Verifying Abstractions of Timed Systems

Serdar Taşıran* Rajeev Alur** Robert P. Kurshan** Robert K. Brayton*

Abstract. Given two descriptions of a real-time system at different levels of abstraction, we consider the problem of proving that the refined representation is a correct implementation of the abstract one. To avoid the complexity of building a representation for the refined system in its entirety, we develop a compositional framework for the implementation check to be carried out in a module-by-module manner using assume-guarantee style proof rules. On the algorithmic side, we show that the problem of checking for timed simulation relations, a sufficient condition for correct implementation, is decidable. We study state homomorphisms as a way of specifying a correspondence between two modules. We present an algorithm for checking if a given mapping is a homomorphism preserving timed behaviors. We have implemented this check in the verifier COSPAN, and applied our method to the compositional verification of an asynchronous queue circuit.

1 Introduction

We address the problem of refinement for real-time systems such as control protocols and asynchronous circuits. We want to prove that, of the two given representations of a system, the more refined one "implements" the more abstract one. By doing so, one would be assured that the properties proved about the abstract description continue to hold in the refined version. This scenario may arise either because the design is being carried out in a top-down fashion by refining the system iteratively, or because the system is too complex and an abstraction of the system needs to be used to verify properties. This work addresses the following two problems: proving that one timed system is an abstraction of the other, and developing a compositional verification framework so that this proof can be carried out modularly.

Typically, a model of a system is described as a collection of coordinating components. The aim of compositional verification is to decompose the problem of verifying refinement into subproblems so that a monolithic representation for the system does not have to be built. Performing verification in this way has the important benefit of scaling with the increasing complexity of real-time systems encountered in practice. With this goal, in Section 2 we present a model for real-time systems. Defining timed language inclusion as the weakest notion for "implementation", we prove compositionality properties and show the soundness of an assumption-guarantee paradigm for modular verification in Section 3. To prove that a refined system with two components A and B implements an abstraction consisting of two components C and D, the compositionality principle asserts that it suffices to prove A implements C and B implements D, separately, while the stronger assumption-guarantee rule asserts that it suffices to prove that

* Department of Electrical Engineering and Computer Sciences, University of California at Berkeley. Supported by SRC under contract DC-324-026.

** Computing Sciences Research Center, Bell Laboratories, Murray Hill, NJ.

(1) the composition of A and D implements C, and (2) the composition of B and C implements D. Note the circularity in the assumption-guarantee rule. Its correctness requires that the processes involved are nonblocking in the sense that they are willing to accept any inputs and do not block the progress of time.

While the relation of timed language inclusion is a natural choice for abstraction relation, it was proved in [AD94] that language containment is undecidable for non-deterministic systems. Since abstract descriptions often involve non-determinism, we propose timed simulation relations as a sufficient condition for implementation. In Section 4 we prove that it is decidable (in EXPTIME) to check whether such a relation exists between two systems.

In Section 5, we investigate homomorphisms, which are restricted forms of simulation relations, as an alternative way of specifying and verifying timed abstractions. This approach requires the user to specify a correspondence between the locations of the abstract and refined versions, but is computationally more feasible, and is a generalization of the existing homomorphism checks supported by the system COSPAN. We present an algorithm for checking if a given mapping between the locations of two timed systems implies inclusion of timed languages. Our algorithm is implemented in COSPAN. We used this algorithm and assumption-guarantee style reasoning to verify abstractions of an asynchronous queue circuit. The algorithm and the results of the experiments are presented in Section 5.

Related research In recent years, many tools have been developed to support verification of real-time systems (for instance, KRONOS [DOY94], ORBITS [Rok93], timed COSPAN [AK96] and UPPAAL [LPY95]). These tools consider the problem model checking, that is, verifying that a mathematical model of a system satisfies its specification. The problem of proving refinements of timed systems has been considered only in the context of manual proofs (see, for instance, [AL91, Sha92, LA92]).

We have emphasized compositionality and modularity. Compositionality means that the implementation relation is a congruence with respect to parallel composition, and is exhibited by any reasonable formalism. Modularity makes explicit distinction between which variables are updated by the system and which are updated by the environment so as to support assume-guarantee style reasoning (an example of such a framework is I/O automata [LT87]). Assume-guarantee style proof rules for untimed systems have been proposed by many researchers (for instance, [GL94, AL93, AH96]). When the rule is symmetric and involves circularity, it is necessary to require that a process is nonblocking [LT87, AH96]. In the case of timed systems, these issues have been considered for timed I/O automata in [GSSL94]. Our framework uses synchronous composition and a much simpler definition of nonblocking without resorting to games.

For timed systems, a variety of implementation relations can be considered. Timed language containment is undecidable [AD94]. Timed bisimulation is decidable [Č92], but is not appropriate for refinements. Time-abstract simulation is considered in [HHK95], but does not preserve timed properties. As far as we know, there is no previous study of timed simulations.

The use of simulation mappings or homomorphisms to prove refinements is common in the literature (see, for instance, [Kur94, Sha92, AL91, LT87]). Among automated tools, COSPAN provides support to check whether the user-supplied mappings actually define a homomorphism. Our work generalizes this to checking the preservation of timed behaviors.

2 Timed Abstractions

Preliminaries Let X be a finite set of real-valued variables. An X-valuation Φ assigns a nonnegative real value $\Phi(x)$ to each variable $x \in X$. Let Φ be an X-valuation. For a real number $\delta \geq 0$, $\Phi+\delta$ denotes the X-valuation that assigns the value $\Phi(x)+\delta$ to each variable x, and $\mathbf{0}$ denotes the X-valuation that assigns the value 0 to all $x \in X$. For a subset $Y \subseteq X$, $\Phi[Y := 0]$ denotes the X-valuation that assigns the value 0 to each $x \in Y$ and the value $\Phi(x)$ to each $x \notin Y$. An X-predicate φ is a boolean combination of constraints of the form $x \diamond k$, where k is a nonnegative integer constant, $x \in X$ is a variable, and $\diamond$ is one of the binary comparison relations: $\leq$, $\geq$, $=$. We write $\Phi \models \varphi$ if the valuation Φ satisfies the formula φ. Note that the set of X-valuations satisfying the X-predicate φ is closed.

Let P be a finite set of variables, each ranging over a finite type. A P-valuation $\mathbf{f}$ is an assignment of values to variables in P. For a P-valuation $\mathbf{f}$ and a subset $Q \subseteq P$, $\mathbf{f}(Q)$ denotes the Q-valuation obtained by the restriction of $\mathbf{f}$ to the variables in Q. A P-event is a pair $\langle \mathbf{f}, \mathbf{f}' \rangle$ consisting of P-valuations $\mathbf{f}$ and $\mathbf{f}'$ denoting the old and the new values of the variables in P. A P-predicate χ is a subset of P-events. While writing P-predicates as formulas, we use primed variables to refer to the updated values. For instance, the P-predicate $p' \neq p$ is the set of all P-events $\langle \mathbf{f}, \mathbf{f}' \rangle$ such that $\mathbf{f}'(p) \neq \mathbf{f}(p)$. We use $\mathit{stutter}(P)$ as an abbreviation for the predicate $\wedge_{p \in P}\ p' = p$.

With these conventions, we proceed to define timed processes as a model for real-time systems. All real-time systems that can be specified in the S/R language of COSPAN can be described as timed processes.

Timed processes A *timed process* A is a tuple $\langle S, S_0, O, I, X, \alpha, \mu, E \rangle$, where

- S is the finite (nonempty) set of locations.
- $S_0 \subseteq S$ is the nonempty set of initial locations.
- X is the finite set of real-valued variables, called *clocks*.
- O is the finite set of output variables, each ranging over a finite type. An *output* of A is a O-valuation.
- I is the finite set of input variables, each ranging over a finite type. It is required that I and O are disjoint. An *input* of A is an I-valuation, an *input-event* is an I-event, and an *input-predicate* is a I-predicate. An *observation* of A is a $(I \cup O)$-valuation, and an *observation-event* of A is a $(I \cup O)$-event.
- α is the invariant function that assigns the X-predicate $\alpha(s)$ to each location $s \in S$. A *state* σ of A is a pair $\langle s, \Phi \rangle$ containing the location $s \in S$ and the X-valuation $\Phi \in \alpha(s)$. The set of all states is denoted Σ_A. The state $\langle s, \Phi \rangle$ is initial if $s \in S_0$ and $\Phi(x) = 0$ for all $x \in X$.
- μ is the output function that assigns the output $\mu(s)$ to each location $s \in S$.

– E is the finite set of edges. Each edge e is a tuple $\langle s,t,\varphi,\chi,Y\rangle$ consisting of the source location s, the target location t, the X-predicate φ, the input-predicate χ, and the set $Y \subseteq X$ of clocks to be reset. It is required that E contains the edge $\langle s,s,true,stutter(I),\emptyset\rangle$ for each location s and for given source and target locations, there is at most one edge between them.

