivalent automaton where transition guards are expressed as constraints over clocks $\{c_1, c_2, \hat{c}\}$ in the form of zones. We then get rid of the internal clocks, after they have served us to restrict the behaviors of the automaton, and project the zones on $\hat{c}$ to obtain the abstract automaton of Figure 1-(c). In this automaton z_1 is constrained to occur in the absolute time interval $[l_1, u_1]$ (which is exact) while z_2 is allowed to take place in the interval $[l_1 + l_2, u_1 + u_2]$ regardless of the occurrence time of z_1 (which is an over-approximation, see Figure 1-(e)).

This example is too small to illustrate the role of timed reachability computation in restricting the range of behaviors of the automaton, it only give an intuition concerning the effect of clock projection. Another simplifying aspect was the fact that we handled only one input event. In this work we address the challenging problem of adapting this idea to *infinite streams* of input events that may occur at arbitrary time points. We let each input event generate its own clock, reset to zero upon arrival, and after the reachability analysis we get rid of the internal clocks and project the timing constraints on the input clocks. Doing so, we generate a reduced abstract model in which the timing of each output event is conditioned on the time elapsed since the input event that has triggered it. We restrict ourselves to *acyclic* networks of timed components in which every component admits a finite upper-bound on its reaction time. In such networks, each input event triggers a *wave* of activity along the network, a wave which progresses from inputs to outputs and leaves the network within a bounded amount of time. This fact, together with bounded frequency assumptions on the inputs, guarantees that the number of co-existing waves is always bounded and, consequently, so is the number of input clocks. Whenever the processing of an event terminates, its clock can be reused by subsequent events. Indeed, the adaptation of the IF toolbox [10] to handle those *dynamic* clocks with varying denotations was the major implementation challenge in this work which goes all the way from conception to realization.

The problem we address is central to the development of embedded systems and no wonder it has attracted many research groups representing different communities and approaches [22, 18], some of which we mention very briefly below. The work around the *real-time calculus* and similar formalisms [24, 26, 14] attempts to adapt analytic tools taken from domains such as queueing theory or max-plus algebra and adapt them to the more irregular setting of embedded systems admitting phenomena such as jitter, burst and discrete decisions of schedulers which are not easily modeled in the classical frameworks. The work and the tools developed in this framework share with ours the overall goal, namely give an abstract and approximate characterization of the timed "transfer function" of components as a transducer of timed streams, however the proposed techniques are radically different. Our approach does not start with clean algebraic framework which scale-up well but is very approximate to start with, but rather starts from the opposite: a state based model represented as a timed automaton which is much stronger in expressive power but is notorious for not scaling up beyond toy problems.[3] It is this clock-explosion challenge that we try to meet in this work. Compositional reasoning has been applied to timed automata in [23] and is advocated for timing and performance issues in the context of interface automata [1, 15] which offers a formal framework for checking whether the performance that a component provides matches the performance expected by other components. The major differences with respect to our work is that we attempt to generate the description of the abstract performance characterization of the component *automatically* from the more detailed model.

The rest of the paper is organized as follows. In Section 2 we present all the prerequisite material on timed automata. In Section 3 we add state variables and additional structure to the automata to facilitate compositional discourse. Section 4 is the core of the paper where we introduce all the steps in our abstraction techniques, namely, the introduction of dynamic events clocks, the reachability computation, the projection of internal clocks and variables and the minimization. In Section 5 we demonstrates the application of the tool chain to a case study consisting of a network of 36 components, modeling a sequential circuit used to illustrate the concept of *wave pipelining*. We apply our technique to find bounds on the

[2]These notions will be explained in the sequel.

[3]We mention [17] as a recent attempt to reconcile these two approaches for cyclic components.

input frequency which guarantee that no elementary component receives an input event before having digested the previous event.

2. TIMED AUTOMATA

In this section we define the variant of timed automata that we use. Notable deviations from the classics, made to fit our modeling and analysis needs, are that we make clock inactivity explicit[4] and that we allow clock copying, not only resets to zero, during transitions. We start with preliminary definitions of clocks, zones, etc., then define timed automata and their semantics and conclude with *forward reachability computation* over *zones*, the method of choice in contemporary tools [16, 12, 19, 10], and its byproduct, the *interpreted timed automaton* which plays an important role in our analysis technique.

2.1 Preliminary Definitions

Let $\mathbb{R}_{\perp}$ denote $\mathbb{R}_+ \cup \{\perp\}$ were $\perp$ is a special symbol meaning *inactive*. We extend addition to $\mathbb{R}_{\perp}$ by letting $\perp + d = \perp$. Let $C = \{c_1, ..., c_n\}$ be a finite set of variables called *clocks* ranging over $\mathbb{R}_{\perp}$. A *clock valuation* is a function $v : C \to \mathbb{R}_{\perp}$ and the set of possible valuations of C is $\mathbb{R}^n_{\perp}$. A clock c is said to be *active* in valuation v iff $v(c) \neq \perp$. Due to the distinction between active and inactive clocks, we have to deal with clock valuations of varying dimensionality. Given some $C' \subseteq C$, we use $\mathcal{V}_{C'}$ to denote the set of valuations in which $v(c) \neq \perp$ iff $c \in C'$. Elements of $\mathcal{V}_{C'}$ are then non-negative real vectors (points) of dimension $|C'| \leq n$. The projection operation $v' = v/_{C'}$ maps elements of $\mathcal{V}_C$ to elements of $\mathcal{V}_{C'}$ by letting $v'(c) = v(c)$ if $c \in C'$ and $v'(c) = \perp$ otherwise.

In timed automata clock valuations change due to two types of activities: *time progress* which happens inside a discrete state and *clock assignments* which take place during discrete transitions. For $d \geq 0$, we say that clock valuation v' is the result of applying *d-time-progress* to v, denoted by $v' = v + d$, if for every clock c, $v'(c) = v(c) + d$. Note that inactive clocks do not change their value. A clock assignment is a function $\gamma : \mathbb{R}^n_{\perp} \to \mathbb{R}^n_{\perp}$ indicating a transformation of clock values upon a transition. Assignments are restricted to compositions of one or more of the following basic assignments: $c_i := 0$ (resetting to zero), $c_i := \perp$ (clock deactivation) and $c_i := c_j$ (clock copying). We use $v' = \gamma(v)$ to denote the fact that v' is the result of applying γ to v and use $\gamma_1 \circ \gamma_2$ for composition of assignments. The set of all assignments thus obtained is denoted by Γ_C. We use the shorthand $r_{C'}$ to denote resetting all the clocks in some $C' \subseteq C$ and $kill_{C'}$ for their deactivation.

Clocks constraints are used to express the influence of clock values on the behavior of the automaton. We use a family of constraints denoted by Ψ_C, defined by the following grammar:

$$\psi ::= true \,|\, c_i \prec k \,|\, c_i - c_j \prec k \,|\, \psi \wedge \psi$$

where $c_i, c_j \in C$, $k \in \mathbb{N}$ and $\prec \in \{<, \leq, =, \geq, >\}$. Constraints of the form $x \prec k$ and $x - y \prec k$ are called *atomic*. The *restriction* of some $\psi \in \Psi_C$ to a set of clocks $C' \subseteq C$, denoted by $\psi/_{C'}$, is the constraint obtained by deleting all atomic constraints that mention a clock outside C'.

We use $v \models \psi$ to denote the fact that valuation v *satisfies* a clock constraint ψ. We will always assume (and guarantee) that constraints evaluated in a state of the automaton mention only clocks active in that state. The set Z_ψ of valuations satisfying ψ is a subset of $\mathbb{R}^m_+$ where $m \leq n$ is the number of clocks active in the state. For any ψ, Z_ψ is a convex polyhedron defined as the intersection of half-spaces which are either orthogonal ($c_i \prec k$) or diagonal ($c_i - c_j \prec k$) with integer k. The set of all such polyhedra that we call *timed zones* will be denoted by $\mathcal{Z}_C$ and its cardinality is finite in any bounded subset of $\mathbb{R}^n$.

