INTEGRATING TIMED CONDITION/EVENT SYSTEMS AND TIMED AUTOMATA FOR THE VERIFICATION OF HYBRID SYSTEMS*

R. HUUCK†, Y. LAKHNECH†, B. LUKOSCHUS†, AND L. URBINA†

S. ENGELL‡, S. KOWALEWSKI‡, AND J. PREUßIG‡

Abstract. In this paper we integrate two different approaches for the specification and verification of timed systems being used in control theory and computer science. These are the timed condition/event systems and the timed automata formalisms. Our main result states that timed condition/event systems can be efficiently transformed into timed automata which then can be analyzed automatically.

1. Introduction. Many computer-controlled systems consist of discrete components interacting with continuously evolving sub-systems. Such systems can be found in avionics, robotics, chemical plants, and other fields. Systems involving discrete as well as continuous or sectionally continuous components are called *hybrid systems.* A very important sub-class of hybrid systems consists of *real-time systems* which are systems depending on hard deadlines and on real-time constraints. A typical real-time constraint is that the response time of the controller to some events generated by the controlled components does not exceed a given amount of time; another typical constraint is that the controlled components do not generate such events too fast. Obviously, the analysis of real-time systems is in general not as involved as the verification of hybrid systems, since these may involve complex differential equations which describe the behavior of the continuous components. However, in many cases hybrid systems can be faithfully considered as real-time systems. Indeed, there are techniques which allow to abstract a hybrid system to obtain a real-time system [HKPS96] such that if the real-time system satisfies the wanted properties, so does the hybrid system too.

Hybrid and real-time systems can often be found in applications where faults can lead to the loss of human lives, pollution of the environment, or important financial loss. Therefore, methods based on mathematical logic should be applied to verify the absence of faults in hybrid systems. An essential ingredient in such a method is a formal model $\mathcal{M}$ which allows to model hybrid systems. In general one can distinguish between operational models for hybrid and real-time systems, on the one hand, and axiomatic models on the other hand. While axiomatic models are usually more suitable for a proof rule-based reasoning, operational models are more suitable for algorithmic reasoning.

In this paper we consider two important mathematical operational models for real-time systems, namely *timed automata* [AD94] and *timed condition/event systems* [EKKP95]. Timed automata are by now a standard model for real-time systems and have been intensively studied from different points of view including the automata theoretic view [AD94], the verification view [ACD90, HNSY94, BL95, Lak96], and the logic view [Wil94, BL95, Lak96]. This led to a variety of tools, for example KRONOS [OY93], UPPAAL [BLL+96], and HyTech [HHWT95], that can analyze real-time systems modeled by timed automata.

It has been demonstrated that timed condition/event systems are suitable for modeling hybrid systems such as chemical plants [Kow95, KEF+95, KP96]. Timed condition event/systems are particularly suitable for a modular specification of the considered system. There are, however, hardly any tools which support the analysis of timed condition/event systems.

Our main objective in the present paper is to show that timed condition/event systems can be transformed into timed automata which then can be analyzed automatically using for instance one of the tools mentioned above. Indeed, we show that, given a timed condition/event system $\mathcal{S}$, we can construct a timed automaton $\Psi(\mathcal{S})$ which admits the same behaviors as $\mathcal{S}$. Moreover, any reachability problem for $\mathcal{S}$, that is, the question whether a particular state of $\mathcal{S}$ is reachable, can be transformed into a reachability problem for $\Psi(\mathcal{S})$.

*This work has been supported by the German Research Council (DFG) within the special program KONDISK (Analysis and Synthesis of Technical Systems with Continuous-discrete Dynamics) under the grants Ro 1122/2 and En 152/19. URL: http://www.informatik.uni-kiel.de/~kondisk/, Email: kondisk@informatik.uni-kiel.de

†Institut für Informatik und Praktische Mathematik der Christian-Albrechts-Universität zu Kiel, Preußerstr. 1–9, D-24105 Kiel, Germany.

‡Lehrstuhl für Anlagensteuerungstechnik, Fachbereich Chemietechnik, Universität Dortmund, Emil-Figge-Straße 70, D-44221 Dortmund, Germany.

Notations. Given a set X, we denote by $2_{\emptyset}^{X}$ the set of non-empty subsets of X. Given a Cartesian product $B_1 \times B_2$ of sets and an element $b = (c, d) \in B_1 \times B_2$, we denote by b_1, respectively, b_2, the projection of b on the first, respectively, second component, i.e. $b_1 = c$ and $b_2 = d$. For a finite-variable function $f : \mathbb{R}_{\geq 0} \to D$, and $t > 0$, we denote by $f(t^-)$ the limit $\lim_{\epsilon \to 0} f(t - \epsilon)$, if it exists. As convention, we define $f(t^-)$ for $t = 0$ as $f(0)$.

2. Condition/Event Systems, their Composition and Real-Time Version.

Condition/event systems (CESs for short) are introduced in [SK91] as a modeling paradigm for discrete event systems (DESs) and have gained increasing interest from the DESs and hybrid systems community (cf. [HR95], [Hol96], [EH96]). The main motivation for developing CESs is to provide a framework for modeling DESs based on block diagrams and signal flows as is standard practice in system theory. A CES allows to model a system as a relation between its input and output signals. The particularity of CESs is that they distinguish between two sorts of signals which correspond to two kinds of interactions between interconnected systems, namely *enabling/disabling* of transitions and, on the other hand, *forcing* of transitions. The first kind of interaction is modeled by *conditions* which can be seen as state variables. Enabling of transitions depends on the values of such conditions. Forcing is implemented by *events*. An enabled transition which is labeled by an event e has to be taken as soon as the event e occurs. When more than one transition is enabled, one of them is nondeterministically chosen.

In this section we recall the definition of condition/event systems and their discrete version. Then, we introduce operators which allow to compose condition/event systems. The operators we consider are variants of the composition operators given in [SK91]. Finally, we define a real-time version of condition/event systems.

2.1. Condition/Event Systems. Condition/event systems describe the behavior of a system in terms of an input-output relation on *condition*, respectively, *event* signals.

DEFINITION 2.1.

- *Given a nonempty finite set U, called* condition alphabet*, of* condition symbols*, a function $s_u : \mathbb{R}_{\geq 0} \to U$ is called* condition signal *over U, if it is right-continuous and finite-variable. A function f is called* finite-variable*, if f has a finite number of discontinuity points in every bounded subinterval of dom(f). The set of all condition signals over U is denoted by $C(U)$.*
- *Given a finite set V, called* event alphabet*, of* event symbols *containing a special* null symbol *$\mathbf{0}_V$, a function $s_v : \mathbb{R}_{\geq 0} \to V$ is called* event signal *over V, if $s_v(0) = \mathbf{0}_V$ and if there are finitely many points $t \in I$ such that $s_v(t) \neq \mathbf{0}_V$, for every bounded interval I in $\mathbb{R}_{\geq 0}$. The set of all event signals over V is denoted by $E(V)$.*

□

Thus, condition signals are piecewise constant functions which associate with each time point a value in U; while event signals take on values indicating events only at single discrete points on the continuous time axis. Henceforth, we write $\mathbf{0}$ instead of $\mathbf{0}_V$, if the alphabet V is understood from the context. Moreover, if V is the product of alphabets $V_1, \cdots, V_n$ then we identify $\mathbf{0}_V$ with $(\mathbf{0}_{V_1}, \cdots, \mathbf{0}_{V_n})$. We call an event e with $e \neq \mathbf{0}$ *proper* event.

DEFINITION 2.2. *A* condition/event system *$\mathcal{S}$ is given by a tuple (U, V, Y, Z, S), where*

- *U is the* input condition alphabet*,*
- *Y is the* output condition alphabet*,*
- *V is the* input event alphabet*,*
- *Z is the* output event alphabet*, and*
- *$S : C(U) \times E(V) \to 2_{\emptyset}^{C(Y) \times E(Z)}$ is the* system behavior function.