Consider a state $\sigma = \langle s,\Phi\rangle$ of the timed process A and a positive time increment δ. The process A can *wait* for δ in state σ, written $wait(\sigma,\delta)$, iff for all $0 \le \delta' < \delta$, $(\Phi+\delta') \models \alpha(s)$. A *timed event* γ of the timed process A is a tuple $\langle \delta,\mathbf{f},\mathbf{f}'\rangle$ consisting of a positive real-valued increment δ and the observation-event $\langle \mathbf{f},\mathbf{f}'\rangle$. Such an event means that the process can wait for the time period δ and then update its output variables from $\mathbf{f}(O)$ to $\mathbf{f}'(O)$ while the environment is updating the input variables from $\mathbf{f}(I)$ to $\mathbf{f}'(I)$. The set of all timed events of A is denoted Γ_A.

The timed process A gives a labeled transition system over the state-space Σ_A with the labels Γ_A. For states $\sigma = \langle s,\Phi\rangle$ and $\tau = \langle t,\Theta\rangle$ in Σ_A, and a timed event $\gamma = \langle \delta,\mathbf{f},\mathbf{f}'\rangle$ in Γ_A, define $\sigma \xrightarrow{\gamma} \tau$ iff $\mathbf{f}(O) = \mu(s)$, $\mathbf{f}'(O) = \mu(t)$, $wait(\sigma,\delta)$, and there exists an edge $\langle s,t,\varphi,\chi,Y\rangle$ such that $(\Phi+\delta) \models \varphi$, $\langle \mathbf{f},\mathbf{f}'\rangle \models \chi$, and $\Theta = (\Phi+\delta)[Y := 0]$. We write $\sigma \xrightarrow{\gamma}$ if $\sigma \xrightarrow{\gamma} \tau$ for some τ.

Proposition 1 (Closure under stuttering). *For $\gamma = \langle \delta,\mathbf{f},\mathbf{f}'\rangle$, if $\sigma \xrightarrow{\gamma} \tau$ then for every $0 < \delta' < \delta$, there exists σ' with $\sigma \xrightarrow{\gamma'} \sigma'$ and $\sigma' \xrightarrow{\gamma''} \tau$ where $\gamma' = \langle \delta',\mathbf{f},\mathbf{f}\rangle$ and $\gamma'' = \langle \delta-\delta',\mathbf{f},\mathbf{f}'\rangle$.*

Example Figure 1 depicts a timed process representing an inertial delay buffer with non-deterministic delay between d_{min} and d_{max}. According to the inertial delay model, an input pulse must have duration at least d_{min} to be reflected at the output, and any input pulse of duration more than d_{max} has to create a corresponding pulse at the output. Note that any logic gate with inertial delay can be modeled as a delayless logic gate followed by an inertial delay buffer. Observe that, in our model of timed processes, a process spends nonzero time

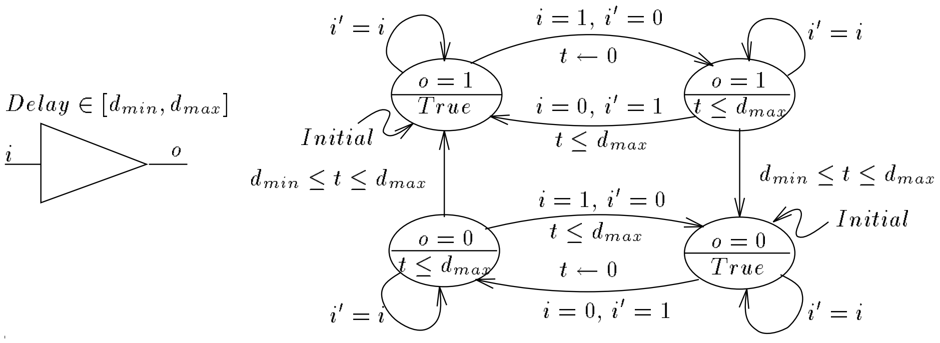

Fig. 1. The timed process corresponding to an inertial delay buffer with non-deterministic delay between d_{min} and d_{max}.

in each location and transitions are instantaneous. The enabling condition on a transition can refer to the old (unprimed) as well as the new (primed) values of the inputs, this makes the use of transient (zero-time) states unnecessary.

Timed language A *timed event sequence* $\overline{\gamma} = \gamma_0, \gamma_1, ..., \gamma_{k-1}$ is a finite sequence of timed events $\gamma_i = \langle \delta_i, \mathbf{f}_i, \mathbf{f}'_i \rangle$ such that $\mathbf{f}_{i+1} = \mathbf{f}'_i$ for $0 \leq i < k$. For such a timed event sequence, define $\Delta_0 = 0$ and $\Delta_i = \sum_{j=0}^{i-1} \delta_j$ for $1 \leq i \leq k$. Each such $\overline{\gamma}$ uniquely defines a function $F_{\overline{\gamma}}$ from the closed interval $[0, \Delta_k]$ to the observations given by $F_{\overline{\gamma}}(t) = \mathbf{f}_i$ for $t \in [\Delta_i, \Delta_{i+1})$ and $F_{\overline{\gamma}}(\Delta_k) = \mathbf{f}'_{k-1}$.

A *run* of A on a timed event sequence $\overline{\gamma}$ is a sequence of states $\sigma_0, \sigma_1, \sigma_2, ..., \sigma_k$ such that $\sigma_0 \xrightarrow{\gamma_0} \sigma_1 \xrightarrow{\gamma_1} \sigma_2 \xrightarrow{\gamma_2} ... \xrightarrow{\gamma_{k-1}} \sigma_k$ in A. The timed event sequence $\overline{\gamma}$ is called a *trace* of A if there exists a run in A on $\overline{\gamma}$ starting from an initial state. The *timed language* of a process A, denoted $\mathcal{L}(A)$, is the set of traces of A.

Timed abstraction relations Consider two timed processes $A = \langle S^A, S_0^A, O^A, I^A, X^A, \alpha^A, \mu^A, E^A \rangle$ and $B = \langle S^B, S_0^B, O^B, I^B, X^B, \alpha^B, \mu^B, E^B \rangle$. The timed process A is *comparable* to B iff $O^B \subseteq O^A$ and $I^B \subseteq I^A$.

Suppose A is comparable to B. Then, a *timed simulation relation* from A to B is a binary relation $\Omega \subseteq \Sigma_A \times \Sigma_B$ among the states of the two processes such that for every $(\sigma, \tau) \in \Omega$ and for every timed event γ in Γ_A if $\sigma \xrightarrow{\gamma} \sigma'$ then there exists $\tau' \in \Sigma_B$ such that $\tau \xrightarrow{\gamma} \tau'$ and (σ', τ') is in Ω. The timed simulation relation Ω is said to be *initialized* iff for every initial state σ of A, there exists an initial state τ of B with $(\sigma, \tau) \in \Omega$. If the timed process A is comparable to the timed process B, and an initialized timed simulation relation from A to B exists, then A is said to *timed-simulate* B, written $A \preceq_S B$.

If timed process A is comparable to B, A is said to be *timed-implement* B, denoted by $A \preceq_L B$, iff for every trace $\overline{\gamma}^A$, there exists $\overline{\gamma}^B$ in $\mathcal{L}(B)$ assigning the same values to the input and output variables of B at all times, i.e., $F_{\overline{\gamma}^A}(I^B \cup O^B) = F_{\overline{\gamma}^B}$. $\preceq_L$ is also referred to as the *language inclusion* relation.

Proposition 2 (Preorder). *The relations $\preceq_S$ and $\preceq_L$ are reflexive and transitive.*

The two timed processes are *timed simulation equivalent*, written $A \cong_S B$, iff both $A \preceq_S B$ and $B \preceq_S A$. It follows that the relation $\cong_S$ is an equivalence relation. Similarly, the timed language equivalence $\cong_L$ is the equivalence induced by $\preceq_L$.

Timed simulation is a stronger requirement than timed implementation.