Zones are closed under intersection and under the operations defined below. The *timed convex hull* of two zones is

$$Z_1 \sqcup Z_2 = \min\{Z \in \mathcal{Z}_C \,|\, (Z_1 \subseteq Z) \wedge (Z_2 \subseteq Z)\}$$

which can be used as an over-approximation of $Z_1 \cup Z_2$. The *forward projection* $Z^{\nearrow} = \{v \in \mathcal{V}_C \,|\, \exists d \geq 0,\ v - d \in Z\}$ consists of all clock valuations that can result from applying arbitrary time progress to elements of Z. The projection of Z on $C' \subseteq C$ is $Z/_{C'} = \{v/_{C'} \,|\, v \in Z\}$, and $\gamma(Z) = \{\gamma(v) \,|\, v \in Z\}$ is the application of clock assignment γ to all elements of Z. In particular $r_{C'}(Z)$ is the *resetting* of the clocks in C' and $kill_{C'}(Z)$ is the *deactivation* of clocks in C'. Note that $kill_{C'}(Z) = Z/_{C \setminus C'}$. The difference between $r_{C'}(Z)$ and $kill_{C'}(Z)$ is that applying $kill_{C'}$ may reduce the dimensionality of the *space* on which the zone is defined while applying $r_{C'}$ may reduce the dimensionality of the *set* but *not* of the space on which it is defined,and after some time progress the set will regain its reduced dimensions. These operation are computed efficiently on a DBM representation of the zones. More details can be found, for example, in [27].

2.2 Timed Automata and their Analysis

DEFINITION 1 (TIMED AUTOMATON). *A timed automaton is a tuple $\mathcal{A} = (Q, q_0, C, I, \Delta)$ where Q is a finite set of discrete states, $q_0 \in Q$ is the initial state, C is a finite set of clocks, $I : Q \to \Psi_C$ associates a staying condition (invariant) with every state q such that the automaton may stay at q as long as the clock constraint $I(q)$ is satisfied, $\Delta \subseteq Q \times \Psi_C \times \Gamma_C \times Q$ is the transition relation consisting of elements of the form $\delta = (q, g, \gamma, q')$ where $q, q' \in Q$ are, respectively, the source and the target of the transition, $g \in \Psi_C$ is the transition guard restricting the execution of the transition to clock valuations that satisfy it and $\gamma \in \Gamma_C$ is a clock assignment occurring upon a transition.*

Note that in this definition we do not have a specific alphabet of transition labels. We view each transition as a *distinct* type of event and postpone transition labeling to the next section where we discuss state variables and composition. Timed automata define infinite transition systems whose states are *configurations* of the form $(q, v) \in Q \times \mathcal{V}_C$. The initial configuration is $(q_0, \perp)$ with all clocks inactive and the transitions are either discrete transitions of the automaton or time-passage transitions, formalized via the notion of a step.

DEFINITION 2 (STEPS). *A step of $\mathcal{A}$ is one of the following:*

- *A* discrete step*: $(q, v) \xrightarrow{\delta} (q', v')$, for some $\delta = (q, g, \gamma, q') \in \Delta$ such that $v \models g$ and $v' = \gamma(v)$,*
- *A* time step*: $(q, v) \xrightarrow{d} (q, v + d)$ for some $d \in \mathbb{R}_+$ such that $v + d \models I(q)$.*

Note that the concatenation of two time steps of durations d_1 and d_2 is a time step of duration $d_1 + d_2$. A *compound step* is a discrete step followed by a time step (possibly of zero duration):

$$(q, v) \xrightarrow{\delta, d} (q', v' + d) \equiv (q, v) \xrightarrow{\delta} (q', v') \xrightarrow{d} (q', v' + d)$$

A *run* of the automaton $\mathcal{A}$ starting from a configuration (q, v) is a sequence of compound steps.We use the notation $(q, v) \xrightarrow{\xi} (q', v')$ for runs. The untiming of ξ, denoted by $\mu(\xi)$, is the sequence of

[4]Unlike [13] where clock activity analysis was used to reduce dimensionality.

transitions taken, regardless of the time durations. The semantics of $\mathcal{A}$, denoted by $[\![\mathcal{A}]\!]$, is the set of runs it may generate from the initial state and its qualitative semantics is $\mu([\![\mathcal{A}]\!]) = \{\mu(\xi) | \xi \in [\![\mathcal{A}]\!]\}$.

2.3 From Timed Automata to Reachability Graphs

The verification of this infinite-state system is done symbolically by computing the *reachability graph*, also known as the *simulation graph*, whose nodes are *symbolic states* of the form $(q, Z) \in Q \times \mathcal{Z}_C$ which are closed under the following operations:

- The *time successor* of (q, Z) is (q, Z') where Z' is the set of clock valuations reachable from Z by letting time progress without violating the staying condition $I(q)$:

$$post^t(q, Z) = (q, (Z^{\nearrow} \cap I(q))) \\ = \{(q, v + d) \,|\, (v \in Z) \wedge (d \geq 0) \wedge ((v + d) \models I(q))\}$$

- The *δ-transition successor* of (q, Z) via a transition $\delta = (q, g, \gamma, q') \in \Delta$ is the set of configurations reachable by taking this transition.

$$post^\delta(q, Z) = \{(q', v') \,|\, \exists v \in Z,\, v \models g \wedge v' = \gamma(v)\} \\ = (q', (\gamma(Z \cap g)))$$

- The *δ-successor* of (q, Z) is the set of configurations reachable from (q, Z) by a δ-transition followed by a time step:

$$succ^\delta(q, Z) = post^t(post^\delta(q, Z)) = (q', (\gamma(Z \cap g)^{\nearrow} \cap I(q'))$$

PROPOSITION 1 (SUCCESSORS). *A configuration (q', v') belongs to $(q', Z') = succ^\delta(q, Z)$ if and only if it is the endpoint of a compound step $(q, v) \xrightarrow{\delta, d} (q', v')$ for some $(q, v) \in (q, Z)$ and some $d \geq 0$.*[5]

The analysis of timed automata is done via a *forward reachability* algorithm which starts with $(q_0, \{\perp\})$ and computes successors until a fixed point is reached, a fact guaranteed by the finiteness of $\mathcal{Z}_C$ in any bounded subset of $\mathbb{R}^n_\perp$. As a byproduct, the algorithm produces the *reachability graph* which can be viewed as the finite state automaton defined below.

DEFINITION 3 (REACHABILITY GRAPH). *The reachability graph associated with a timed automaton $\mathcal{A} = (Q, q_0, C, I, \Delta)$ is a finite automaton $\mathcal{G} = (\Delta, S, s_0, h)$ such that S is the smallest set of symbolic states containing the initial state $s_0 = (q_0, \{\perp\})$ and closed under $\{succ^\delta\}_{\delta \in \Delta}$. The transition relation $h \subseteq S \times \Delta \times S$ consists of all triples of the form $((q, Z), \delta, (q', Z'))$ such that $(q', Z') = succ^\delta(q, Z)$.*

The fundamental property of the reachability graph is that it preserves the qualitative semantics in the sense that it may generate a sequence of transitions $w \in \Delta^*$ iff $w \in \mu([\![\mathcal{A}]\!])$. This implies that any (untimed) linear-time property verified on $\mathcal{G}$ holds also for $\mathcal{A}$. Note that this is *not true* for the automaton obtained by taking $\mathcal{A}$ and removing its clocks. The reachability process excludes behaviors that cannot be realized due to timing constraints and *only after that* it is safe to remove the clocks. Our technique is inspired by this insight, and provides different levels of relaxation of the timing constraints after reachability. To this end we define the *interpreted timed automaton* which enjoys the properties of the reachability graph, but preserves also the quantitative timing information and is semantically equivalent to the timed automaton $\mathcal{A}$ from which it was derived.