□

We identify the function S with a *behavior relation* $S_R \subseteq \mathcal{B}(U, V, Y, Z) = C(U) \times E(V) \times C(Y) \times E(Z)$. An element of $\mathcal{B}(U, V, Y, Z)$ is called *behavior*. Henceforth, given a behavior $s \in \mathcal{B}(U, V, Y, Z)$, we denote by s_u, s_v, s_y, and s_z the signals such that $s = (s_u, s_v, s_y, s_z)$.

2.2. Discrete Condition/Event Systems. Condition/event systems with discrete state realizations are condition/event systems that can be described operationally.

DEFINITION 2.3. *A* discrete condition/event system *(DCES) is given by a tuple $D = (U, V, \mathcal{Q}, Y, Z, f, g, h, q_0)$, where*

- U, V, Y *and* Z *are as above,*
- $\mathcal{Q}$ *is a finite set of* states,
- $f : \mathcal{Q}\times U\times V \to 2^{\mathcal{Q}}_{\emptyset}$ *is the* state transition function,
- $g : \mathcal{Q}\times U \to Y$ *is the* condition output function,
- $h : \mathcal{Q}\times\mathcal{Q}\times V \to Z$ *is the* event output function, *and*
- $q_0 \in \mathcal{Q}$ *is the* initial state.

□

An event input signal can be used to force state transitions in a CES in the sense that a state transition has to take place simultaneously with a synchronizing event indicated by the event input signal. Whereas, a condition input signal can be used to enable/disable transitions.

DEFINITION 2.4. *A DCES is called* well-behaved, *if it satisfies the following conditions:*

- Stuttering*:* $\forall q \in \mathcal{Q}.\ \forall u \in U.\ q \in f(q, u, \mathbf{0})$, *i.e., the system is allowed to stay in the same state as long as the value of the event input signal does not show a proper event.*
- Output triggering*:* $\forall q \in \mathcal{Q}.\ h(q, q, \mathbf{0}) = \mathbf{0}$, *i.e., the system can only output a proper event while remaining in the same state when it receives a proper input event; this is a causality condition.*

□

Henceforth, we only consider well-behaved DCESs and call them simply DCESs. To assign a semantics to DCESs, we introduce the notion of a *run.*

DEFINITION 2.5. *Given a DCES* $D = (U, V, \mathcal{Q}, Y, Z, f, g, h, q_0)$, *we call a* run *of* D *over* $s \in \mathcal{B}(U, V, Y, Z)$ *any right-continuous and finite-variable function* $r : I\!R_{\geq 0} \to \mathcal{Q}$ *which satisfies the following conditions for each* $t \in I\!R_{\geq 0}$*:*

- $r(t) \in f(r(t^-), s_u(t^-), s_v(t))$,
- $s_y(t) = g(r(t), s_u(t))$,
- $s_z(t) = h(r(t^-), r(t), s_v(t))$, *and*
- $r(0) = q_0$.

□

We define the behavior relation S_D of a discrete condition/event system D as the set consisting of all behaviors $s \in \mathcal{B}(U, V, Y, Z)$ such that there exists a run of D over s. We also write $\mathcal{S}_D$ to denote the condition/event system (U, V, Y, Z, S_D).

2.3. Composition of Condition/Event Systems. When a complex system model is designed, it is often divided into smaller parts, called *modules*, which more or less represent "real world" parts of the system. Interaction of these parts is achieved by connections between them. When using condition/event systems as the underlying model, these connections are condition and event signals.

When dealing with a network of interconnected condition/event systems, one is interested in the semantics of this whole network. There are mainly two ways of defining this. One way is to define a *global* semantics which takes all single systems and a description of their interconnections and computes the network semantics following a given rule. In this case, the network is composed in a single step, and hence, reflects a flat structure. The other way is to define composition operators on condition/event systems which take some of the systems and descriptions of how they are connected and replace these systems by a single one having their "overall" behavior. The result is a network of systems equivalent to the original one, but with a reduced number of single systems. Repeated application of these operators finally results in one condition/event system having the overall semantics of the original network. Thus, a network described using such operators is composed in several steps to a single system and reflects a certain composition structure.

In [SK91] two composition operators are defined: the *cascade interconnection* and the *feedback connection.* In the following, we introduce and discuss slight variants of these operators.

2.3.1. Cascade Interconnection. Given two condition/event systems, the cascade interconnection describes the connection of the first system's outputs with the second system's inputs. Obviously, the alphabets of the connected signals have to be equal. The result of this connection is a system having the first system's inputs and the second system's outputs, and as semantics we have the composition of the single systems' behaviors. In contrast to Sreenivas and Krogh's definition, according to our definition, the "intermediate" signals between the two systems remain visible in the resulting system (see Fig. 2.1).

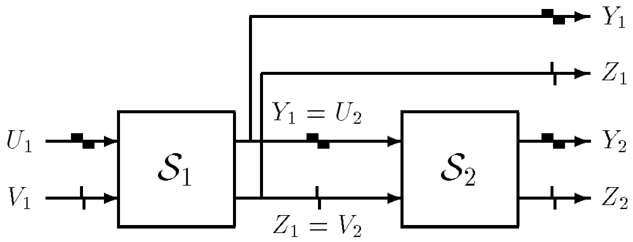

FIG. 2.1. *Condition/Event System Cascade Interconnection of $\mathcal{S}_1$ and $\mathcal{S}_2$*

First, we define the cascade interconnection for condition/event systems. Given condition/event systems $\mathcal{S}_i = (U_i, V_i, Y_i, Z_i, S_i)$, for $i = 1, 2$, we say that $\mathcal{S}_1$ and $\mathcal{S}_2$ are *compatible*, if $Y_1 = U_2$ and $Z_1 = V_2$.

DEFINITION 2.6. *Let $\mathcal{S}_i = (U_i, V_i, Y_i, Z_i, S_i)$, for $i = 1, 2$, be two compatible C/E systems. A condition/event system $\mathcal{S} = (U, V, Y, Z, S)$ is called* cascade interconnection *of $\mathcal{S}_1$ and $\mathcal{S}_2$, if and only if the following conditions hold:*

- *$U = U_1$, $V = V_1$, $Y = Y_1 \times Y_2$, $Z = Z_1 \times Z_2$,*
- *for any $s \in \mathcal{B}(U, V, Y, Z)$, we have $s \in S$ iff $(s_{y\,1}, s_{z\,1}) \in S_1(s_u, s_v)$ and $(s_{y\,2}, s_{z\,2}) \in S_2(s_{y\,1}, s_{z\,1})$.*

□

Having defined the cascade interconnection for condition/event systems, we are interested in the following question: If we have two compatible *discrete* condition/event systems, can we always construct a *discrete* condition/event system that is cascade interconnection of them? The answer is positively answered in [SK91]. For the sake of self-containment we restate the theorem and recall its proof.

THEOREM 2.7. *[SK91] Given two compatible DCESs $D_i = (U_i, V_i, \mathcal{Q}_i, Y_i, Z_i, f_i, g_i, h_i, q_{0,i})$, for $i = 1, 2$, a DCES D can be effectively constructed such that $\mathcal{S}_D$ is a cascade interconnection of $\mathcal{S}_{D_1}$ and $\mathcal{S}_{D_2}$.* □

Proof. Let D_1 and D_2 be given as above. We construct a DCES D such that $\mathcal{S}_D$ is cascade interconnection of $\mathcal{S}_{D_1}$ and $\mathcal{S}_{D_2}$. Let $D = (U, V, \mathcal{Q}, Y, Z, f, g, h, q_0)$ be the DCES with $U = U_1$, $V = V_1$, $Y = Y_1 \times Y_2$, $Z = Z_1 \times Z_2$, $\mathcal{Q} = \mathcal{Q}_1 \times \mathcal{Q}_2$, $q_0 = (q_{0,1}, q_{0,2})$, and for all $q = (q_1, q_2) \in \mathcal{Q}$, $q' = (q'_1, q'_2) \in \mathcal{Q}$, $u \in U$, $v \in V$,

- $q' \in f(q, u, v)$ iff $q'_1 \in f_1(q_1, u, v)$ and $q'_2 \in f_2\big(q_2, g_1(q_1, u), h_1(q_1, q'_1, v)\big)$,
- $g(q, u) = \Big(g_1(q_1, u), g_2\big(q_2, g_1(q_1, u)\big)\Big)$,
- $h(q, q', v) = \Big(h_1(q_1, q'_1, v), h_2\big(q_2, q'_2, h_1(q_1, q'_1, v)\big)\Big)$.