Proposition 3 (Languages and timed simulation). *If $A \preceq_S B$, then $A \preceq_L B$.*

Composition of timed processes The key operator to build complex processes from simpler ones is parallel composition. Consider two timed processes $A = \langle S^A, S_0^A, O^A, I^A, X^A, \alpha^A, \mu^A, E^A \rangle$ and $B = \langle S^B, S_0^B, O^B, I^B, X^B, \alpha^B, \mu^B, E^B \rangle$. We assume that the clocks X^A and X^B are disjoint (this can be achieved by renaming). The two timed processes A and B are *composable* iff their output variables O^A and O^B are disjoint. For two such composable timed processes, their parallel composition $A \parallel B$ is the timed process with the following components:

- The set S of locations equals $S^A \times S^B$.
- The set S_0 of initial locations equals $S_0^A \times S_0^B$.
- The set O of output variables equals $O^A \cup O^B$.
- The set I of input variables equals $(I^A \cup I^B) \setminus O$.
- The set X of clock variables equals $X^A \cup X^B$.
- For every $s \in S^A$ and $t \in S^B$, the invariant $\alpha(\langle s, t\rangle)$ equals the conjunction $\alpha^A(s) \wedge \alpha^B(t)$.
- For every $s \in S^A$ and $t \in S^B$, the output $\mu(\langle s, t\rangle)$ equals $\mu^A(s) \cup \mu^B(t)$.
- For every edge $e = \langle s, t, \varphi, \chi, Y\rangle$ in E^A and $e' = \langle s', t', \varphi', \chi', Y'\rangle$ in E^B, the set E contains the edge $\langle\langle s, s'\rangle, \langle t, t'\rangle, \varphi \wedge \varphi', \chi'', Y \cup Y'\rangle$, where the I-event $\langle \mathbf{f}, \mathbf{f}'\rangle$ is in χ'' iff $\langle \mathbf{f} \cup \mu^B(t), \mathbf{f}' \cup \mu^B(t')\rangle \models \chi$ and $\langle \mathbf{f} \cup \mu^A(s), \mathbf{f}' \cup \mu^A(s')\rangle \models \chi'$.

First observe that the parallel composition operator is commutative and associative:

Proposition 4. *$A \parallel B \cong_S B \parallel A$, and $A \parallel (B \parallel C) \cong_S (A \parallel B) \parallel C$.*

The composed process is an implementation of each component:

Proposition 5. *$A \parallel B \preceq_S A$.*

Both the timed equivalence relations are congruences with respect to the composition operator $\parallel$:

Proposition 6 (Compositionality). *If $A \preceq_S B$ then $A \parallel C \preceq_S B \parallel C$; if $A \preceq_L B$ then $A \parallel C \preceq_L B \parallel C$.*

The compositionality principle tells us that to prove that $A \parallel B$ is a timed refinement of $C \parallel D$ it suffices to show separately that A is a timed refinement of C and B is a timed refinement of D.

3 Modularity

In this section, we examine assumption-guarantee style reasoning principles for the abstraction relations. It is not clear if such principles hold for the simulation preorder, however, for the language preorder, with certain restrictions, a simple and powerful modularity principle can be obtained.

A timed process $A = \langle S, S_0, O, I, X, \alpha, \mu, E\rangle$ is said to be *nonblocking* iff for all states σ in Σ_A,

(1) $\sigma \xrightarrow{\gamma}$ for some timed event γ, and
(2) if $\sigma \xrightarrow{\gamma}$ for some timed event $\gamma = \langle \delta, \mathbf{f}, \mathbf{f}'\rangle$ then $\sigma \xrightarrow{\langle \delta, \mathbf{g}, \mathbf{g}'\rangle}$ for all $\mathbf{g}$ and $\mathbf{g}'$ with $\mathbf{g}(O) = \mathbf{f}(O)$ and $\mathbf{g}'(O) = \mathbf{f}'(O)$.

The intuition behind this definition is that a non-blocking process should be able to generate a trace no matter what the sequence of input events is. The execution of a nonblocking timed process can be viewed operationally as follows. Consider the timed process in state $\sigma = \langle s, \Phi\rangle$ with output $\mathbf{f} = \mu(s)$. The process chooses a timed delay δ' such that $wait(\sigma, \delta')$, and simultaneously, the environment chooses a time delay δ''. Let δ be the minimum of δ' and δ''. The next observable event happens after a delay of δ. The timed process decides to update its output from $\mathbf{f}$ to $\mathbf{f}'$. The environment decides to update the input from $\mathbf{g}$ to $\mathbf{g}'$ independently. The timed process updates its state to $\sigma' = \langle s', \Phi'\rangle$ with $\mu(s') = \mathbf{f}'$ by choosing an appropriate edge. Such an update always exists due to the nonblocking requirement. Thus, while the update of outputs is independent of the update of inputs, the update of internal variables (i.e., the location and the clocks) depends on it.

Proposition 7 (Closure under composition). *If A and B are nonblocking, then so is $A \parallel B$.*

The following assumption-guarantee rule is useful in modular reasoning.

Proposition 8 (Assumption-Guarantee). *For nonblocking timed processes, if $A \parallel D \preceq_L C$ and $C \parallel B \preceq_L D$ then $A \parallel B \preceq_L C \parallel D$.*

Observe the apparent circularity in the rule: to prove that $A \parallel B$ is a refinement of $C \parallel D$, it suffices to prove that (1) A is a refinement of C assuming that the environment behaves like D, and (2) B is a refinement of D assuming that the environment behaves like C. The proof relies on the fact that all processes are nonblocking. Let us note a few observations before we give a detailed proof. First, the rule is incorrect if we remove the requirement of nonblocking. Second, the rule is incorrect if we replace $\preceq_L$ by the simulation preorder $\preceq_S$. Third, recall that we have required all X-predicates to be closed, (i.e. all invariants labeling the locations are conjunctions of non-strict inequalities). If we allow predicates that define open sets (e.g. invariants of the form $x < 5$), the rule fails again. In the proof below, we show that a timed process cannot force infinitely many transitions within a finite interval (the so-called condition of non-Zenoness). This does not hold automatically with open invariants. In this case, the non-blocking requirement needs to be strengthened by replacing requirement (1) in the definition of nonblocking using games. In [GSSL94], such a framework is developed for (asynchronous) timed I/O automata. However, that complicates the development considerably (indeed, the proof that the nonblocking requirement is preserved under composition runs many pages in [GSSL94]).

Proof. The proof is by contradiction. Assume that there exists a timed event sequence $\overline{\gamma} = \gamma_0, \gamma_1, ..., \gamma_{n-1}$ in $\mathcal{L}(A \parallel B)$ and not in $\mathcal{L}(C \parallel D)$. Let $\overline{\gamma_k} = \gamma_0, \gamma_1, ..., \gamma_{k-1}$, where $k < n$ be the longest prefix of $\overline{\gamma}$ that is a trace of $C \parallel D$. Let $\tau_0 \xrightarrow{\gamma_0} \tau_1 \xrightarrow{\gamma_1} ... \xrightarrow{\gamma_{k-1}} \tau_k$ be in $C \parallel D$. Since $\gamma_0, \gamma_1, ..., \gamma_n \in \mathcal{L}(A \parallel B)$, we have a run in $A \parallel B$ of the form $\sigma_0 \xrightarrow{\gamma_0} ... \xrightarrow{\gamma_{k-1}} \sigma_k \xrightarrow{\gamma_k} ... \xrightarrow{\gamma_{n-1}} \sigma_n$. Let $\sigma_i = \langle \sigma_i^A, \sigma_i^B \rangle$ and $\tau_i = \langle \tau_i^C, \tau_i^D \rangle$ for all i. From the above, we have

$$\langle \sigma_0^A, \tau_0^D \rangle \xrightarrow{\gamma_0} \langle \sigma_1^A, \tau_1^D \rangle \xrightarrow{\gamma_1} ... \xrightarrow{\gamma_{k-1}} \langle \sigma_k^A, \tau_k^D \rangle \text{ in } A \parallel D$$

and

$$\langle \tau_0^C, \sigma_0^B \rangle \xrightarrow{\gamma_0} \langle \tau_1^C, \sigma_1^B \rangle \xrightarrow{\gamma_1} ... \xrightarrow{\gamma_{k-1}} \langle \tau_k^C, \sigma_k^B \rangle \text{ in } C \parallel B$$

The partitioning of the input and output variables of the processes in Figure 2

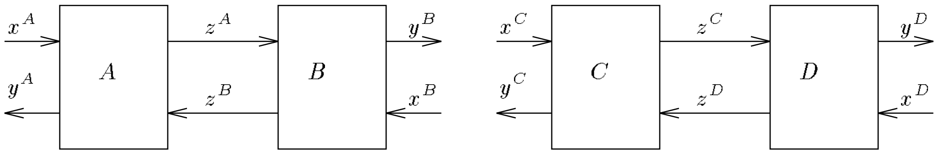

Fig. 2. The partitioning of the input and output variables.

will be useful in the rest of the proof. Let σ_k^A, σ_k^B, τ_k^C and τ_k^D be denoted by

$\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$ and $\boldsymbol{d}$ respectively. Also let $\gamma_k = \gamma = \langle \delta, \mathbf{f}, \mathbf{f}' \rangle$. We have $\langle \boldsymbol{a}, \boldsymbol{b} \rangle \xrightarrow{\gamma} \langle \boldsymbol{a}', \boldsymbol{b}' \rangle$ for some $\langle \boldsymbol{a}', \boldsymbol{b}' \rangle \in \Sigma_{A \| B}$ since $\overline{\gamma}$ is in the language. Consider $\langle \boldsymbol{a}, \boldsymbol{d} \rangle \in \Sigma_{A \| D}$ and $\langle \boldsymbol{c}, \boldsymbol{b} \rangle \in \Sigma_{C \| B}$. From the above, A and B can select an increment of δ at states $\boldsymbol{a}$ and $\boldsymbol{b}$ respectively. The rest of the proof splits into two cases.