DEFINITION 4 (INTERPRETED TIMED AUTOMATON). *Let $\mathcal{A} = (Q, q_0, C, I, \Delta)$ be a timed automaton and let $\mathcal{G} = (\Delta, S, s_0, h)$ be its reachability graph. The interpreted timed automaton for $\mathcal{A}$ is $\mathcal{A}^r = (S, s_0, C, I^r, \Delta^r)$ such that $I^r(q, Z) = Z$ and for every transition $((q, Z), \delta, (q', Z')) \in h$, corresponding to a transition $\delta = (q, g, \gamma, q') \in \Delta$ we define a transition $((q, Z), g \cap Z, \gamma, (q', Z')) \in \Delta^r$.*

Clearly, the semantics of $\mathcal{A}$ and of $\mathcal{A}^r$ are identical and even if we later relax the timing constraints in $\mathcal{A}^r$ and introduce more behaviors, we do not introduce any new *qualitative* behavior.

3. TIMED AUTOMATA WITH STATE VARIABLES

The timed automaton model presented so far was based on a *flat* state space, without mentioning state variables nor a Cartesian product structure. For the compositional treatment of networks of automata we need a richer formalism that will facilitate the discussion of composition, hiding of internal actions and the like. To this end we associate with the automata state variables so that each transition is associated in a change in the valuation of a variable and the semantics consists of the evolution of these valuations over time.

3.1 Variables, Valuations and Assignments

Let $X = \{x_1, \ldots, x_m\}$ be a finite set of variables ranging over a domain $\mathbb{D}$. An X-valuation is a function $\mathbf{v} : X \to \mathbb{D}$ assigning to each x_i a value $\mathbf{v}(x_i)$. The set of X-valuations, which are just elements of $\mathbb{D}^m$, is denoted by $\mathbf{V}_X$. Every $X' \subseteq X$ defines a *projection* function $_{/X'} : \mathbf{V}_X \to \mathbf{V}_{X'}$ such that $\mathbf{v}' = \mathbf{v}_{/X'}$ if $\mathbf{v}'(x) = \mathbf{v}(x)$ for every $x \in X'$. When states are associated with X-valuations, transitions correspond to changes in one or more state variables, formalized as *assignments*.

DEFINITION 5 (ASSIGNMENTS). *Let $\mathbf{v}$ and $\mathbf{v}'$ be two valuations. The function $f_\mathbf{v} : \mathbf{V}_X \to \mathbf{V}_X$ is the constant function defined for every $\mathbf{u} \in \mathbf{V}_X$ as $f_\mathbf{v}(\mathbf{u}) = \mathbf{v}$. The function $f_{\mathbf{v},\mathbf{v}'} : \mathbf{V}_X \to \mathbf{V}_X$ is such that for every $\mathbf{u}, \mathbf{u}' \in \mathbf{V}_X$, $\mathbf{u}' = f_{\mathbf{v},\mathbf{v}'}(\mathbf{u})$ when*

$$\mathbf{u}'(x) = \begin{cases} \mathbf{v}'(x) & \text{if } \mathbf{v}(x) \neq \mathbf{v}'(x) \\ \mathbf{u}(x) & \text{if } \mathbf{v}(x) = \mathbf{v}'(x) \end{cases}$$

In other words, $f_{\mathbf{v},\mathbf{v}'}$ acts as the identity on all the variables on which $\mathbf{v}$ and $\mathbf{v}'$ agree and assigns to the other variables their value in $\mathbf{v}'$. Needless to say, $\mathbf{v}' = f_{\mathbf{v},\mathbf{v}'}(\mathbf{v})$. In the sequel we restrict ourselves to the Boolean domain and use the notation $x^\downarrow$ for $x := 0$ and $x^\uparrow$ for $x := 1$. For example if $\mathbf{v} = (0, 1, 1, 0)$ and $\mathbf{v}' = (1, 1, 0, 0)$, then $f_{\mathbf{v}'} = \{{x_1}^\uparrow, {x_2}^\uparrow, {x_3}^\downarrow, {x_4}^\downarrow\}$, and $f_{\mathbf{v},\mathbf{v}'} = \{{x_1}^\uparrow, {x_3}^\downarrow\}$. We denote the set of X-assignments by F_X. The projection $f_{/\mathbf{v}'}$ of an assignment f on a subset X' of the variables is defined naturally via restriction. When valuations and assignments are associated with states and transitions we need to check, when composing automata, the compatibility between valuations over two non-disjoint sets of variables.

DEFINITION 6 (COMPATIBILITY). *Let X^1 and X^2 be two sets of variables and let $X = X^1 \cup X^2$ and $\underline{X} = X^1 \cap X^2$. Two valuations $\mathbf{v}^1 \in \mathbf{V}_{X^1}$ and $\mathbf{v}^2 \in \mathbf{V}_{X^2}$ are said to be compatible if they agree on shared variables, that is, $\mathbf{v}^1_{/\underline{X}} = \mathbf{v}^2_{/\underline{X}}$. From two such*

[5]Note, however, that not all elements of (q, Z) are the start points of such compound steps because they do not necessarily satisfy g.

compatible valuation one can naturally construct a joint valuation $\mathbf{v}^1 \parallel \mathbf{v}^2 : X \to \mathbf{V}_X$ *which agrees with* $\mathbf{v}^1$ *and* $\mathbf{v}^2$ *on all variables. Likewise, two assignments* $f^1 \in F_{X^1}$ *and* $f^2 \in F_{X^2}$ *are compatible if* $f^1_{/\underline{X}} = f^2_{/\underline{X}}$ *and one can construct from them a joint assignment* $f^1 \parallel f^2 : \mathbf{V}_X \to \mathbf{V}_X$.

3.2 Signals

As timed behaviors of a system defined over X we consider signals, functions from the time domain $\mathbb{R}_+$ to $\mathbf{V}_X$.[6]

DEFINITION 7 (X-VALUED SIGNALS). *A signal over a set* X *of variables ranging over a discrete domain is a partial function* $\xi : \mathbb{R}_+ \to \mathbf{V}_X$ *defined over some interval* $[0, t)$, $t \in \mathbb{R}_+ \cup \{\infty\}$, *which can be partitioned into a countable sequence of left-closed right-open intervals* $\mathcal{J} = J_1, J_2, J_3, \ldots$ *each of the form* $J_k = [t_{k-1}, t_k)$, *such that* $\xi(t) = \xi(t') = \xi(J)$ *if* t *and* t' *belong to the same interval* J.

We denote the set of finite and infinite X-signals by $\mathcal{S}^*(X)$ and $\mathcal{S}^\omega(X)$, and let $\mathcal{S}(X) = \mathcal{S}^*(X) \cup \mathcal{S}^\omega(X)$. The *concatenation* operation $\xi_1 \cdot \xi_2$ is defined naturally for any $\xi_1 \in \mathcal{S}^*(X)$, $\xi_2 \in \mathcal{S}(X)$, as it is defined for sequences. Signals can be represented in a state-based fashion, as a concatenation of constant signals where $\mathbf{v}_1^{r_1} \cdot \mathbf{v}_2^{r_2} \cdot \mathbf{v}_3^{r_3} \cdots$ representing a signal which has value $\mathbf{v}_1$ for duration r_1, then value $\mathbf{v}_2$ for duration r_2, etc. or in an event-based fashion as a concatenation of time durations and assignments: $f_{\mathbf{v}_1} \cdot r_1 \cdot f_{\mathbf{v}_1,\mathbf{v}_2} \cdot r_2 \cdot f_{\mathbf{v}_2,\mathbf{v}_3} \cdot r_3 \cdots$, which is the representation we will use. For example, signal which in state-based representation looks like

$$\xi = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}^2 \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}^5 \cdot \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}^3 \cdot \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}^7 \cdots,$$

will be written in an event-based form as:

$$\xi = \{x_1{}^{\uparrow}, x_2{}^{\downarrow}, x_3{}^{\downarrow}\} \cdot 2 \cdot \{x_1{}^{\downarrow}, x_2{}^{\uparrow}\} \cdot 5 \cdot \{x_2{}^{\downarrow}\} \cdot 3 \cdot \{x_1{}^{\uparrow}, x_3{}^{\uparrow}\} \cdot 7 \cdots$$