It is easy to check that $\mathcal{S}_D$ is cascade interconnection of $\mathcal{S}_{D_1}$ and $\mathcal{S}_{D_2}$. □

Finally, we define the cascade interconnection operator for discrete condition/event systems.

DEFINITION 2.8. *Let D_1 and D_2 be two compatible DCESs as in Theorem 2.7. The DCES D gained from the construction above is called* discrete cascade interconnection *of D_1 and D_2, denoted by $D_1 \to D_2$.* □

2.3.2. Feedback Connection. Given a condition/event system where the input and output alphabets are equal, for both condition and event signals, the feedback connection describes the connection of the system's outputs with the system's inputs ("feedback loop"). The signals in this loop remain visible. In contrast to Sreenivas and Krogh's definition, we have additional inputs and outputs in the system (see Fig. 2.2). As we discuss later this makes the condition under which the feedback connection of a condition/event system is well-defined slightly more complicated.

DEFINITION 2.9. *Let $\mathcal{S} = (U_1 \times U_2, V_1 \times V_2, Y_1 \times Y_2, Z_1 \times Z_2, S)$ be a condition/event system with $Y_2 = U_2$ and $Z_2 = V_2$. A condition/event system $\mathcal{S}' = (U, V, Y, Z, S')$ is called* feedback connection *of $\mathcal{S}$, if and only if the following conditions hold:*

- $U = U_1$, $V = V_1$, $Y = Y_1 \times Y_2$, $Z = Z_1 \times Z_2$,
- *for any* $s \in \mathcal{B}(U, V, Y, Z)$, *we have* $s \in S'$ *iff* $\big((s_u, s_{y_2}), (s_v, s_{z_2}), s_y, s_z\big) \in S$.

□

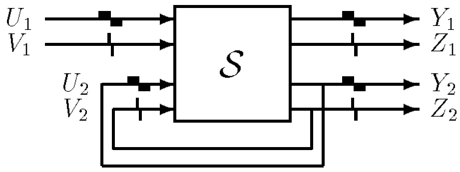

FIG. 2.2. *Condition/Event System Feedback Connection of* $\mathcal{S}$

In contrast to cascade interconnection not every *discrete* condition/event system (with adequate alphabets) can be transformed to a *discrete* condition/event system that is a feedback connection of the original one. This is the case because there is not always a solution to the equations $g(x, u)_2 = u_2$ and $h(x, x', v)_2 = v_2$. Therefore, we have to require that there exists a *unique* solution to each of the equations. However, it would be sufficient to require that there is *at least one* solution to each of the equations. This weaker requirement is used in Sreenivas and Krogh's definition, leading to a larger state space $\mathcal{Q}$ in the new system. For our construction, we use the *unique solution property.*

THEOREM 2.10. *Given a DCES* $D = (U_1 \times U_2, V_1 \times V_2, \mathcal{Q}, Y_1 \times Y_2, Z_1 \times Z_2, f, g, h, q_0)$, *with* $Y_2 = U_2$ *and* $Z_2 = V_2$, *a DCES* D' *can be effectively constructed such that* $\mathcal{S}_{D'}$ *is feedback connection of* $\mathcal{S}_D$, *if the following* unique solution property *holds for* D:

- *For all* $q \in \mathcal{Q}$ *and* $u_1 \in U_1$ *there exists a unique* $y_2 \in Y_2$ *such that* $g\big(q, (u_1, y_2)\big)_2 = y_2$.
- *For all* $q, q' \in \mathcal{Q}$ *and* $v_1 \in V_1$ *there exists a unique* $z_2 \in Z_2$ *such that* $h\big(q, q', (v_1, z_2)\big)_2 = z_2$.

□

Proof. Let D be given as above and let the unique solution property hold. Let $G(q, u_1)$ and $H(q, q', v_1)$ be the unique solutions y_2 and z_2, respectively. We construct a DCES D' such that $\mathcal{S}_{D'}$ is a feedback connection of $\mathcal{S}_D$. Let $D' = (U_1, V_1, \mathcal{Q}, Y_1 \times Y_2, Z_1 \times Z_2, f', g', h', q_0)$ be the DCES with

- $q' \in f'(q, u_1, v_1)$ if and only if $q' \in f\Big(q, \big(u_1, G(q, u_1)\big), \big(v_1, H(q, q', v_1)\big)\Big)$,
- $g'(q, u_1) = g\Big(q, \big(u_1, G(q, u_1)\big)\Big)$,
- $h'(q, q', v_1) = h\Big(q, q', \big(v_1, H(q, q', v_1)\big)\Big)$,

for all $q, q' \in \mathcal{Q}$, $u_1 \in U_1$, $v_1 \in V_1$.

It is easy to check that $\mathcal{S}_{D'}$ is feedback connection of $\mathcal{S}_D$. □

Finally, we define the feedback connection operator for discrete condition/event systems.

DEFINITION 2.11. *Let* D *be given as in Theorem 2.10. The DCES* D' *gained from the construction above is called the* discrete feedback connection *of* D, *denoted by* $D \hookleftarrow$. □

2.4. Timed Condition/Event Systems. Since the behavior of condition/event systems is defined over continuous time, it is straightforward to incorporate quantitative timing or continuous dynamics. Because the communication between condition/event systems is already defined in terms of continuous time signals, it is a natural way to add so-called *c/e timers* to the model as a special class of c/e systems communicating with the original untimed ones. A c/e timer behaves like an alarm clock, that is, it counts the elapsed time since the last time the timer has been reset. Moreover, time can be compared to certain thresholds and the timer sends an alarm if some specified threshold is reached. To formally define the behavior of a c/e timer we use a function τ_θ, which models for a given event signal s the time elapsed since the last reset point.

DEFINITION 2.12. *Let* s *be an event signal. Let* $\alpha = t_0 t_1 \cdots$ *be the maximal ordered finite or infinite sequence of time points such that* $s(t_i) \neq \mathbf{0}$. *Let* $t_{-1} = 0$ *and let* k *be the length of* α *(thus* $k \in \omega \cup \{\omega\}$*). A* timer function $\tau_\theta(s) : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ *associated to* s *is determined by*

$$\tau_\theta(s)(t) = \begin{cases} t - t_{j-1} & \text{if } \exists j \geq 0.\ t_{j-1} < t \leq t_j \wedge j < k \\ t - t_j & \text{if } k \in \omega \wedge j = k - 1 \wedge t > t_j \end{cases} .$$

□

Intuitively, we interpret $\tau_\theta(s)(t)$ as the value of the timer at time point t. Indeed, since the occurrence of a proper event resets the timer to zero, between two resets time increases by the distance between the times at which these resets occur.

DEFINITION 2.13. *A* condition/event timer *(c/e timer) θ is a pair consisting of a condition/event system*[1] *$(Res_\theta, Int_\theta, Ala_\theta, S_\theta)$ and a vector $T_\theta = (T_\theta^1, \cdots, T_\theta^m)$ of constants $T_\theta^i \in I\!N_{>0}$ such that $T_\theta^1 < \cdots < T_\theta^m$ and*

- *$Res_\theta = \{$ "$\theta := 0$", $\mathbf{0}\}$ is the input event alphabet,*
- *$Int_\theta = \{$"$\theta < T_\theta^1$", "$T_\theta^1 \leq \theta < T_\theta^2$", $\cdots$, "$T_\theta^{m-1} \leq \theta < T_\theta^m$", "$\theta \geq T_\theta^m$"$\}$ is the output condition alphabet,*
- *$Ala_\theta = \{$"$\theta = T_\theta^i$" $\mid i = 1, \cdots, m\} \cup \{\mathbf{0}\}$ is the output event alphabet, and*
- *for every $(s_v, s_y, s_z) \in \mathcal{B}(Res_\theta, Int_\theta, Ala_\theta)$, $(s_y, s_z) \in S_\theta(s_v)$ iff for every $t \in I\!R_{\geq 0}$ the signal s_y satisfies*

$$s_y(t) = \begin{cases} \text{“}\theta < T_\theta^1\text{”} & \text{if } \tau_\theta(s_v)(t) < T_\theta^1 \\ \text{“}T_\theta^i \leq \theta < T_\theta^{i+1}\text{”} & \text{if } i \in \{1, \cdots, m-1\} \wedge \tau_\theta(s_v)(t) \in [T_\theta^i, T_\theta^{i+1}) \\ \text{“}\theta \geq T_\theta^m\text{”} & \text{if } \tau_\theta(s_v)(t) \geq T_\theta^m \end{cases}$$

and the signal s_z satisfies

$$s_z(t) = \begin{cases} \text{“}\theta = T_\theta^i\text{”} & \text{if } i \in \{1, \cdots, m\} \wedge \tau_\theta(s_v)(t) = T_\theta^i \\ \mathbf{0} & \text{otherwise} \end{cases} .$$