Case 1. C and D can both select time increments of δ or greater in states $\boldsymbol{c}$ and $\boldsymbol{d}$. In this case, letting the environment choose the inputs x^A and x^B according to γ_k, we have

(I) $\langle \boldsymbol{a}, \boldsymbol{d} \rangle \xrightarrow{\langle \delta, \mathbf{g}, \mathbf{g}' \rangle} \langle \boldsymbol{a}'', \boldsymbol{d}' \rangle$ for some $\boldsymbol{a}''$ and $\boldsymbol{d}'$ such that $\mathbf{g}$ and $\mathbf{g}'$ agree with $\mathbf{f}$ and $\mathbf{f}'$ on x^A, y^A and z^A respectively. Thus $\gamma_0, \gamma_1, \ldots, \gamma_{k-1}, \langle \delta, \mathbf{g}, \mathbf{g}' \rangle$ is a trace of $A \parallel D$, and since $A \parallel D \preceq_L C \parallel D$, it is a trace of $C \parallel D$.

(II) $\langle \boldsymbol{c}, \boldsymbol{b} \rangle \xrightarrow{\langle \delta, \mathbf{h}, \mathbf{h}' \rangle} \langle \boldsymbol{c}', \boldsymbol{d}'' \rangle$ for some $\boldsymbol{c}'$ and $\boldsymbol{d}''$ such that $\mathbf{h}$ and $\mathbf{h}'$ agree with $\mathbf{f}$ and $\mathbf{f}'$ on x^B, y^B and z^B respectively. We conclude that $\gamma_0, \gamma_1, \ldots, \gamma_{k-1}, \langle \delta, \mathbf{h}, \mathbf{h}' \rangle$ is a trace of $C \parallel D$ by a similar reasoning.

(I) and (II), together with the fact that C and D are nonblocking imply that $\gamma_0, \gamma_1, \ldots, \gamma_{k-1}, \gamma_k$ is a trace of $C \parallel D$, which contradicts our assumption that $\gamma_0, \gamma_1, \ldots, \gamma_{k-1}$ is maximal.

Case 2. Either C or D can not select a time increment of δ or greater in states $\boldsymbol{c}$ and $\boldsymbol{d}$. Let δ^C and δ^D be the largest increments that C and D can select and let Δ_0 be the minimum of δ^C and δ^D. Since $\Delta_0 < \delta$, for $\theta_0 = \langle \Delta_0, \mathbf{f}, \mathbf{f} \rangle$, we have $\langle \boldsymbol{a}, \boldsymbol{b} \rangle \xrightarrow{\theta_0} \langle \boldsymbol{a}_0^*, \boldsymbol{b}_0^* \rangle$ for some $\langle \boldsymbol{a}_0^*, \boldsymbol{b}_0^* \rangle$. Then,

(I) $\langle \boldsymbol{a}, \boldsymbol{d} \rangle \xrightarrow{\langle \Delta_0, \mathbf{g}_0, \mathbf{g}_0' \rangle} \langle \boldsymbol{a}_0, \boldsymbol{d}_0 \rangle$ for some $\boldsymbol{a}_0$ and $\boldsymbol{d}_0$ such that $\mathbf{g}_0$ and $\mathbf{g}_0'$ agree with $\mathbf{f}$ on x^A, y^A and z^A. With a similar reasoning to (I) in Case 1, we have that $\gamma_0, \gamma_1, \ldots, \gamma_{k-1}, \langle \Delta_0, \mathbf{g}_0, \mathbf{g}_0' \rangle$ is a trace of $C \parallel D$.

(II) $\langle \boldsymbol{c}, \boldsymbol{b} \rangle \xrightarrow{\langle \Delta_0, \mathbf{h}_0, \mathbf{h}_0' \rangle} \langle \boldsymbol{c}_0, \boldsymbol{b}_0 \rangle$ for some $\boldsymbol{c}_0$ and $\boldsymbol{b}_0$ such that $\mathbf{h}_0$ and $\mathbf{h}_0'$ agree with $\mathbf{f}$ on x^B, y^B and z^B. From this, we obtain $\gamma_0, \gamma_1, \ldots, \gamma_{k-1}, \langle \Delta_0, \mathbf{h}_0, \mathbf{h}_0' \rangle$ as a trace of $C \parallel D$.

From (I) and (II), we derive that $\gamma_0, \gamma_1, \ldots, \gamma_{k-1}, \theta_0$ is a trace of $C \parallel D$.

Note that $\overline{\gamma_k}$ is equivalent modulo stuttering to $\gamma_0, \gamma_1, \ldots, \gamma_{k-1}, \theta_0, \gamma_k^{(0)}$, where $\gamma_k^{(0)} = \langle \delta - \Delta_0, \mathbf{f}, \mathbf{f}' \rangle$. If we can show that the latter is a trace of $C \parallel D$, we will have reached a contradiction.

Consider $\overline{\phi_0} = \gamma_0, \gamma_1, \ldots, \gamma_{k-1}, \theta_0$ and some run of $C \parallel D$ on this trace. Let $\langle \boldsymbol{c}_0^*, \boldsymbol{d}_0^* \rangle$ be the state reached at the end of this run. We can replace $\overline{\gamma_k}$ by $\overline{\phi_0}$, and $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$ and $\boldsymbol{d}$ with $\boldsymbol{a}_0^*$, $\boldsymbol{b}_0^*$, $\boldsymbol{c}_0^*$ and $\boldsymbol{d}_0^*$ and repeat the argument in the proof so far. If it is the case that it is possible to select a time increment of $\delta - \Delta_0$ at states $\boldsymbol{c}_0^*$ and $\boldsymbol{d}_0^*$, we conclude similarly that $\gamma_0, \gamma_1, \ldots, \gamma_{k-1}, \theta_0, \gamma_k^{(0)}$ is a trace of $C \parallel D$, which contradicts the maximality of $\gamma_0, \gamma_1, \ldots, \gamma_{k-1}, \theta_0$. Otherwise, we can obtain $\theta_1 = \langle \Delta_1, \mathbf{f}, \mathbf{f} \rangle$ such that $\Delta_1 < \delta - \Delta_0$ and $\gamma_0, \gamma_1, \ldots, \gamma_{k-1}, \theta_0, \theta_1$ is a trace of $C \parallel D$. Then, the whole argument can be repeated again. If the process terminates after finitely many repetitions, we obtain $\gamma_0, \gamma_1, \ldots, \gamma_{k-1}, \theta_0, \theta_1, \ldots, \theta_m, \gamma_k^{(m)}$, where $\gamma_k^{(m)} = \langle \delta - \sum_{i=0}^{m} \Delta_i, \mathbf{f}, \mathbf{f}' \rangle$, which is equivalent modulo stuttering to $\overline{\gamma_k}$, the contradiction we sought. Otherwise, we must have an infinite sequence $\gamma_0, \gamma_1, \ldots, \gamma_{k-1}, \theta_0, \theta_1, \theta_2, \theta_3, \ldots$

where $\theta_i = \langle \Delta_i, \mathbf{f}, \mathbf{f} \rangle$ and $\sum_{i=0}^{\infty} \Delta_i < \delta$. We will now show that this can not be the case. Since the sum of the Δ_i's converges, there must exist l such that $\sum_{i=l}^{\infty} \Delta_i < 1$. Let $r > l$ be such that no clock value assumes a non-zero integer value after the edge corresponding to θ_r. Such an r must exist for the following reason: The non-zero integer crossing points corresponding to a clock must be separated by 1 time unit, so there can not be more than one such crossing per clock after θ_r. But there are a finite number of clocks, therefore only finitely many such points, which means there is a last one. We argue that after θ_r, the same set of clock predicates of the form $x \diamond k$ are satisfied. This is because no clocks cross integer boundaries other than 0, and the clocks that get reset after θ_r satisfy $0 < x \leq 1$ on all edges after θ_r. Let us choose any $q > r$. By the preceding argument, Δ_q could have been increased to $\sum_{i=q}^{\infty} \Delta_i$, which contradicts the maximality of Δ_q.

4 Decidability of Timed Simulation

A language inclusion check between non-deterministic timed processes is undecidable in the general case[3] while [Č92] has proved that computing timed bisimulation is decidable. We have shown the existence of a timed simulation relations is a sufficient condition for language inclusion. In the following, we will show that the problem of checking the existence of a timed simulation relation is decidable. We achieve this by converting this check to a finite check on the finitely many equivalence classes of an equivalence relation defined on $\Sigma_{A\|B}$. Our argument generalizes that of [Č92] for the decidability of bisimulations.