The *projection* $\xi' = \xi_{/X'}$ of an X-signal ξ on some $X' \subseteq X$ is defined naturally by deleting variables outside X' from the assignments. This operation is a natural part of the process of hiding internal actions and may results in eliminating certain discrete steps and merging time steps. The projection of ξ on $\{x_1, x_3\}$, is represented as

$$\xi' = \{x_1{}^{\uparrow}, x_3{}^{\downarrow}\} \cdot 2 \cdot \{x_1{}^{\downarrow}\} \cdot 5 \cdot \{\} \cdot 3 \cdot \{x_1{}^{\uparrow}, x_3{}^{\uparrow}\} \cdot 7 \cdots,$$

and since the assignment which involved only a change in x_2 became silent we can merge two time steps into one and obtain

$$\xi' = \{x_1{}^{\uparrow}, x_3{}^{\downarrow}\} \cdot 2 \cdot \{x_1{}^{\downarrow}\} \cdot 8 \cdot \{x_1{}^{\uparrow}, x_3{}^{\uparrow}\} \cdot 7 \cdots$$

The *untiming* of a signal is the sequence $\mu(\xi) = f_{\mathbf{v}_1} \cdot f_{\mathbf{v}_1,\mathbf{v}_2} \cdot f_{\mathbf{v}_2,\mathbf{v}_3} \cdots$, that is, the *qualitative behavior* described previously.

3.3 Timed Automata with Variables

We associate every timed automaton with a set X of variables, partitioned into disjoint subsets of *input* and *state* variables $X = X_{in} \uplus X_{st}$. The difference between them is that those in X_{in} are controlled by the *external environment* of the automaton which follows their evolution passively, reacting to changes in their values. On the other hand, variables in X_{st} are owned *exclusively* by the automaton which controls their values. A subset $X_{ou} \subseteq X_{st}$ of the state variables, called *output* variables, are observable to the outside and may serve as inputs for other components. The *interface* variables of the automaton are its input and output variables $X_{io} = X_{in} \uplus X_{ou}$.

The connection between states and valuations is defined via the function $\lambda : Q \to \mathbf{V}_X$ associating with every state an X-valuation $\mathbf{v} = \lambda(q)$. The projections of λ on the interface variables is denoted by λ_{io}. Two points are worth mentioning: 1) we allow multiple states to be mapped to the *same* valuation to accommodate for future use in the interpreted timed automaton which may have several symbolic states corresponding to the same discrete state; 2) We consider the values of input variables to be part of the state, hence *every change* in the input will cause a transition to a different state, and there will be only one input valuation associated with each state. This amounts to composing every automaton with an unconstrained generator of its inputs. At a first glance this looks like an unnecessary blow up of the state space, but, when analyzed, the automaton should be composed anyhow with a generator of its inputs, so this overhead is an illusion.

DEFINITION 8 (TIMED AUTOMATA OVER VARIABLES). *A timed automaton over a set* X *of variables is a triple* $\mathcal{A}_X = (\mathcal{A}, X, \lambda)$ *where* $\mathcal{A} = (Q, q_0, C, I, \Delta)$ *is a timed automaton,* X *is a set of variables, and* $\lambda : Q \to \mathbf{V}_X$ *maps states to* X*-valuations. This function induces a mapping between transition labels and* X*-assignments where every transition* $\delta = (q, g, \gamma, q')$ *such that* $\lambda(q) = \mathbf{v}$ *and* $\lambda(q') = \mathbf{v}'$ *is labeled by* $\lambda(\delta) = f_{v,v'}$.

We impose some additional properties that hold, for example, for the automata introduced in [20] for modeling Boolean circuits with delays, on which our example in Section 5 is based, and other automata obtained by well-structured composition of timed components.

1. Every transition in Δ changes the value of *exactly one* variable. Consequently we can partition Δ into sets of *input* and *state* transitions $\Delta = \Delta_{in} \uplus \Delta_{st}$.

2. The automata are input-enabled (or receptive) in the sense that every change of an input variable is possible in every state and all transitions in Δ_{in} are guarded by $true$.

3. All transitions in Δ_{st} are guarded by clock constraints of the form $c \in [l, u]$ involving only a *single* clock.

The *semantics* of $\mathcal{A}_X$ is defined in terms of *steps* and *runs*, inherited from the definition of $\mathcal{A}$, and in terms of the signals *carried* by these runs. With each compound step $(q, v) \xrightarrow{\delta,t} (q', v')$ such that $\lambda(q) = \mathbf{v}$ and $\lambda(q) = \mathbf{v}'$, we associate the X-signal $f_{\mathbf{v},\mathbf{v}'} \cdot t$. The signal carried by a run is the concatenation of the signals carried by its steps. We denote by $[\![\mathcal{A}]\!] \subseteq \mathcal{S}(X)$ the set of signals carried by all runs of $\mathcal{A}$ and use $[\![\mathcal{A}]\!]_{in}$, $[\![\mathcal{A}]\!]_{ou}$ and $[\![\mathcal{A}]\!]_{io}$ for the sets of their projections on the respective sets of variables. In particular, $[\![\mathcal{A}]\!]_{io} \subseteq \mathcal{S}(X_{io})$ is the *observable behavior* of $\mathcal{A}$. Likewise we will use $\mu([\![\mathcal{A}]\!])$ and $\mu([\![\mathcal{A}]\!]_{io})$ for the untiming of those signal languages.

Given two timed automata $\mathcal{A}$ and $\mathcal{A}'$ having the same set X_{io} of interface variables, we say that $\mathcal{A}'$ is an *over approximation* of $\mathcal{A}$, denoted by $\mathcal{A} \sqsubseteq \mathcal{A}'$, if $[\![\mathcal{A}]\!]_{io} \subseteq [\![\mathcal{A}']\!]_{io}$. This will always be the case when $\mathcal{A}'$ is obtained from $\mathcal{A}$ by relaxing some of the timing constraints, or by merging two or more discrete states.

[6] We use signals in this paper because our case study is a circuit for which signals are more appropriate. More theoretical background on the relation between signals and the alternative semantic domain for timed systems, *time-event sequences* consisting of *events* separated by time durations (which are also equivalent to the *timed traces* of [3]) can be found in [4].

3.4 Composition

We now define the composition of two automata with variable sets X^1 and X^2 such that $X^1_{st} \cap X^2_{st} = \emptyset$ where interaction between components is realized via the *shared variables*

$$\underline{X} = X^1 \cap X^2 = (X^1_{in} \cap X^2_{in}) \uplus (X^1_{ou} \cap X^2_{in}) \uplus (X^2_{ou} \cap X^1_{in}).$$

When one automaton takes a transition that changes the value of a shared variable, the other automaton must take a similar transition as well. If the variable is an input for both it remains an input of the composed system. On the other hand, a variable which is an output variable of one automaton and an input for the other, becomes a state variable of the composed system.

A composition is *cyclic* is there are loops of influence between components: $(X^1_{out} \cap X^2_{in} \neq \emptyset) \wedge (X^2_{out} \cap X^1_{in} \neq \emptyset)$. Cyclic composition may introduce, in certain circumstances, unstable behaviors consisting of spontaneous oscillations between values without any external stimulus. Our implementation uses the algorithm of [21] to check stability of cyclic compositions, but to simplify the presentation we restrict ourselves to acyclic composition.