□

In words, a c/e timer θ is a condition/event system with specific input and output signals and an additional set of thresholds $T_\theta = (T_\theta^1, \cdots, T_\theta^m)$. The behavior of the c/e timer is completely determined by its input event signal. A c/e timer behaves as follows: The internal time given by the timer function is always compared to the thresholds and the result is passed to the environment. The output condition signal always states the interval the timer function is in at the moment, i.e. it produces the conditions "$\theta < T_\theta^1$", "$T_\theta^i \leq \theta < T_\theta^{i+1}$", and "$\theta \geq T_\theta^m$". The output event signal alarms the environment by generating the output event "$\theta = T_\theta^i$" when a specific threshold T_θ^i is reached. The value of the timer function is set to 0 when the c/e timer receives "$\theta := 0$" as input event.

Since a c/e timer is completely determined by its threshold vector we identify a c/e timer with this vector. Moreover, when we consider a set $\{\theta_i \mid 1 \leq i \leq n\}$ of c/e timers, we denote by $T_{\theta_i} = (T_i^1, \cdots, T_i^{m_i})$ the threshold vector of the c/e timer θ_i, and by $\mathrm{Res}_i, \mathrm{Int}_i, \mathrm{Ala}_i$ its signal alphabets. For the sake of presentation we use the following abbreviations: U^+ for $U \times \Pi_{i=1}^n \mathrm{Int}_i$, V^+ for $V \times \Pi_{i=1}^n \mathrm{Ala}_i$, and Z^+ for $Z \times \Pi_{i=1}^n \mathrm{Res}_i$. Then, we define timed condition/event systems as condition/event systems which are determined by a set of timers and a discrete condition/event system D. The behavior of the discrete condition/event system D depends on the output signals of the timers and is able to reset these timers by sending a reset signal.

DEFINITION 2.14. *A* timed condition/event system *(TCES) is a tuple $\mathcal{T} = (U, V, Y, Z, D, \Theta)$, where*

- *$\Theta = \{\theta_i \mid 1 \leq i \leq n\}$ is a set of c/e timers and*
- *$D = (U^+, V^+, \mathcal{Q}, Y, Z^+, f, g, h, q_0)$ is a discrete condition/event system.*

□

In order to assign a semantics to a timed condition/event system, we associate to each timed condition/event system $\mathcal{T}$ a condition/event system $\mathcal{S_T}$ with behavior function $S_\mathcal{T}$.

DEFINITION 2.15. *Given a timed condition/event system $\mathcal{T} = (U, V, Y, Z, D, \Theta)$. Its* behavior function *$S_\mathcal{T} : C(U) \times E(V) \to 2_\emptyset^{C(Y) \times E(Z)}$ is defined as follows:*

For any $s \in \mathcal{B}(U, V, Y, Z)$, we have $(s_y, s_z) \in S_\mathcal{T}(s_u, s_v)$ iff there exist $s'_u \in \Pi_{i=1}^n C(Int_i)$, $s'_v \in \Pi_{i=1}^n E(Ala_i)$, and $s'_z \in \Pi_{i=1}^n E(Res_i)$, such that the following conditions are satisfied:

[1]The input condition alphabet of this system is a singleton and therefore omitted.

- $(s'_{u_i}, s'_{v_i}) \in S_{\theta_i}(s'_{z_i})$, *for each* $i = 1, \cdots, n$, *and*
- $(s_y, (s_z, s'_z)) \in S_D((s_u, s'_u), (s_v, s'_v))$.

□

In the above definition the first condition ensures that the timer signals associated to the c/e timers are legal and the second condition ensures the same for the DCES. Thus, a timed condition/event system is a connection between a finite set of timers, each one behaving like a condition/event system and a DCES also behaving like a condition/event system. The connection is established as follows: The output signals of the timers are part of input signals of the TCES and the input event signals of the timers are part of the output event signals of the TCES. Since the signals from timers are not observable for an environment, we call them *internal signals*. On the other hand, we call the remaining observable signals *external*. We use these notions for the corresponding alphabets, too. One can easily prove that timed condition/event systems are condition/event systems.

LEMMA 2.16. *For any timed condition/event system* $\mathcal{T}$, $\mathcal{S}_{\mathcal{T}}$ *is a condition/event system.* □

2.5. Composition of Timed Condition/Event Systems. Timed condition/event systems can be composed using a feedback and a cascade operator in the same way as condition/event systems. In the full paper [HLL+98] we show that Theorem 2.7 and Theorem 2.10 can be extended to timed condition/event systems.

3. Timed Automata. *Timed automata* (TAs) [AD94] are finite state automata extended by real-valued variables. Semantically, timed automata allow two kinds of transitions, namely *discrete* and *time-pass transitions*. The discrete transitions are specified by the edges between the control locations and guarded by conditions on the variables. A discrete transition can only be taken when the values of the variables satisfy its guard. Moreover, when a discrete transition is taken, some of the variables may be reset. While control resides at a control location of the automaton time can pass and the values of the variables increase by the amount of time that passed. Therefore, these real-valued variables are called *clocks*.

The verification problem for timed automata has been intensively investigated and many interesting logics have been identified for which this problem is decidable [ACD90, AFH96, BL95, BLY96, Lak96]. Also the synthesis problem for timed automata is decidable [MPS95].

In this section we define *timed words* as well as *timed automata* which recognize sets of timed words. Since timed condition/event systems do not include any acceptance condition we only consider *safety timed automata* [HNSY94].

3.1. Timed Languages. Let Σ be an alphabet. The behavior of a real-time system corresponds to a timed word over Σ. As we are interested in a *dense-time* model, we choose the set $\mathbb{R}_{\geq 0}$ of non-negative reals as time domain.

DEFINITION 3.1. *A* time sequence $\tau = (\tau_i)_{i\in\omega}$ *is an infinite sequence of time values* $\tau_i \in \mathbb{R}_{\geq 0}$ *that satisfies the following conditions:*

- Monotonicity: τ *increases monotonically; that is,* $\tau_{i+1} > \tau_i$, *for every* $i \in \omega$.
- Divergence: τ *diverges, that is, for every* $t \in \mathbb{R}_{\geq 0}$ *there is* $i \in \omega$ *such that* $\tau_i \geq t$.

A timed word *over an alphabet* Σ *is a pair* (σ, τ), *where* σ *is an infinite word in* $(2^{\Sigma}_{\emptyset})^{\omega}$ *and* τ *is a time sequence. A* timed language *over* Σ *is a set of timed words over* Σ. □

The intuitive interpretation of a timed word (σ, τ) is that at time point τ_i we observe the symbols in σ_i.

3.2. Safety Timed Automata. Let $\mathcal{V}$ be a set of real-valued variables called *clocks*. A *constraint* γ over $\mathcal{V}$ is a boolean combination of formulas of the form $x\#c$, where $x \in \mathcal{V}$, $\# \in \{<, \leq, >\}$, and $c \in \mathbb{N}$. Let $\mathcal{C}(\mathcal{V})$ denote the set of constraints over $\mathcal{V}$. A *valuation* over $\mathcal{V}$ is a function $\nu : \mathcal{V} \to \mathbb{R}_{\geq 0}$. The satisfaction relation $\nu \models \gamma$ between valuations and constraints is defined as usual. Given a valuation ν and a set $R \subseteq \mathcal{V}$, we denote by $\nu[R \mapsto 0]$ the valuation ν' which associates with each clock in R the value 0 and coincides with ν on all the other clocks. For $t \in \mathbb{R}_{\geq 0}$ we denote by $\nu + t$ the valuation ν' such that $\nu'(x) = \nu(x) + t$, for all $x \in \mathcal{V}$. Moreover, we denote by ν_0 the clock valuation which maps every clock to 0. We are now ready to define safety timed automata.