Preliminaries Let A be a timed process and for each $x \in X$ let K_x denote the integer such that $x \diamond K_x$ appears in an X-predicate on some edge of A. Also let $Fr(x) = x - \lfloor x \rfloor$, the fractional part of x. The region equivalence [AD94] relation $\equiv$ on Σ_A is defined as follows: $\langle s, \Phi \rangle \equiv \langle t, \Theta \rangle$ iff $s = t$, and for all $x \in X$ either both $\Phi(x)$ and $\Theta(x)$ are larger than K_x, or the following hold

- $\lfloor \Phi(x) \rfloor = \lfloor \Theta(x) \rfloor$, and
- For all $x' \in X$, $Fr(\Phi(x)) \geq Fr(\Phi(x'))$ iff $Fr(\Theta(x)) \geq Fr(\Theta(x'))$, and
- $Fr(\Phi(x)) = 0 \Leftrightarrow Fr(\Theta(x)) = 0$.

The following lemma will enable us to pick an arbitrary representative of an equivalence class in the rest of the paper.

Lemma 9. *If* $\langle s, \Phi \rangle \stackrel{\langle \delta, \mathbf{f}, \mathbf{f}' \rangle}{\rightarrow} \langle t, \Phi' \rangle$ *and* $\langle s, \Phi \rangle \equiv \langle s, \Theta \rangle$*, then* $\langle s, \Theta \rangle \stackrel{\langle \delta', \mathbf{f}, \mathbf{f}' \rangle}{\rightarrow} \langle t, \Theta' \rangle$ *for some* $\delta' > 0$ *and* Θ' *such that* $\langle s, \Phi' \rangle \equiv \langle s, \Theta' \rangle$.

Simulation relations and equivalence classes of $\equiv$. The following theorem stipulates that any simulation relation can be extended to one consisting of equivalence classes of $\equiv$.

[3] The halting problem for non-deterministic 2-counter machines can be reduced to the language inclusion problem in a similar manner to [AD94]. There are two key differences from [AD94]: the non-halting computations of the 2-counter machine are encoded as finite strings and accepting states of automata are mimicked by using output variables. The details of the reduction are too lengthy to present here.

Theorem 10. *Let Ω be a timed-simulation from A to B and let X^A and X^B be the clocks of A and B. Define*

$$\Omega' = \left\{ \langle \sigma^A, \sigma^B \rangle \;\middle|\; \langle \sigma^A, \sigma^B \rangle \equiv \langle \tau^A, \tau^B \rangle \text{ for some } \langle \tau^A, \tau^B \rangle \in \Omega \right\}$$

where $\equiv$ is interpreted over $X^A \cup X^B$. Then, Ω' is a timed-simulation relation from A to B. Furthermore, if Ω is initialized, then so is Ω'.

Theorem 10 is important, because it implies that the maximal timed simulation relation consists of a union of equivalence classes of $\equiv$. In the following, we will develop machinery to show that given A and B, the problem of deciding if a timed simulation from A to B exists can be converted to a condition on the equivalence classes of $\equiv$ which can be checked in finitely many steps. From this we will infer that the problem is decidable.

Symbolic simulations For timed processes A and B, let $\mathcal{EQ}_{A\|B}$ be the set of equivalence classes of $\equiv$ on $\Sigma_{A\|B}$. With $\mathcal{EQ}(\sigma^A, \sigma^B)$, denote the equivalence class that the state $\langle \sigma^A, \sigma^B \rangle \in \Sigma_{A\|B}$ belongs to, where $\sigma^A \in \Sigma_A$ and $\sigma^B \in \Sigma^B$. For $\langle \sigma^A, \sigma^B \rangle$, assume that $u_0, u_1, ..., u_k$ are the fractional parts of clock values of σ^A and σ^B in ascending order. Let k_A be such that u_{k_A} is the largest fractional part of a clock value of σ^A. Define

$$Times(\langle \sigma^A, \sigma^B \rangle) = \{(1 - u_i) \mid k_A \leq i \leq k\} \cup \{(1 - 0.5(u_i + u_{i+1})) \mid k_A \leq i \leq k\}$$

with the convention that $u_{k+1} = 1$. If we imagine the fractional parts of clocks ordered on the real line and if we also include the mid-points between each two adjacent fractional parts, $Times$ consists of the distances between these points and the next integer point.

We say that $\mathcal{X} \subseteq \mathcal{EQ}_{A\|B}$ is a *symbolic simulation* from A to B iff for each $\mathcal{EQ}(\sigma^A, \sigma^B) \in \mathcal{X}$ the following condition is satisfied. For every $t \in Times(\langle \sigma^A, \sigma^B \rangle)$ and a timed event $\gamma = \langle t, \mathbf{f}, \mathbf{f}' \rangle$, if $\sigma^A \xrightarrow{\gamma} \tau^A$, then

(1) $\sigma^B \xrightarrow{\gamma} \tau^B$ for some τ^B such that $\mathcal{EQ}(\tau^A, \tau^B) \in \mathcal{X}$, and
(2) If A can wait in σ^A for time t, then so can B in state σ^B.

Note that, by Lemma 9, the conditions above are independent of what representative is chosen for each equivalence class.

Theorem 11. *Given $\mathcal{X} \subseteq \mathcal{EQ}_{A\|B}$ let $\mathcal{R}_{\mathcal{X}} = \left\{ \langle \sigma^A, \sigma^B \rangle \;\middle|\; \mathcal{EQ}(\sigma^A, \sigma^B) \in \mathcal{X} \right\}$. $\mathcal{R}_{\mathcal{X}}$ is a timed simulation relation from A to B iff $\mathcal{X}$ is a symbolic simulation from A to B.*

Proof. The ($\Rightarrow$) direction is straightforward. We will prove the ($\Leftarrow$) direction. Suppose that $\mathcal{X}$ is a symbolic simulation from A to B. Let $\langle \sigma^A, \sigma^B \rangle \in \mathcal{R}_{\mathcal{X}}$ and let $\sigma^A \xrightarrow{\gamma} \tau^A$ for some timed event $\gamma = \langle t, \mathbf{f}, \mathbf{f}' \rangle$. We need to show that there exists a τ^B such that $\sigma^B \xrightarrow{\gamma} \tau^B$ and $\langle \tau^B, \tau^B \rangle \in \mathcal{R}_{\mathcal{X}}$. If $t \in Times(\langle \sigma^A, \sigma^B \rangle)$ the claim holds by the definition of a symbolic simulation. Otherwise, let t_{max} be the largest element of $Times(\langle \sigma^A, \sigma^B \rangle)$. We will denote the latter with $Times$ in the rest of the proof. There are two cases.

(I) $t \leq t_{max}$. Then, $p < t < q$ for some p and q in $Times$. One of p and q has the form $1 - 0.5(u_i + u_{i+1})$ (call this one r) and the other has the form $1 - u_j$, for some i and j. Let $\sigma^A = \langle s^A, \Phi^A \rangle$ and $\sigma^B = \langle s^B, \Phi^B \rangle$. Define $\theta = \langle r, \mathbf{f}, \mathbf{f}' \rangle$, i.e., the time increment in γ replaced with r. $\sigma^A \xrightarrow{\theta} \tau^A$, since r and t are both less than $1-u_{k_A}$, which means that the same timing predicates are satisfied by Φ^A, Φ^A+t and Φ^A+r. By the fact that $\mathcal{X}$ is a symbolic simulation, and $r \in Times$, $\sigma^B \xrightarrow{\theta} \tau^B$ for some τ^B such that $\langle \tau^A, \tau^B \rangle \in \mathcal{R}_\mathcal{X}$. We claim that $\sigma^B \xrightarrow{\gamma} \tau^B$, which will imply the desired result. This claim follows from the fact that Φ^B+t satisfies the same clock predicates as Φ^B+r, since no clock value crosses an integer value between these two clock valuations.

(II) $t > t_{max}$. Let us consider all points in time when a clock of A takes on an integer value as A waits for time t starting from state σ^A. Clearly there must be a finite number of such points since t is finite and there are a finite number of clocks. More precisely, let $\sigma_1^A, \sigma_2^A, ..., \sigma_k^A$ be the maximal sequence of states such that (1) σ_i^A is reached from σ^A by waiting for time δ_i, and has some clock with an integer value, and (2) $\sigma_k^A \xrightarrow{\langle \delta, \mathbf{f}, \mathbf{f}' \rangle} \tau^A$, where $\delta = t - \delta_k$. We will prove the original claim in ($\Leftarrow$) by induction on k. The case where $k=0$ follows from (I). Let us assume that the claim holds for k. We will show that the claim holds for $k+1$. Consider $\sigma_1^A, \sigma_2^A, ..., \sigma_k^A, \sigma_{k+1}^A$. Since $\mathcal{X}$ is a symbolic simulation, and since A can wait for δ_1 at σ^A to reach σ_1^A, B can wait at σ^B for δ_1 to reach σ_1^B such that $\mathcal{EQ}(\sigma_1^B, \sigma_1^B) \in \mathcal{X}$. Applying the induction assumption to the transition $\sigma_1^A \xrightarrow{\langle t-\delta_1, \mathbf{f}, \mathbf{f}' \rangle} \tau^A$, we conclude that there exists τ^B such that $\sigma_1^B \xrightarrow{\langle t-\delta_1, \mathbf{f}, \mathbf{f}' \rangle} \tau^B$. Thus, at state σ^B, B can wait for a total of t and take a transition to τ^B, such that $\mathcal{EQ}(\tau^A, \tau^B) \in \mathcal{X}$, i.e., $\langle \tau^A, \tau^B \rangle \in \mathcal{R}_\mathcal{X}$ as desired.