DEFINITION 9 (COMPOSITION: VARIABLE CLASSIFICATION). *The set of variables in a composition of two timed automata over variable sets X^1 and X^2, is $X = X^1 \cup X^2$, where $X_{st} = X^1_{st} \uplus X^2_{st}$, $X_{in} = (X^1_{in} - X^2_{ou}) \cup (X^2_{in} - X^1_{ou})$, and $X_{ou} \subseteq X^1_{ou} \uplus X^2_{ou}$.*

Note that the definition of X_{ou} is not unique, permitting any subset of $X^1_{ou} \uplus X^2_{ou}$ to reflect the liberty in deciding which variables in a component are to be exposed to the external world. We do not include this choice of X_{ou} in the formal definition of the composition in order to keep it associative. When we apply it in practice we compose all the automata together and specify explicitly the set of output variables of the product. The following definition identifies global states of the product as pairs of states whose valuations are compatible.

DEFINITION 10 (CONSISTENT GLOBAL STATES). *Let $\mathcal{A}_{X^1} = (\mathcal{A}^1, X^1, \lambda^1)$ and $\mathcal{A}_{X^2} = (\mathcal{A}^2, X^2, \lambda^2)$ be two timed automata. We say that a pair of states $(q^1, q^2) \in Q^1 \times Q^2$ is consistent if $\lambda^1(q^1)_{/\underline{X}} = \lambda^2(q^2)_{/\underline{X}}$. We denote the set of consistent global states by $\overline{Q^1 \times Q^2}$.*

DEFINITION 11 (COMPOSITION OF TIMED AUTOMATA). *Let $\mathcal{A}_{X^1} = (\mathcal{A}^1, X^1, \lambda^1)$ with $\mathcal{A}^1 = (Q^1, q^1_0, C^1, I^1, \Delta^1)$ and $\mathcal{A}_{X^2} = (\mathcal{A}^2, X^2, \lambda^2)$ with $\mathcal{A}^2 = (Q^2, q^2_0, C^2, I^2, \Delta^2)$, be two timed automata with variables, such that (q^1_0, q^2_0) is consistent. Their composition is $\mathcal{A}_{X^1} \parallel \mathcal{A}_{X^2} = \mathcal{A}_X = (\mathcal{A}, X, \lambda)$ with $\mathcal{A} = (Q, q_0, C, I, \Delta)$ where $X = X^1 \cup X^2$, classified according to Definition 9, $Q = \overline{Q^1 \times Q^2}$, $q_0 = (q^1_0, q^2_0)$, $C = C^1 \uplus C^2$, $I((q^1, q^2)) = I^1(q^1) \wedge I^2(q^2))$ for every (q^1, q^2) and the transition relation consists of all transitions of the form $\delta = ((q^1, q^2), g, \gamma, (q'^1, q'^2))$ for which one of the following holds:*

1. *$q'^2 = q^2$ and there is a transition $\delta = (q^1, g, \gamma, q'^1) \in \Delta^1$ such that $\lambda^1(\delta) = f$ and $f_{/\underline{X}}$ is the identity function*
2. *$q'^1 = q^1$ and there is a transition $\delta = (q^2, g, \gamma, q'^2) \in \Delta^2$ such that $\lambda^2(\delta) = f$ and $f_{/\underline{X}}$ is the identity function*
3. *there are two transitions $\delta^1 = (q^1, g^1, \gamma^1, q'^1) \in \Delta^1$ and $\delta_2 = (q^2, g^2, \gamma^2, q'^2) \in \Delta^2$ such that $\lambda^1(\delta^1) = f^1$, $\lambda^2(\delta^2) = f^2$, $f^1_{/\underline{X}} = f^2_{/\underline{X}}$ is not the identity function, $g = g^1 \wedge g^2$ and $\gamma = \gamma^1 \circ \gamma^2$*

The mapping $\lambda : Q \to X$ is defined for every global state as $\lambda((q^1, q^2)) = \lambda^1(q^1) \parallel \lambda^2(q^2)$.

The first two cases in the definition of Δ correspond to local transitions of $\mathcal{A}^1$ and $\mathcal{A}^2$ that do not touch shared variables and to which we apply the interleaving semantics. The third case corresponds to two transitions, one in each automaton, whose respective assignments change the values of a shared variable in a consistent way. It is not hard to see that state consistency as well as properties 1-3 above are preserved under composition. We mention a special class of timed automata which is closed under acyclic composition.

DEFINITION 12 (INPUT-DEPENDENT TIMED AUTOMATA). *A timed automaton $\mathcal{A}$ is input dependent if every non-trivial cycle in its transition graph involves at least one transition which changes the value of some input variable.*

PROPOSITION 2 (CLOSURE UNDER ACYCLIC COMPOSITION). *If both $\mathcal{A}_{X^1}$ and $\mathcal{A}_{X^2}$ are input-dependent, so is their acyclic composition $\mathcal{A} = \mathcal{A}_{X^1} \parallel \mathcal{A}_{X^2}$.*

PROOF. Suppose, without loss of generality, that $X^1_{ou} \cap X^2_{in} \neq \emptyset$. Suppose $\mathcal{A}$ does have a non-trivial cycle without change in its input. Then, at least one of its projections on $\mathcal{A}^1$ or $\mathcal{A}^2$ must be a non-trivial cycle. Such a cycle in $\mathcal{A}^1$ will contradict the fact that $\mathcal{A}^1$ is input-dependent. Otherwise, the projection on $\mathcal{A}^1$ is a trivial cycle without a change in its output, and the induced cycle in $\mathcal{A}^2$ contradicts the fact that $\mathcal{A}^2$ is input-dependent. ∎ ∎

An important property of input-dependent automata is that there is a constant k such that the number of output events is at most k times the number of input events.

4. THE ABSTRACTION TECHNIQUE

At this point we can describe our abstraction procedure, starting with the automaton $\mathcal{A}_X$ modeling a network of timed components and applying the following sequence of steps

$$\mathcal{A}_X \Rightarrow \mathcal{A}^{+\hat{C}}_X \Rightarrow \mathcal{A}^r_X \Rightarrow \mathcal{A}^{\hat{C}}_X \Rightarrow \mathcal{A}_{X_{io}} \Rightarrow \mathcal{A}^m_{X_{io}}.$$

1. From $\mathcal{A}_X$ we construct $\mathcal{A}^{+\hat{C}}_X$ by adding auxiliary input clocks that do not participate in transition guards or invariants, but only *observe* the dynamics of the automaton and measure the time elapsed since each input event. An input clock is discarded after a finite amount of time when the chain of reactions triggered by its event terminates.

2. We apply the forward reachability algorithm to $\mathcal{A}^{+\hat{C}}_X$ to obtain the interpreted timed automaton $\mathcal{A}^r_X$ having the same semantics as $\mathcal{A}_X$. It is however, annotated with additional (redundant) timing constraints involving clocks in $\hat{C}$.

3. We relax the timing constraints of $\mathcal{A}^r_X$ by *projecting* them on clocks in $\hat{C}$ to obtain $\mathcal{A}^{\hat{C}}_X$. To be more precise, each transition guard is projected on the clock associated with the input event that has triggered it.

4. We project $\mathcal{A}^{\hat{C}}_X$ on the *interface* variables to obtain $\mathcal{A}_{X_{io}}$ thus making some internal transitions *silent*.

5. We then reduce the discrete state space of $\mathcal{A}_{X_{io}}$ by merging states which are equivalent in terms of the untimed behaviors they admit and after merging transitions guards we obtain the reduced automaton $\mathcal{A}^m_{X_{io}}$.

At the end of the process we have the following relationships between the observable semantics of these automata:

$$[\![\mathcal{A}_X]\!]_{io} = [\![\mathcal{A}_X^{+\hat{C}}]\!]_{io} = [\![\mathcal{A}_X^r]\!]_{io} \subseteq [\![\mathcal{A}_X^{\hat{C}}]\!]_{io} = [\![\mathcal{A}_{X_{io}}]\!] \subseteq [\![\mathcal{A}_{X_{io}}^m]\!]$$

and the following relation between the qualitative semantics

$$\mu([\![\mathcal{A}_X]\!]_{io}) = \mu([\![\mathcal{A}_X^{+\hat{C}}]\!]_{io}) = \mu([\![\mathcal{A}_X^r]\!]_{io}) = \mu([\![\mathcal{A}_X^{\hat{C}}]\!]_{io}) = \mu([\![\mathcal{A}_{X_{io}}]\!]) = \mu([\![\mathcal{A}_{X_{io}}^m]\!]).$$

In other words, we preserve the qualitative semantics of the automaton and relax, to some extent, its timing constraints.