DEFINITION 3.2. *A safety timed automaton (TA) over an alphabet Σ is given by a tuple $A = (\mathcal{Q}, q_0, \mathcal{X}, \mathcal{E}, Urg)$, where*

- *$\mathcal{Q}$ is a finite set of* locations*,*
- *$q_0 \in \mathcal{Q}$ is the* initial location*,*
- *$\mathcal{X}$ is a finite set of* clocks*,*
- *$\mathcal{E} \subseteq \mathcal{Q} \times \mathcal{C}(\mathcal{X}) \times 2^{\Sigma}_{\emptyset} \times 2^{\mathcal{X}} \times \mathcal{Q}$ is a set of* edges*, and*
- *$Urg \subseteq \mathcal{E}$ is a set of* urgent edges*.* □

An edge $(q, \gamma, \sigma, \xi, q')$ stands for a transition from location q to location q' which is guarded by the constraint γ, labeled by σ, and which resets the clocks in ξ. Intuitively, an urgent transition has a higher priority than a non-urgent transition and also higher than time passing, i.e., when urgent and non-urgent transitions are enabled at the same time, then one of the urgent transitions is taken. Moreover, time is not allowed to pass beyond a time point at which an urgent transition is enabled.

A *state* of a timed automaton A is a pair (q, ν) where q is a location of A and ν is a valuation of the clocks of A. Let Ξ_A denote the set consisting of the states of A. To define the semantics of safety timed automata we associate to every timed automaton A a labeled transition system T_A having the same states as A. Therefore, we introduce the following definitions.

DEFINITION 3.3. *A* labeled transition system *is given by a tuple $\mathrm{T} = (\mathcal{X}, L, q_0, \rightarrow)$, where*

- *$\mathcal{X}$ is a set of* states*,*
- *L is a set of* labels*,*
- *$q_0 \in \mathcal{X}$ is the* initial state*, and*
- *$\rightarrow \subseteq \mathcal{X} \times L \times \mathcal{X}$ is the* transition relation*.*

A run *r* of T *over a word $(l_i)_{i\in\omega}$ is a sequence $(q'_i)_{i\in\omega}$ such that $q_0 = q'_0$ and $q'_i \xrightarrow{l_i} q'_{i+1}$, for every $i \in \omega$. The* language of T*, denoted by $\mathcal{L}(\mathrm{T})$, is the set of words α such that there exists a run of* T *over α.* □

To relate timed automata and transition systems we need to introduce following definition. Let A be a timed automaton and $L = 2^{\Sigma}_{\emptyset} \times I\!R_{>0}$. The relation $\rightsquigarrow \subseteq \Xi_A \times L \times \Xi_A$ is defined as follows: $(q, \nu) \overset{(\sigma,\delta)}{\rightsquigarrow} (q', \nu')$ iff there is an edge $(q, \gamma, \sigma, \xi, q')$ in A such that the following conditions hold.

- $\nu + \delta$ satisfies the constraint γ and
- $\nu' = (\nu + \delta)[\xi \mapsto 0]$.

We use the relation $\rightsquigarrow$ to define the semantics of timed automata.

DEFINITION 3.4. *With every timed automaton A we associate a labeled transition system $\mathrm{T}_A = (\Xi_A, L, (q_0, \nu_0), \rightarrow)$, where $L = 2^{\Sigma}_{\emptyset} \times I\!R_{>0}$ is the set of labels of T_A and the transition relation $\rightarrow \subseteq \Xi_A \times L \times \Xi_A$ is defined as follows:*

$(q, \nu) \xrightarrow{(\sigma,\delta)} (q', \nu')$ iff there is an edge $e = (q, \gamma, \sigma, \xi, q')$ in A such that the following conditions are satisfied:

- *$(q, \nu) \overset{(\sigma,\delta)}{\rightsquigarrow} (q', \nu')$,*
- *if $e \in Urg$, then there is no $(q, \gamma', \sigma', \xi', q'') \in Urg$, valuation ν'' and $\delta' < \delta$ such that $(q, \nu) \overset{(\sigma',\delta')}{\rightsquigarrow} (q'', \nu'')$, and*
- *if $e \notin Urg$, then there is no $(q, \gamma', \sigma', \xi', q'') \in Urg$, valuation ν'' and $\delta' \leq \delta$ such that $(q, \nu) \overset{(\sigma',\delta')}{\rightsquigarrow} (q'', \nu'')$.*

□

We define the language $L(A)$ of a timed automaton A as the set of timed words $(\sigma_i, t_i)_{i\geq 0}$ such that there exists $(\delta_i)_{i\geq 0}$ with $(\sigma_i, \delta_i)_{i\geq 0} \in \mathrm{T}_A$ and $t_i = \Sigma_{j\leq i}\delta_j$.

Later we will apply the standard notion of bisimulation [Par81, Mil89] to transition systems to compare timed automata. Let $\mathrm{T} = (\mathcal{Q}, L, q_0, \rightarrow_{\mathrm{T}})$ and $\mathrm{T}' = (\mathcal{Q}', L, q'_0, \rightarrow_{\mathrm{T}'})$ be two given labeled transition systems.

DEFINITION 3.5. *A symmetric relation $R \subseteq (\mathcal{Q} \times \mathcal{Q}') \cup (\mathcal{Q}' \times \mathcal{Q})$ is called a* bisimulation *between* T *and* T$'$*, if for every $q, \bar{q} \in \mathcal{Q}$, $q' \in \mathcal{Q}'$, and $l \in L$ with $(q, q') \in R$ the following condition is satisfied:*

- *Whenever $q \xrightarrow{l}_{\mathrm{T}} \bar{q}$ there exists a state $\bar{q}'$ such that $q' \xrightarrow{l}_{\mathrm{T}'} \bar{q}'$ and $(\bar{q}, \bar{q}') \in R$.*

□

The largest bisimulation between T and T$'$ (which exists by Tarski's theorem on fixpoints) is denoted by $\sim$. We say that T and T$'$ are *bisimilar*, denoted by $\mathrm{T} \sim \mathrm{T}'$, if $q_0 \sim q'_0$. We say that two timed automata A and B are *timed bisimilar*, if $\mathrm{T}_A \sim \mathrm{T}_B$.

Composing timed automata. Timed automata are composed using the parallel operator $||$. Timed automata A and A' over alphabets Σ and Σ' synchronize on common events when put in parallel. The formal semantics of $A \parallel A'$, where A and A' have disjoint sets of clocks, is given by the timed automaton $B = (\mathcal{Q} \times \mathcal{Q}', (q_0, q_0'), \mathcal{X} \cup \mathcal{X}', \mathcal{E}_B, \mathrm{Urg}_B)$ over the alphabet $\Sigma \cup \Sigma'$ such that $e = ((q_1, q_1'), \gamma \wedge \gamma', \sigma \cup \sigma', \xi \cup \xi', (q_2, q_2')) \in \mathcal{E}_B$ iff $e_1 = (q_1, \gamma, \sigma, \xi, q_2) \in \mathcal{E}$, $e2 = (q_1', \gamma', \sigma', \xi', q_2') \in \mathcal{E}'$, and $\sigma \cap L' = \sigma' \cap L$. Moreover, $e \in \mathrm{Urg}_B$ iff $e_1 \in \mathrm{Urg}_1$ or $e_2 \in \mathrm{Urg}_2$.