Thus, the problem of checking whether a timed process A timed-simulates another process B can be reduced to computing the maximal symbolic simulation relation over the equivalence classes $\mathcal{EQ}_{A\|B}$. For this purpose, any of the existing algorithms for computing simulation (eg. see [KS90, HHK95]) can be adopted to obtain an algorithm with complexity polynomial in the size of $\mathcal{EQ}_{A\|B}$. The size of $\mathcal{EQ}_{A\|B}$ is polynomial in the number of locations and exponential in the number of clocks and the size of binary encodings of the clock constraints. This gives an exponential algorithm for checking timed simulation:

Theorem 12. *Given two timed processes A and B, the problem of checking whether A timed-simulates B is solvable in* EXPTIME.

We conjecture that EXPTIME is also a lower bound for this problem.

5 Verification Using Homomorphisms

In this section, we propose homomorphisms as an alternative way of proving timed refinement. Homomorphisms can roughly be viewed as mappings from the locations of one process to those of the other which conserve the transition structure. There are several reasons for exploring this alternative approach. First, the algorithm of the previous section is computationally expensive, and we do not yet have good heuristics to proceed with an implementation. Second, the tool COSPAN already supports the use of homomorphisms for proving refinements for untimed systems, and thus, for us it is a natural to generalize this to timed systems. Third, homomorphisms capture the user's intuition of the correspondence between the two levels, and it is desirable to make use of this knowledge[4].

For a timed process $A = \langle S, S_0, O, I, X, \alpha, \mu, E \rangle$, let A_{ut} denote the untimed transition structure obtained by removing all the clocks and the timing constraints. More precisely, let $A_{ut} = \langle S, S_0, O, I, E_{ut} \rangle$, where $\langle s, t, \chi \rangle \in E_{ut}$ iff $\langle s, t, \varphi, \chi, Y \rangle \in E$ for some φ and Y. The states of A_{ut} are locations in S and the events of A_{ut} are observation events of A. The runs of A_{ut} are called the untimed runs of A. A mapping $h : S^A \rightarrow S^B$ is said to *preserve untimed behavior* iff for each run $s_0 \xrightarrow{\theta_0} s_1 \xrightarrow{\theta_1} \cdots \xrightarrow{\theta_{k-1}} s_k$ of A_{ut} with $s_0 \in S_0^A$, $h(s_0) \xrightarrow{\theta_0} h(s_1) \xrightarrow{\theta_1} \cdots \xrightarrow{\theta_{k-1}} h(s_k)$ is a run of B_{ut} with $h(s_0) \in S_0^B$. h preserves untimed behavior iff h is a state homomorphism from the reachable locations of A_{ut} to B_{ut}. (See [Kur94] for the definition of homomorphisms for untimed systems). h is a mapping to be supplied by the user capturing the intuition regarding the correspondence of locations at different levels. The syntax of S/R, the input language of COSPAN, allows specifications of such maps. The verifier COSPAN checks, using either an on-the-fly depth-first-search or a BDD-based symbolic search of the product of the two processes, whether the user-supplied mapping h preserves untimed behavior. Let us generalize this to timed behavior.

A mapping $h : S^A \rightarrow S^B$ is said to *preserve timed behavior* iff for each run $\langle s_0, \Phi_0 \rangle \xrightarrow{\gamma_0} \langle s_1, \Phi_1 \rangle \xrightarrow{\gamma_1} \cdots \xrightarrow{\gamma_{k-1}} \langle s_k, \Phi_k \rangle$ of A starting from an initial state, there exist $\Theta_0, \ldots, \Theta_k$ such that $\langle h(s_0), \Theta_0 \rangle \xrightarrow{\gamma_0} \langle h(s_1), \Theta_1 \rangle \xrightarrow{\gamma_1} \ldots \xrightarrow{\gamma_{k-1}} \langle h(s_k), \Theta_k \rangle$ is a run of B that starts from an initial state. Note that if there exist such $\Theta_0, \ldots, \Theta_k$, then they are uniquely determined. It is easy to show that

Proposition 13. *If $h : S^A \rightarrow S^B$ preserves timed behavior, then $A \preceq_L B$.*

Observe that for a mapping to preserve timed or untimed behavior, it must map initial locations to initial locations. We restrict our attention to such mappings.

The remaining part of this section describes a method for checking if a given mapping h preserves timed behavior. This is achieved by converting the problem to an untimed homomorphism check, which can then be performed by a tool such as COSPAN [Kur94]. The basic idea is to construct an untimed transition structure $\mathcal{R}$ in the fashion of the region automaton of [AD94]. This approach has the added advantage that we use the homomorphism checking in COSPAN as a black-box, and thus, we can use either an on-the-fly DFS or a BDD-based symbolic search.

$\mathcal{R}$ has no outputs and the same inputs as A, with an additional input specifying the location that B is at. For notational convenience, an input event of $\mathcal{R}$ will be given as an input event of A together with the old and new locations of B, s^B and t^B, denoted as $\langle \mathbf{f} + s^B, \mathbf{f}' + t^B \rangle$. The location of $\mathcal{R}$ keeps track of the location of A and the equivalence class that the clock evaluations of A and B fall in. Let $[\Phi]$ denote the equivalence class of $\equiv$ that Φ belongs to. The locations

[4] A user-given simulation relation has these properties as well, but is harder to specify.

of $\mathcal{R}$ are given by

$S^{\mathcal{R}} = \{\langle s^A, [\Phi^A, \Phi^B]\rangle \mid s^A \in S^A$ and Φ^A and Φ^B are X^A and X^B valuations$\}\cup\{Bad\}$

The initial locations of $\mathcal{R}$ are all locations of the form $\langle s_0^A, [\mathbf{0}, \mathbf{0}]\rangle$ where $s_0^A \in S_0^A$.

Consider a sequence of (untimed) observation events $\overline{\gamma'} = \gamma'_0, \gamma'_1, ..., \gamma'_{k-1}$ where $\gamma'_i = \langle \mathbf{f}_i, \mathbf{f}'_i\rangle$. Note that there are many timed event sequences $\overline{\gamma} = \gamma_0, \gamma_1, ...,$ γ_{k-1} such that for all i, $\gamma_i = \langle \delta_i, \mathbf{f}_i, \mathbf{f}'_i\rangle$ for some $\delta_i > 0$. Imagine a run of timed processes A and B on $\overline{\gamma}$, where $\overline{s} = s_0^B, s_1^B, ..., s_k^B$ is such that s_i^B is the location of B after timed event γ_i. Each location that $\mathcal{R}$ can reach on $\overline{\gamma'}$ and $\overline{s}$ corresponds to some such $\overline{\gamma}$ and carries the information about the location that A reaches and the equivalence classes that the clock values of A and B are in.

$\mathcal{R}$ has an edge from $\langle s^A, [\Phi^A, \Phi^B]\rangle$ to $\langle t^A, [\Theta^A, \Theta^B]\rangle$ on observation event $\gamma' = \langle \mathbf{f}+s^B, \mathbf{f}'+t^B\rangle$ iff for some timed event $\gamma = \langle \delta, \mathbf{f}, \mathbf{f}'\rangle$ we have (1) $\langle s^A, \Phi^A\rangle \xrightarrow{\gamma}$ $\langle t^A, \tilde{\Theta}^A\rangle$ and (2) $\langle s^B, \Phi^B\rangle \xrightarrow{\gamma} \langle t^B, \tilde{\Theta}^B\rangle$ for some $[\tilde{\Theta}^A, \tilde{\Theta}^B] = [\Theta^A, \Theta^B]$. $\mathcal{R}$ has a transition to location Bad from $\langle s^A, [\Phi^A, \Phi^B]\rangle$ on γ' if for some γ as above (1) holds for some $\tilde{\Theta}^A$ and (2) does not hold for any $\tilde{\Theta}^B$. Bad has a self-loop on all observation events.

By Lemma 9, the choice of the representatives from equivalence classes is immaterial. Also note that, since clock predicates on edges cannot distinguish clock valuations belonging to the same equivalence class, one need only consider values of δ that lead to distinct equivalence classes, i.e., different $[\Phi^A+\delta, \Phi^B+\delta]$.