4.1 Adding Input Clocks

The most original and hard to implement part of our procedure is the construction of $\mathcal{A}^{+\hat{C}}$ which requires monitoring the propagation of input events in an acyclic network of timed components. When an input event occurs it excites one or more of the components to which it is a direct input. Each of those components will react within some $t \in [l, u]$ and emit an output event, 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 that 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 serially-connected components, and u_* is the maximal delay upper bound of the components.

For an input event to be alive in the system there must be at least one active clock triggered by it, and since each clock is reset by one event, the number of live events is bounded by the number of clocks in the system which is equal to the number of timed components. This is an upper bound and in practice the maximal number of live events can be much smaller due to logical interference or additional bounded-variability assumptions concerning the external environment. In the sequel we assume m to be the upper bound on the number of live input events for each input variable.

For every transition $\delta \in \Delta$ we let $\chi(\delta)$ be the set of clocks reset to zero by the transition. We say that a transition is *exciting* if $\chi(\delta) \neq \emptyset$. An exciting transition is a transition which triggers processes that will eventually be concluded by transitions guarded by clocks in $\chi(\delta)$. We will augment every discrete state of the automaton with additional machinery that keeps track of the events which are alive in this state, and relates pending changes of states (represented by active clocks) to these events. Let $M = \{0, \ldots, m\}$ where m is the maximal number of live events in the automaton. The additional structure consists of:

- An *event-recording table* (ℓ, θ) consisting of
  - A *live-event counter* $\ell : X_{in} \to M$ indicating for each input variable x how many of its events are alive.
  - An *association function* $\theta : C \to (X_{in} \times M) \cup \{\bot\}$ relating each active clock to the input event responsible for its activation. Event (x, i) is understood to be the i^{th}-oldest x-event still alive in the system.

  We denote the (finite) set of event-recording tables by $\mathcal{E}$.
- A set of *auxiliary input clocks*: $\hat{C} = \{c_x[i] \mid x \in X_{in}, i \in M\}$ with the intended meaning that $c_x[i]$ is the time elapsed since the occurrence of event (x, i). A valuation of these clocks is a function $\hat{v} : \hat{C} \to \mathbb{R}_\bot$. We will refer to the original clocks of the components as *internal* clocks.

Computing $\Delta^{+\hat{C}}$

Input: An extended state $(q, \ell, \theta) \in Q^{+\hat{C}}$
and a transition $\delta = (q, g, \gamma, q') \in \Delta$

Output: A transition $\delta^{+\hat{C}} = ((q, \ell, \theta), g, \hat{\gamma}, (q', \ell', \theta')) \in \Delta^{+\hat{C}}$

$\gamma' := \gamma$; $\ell' := \ell$; $\theta' := \theta$

if $\chi(\delta) \neq \emptyset$

  if $\delta \in \Delta_{in}$ changing input variable x /* new event creation */

    $\ell'(x) := \ell'(x) + 1$

    $e := (x, \ell'(x))$

    $\gamma' := \gamma' \cup \{c_x[\ell(x)] := 0\}$

  elsif $\delta \in \Delta_{st}$ guarded by clock c /* event propagation */

    $e := \theta(c)$

  for each $c' \in \chi(\delta)$

    $\theta'(c') := e$

for every non-resetting clock assignment in γ /* book keeping */

  if of the form $c := \bot$

    $\theta'(c) := \bot$

  elsif of the form $c_1 := c_2$

    $\theta'(c_1) := \theta(c_2)$

for each $x \in X_{in}$ /* clock killing and shifting */

  for $i = 1$ to $\ell'(x)$

    if $\theta'^{-1}(x, i) = \emptyset$

      for $j = i$ to $\ell'(x) - 1$

        $\gamma' := \gamma' \cup \{c_x[j] := c_x[j + 1]\}$

        for each $c \in \theta'^{-1}(x, j + 1)$

          $\theta'(c) := (x, j)$

      $\ell'(x) := \ell'(x) - 1$

Table 1: The algorithm for computing $\Delta^{+\hat{C}}$.

We can now proceed to the construction of the automaton. Since this construction does not affect the mapping of states to variable valuations we define it in terms of $\mathcal{A}$ from which we construct $\mathcal{A}^{+\hat{C}} = (Q^{+\hat{C}}, q_0^{+\hat{C}}, C \cup \hat{C}, I^{+\hat{C}}, \Delta^{+\hat{C}})$ where $Q^{+\hat{C}} = Q \times \mathcal{E}$ and $q_0^{+\hat{C}} = (q_0, \ell_0, \theta_0)$ where $\ell_0(x) = 0$ for every x, all clocks in $\hat{C}$ are inactive and $\theta_0(c) = \bot$ for every $c \in C$.

The transition relation $\Delta^{+\hat{C}}$ consists of transitions of the form $\delta' = ((q, \ell, \theta), g, \hat{\gamma}, (q', \ell', \theta'))$ based on transitions of the form $\delta = (q, g, \gamma, q') \in \Delta$ where $\hat{\gamma}$ is a clock assignment on $C \cup \hat{C}$ whose projection on C is γ. The updated event recoding table (ℓ', θ') and the extended assignment $\hat{\gamma}$ are computed from (ℓ, θ), g and γ according to the algorithm in Table 1. This procedure has four major parts.

1. Initialization: γ' inherits the clock assignments for the internal clocks from γ and the initial event-recording table is the same as in the source state.
2. Treatment of excitation: when a transition triggers a new internal processes by resetting clocks, these clocks should be associated with the input event responsible for their excitation. There are two cases:
   (a) New event generation: the transition is due to a new input event on x and in this case we increment the x event counter, initialize a new input clock and keep the responsible event in a temporary variable e.
   (b) Event propagation: the exciting transition is not related to a new input event, and in this case the responsible

input event is inherited from the clock c guarding the transition.

Then every clock reset by the transition is associated with the input event in e.

3. Association book keeping: when clocks become inactive or when they are shifted, their association is updated.

4. Event death detection: if no clock points via θ to event (x,i) the event is no longer alive and we discard clock $c_x[i]$. To keep the number of clocks finite we "shift" clocks $c_x[i+1],\ldots,c_x[\ell(x)]$ to the left and modify the association function accordingly. We use $kill(x,i)$ to denote this operation.

This procedure maintains the event-detection table well-formed:

1. Only active internal clocks are associated with input events: $\theta(c)\neq\perp$ iff $v(c)\neq\perp$.

2. These clocks are associated only with live events: $\theta(c)=(x,i)$ only if $i\leq\ell(x)$.

3. Each live event has at least one internal clock associated with it: $i\leq\ell(x)$ only if $\exists c\,\theta(c)=(x,i)$.

4. Only input clocks associated with live events are active: $\hat{v}(c_x[i])\neq\perp$ iff $i\leq\ell(x)$.

4.2 Reachability Computation and Projection

Automaton $\mathcal{A}^{+\hat{C}}$ is an ordinary timed automaton except for the fact that the denotation of its input clocks may vary from one state to another due to clock shifting. Its configurations are of the form $((q,\ell,\theta),(v,\hat{v}))$ where $(v,\hat{v})$ is a joint valuation of $C\cup\hat{C}$. After reachability computation we obtain the interpreted timed automaton whose states are of the form $((q,\ell,\theta),Z))$ where Z is a zone over $C\cup\hat{C}$. As described in Section 2, we intersect invariants and transition guards with these zones to obtain the interpreted timed automaton $\mathcal{A}^r_X=(\mathcal{A}^r,X,\lambda)$ with $\mathcal{A}^r=(Q^r,X,q^r_0,C\cup\hat{C},I^r,\Delta^r)$.