4. From Timed Condition/Event Systems to Timed Automata. In this section, we show that timed condition/event systems can be effectively transformed into timed automata while preserving the same real-time behavior. To do so, we present a function Ψ that transforms timed condition/event systems into timed automata. A theoretical consequence of this result is that timed automata are at least as expressive as TCESs, i.e. every system that can be modeled by a TCES can also be modeled by a timed automaton. Here, we take language inclusion as our criterion for comparing expressiveness. A practical approach of this result is that existing tools, like KRONOS [OY93], UPPAAL [BLL+96], and HyTech [HHWT95], for the analysis of timed automata, can now be used to analyze TCESs. The transformation Ψ we present is, however, not very efficient since it transforms a TCES into a timed automaton which has much more discrete states. Therefore, we present a more efficient transformation Ψ' which avoids this increase of discrete states whenever possible. The relationship between both transformations is that they produce for a given TCES two timed automata that are timed bisimilar. It is also worth to mention that the presented transformations also allow to reduce a reachable problem for TCESs, that is, given a TCES and a discrete state whether there is a run of the given TCES that reaches this discrete state, into a reachability problem for timed automata. There are tools that can solve the latter problem efficiently.

Relating timed words and TCES-behaviors. Since the behavior of a TCES is defined as a relation associating to input signals a set of output signals whereas the language of a TA is a set of timed words, we first have to relate both concepts. This is done by the function β, which assigns to each timed word $w = (\sigma_i, t_i)_{i \in \omega}$ a set of behaviors such that for each behavior $s \in \beta(w)$ the following conditions are true:

- for all $t \in [0, t_0)$ we have $s(t) = \sigma_0$ and
- for all $t \in [t_i, t_{i+1})$, $i \geq 0$, we have $s(t) = \sigma_i$.

The mapping β is pointwisely extended to sets of behaviors.

The basic transformation. The transformation function Ψ from TCES into timed automata and its correctness are illustrated in Fig. 4.1. That is, if B is a timed condition/event system and $L(\Psi(\mathrm{B}))$ is the language of the resulting timed automaton $\Psi(\mathrm{B})$, then the resulting relation $\beta(L(\Psi(\mathrm{B})))$ equals the relation S_{B} generated by B, i.e. $\beta(L(\Psi(\mathrm{B}))) = S_{\mathrm{B}}$. In [KEP+97] we illustrate the analysis of timed condition/event systems by tools for timed automata using the railroad crossing example.

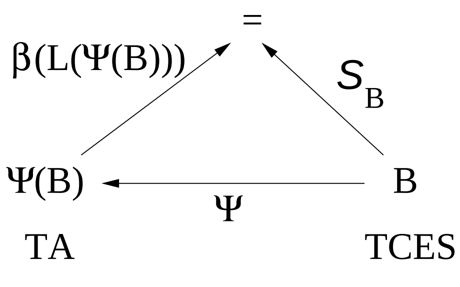

FIG. 4.1. *Transformation from Timed Condition/Event Systems to Timed Automata*

Formally, we can state the following result:

THEOREM 4.1. *There exists an effective function Ψ such that for every timed condition/event system $\mathcal{T}$, we have $\beta(L(\Psi(\mathcal{T}))) = S_{\mathcal{T}}$.* □

In the sequel, we present the transformation function Ψ and explain its correctness. First of all, the transformation Ψ has the following characteristics: For each c/e timer θ of the timed condition/event system there exists exactly one clock x in $\Psi(\mathcal{T})$. This clock x is reset if and only if θ is reset. Furthermore, Ψ transforms $\mathcal{T}$ with the set of states $\mathcal{Q}$ into a timed automaton A with the set of states $\mathcal{Q} \times U \cup \{q_{ini}\}$, because in A it is always necessary to remember the last external condition symbol in U to imitate the behavior of $\mathcal{T}$. The additional state $\{q_{ini}\}$ is necessary to build A where initially every condition symbol is possible as it is in $\mathcal{T}$. A major step in the transformation is to determine the guards of the transitions in A. This is realized by the function $cons_\theta$ mapping condition and event symbols in $\mathrm{Int}_\theta \cup \mathrm{Ala}_\theta$ to clock constraints. For each $a \in \mathrm{Int}_\theta \cup \mathrm{Ala}_\theta$, the constraint $cons_\theta((a,x))$ is in the table below.

$$cons_\theta((a,x)) = \begin{cases} x \le T^1 & \text{if } a = \text{“}\theta < T^1\text{”} \\ x > T^m & \text{if } a = \text{“}\theta \ge T^m\text{”} \\ x = T^j & \text{if } a = \text{“}\theta = T^j\text{”} \\ T^j < x \le T^{j+1} & \text{if } a = \text{“}T^j \le \theta < T^{j+1}\text{”} \\ \bigwedge_{j=1}^m x \ne T^j & \text{if } a = \mathbf{0} \end{cases}$$

TABLE 4.1
Definition of $cons_\theta$

To understand this translation note that, since s_u is a piecewise function, we have $s_u(t^-) = u^-$ iff there exists an interval I_t left from t such that for each $t' \in I_t$, $s_u(t') = u^-$.

Moreover, consider a time point t in a run r of a TCES. Then, we have $r(t) \in f(r(t^-), s_u(t^-), s_v(t))$. In order to determine the condition output $s_y(t) = g(r(t), s_u(t))$ for some timer θ we have to determine $s_u(t)$. Therefore, we define a function $next_\theta : (\mathrm{Int}_\theta \times \mathrm{Ala}_\theta \times \mathrm{Res}_\theta) \to \mathrm{Int}_\theta$ determining the timer condition in t for a given timer condition in t^- and the timer events in t. For a given set of thresholds $\{T^1, \ldots, T^m\}$ the function $next_\theta$ is determined by Table 4.2, where “*” abbreviates that any possible value is allowed.

u	v	z	$next_\theta(u,v,z)$
$*$	$*$	“$\theta := 0$”	“$\theta < T^1$”
$*$	$\mathbf{0}$	$\mathbf{0}$	u
“$\theta < T^1$”	“$\theta = T^1$	$\mathbf{0}$	“$T^1 \le \theta < T^2$” if $m > 1$
“$\theta < T^1$”	“$\theta = T^1$	$\mathbf{0}$	“$\theta \ge T^1$” if $m = 1$
“$T^j \le \theta < T^{j+1}$”	“$\theta = T^{j+1}$”	$\mathbf{0}$	“$T^{j+1} \le \theta < T^{j+2}$” if $m \ge j+2$
“$T^j \le \theta < T^{j+1}$”	“$\theta = T^{j+1}$”	$\mathbf{0}$	“$\theta \ge T^{j+1}$” if $m < j+2$

TABLE 4.2
Definition of $next_\theta$

The label of a transition in the constructed timed automaton is the set of symbols determined by the external signals of the timed condition/event system at the time point t, where the corresponding state transformation, given by f, takes place. Now, we define Ψ explicitly.

DEFINITION 4.2. *Consider a TCES $\mathcal{T} = (U, V, Y, Z, D, \Theta)$, where $D = (U, V, \mathcal{Q}, Y, Z, f, g, h, q_0)$ is the DCES and $\Theta = \{\theta_i \mid 1 \le i \le n\}$ is the set of clocks. We define $\Psi(\mathcal{T}) = (\mathcal{Q}, q_{ini}, \mathcal{X}, \mathcal{E}, Urg)$ by*

- *$\mathcal{Q} = (Q \times U) \cup \{q_{ini}\}$, $\mathcal{X} = \{x_1, \cdots, x_n\}$, and*
- *$\mathcal{E}$ and Urg are given by*
  - *$e = (q_{ini}, \bigwedge_{i=1}^{n} x_i < T_i^1, \sigma_0, \emptyset, (q_0, u_0)) \in \mathcal{E}$ and $e \notin Urg$, where $\sigma_0 = \{u_0, \mathbf{0}_V, y_0, \mathbf{0}_Z\}$, $u_0 \in U$, and $y_0 = g(q_0, \vec{u}_0) \in Y, \vec{u}_0 = (u_0, \text{“}\theta_1 < T_1^1\text{”}, \cdots, \text{“}\theta_n < T_n^1\text{”}) \in U^+$,*

– $e = ((q, u_0), \gamma, \sigma, \xi, (q', u'_0)) \in \mathcal{E}$ *if there exists*
  * $\vec{u} = (u_0, u_1, \cdots, u_n) \in U^+$,
  * $\vec{u}' = (u'_0, u'_1, \cdots, u'_n) \in U^+$, *and*
  * $\vec{v} = (v_0, v_1, \cdots, v_n) \in V^+$,

such that the following requirements are satisfied:
1. $q' \in f(q, \vec{u}, \vec{v})$,
2. $\gamma = \bigwedge_{i=1}^{n} cons_{\theta_i}(u_i, x_i) \wedge \bigwedge_{i=1}^{n} cons_{\theta_i}(v_i, x_i)$,
3. $\sigma = \{u'_0, v_0, g(q', \vec{u}'), h(q, q', \vec{v})_0\}$,
4. $\xi = \{x_i \in \mathcal{X} \mid h(q, q', \vec{v})_i \neq \mathbf{0}\}$.
5. $u'_i = next_{\theta_i}(u_i, v_i, h(q, q', \vec{v})_i)$ *for all* $i = 1, \cdots, n$

Moreover, $e \in Urg$, *if and only if there exists* $i \in \{1, \cdots, n\}$ *such that* $v_i \neq \mathbf{0}$.