Theorem 14. *Let h be a mapping from A to B and let $h' : (S^{\mathcal{R}} \setminus \{Bad\}) \to S^B$ given by $h'(\langle s^A, [\Phi^A, \Phi^B]\rangle) = h(s^A)$. h preserves timed behavior iff, assuming that the location of B is specified by h' at all times, (1) h' preserves untimed behavior and (2)Bad is unreachable in $\mathcal{R}$.*

Proof.($\Rightarrow$) Assume that h preserves timed behavior. Let $\overline{\gamma'} = \gamma'_0, \gamma'_1, ..., \gamma'_{k-1}$ be a sequence of observation events for A. We show by induction on the length of $\overline{\gamma'}$ that conditions (1) and (2) of the theorem are not violated by any run on $\overline{\gamma'}$. If $k = 0$, then both claims hold trivially. Assume that we have the desired properties for k. Consider $k+1$. By the induction assumptions on (1) and (2), observe that all runs of $\mathcal{R}$ on $\gamma'_0, \gamma'_1, ..., \gamma'_{k-1}$ where the location of B is given by h' must have the form $t_0, ..., t_k$ where $t_i = \langle s_i, [\tilde{\Phi}_i, \tilde{\Theta}_i]\rangle$. Let us take one such run. By the construction of $\mathcal{R}$ we can find runs $\langle s_0, \Phi_0\rangle \xrightarrow{\gamma_0} \langle s_1, \Phi_1\rangle \xrightarrow{\gamma_1}$ $... \xrightarrow{\gamma_{k-1}} \langle s_k, \Phi_k\rangle$ of A, and $\langle h(s_0), \Theta_0\rangle \xrightarrow{\gamma_0} \langle h(s_1), \Theta_1\rangle \xrightarrow{\gamma_1} ... \xrightarrow{\gamma_{k-1}} \langle h(s_k), \Theta_k\rangle$ of B such that $[\Phi_i, \Theta_i] = [\tilde{\Phi}_i, \tilde{\Theta}_i]$ for $1 \le i \le k$ where $\gamma_i = \langle \delta_i, \mathbf{f}_i, \mathbf{f}'_i\rangle$ for all i. Now consider γ'_k. If $\mathcal{R}$ has a transition on γ'_k, then there must exist $\gamma_k = \langle \delta_k, \mathbf{f}_k, \mathbf{f}'_k\rangle$ such that $\langle s_k, \Phi_k\rangle \xrightarrow{\gamma_k} \langle s_{k+1}, \Phi_{k+1}\rangle$. Let us pick any such transition. Since h preserves timed behavior and since h and $\gamma_0, ..., \gamma_{k-1}$ uniquely determine B's run up to $\langle h(s_k), \Theta_k\rangle$, we must have $\langle h(s_k), \Theta_k\rangle \xrightarrow{\gamma_k}$ $\langle h(s_{k+1}), \Theta_{k+1}\rangle$. Thus, conditions (1) and (2) are satisfied for k as well.

($\Leftarrow$) Assume (1) and (2). Let $\gamma_0, \gamma_1, ..., \gamma_{k-1}$ be a timed event sequence, where $\gamma_i = \langle \delta_i, \mathbf{f}_i, \mathbf{f}'_i\rangle$ for all i. We will proceed by induction on k. For $k = 0$, ($\Leftarrow$) holds by the fact that initial locations are mapped to initial locations by h.

Assume that ($\Leftarrow$) holds for k. Consider $k+1$. Let $\langle s_0, \Phi_0\rangle \xrightarrow{\gamma_0} \langle s_1, \Phi_1\rangle \xrightarrow{\gamma_1} \ldots \langle s_k, \Phi_k\rangle \xrightarrow{\gamma_k} \langle s_{k+1}, \Phi_{k+1}\rangle$ be a run of A. By the induction assumption there exist $\Theta_0, \ldots, \Theta_k$ such that $\langle h(s_0), \Theta_0\rangle \xrightarrow{\gamma_0} \langle h(s_1), \Theta_1\rangle \xrightarrow{\gamma_1} \ldots \xrightarrow{\gamma_{k-1}} \langle h(s_k), \Theta_k\rangle$ is a run of B. Then, by (2) and the construction of $\mathcal{R}$, $t_0 \xrightarrow{\gamma_0'} t_1 \xrightarrow{\gamma_1'} \ldots \xrightarrow{\gamma_{k-1}'} t_k$ is in $\mathcal{R}$, where $\gamma_i' = \langle \mathbf{f}_i, \mathbf{f}_i'\rangle$ and $t_i = \langle s_i, [\Phi_i, \Theta_i]\rangle$ for all i, and the location of B is specified by h'. Since $\langle s_k, \Phi_k\rangle \xrightarrow{\gamma_k} \langle s_{k+1}, \Phi_{k+1}\rangle$ in A, by the construction of $\mathcal{R}$, $t_k \xrightarrow{\gamma_k'} t_{k+1}$ for some t_{k+1}, while the location of B changes to $h(s_{k+1})$ from $h(s_k)$. By (2), *Bad* is unreachable from t_k in this case. Then, by the construction of $\mathcal{R}$ and (1), there must exist Θ_{k+1} such that $\langle h(s_k), \Theta_k\rangle \xrightarrow{\gamma_k} \langle h(s_{k+1}), \Theta_{k+1}\rangle$, which is what was required.

$\mathcal{R}$ has size exponential in the number of clocks in A and B and the constants used in the timing predicates, and linear in $|S_A \times S_B|$. The complexity of checking (1) and (2) in the theorem is then linear in the reachable part of $\mathcal{R}$.

Implementation and Modular Verification Example We have implemented an algorithm within the Cospan [Kur94] system to construct $\mathcal{R}$ from A and B. Cospan supports timing verification and homomorphism checks (see [AK96] for an overview). Our implementation integrates these two capabilities. Given $\mathcal{R}$ and B, conditions (1) and (2) of Theorem 14 can be checked simultaneously by Cospan by an on-the-fly search. If (1) fails, this points to the fact that h does not preserve untimed behavior in the reachable parts of A and B. If (2) fails, on the other hand, then B can not follow A due to timing constraints.

We applied the assumption-guarantee reasoning paradigm and the homomorphism checking algorithm described above to the verification of an asynchronous circuit: the Seitz queue element given in Figure 3. The Seitz queue element is a self-timed circuit that constitutes one stage of a FIFO queue. It was studied in detail in [Rok93]. The functioning of this circuit depends critically on the ranges of gate delays: if the deviation of the gate delays from the specified values exceeds 20%, the circuit does not function correctly [Rok93]. We model each gate by a

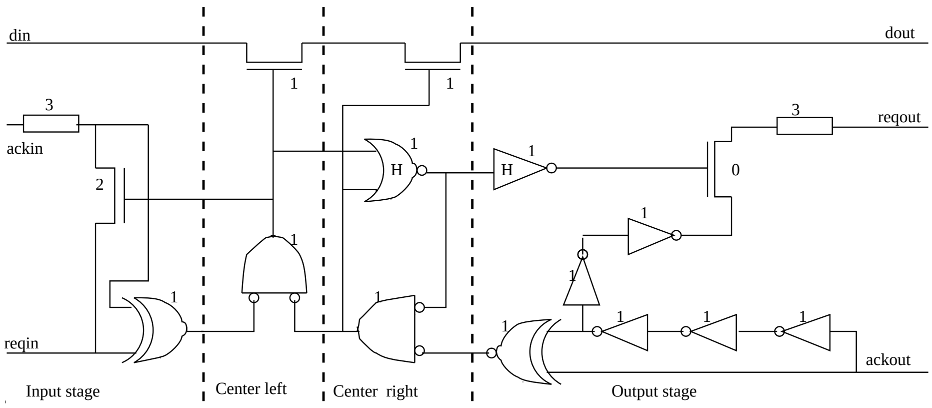

Fig. 3. The Seitz queue element. The delay for each component is indicated next to it.

timed process. Each such process involves one clock, except for gates marked "H" which can pass one hazardous pulse (see [Rok93]), which involve two clocks each. This is not necessarily the most efficient way of modeling this circuit. The main purpose of this example was to demonstrate the techniques that we proposed for assumption-guarantee reasoning and homomorphism checks.

The circuit can be viewed as consisting of four blocks. The block marked "input stage" is responsible for passing on to the rest of the circuit the request for new data to be read in, and signaling an acknowledge to the requester when the queue element is ready to receive the data. Similarly, "output stage" signals a request for passing on the data to the circuit connected to "reqout", and when this request is acknowledged and data passed on, signals to the rest of the circuit that new data can be read in. The "center left" and "center right" stages are responsible for storing the input data and passing it to the output, and isolating the stored data from the data that is requested to be input. The timed process for each block consisted of the composition of the timed processes for its gates. Let $\mathcal{A}_{inp}$, $\mathcal{A}_{cl}$, $\mathcal{A}_{cr}$ and $\mathcal{A}_{out}$ be the timed processes describing the input, center left, center right and output stages respectively.