The automaton $\mathcal{A}^{\hat{C}}=(Q^r,q^r_0,\hat{C},I^{\hat{C}},\Delta^{\hat{C}})$ is constructed from $\mathcal{A}^r$ by projecting the timing constraints on clocks $\hat{C}$. For each state $p=((q,\ell,\theta),Z)\in Q^r$ let $\hat{C}(p)$ be the set of input clocks active at p. The invariant of p in $\mathcal{A}^{\hat{C}}$ is $I^{\hat{C}}(p)=I^r(p)/_{\hat{C}(p)}$. For every transition $\delta=(p,g,\gamma,p')\in\Delta^r$ we define a transition $\delta^{\hat{C}}=(p,g^{\hat{C}},\gamma^{\hat{C}},p')\in\Delta^{\hat{C}}$ by letting $\gamma^{\hat{C}}=\gamma/_{\hat{C}}$ and

$$g^{\hat{C}}=\begin{cases}\gamma/_{\hat{C}} & \text{if } \delta\in\Delta^r_{in}\\ \gamma/_{\theta(c)} & \text{if } \delta\in\Delta^r_{in} \text{ and is guarded by } c\end{cases}$$

We project $\mathcal{A}^{\hat{C}}$ on the set of interface variables $X_{io}=X_{in}\uplus X_{ou}$ to obtain $\mathcal{A}_{X_{io}}=(\mathcal{A}^{\hat{C}},X_{io},\lambda_{io})$. This operation renders some transition invisible in the sense of not affecting variables but, unfortunately, not all of those can be eliminated. Consider an input event (x,i) which is responsible for the excitation of an internal timed component. It may happen that the reaction of that component has no further consequence and hence event (x,i) dies and its clock should be removed. The change in clock denotation due to shifting should be visible in the reduced model in order to preserve its intended semantics.

4.3 Minimization

The next step is the reduction of the discrete state space by merging states which are equivalent in terms of the qualitative behaviors they admit. This work is inspired by the minimization with elimination of silent transitions for the untimed case [9] and its extension to timed systems [25]. From the previous step we have a timed automaton whose transitions are decorated by clock assignments on $\hat{C}$ and by assignments on X_{io}. We minimize the automaton with respect to the alphabet $\Sigma=F_{X_{io}}\times\Gamma(\hat{C})$. We use τ to denote the silent element (f,γ) where both f and γ are the identity functions and use symbols like a to denote the other elements of Σ which are visible. To simplify notation we describe the minimization with respect to a Σ-labeled timed automaton $\mathcal{A}=(\Sigma,Q,q_0,\hat{C},I,\Delta)$ with transitions of the form (q,a,g,γ,q'). We use notation $q\xrightarrow{a}q'$ to denote the existence of an a-labeled transition for q to q'. The minimization is a sequence of three steps $\mathcal{A}\Rightarrow\mathcal{A}^o\Rightarrow\mathcal{A}^b\Rightarrow\mathcal{A}^m$ explained below.

1. To remove silent transitions we identify states that are the target of an observable transition and collapse into them all their τ-successors to construct a state whose invariant is the union of all the invariants of these states. Every sequence of silent transitions is thus replaced by a single time step.

2. We compute the bisimulation relation $\sim$ with respect to Σ on states of $\mathcal{A}^o$ and let the states of $\mathcal{A}^b$ be the equivalence classes of $\sim$. The invariant of each class is the *convex hull* of the invariants of its members. All transitions are kept.

3. For every pair of states we merge all the transitions between them having the same a-label into one transition whose guard is the *convex hull* of the guards of those transitions.

4.3.1 Eliminating Silent Transitions

A state q in $\mathcal{A}=(\Sigma,Q,q_0,I,\Delta)$ is an *observable entry point* if it is the initial state q_0 or there is some observable a and state q' such that $q'\xrightarrow{a}q$. We denote these states by $D(Q)$. A state $q'\notin D(Q)$ is a *silent-successor* of $q\in D(q)$ if there is a path $q\xrightarrow{\tau}q_1\xrightarrow{\tau}\cdots q_k\xrightarrow{\tau}q'$ in the automaton. We denote by $\langle q\rangle$ the set containing q and all its silent successors. Note that all states in $\langle q\rangle$ admit the same variable valuation over X_{io}.

DEFINITION 13 (AUTOMATON $\mathcal{A}^o$).
The automaton $\mathcal{A}^o=(\Sigma,Q^o,q^o_0,\hat{C},I^o,\Delta^o)$ *is constructed from* $\mathcal{A}=(\Sigma,Q,q_0,\hat{C},I,\Delta)$ *as follows:*

- $Q^o=\{\langle q\rangle : q\in D(Q)\}$, $q^o_0=\langle q_0\rangle$
- $I^o(\langle q\rangle)=\bigcup_{p\in\langle q\rangle}I(p)$
- Δ^o *contains a transition* $(\langle q\rangle,a,g,\gamma,\langle q'\rangle)$ *for every transition* $(q'',a,g,\gamma,q')\in\Delta$ *such that* $q''\in\langle q\rangle$.

It is not hard to see that the invariants thus obtained are convex, because clocks in $\hat{C}$ are never reset by silent transitions. Hence $[\![\mathcal{A}^o]\!]$ and $[\![\mathcal{A}]\!]$ coincide. Note also that $\mathcal{A}^o$ has the same property as the reachability graph in the sense that every sequence of transitions is realizable in the original automaton.

4.3.2 Merging Bisimilar States

Let $\mathcal{A}=(\Sigma,Q,\hat{C},q_0,I,\Delta)$ be a timed automaton without silent transitions. The *qualitative bisimulation* relation over Q is the largest equivalence relation satisfying

$$q\sim q' \;\equiv\; \forall a\in\Sigma\,(q\xrightarrow{a}p\Rightarrow\exists p'(q'\xrightarrow{a}p'\wedge p\sim p')).$$

In other words $q \sim q'$ if the same qualitative observable behaviors can be generated from both. We let $[q]$ denote the $\sim$ equivalence classes of q and denote the set of such classes by $Q/\sim$. All elements of $[q]$ admit the same variable labeling.

DEFINITION 14 (AUTOMATON $\mathcal{A}^b$).
The automaton $\mathcal{A}^b = (\Sigma, Q^b, q_0^b, \hat{C}, I^b, \Delta^b)$ *is constructed from* $\mathcal{A}^o = (\Sigma, Q^o, q_0^o, \hat{C}, I^o, \Delta^o)$ *as follows:*

- $Q^b = Q^o/\sim,\ q_0^b = [q_0^o]$
- $I^b([q]) = \bigsqcup_{q' \in [q]} I^o(q')$
- Δ^b *contains a transition* $([q], a, g, \gamma, [q'])$ *for every transition* (p, a, g, γ, p') *such that* $p \in [q]$ *and* $p' \in [q']$.

This transformation, by definition, preserves qualitative behavior, and may over-approximate the timed behavior due to the use of convex hull.

4.3.3 Merging Transitions

The last step is to merge transitions between every pair of states which agree on the transition label.

DEFINITION 15 (AUTOMATON $\mathcal{A}^m$).
Automaton $\mathcal{A}^m = (\Sigma, Q^b, q_0^b, \hat{C}, I^b, \Delta^m)$ *is obtained from* $\mathcal{A}^b = (\Sigma, Q^b, q_0^b, \hat{C}, I^b, \Delta^b)$ *by letting* Δ^m *consist of transitions of the form* (q, a, g, γ, q') *where*

$$g = \bigsqcup \{g' \mid (q, a, g', \gamma, q') \in \Delta^b\}.$$

As in the previous step, qualitative semantics is preserved while the timed semantics may grow due to the use of convex hull. Let us remark that in many cases, $\bigcup g'$ is already convex and the timed semantics is preserved by this step. We can guarantee preservation of timed semantics by this step if we restrict the merging of transitions and states to those whose corresponding unions are convex.