□

By the transformation Ψ we obtain a timed automaton the states of which except q_{ini} are in the Cartesian product of the set of control locations of the timed condition/event system and the external input condition alphabet. In general, this is not necessary because often we cannot distinguish two condition symbols in the sense that they have the same effect on f and g. Thus, the state space can be reduced to a product of control locations with equivalent classes of condition symbols. To obtain a more efficient transformation we define an equivalence on condition symbols.

DEFINITION 4.3. *For every state* $q \in \mathcal{Q}$ *in a given TCES we call two condition symbols* $u, u' \in U$ equivalent at q, *denoted by* $u \simeq_q u'$, *if for all* $u_\theta \in \Pi_{i=1}^{n} Int_i$ *the following requirements are satisfied:*
- *for all* $\vec{v} \in V^+$, $f(q, (u, u_\theta), \vec{v}) = f(q, (u', u_\theta), \vec{v})$, *and*
- $g(q, (u, u_\theta)) = g(q, (u', u_\theta))$. □

Let $U|_{\simeq q}$ denote the set of all equivalent classes of condition symbols for a state q. We define another transformation Ψ' from timed condition/event systems to timed automata exactly as before, but in this case we restrict the states of the timed automaton to $(\bigcup_{q \in \mathcal{Q}}(\{q\} \times U|_{\simeq q})) \cup \{q_{ini}\}$ and define its transition relation on states $(q, u_{\simeq q})$, where $u_{\simeq q} \in U|_{\simeq q}$. In comparison to the first transformation Ψ, we reduce the number of resulting states by summarizing all states (q, u^i), $i \in I\!N$, where all u^i and u^j are equivalent, into one state $(q, u_{\simeq q})$. $\Psi(\mathcal{T})$ and $\Psi'(\mathcal{T})$ are related by the following lemma.

LEMMA 4.4. *For a timed condition/event system* $\mathcal{T}$ *the timed automata* $\Psi(\mathcal{T})$ *and* $\Psi'(\mathcal{T})$ *are timed bisimilar, i.e.* $\Psi(\mathcal{T}) \sim \Psi'(\mathcal{T})$. □

Returning to the verification problem, we see that language inclusion for timed condition event systems $\mathcal{T}$ and $\mathcal{T}'$ can be checked directly for their corresponding timed automata $\Psi(\mathcal{T})$ and $\Psi(\mathcal{T}')$, respectively. However, to check reachability that is not straightforward to apply, since the state space of the resulting timed automata is not just the set of control location $\mathcal{Q}$, but its Cartesian product with the external input condition alphabet. Thus, we have to check whether the pair consisting of location and external input condition symbol is reachable. This leads to the following: For a given timed condition/event system $\mathcal{T}$ a control location $q \in \mathcal{Q}$ is reachable, if and only if there exists an external input condition symbol $u \in U$ such that (q, u) is reachable in $\Psi(\mathcal{T})$, respectively $\Psi'(\mathcal{T})$.

5. Example. To illustrate timed condition/event systems, their composition and transformation to timed automata we consider the laboratory batch plant example [KP96], the so-called Evaporator System. The system is shown in Figure 5.1, where the following production sequence takes place: "A salt solution is filled into tank 1 T1 via valve V12 and then evaporated until a desired concentration is reached. During evaporation, the condenser C1 is in operation and captures the steam coming from T1. When the desired concentration is reached, the material is drained from T1 into tank 2 T2 via valve V15 as soon as T2 becomes empty. A post-processing step takes place in T2, before the material can be pumped out of T2 to a subsequent part of the plant via valve V18."

We suppose that in the undisturbed case the operation sequence described above is ensured by the controller and direct our interest to the problem of appropriate control reaction to disturbances. In particular, we look at the consequences of a cooling breakdown in the condenser. This failure can lead to a dangerously high pressure in the condenser tube, if the evaporating is continued. On the other hand, when the heater in T1 is switched off, the pressure in C1 will not rise anymore.

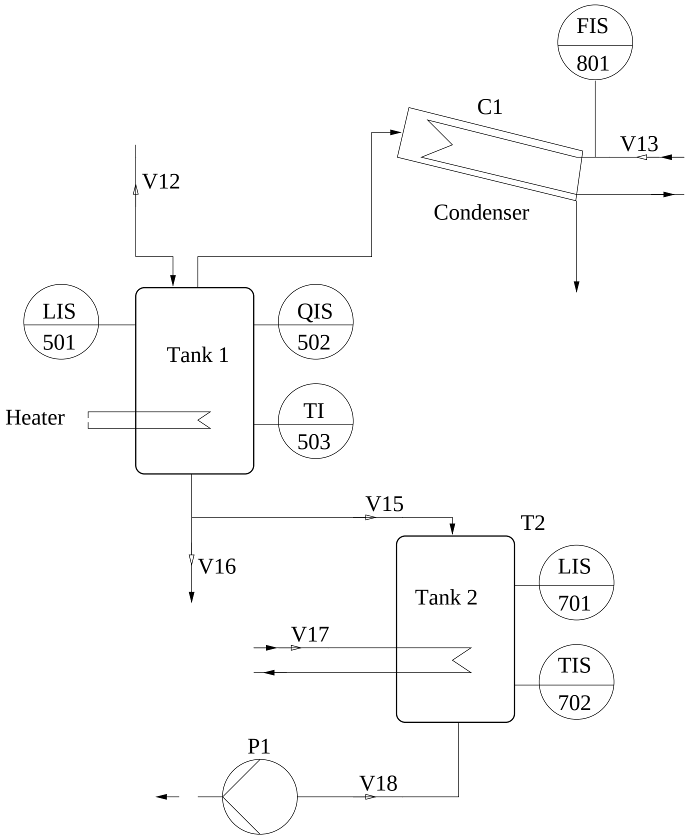

FIG. 5.1. *The Evaporator System*

We assume that the material in T1 becomes solid after a certain time when it cools down and cannot be drained into T2 via V15. This situation is of course not desired. As a consequence, the controller should not switch off the heater immediately after a cooling breakdown. But, if T2 is still full, switching off the heater must be delayed to ensure that T1 can be drained via V15 before the material becomes solid. However, a controller should start draining T2 via V18 as soon as a cooling breakdown occurs, so that no time is lost.

In the sequel we give a model of the evaporator system as a composition of timed condition/event systems. Moreover, for the sake of illustration, we give the result of the transformation of section 4 when applied to one of the components of the system.

5.1. Specification. We first specify the evaporator system as the composition of the TCESs modules *Tank1*, *Solution1*, *Heater*, *Tank2*, *Solution2*, *Condenser*, *Pressure*, and *Controller*. To represent TCESs a graphical description is used which is quite similar to that used for input/output automata. First, each state is labeled with a possible output condition in that state. Second, each transition is labeled with $(ic, ie)/oe$, where ic is an input condition, ie is an input event and oe is an output event. Sometimes, when there is no ic or no ie we simply omit the parenthesis and write only the remaining components. The same holds if there is no oe. In this case we also omit the slash symbol. Moreover, if in a state the label of an input or output symbol of a transition from that state is arbitrary we shortly write '*'. It represents each possible symbol of the corresponding alphabet.