The verification consisted of two phases. In the first phase, an abstraction for each circuit block was constructed by hand. Let these be denoted by $\mathcal{A}_{inp}^{abs}$, $\mathcal{A}_{cl}^{abs}$, $\mathcal{A}_{cr}^{abs}$ and $\mathcal{A}_{out}^{abs}$. These abstractions described the qualitative functioning of each block, and hid the information about particular signals. In other words, the abstract processes encapsulated the "interface timing behavior" of each block. Untimed state homomorphisms were specified and it was verified as described above that each circuit block is an implementation of its abstraction. For this, we used the assumption-guarantee rule of Proposition 8. For each block, we showed that $\mathcal{A}_{.} \preceq_S \mathcal{A}_{.}^{abs}$ in the context of the abstractions of the rest of the circuit blocks. For instance, for the input stage we showed

$$\mathcal{A}_{inp} \parallel \mathcal{A}_{cl}^{abs} \parallel \mathcal{A}_{cr}^{abs} \parallel \mathcal{A}_{out}^{abs} \quad \preceq_S \quad \mathcal{A}_{inp}^{abs} \parallel \mathcal{A}_{cl}^{abs} \parallel \mathcal{A}_{cr}^{abs} \parallel \mathcal{A}_{out}^{abs} \tag{1}$$

Since the state space of the composition of the abstractions was too big, we actually showed

$$\mathcal{A}_{inp} \parallel \mathcal{A}_{inp}^{env} \quad \preceq_S \quad \mathcal{A}_{inp}^{abs} \parallel \mathcal{A}_{inp}^{env} \tag{2}$$

where $\mathcal{A}_{inp}^{env}$ is a timed process that incorporates the information about $\mathcal{A}_{cl}^{abs} \parallel \mathcal{A}_{cr}^{abs} \parallel \mathcal{A}_{out}^{abs}$ that is necessary for proving (2). We then verified

$$\mathcal{A}_{cl}^{abs} \parallel \mathcal{A}_{cr}^{abs} \parallel \mathcal{A}_{out}^{abs} \quad \preceq_S \quad \mathcal{A}_{inp}^{env} \tag{3}$$

With some manipulation, we can infer (1) from (2) and (3) by Propositions 5 and 6. The same methodology was used for each of the circuit blocks. In some cases, when checking the equivalent of (3) it was possible to "free" some of the processes on the left hand side, i.e., disregard their transition structures and consider them as processes that pose no restriction on their inputs and outputs. This was useful in reducing the computation involved in checking (3). The conclusion from the first phase was that

$$\mathcal{A}_{inp} \parallel \mathcal{A}_{cl} \parallel \mathcal{A}_{cr} \parallel \mathcal{A}_{out} \quad \preceq_S \quad \mathcal{A}_{inp}^{abs} \parallel \mathcal{A}_{cl}^{abs} \parallel \mathcal{A}_{cr}^{abs} \parallel \mathcal{A}_{out}^{abs}$$

In the second phase of the verification, a high level abstraction $\mathcal{A}^{all}$ of the whole circuit was constructed, and it was shown that $\mathcal{A}_{inp}^{abs} \parallel \mathcal{A}_{cl}^{abs} \parallel \mathcal{A}_{cr}^{abs} \parallel \mathcal{A}_{out}^{abs} \preceq_S$

$\mathcal{A}^{all}$ by constructing and verifying a homomorphism as before. We will refer to this as the "overall" abstraction.

The Seitz circuit has the property that the center stage can be repeated n times to construct a FIFO queue of length n. Verification techniques similar to the above can be used to verify such a queue with only a polynomial increase in complexity, contrasted with the exponential dependency of the state space on n.

The data from the experiments is listed in Table 1. The homomorphism check was performed within COSPAN using BDDs as an implicit representation for sets and relations. COSPAN has the capability to return a counterexample if the desired relation does not hold. This was instrumental in obtaining a correct abstraction by iteration. Tests were run on an SGI machine with 1 GB memory. To act as a comparison, we attempted to run timed reachability analysis on

	Num. of clocks	Largest timing constant	Num. of loc.s of $\mathcal{R}$ (in thousands)	BDD nodes (in thousands)	CPU time (seconds)
Input	6	32	.24	15	39
Input (Env)	3	12	6	13	15
Center Left	11	22	18	131	403
Center Right	11	22	19	142	4879
Center (Env)	3	32	.12	7	30
Output	8	22	478	235	1029
Output (Env)	3	12	4.9	15	16
Overall	9	32	101	262	8115

Table 1. (Env) denotes the verification corresponding to equation (3). The same environment abstraction was used for both center blocks. Since timing constants must be integers, a 20% variation in a gate delay of k is represented by specifying the delay to be between $5k$ and $6k$.

the complete circuit without using any compositional rules and abstraction. The computation did not complete: the computer ran out of memory after running for several hours.

6 Conclusion and Future Work

We proposed a framework for hierarchical reasoning about real-time systems. Our framework supports modular and compositional verification rules. We proved that the problem of checking for timed simulation relations is decidable. On the application side, we generalized the notion of state homomorphisms to timed processes, and gave an algorithm to check if a given map between the locations of two processes preserves timed behavior. The proposed algorithm was implemented in COSPAN, and as a case study, we dealt with the Seitz queue circuit.

Our experience was that it can be difficult to specify a correct homomorphism and one usually has to go through several iterations. Therefore, it would be valuable to have the capability to check for timed simulation relations without the user needing to provide the relation. The only concern is that such an algorithm may be too complex. Heuristics can be devised for certain application domains.

The construction of $\mathcal{R}$, the process encapsulating the timing information, is not as efficient as it could be. Any method for making timing verification more efficient, such as partial order reductions, could in principle be incorporated into our current scheme.

We believe that our work will be useful in applications such as asynchronous circuits and hardware software co-design, where abstract descriptions of systems necessarily include timing information.

References

[AD94] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[AH96] R. Alur and T.A. Henzinger. Reactive modules. In *Proceedings of the 11th IEEE Symposium on Logic in Computer Science*, 1996.

[AK96] R. Alur and R.P. Kurshan. Timing analysis in COSPAN. In *Hybrid Systems III*, Lecture Notes in Computer Science. Springer-Verlag, 1996.

[AL91] M. Abadi and L. Lamport. An old-fashioned recipe for real time. In *Real-Time: Theory in Practice, REX Workshop*, LNCS 600, pages 1–27. Springer-Verlag, 1991.

[AL93] M. Abadi and L. Lamport. Composing specifications. *ACM TOPLAS*, 15(1):73–132, 1993.

[DOY94] C. Daws, A. Olivero, and S. Yovine. Verifying ET-LOTOS programs with KRONOS. In *Formal Description Techniques VII, Proceedings of FORTE'94*, pages 227–242, 1994.

[GL94] O. Grümberg and D.E. Long. Model checking and modular verification. *ACM Transactions on Programming Languages and Systems*, 16(3):843–871, 1994.

[GSSL94] R. Gawlick, R. Segala, J. Sogaard-Andersen, and N. Lynch. Liveness in timed and untimed systems. In *Automata, Languages, and Programming, Proceedings of the 21st ICALP*, LNCS 820, pages 166–177. 1994.

[HHK95] M.R. Henzinger, T.A. Henzinger, and P.W. Kopke. Computing simulations on finite and infinite graphs. In *Proceedings of the 36th IEEE Symposium on Foundations of Computer Science*, pages 453–462, 1995.

[KS90] P. Kanellakis and S.A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86(1):43–68, 1990.

[Kur94] R.P. Kurshan. *Computer-aided Verification of Coordinating Processes: the automata-theoretic approach.* Princeton University Press, 1994.

[LA92] N.A. Lynch and H. Attiya. Using mappings to prove timing properties. *Distributed Computing*, 6:121–139, 1992.

[LPY95] K. Larsen, P. Pettersson, and W. Yi. Compositional and symbolic model-checking of real-time systems. In *Proceedings of the 16th IEEE Real-Time Systems Symposium*, 1995.

[LT87] N.A. Lynch and M. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the Seventh ACM Symposium on Principles of Distributed Computing*, pages 137–151, 1987.

[Rok93] T. Rokicki. *Representing and modeling digital circuits.* PhD thesis, Stanford University, 1993.

[Sha92] A.U. Shankar. A simple assertional proof system for real-time systems. In *Proceedings of the 13th IEEE Real-Time Systems Symposium*, pages 167–176, 1992.

[Č92] K. Čerāns. Decidability of bisimulation equivalence for parallel timer processes. In *Proceedings of the Fourth Workshop on Computer-Aided Verification*, LNCS 663, pages 302–315, 1992.