5. EXPERIMENTAL RESULTS

For lack of space we refer the reader to [6] for some details on the implementation of the full tool chain starting from high-level description of the network of timed component down to the realization of all the steps described in the paper (35K lines of C++ code) inside the IF toolbox.

To demonstrate our approach and tool chain consider the circuit depicted in Figure 2, a part of a multiplier adapted from the tutorial [11] on wave pipelining, a technique for improving the throughput of sequential circuits beyond what is possible by a purely synchronous approach. Each tick of a periodic clock defines a temporal window in which each of the inputs may change its value. Jitter is modeled by the fact that each input may choose a different point in that window. Then the changes in the input propagate through 36 logic gates, each modeled by a timed automaton with one clock based on the modeling approach of [20] for modeling circuits with bi-bounded delays. In a purely-synchronous regime the frequency of input waves should be greater than the worst-case propagation time of the whole circuit. Wave pipelining allows one to use a higher frequency and let several waves be present simultaneously in different parts of the circuit. A crucial question in this methodology is to find the highest frequency in which waves do not interfere, that is, avoidance of situations where an elementary gate receives an input from wave $i+1$ while it has not yet finished processing wave i. Questions of a similar nature exist at higher levels of abstraction where gates are replaced by software modules and the effect of inputs arriving at a too high rate can be, for example, buffer overflow.

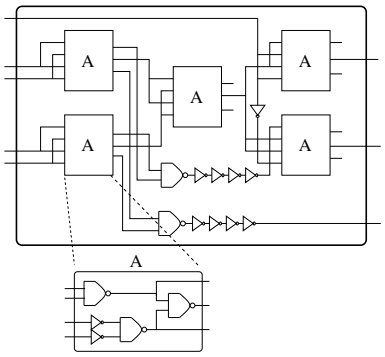

Figure 2: A wave-pipelining circuit

We first explore the limits of a non-compositional approach applied directly to a product of 36 timed automata. The longest delay path in the circuit is 1086 and we first compose it with an input generator of period 1200 and jitter of 10 which generates only one wave through the circuit. In this setting, using our interleaving reduction [7] we manage to complete the analysis within 7 minutes to obtain a reachability graph with more than 26K symbolic states and 50K transitions. When we slightly reduce the input period to 1000 to allow two simultaneous waves, the analysis gets stuck for hours and we need to resort to a compositional approach. As can be seen in the figure, the circuits admits 5 instances of the sub-circuit denoted by A, two of which are connected to the primary inputs. We compose this sub circuit, whose longest path is 362 with an input generator of period 247 and jitter 10 which may induce 2 simultaneous waves in the circuit. The reachability graph obtained has $213/321$ states/transtions which are reduced by our technique to $34/58$ from which we deduce an output jitter of 32. To obtain a model of the third instance of A which is exposed to the output of the first two, we compose A with an input generator of jitter 32 and repeat the process, this time going from $270/411$ transitions/states in the reachability graph to an abstract model with $36/62$ and output jitter of 54. To treat the last two instances of A we compose it with an input generator of jitter 54 and get exactly the same results as the previous step in terms of size or the graph and the reduced model. No interference is detected and we can deduce that the whole circuit is safe with an input frequency with period 247. When we repeat the same sequence of steps with a input of period 246, an interference is detected.

6. DISCUSSION

We have developed a new original technique for abstracting the behavior of timed components. The essence of this technique is to *use the internal clocks* to compute what the component can and cannot do and then *get rid of these clocks* by projecting on the observable input clocks that we introduce into the model. As a result we obtain a model which focuses on what the potential users of the component care about: the *relation between the timing of input and output events*. This model is faithful to the qualitative semantics of the component but may relax the temporal correlation between output events that depend on the same internal event. Such a reduced model can serve as a specification of the component in a component library and, as we have demonstrated in the case study, to allow us to analyze timed automata of unpreceded size and complexity.

Finding ways to evaluate the *quality* of the abstractions obtained by our technique is one of the items on our agenda. Over-approximation of a system by another can be obtained trivially and the quality of the abstract model should be be judged either by whether it is sufficiently detailed to prove some properties, or by quantitative means: how many spurious behaviors have been added. For the first criterion, it is hard to find large examples due to the intractability of timed automata analysis, and we intend to take some cases studied using analytical methods for performance analysis and see whether we can obtain tighter results. For the second approach, recent results on volume and entropy of timed languages [5] may provide a tool for measuring the degree of over-approximation.

7. REFERENCES

[1] L. de Alfaro, T.A. Henzinger, and M. Stoelinga, Timed Interfaces, *EMSOFT'02*, LNCS 2491, 108-122, 2002.

[2] R. Alur, Timed Automata, *CAV'99*, LNCS 1633, 8–22, 1999.

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

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

[5] E. Asarin and A. Degorre, Volume and Entropy of Regular Timed Languages, *hal*-00369812, 2009.

[6] R. Ben Salah, *On Timing Analysis of Large Systems*, PhD Thesis, INPG Grenoble, October 2007.

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

[8] R. Ben Salah, M. Bozga and O. Maler, On Timed Components and their Abstraction *SAVCBS'07*, 63-71, 2007.

[9] A. Bouajjani, J.-C. Fernandez, S. Graf, C. Rodriguez, and J. Sifakis, Safety for Branching Time Semantics, *ICALP'91*, 1991.

[10] M. Bozga, S. Graf and L. Mounier, IF-2.0: A Validation Environment for Component-Based Real-Time Systems, *CAV'02*, 343-348, LNCS 2404, 2002.

[11] W. Burleson, M. Ciesielski, F. Klass and W. Liu, Wave-pipelining: A Tutorial and Survey of Recent |Research, *IEEE Trans. on VLSI* 6, 464-474, 1998.

[12] C. Daws, A. Olivero, S. Tripakis, and S. Yovine, The tool KRONOS, In *Hybrid Systems III*, LNCS 1066, 208-219, 1995.

[13] C. Daws and S. Yovine, Reducing the Number of Clock Variables of Timed Automata, *RTSS'96*, 73–81, 1996.

[14] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, System level performance analysis-the SymTA/S approach, *IEE Proc. CDT* 152, 148-166, 2005.

[15] T. Henzinger, S. Matic, An Interface Algebra for Real-Time Components, RTAS'06, 253-263, 2006.

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

[17] B. Jonsson, S. Perathoner, L. Thiele and W. Yi, Cyclic Dependencies in Modular Performance Analysis *EMSOFT'08*, 179-188, 2008.

[18] S. Kunzli, F. Poletti, L. Benini and L. Thiele, Combining Simulation and Formal Methods for System-Level Performance Analysis *DATE'06*, 236-241, 2006.

[19] K.G. Larsen, P. Pettersson, and W. Yi, UPPAAL in a Nutshell, *STTT* 1, 134-152, 1997.

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

[21] S. Malik, Analysis of cyclic combinational circuits, *IEEE Transactions on Computer-Aided Design* 13, 950-956, 1994.

[22] S. Perathoner, E. Wandeler, L. Thiele, A. Hamann, S. Schliecker, R.Henia, R. Racu, R. Ernst, M. Gonzalez-Harbour, Influence of Different System Abstractions on the Performance Analysis of Distributed Real-Time Systems, *EMSOFT'07*, 193-202, 2007.

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

[24] L. Thiele and E. Wandeler, Performance Analysis of Embedded Systems, *Embedded System Handbook*, CRC Press, 2005.

[25] S. Tripakis and S. Yovine, Analysis of timed systems using time-abstracting bisimulations, *FMSD* 18, 25-68, 2001.

[26] E. Wandeler, L. Thiele, M. Verhoef, P. Lieverse, System Architecture Evaluation Using Modular Performance Analysis - A Case Study, *STTT* 8, 649-667, 2006.

[27] S. Yovine, Kronos: A Verification Tool for Real-time Systems, *STTT* 1 123-133, 1997.