Tank1 and Solution1. The block diagram of the *Tank1* module is shown in Figure 5.2. It has input event alphabet $Drain1 = \{drain1\}$ and output condition alphabet $T1\_Status = \{e1, f1\}$. The graphical representation of *Tank1* is shown in Figure 5.3. The set of states is $\mathcal{Q} = \{empty, full\}$. The module *Tank1* describes the filling and draining processes of T1. At the beginning, *Tank1* is *empty* and can spontaneously get filled by going to *full*. It can stay there or go back to *empty* when

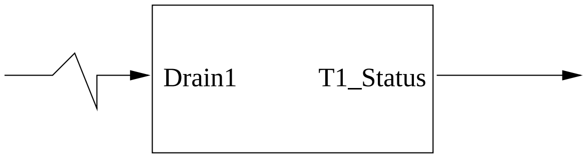

FIG. 5.2. *Block Diagram for Tank 1*

it receives the event "*drain1*" i.e. T1 was drained. Being *empty* and receiving "*drain1*" it remains *empty*. *Tank1* is a pure discrete condition/event system. There is no timer.

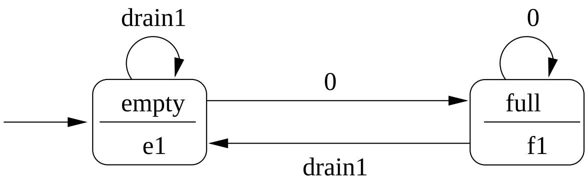

FIG. 5.3. *Tank 1*

The graphical representation of *Solution1* is shown in Figure 5.4. It has input event alphabet

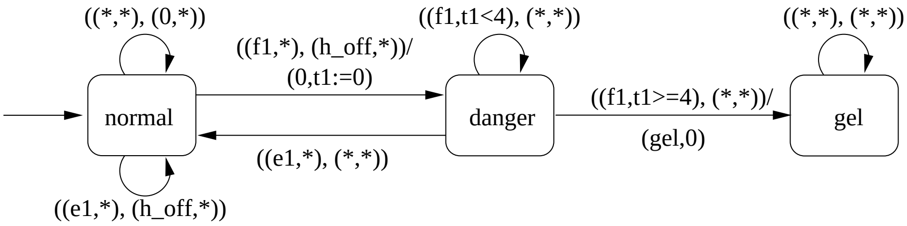

FIG. 5.4. *Solution 1*

$H\_Off = \{h\_off\}$, input condition alphabet $T1\_Status = \{e1, f1\}$ and output event alphabet $Gel = \{gel\}$. Moreover, we use a timer *t1* with threshold vector (4) in order to measure the time passed. So, the label of a transition takes the form $((T1\_Status, \mathrm{Int}_{t1}), (H\_Off, \mathrm{Ala}_{t1}))/(Gel, \mathrm{Res}_{t1})$, where $\mathrm{Int}_{t1} = \{\text{“}t1 < 4\text{”}, \text{“}t1 \geq 4\text{”}\}$, $\mathrm{Ala}_{t1} = \{\text{“}t1 = 4\text{”}, \mathbf{0}\}$, and $\mathrm{Res}_{t1} = \{\text{“}t1 := 0\text{”}, \mathbf{0}\}$, are the intervals, alarm and reset sets of the timer *t1*, respectively. *Solution1* describes the different states of the salt solution inside *Tank1*. At the beginning, the salt solution in T1 is normally heated, state *normal*. However, if the status of *Tank1* is full, condition "*f1*", and the heater was switched off, the module goes to *danger*. The timer *t1* is reset. In *danger*, *Solution1* has 4 time units to go back to *normal*, if *Tank1* was in "*e1*". If these 4 time units passed before *Tank1* could be drained, then *Solution1* goes into *gel*, i.e. the salt solution gelatinize.

Notice that *Tank1* and *Solution1* have the alphabet *T1_Status* in common, and that *T1_Status* is an output condition alphabet from *Tank1* and an input condition alphabet from *Solution1*, such that we can compose both modules by the cascade interconnection operator: $\mathcal{S}_1 = Tank1 \rightarrow Solution1$. Moreover, since *T1_Status* does not need to be visible outside this composition we hide it. One obtains a module with input alphabets *Drain1*, *H_Off* and output alphabet *Gel*.

Tank 2 and Solution 2. Similarly to *Tank1* the *Tank2* module describes the two possible statuses of tank 2, T2, namely, *empty* and *full*. The status of *Tank2* can be changed from *empty* to *full* if T1 is drained, event "*drain1*". On the other hand, it can change from *full* to *empty* just when T2 was successfully drained, that is, it receives the event "*empty2*".

Assuming that T2 has 10 time units to be successfully drained if an emergency situation occurs, the module *Solution2* exactly describes that, if a failure in the whole system occurs T2 has to be emergency drained within 10 time units; otherwise, an overflow occurs. *Solution2* has input event alphabet $Emergency = \{emer\_drain\}$, input condition alphabet $T2\_Status = \{e2, f2\}$ and output event alphabet *Empty2* $= \{$*empty2*$\}$. Moreover, we use a timer *t2* with threshold vector (10) in order to measure the time passed after the emergency has occurred.

Tank2 and *Solution2* can be composed by cascade interconnection of both modules and subsequent feedback of the resulting module, i.e. $\mathcal{S}_2 = (\mathit{Tank2} \to \mathit{Solution2}) \hookleftarrow$.

Heater. This module is shown in Figure 5.5. It has input event alphabet *Switch* $= \{$*switch_off*$\}$,

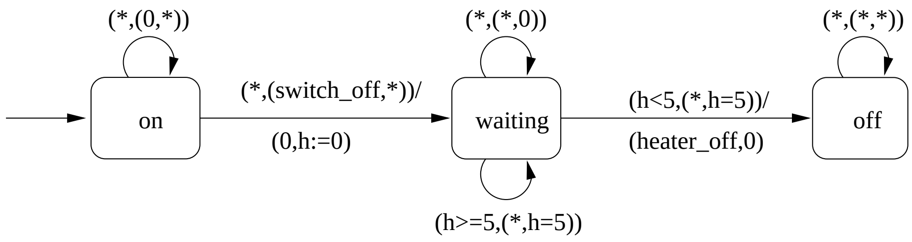

FIG. 5.5. *Heater*

output event alphabet *H_Off* $= \{$*heater_off*$\}$ and timer h with threshold vector (5). Assuming that the heater is switched off with a delay of 5 time units the module Heater exactly describes this process. Here, after receiving *switch_off* the module delays the time to switch off the heater by 5 time units. The heater is switched off by the transition from *waiting* to *off*. The event *heater_off* is sent.

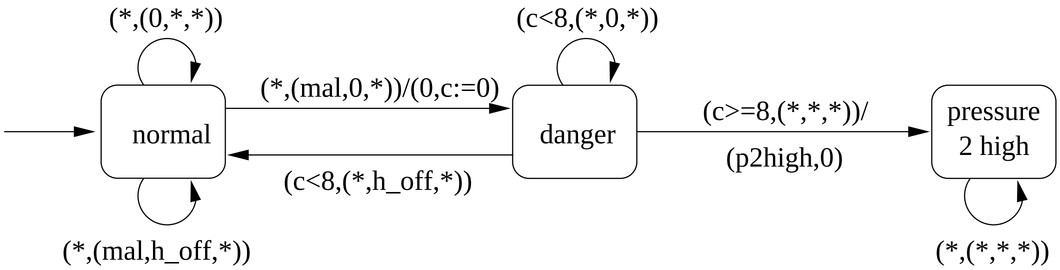

FIG. 5.6. *Pressure*

Condenser and Pressure. Condenser is a discrete condition event system with only an output event alphabet *Mal* $= \{$*mal*$\}$. It describes the situation that in C1 a failure can spontaneously occur, event "*mal*". On the other hand, the module *Pressure* describes the process that, if such a failure occurs in C1 (event "*mal*") the heater must be switched off within 8 time units. Otherwise, the pressure in C1 will be too high. *Condenser* and *Pressure* can be composed by cascade interconnection of both modules, i.e. $\mathcal{S}_3 = \mathit{Condenser} \to \mathit{Pressure}$. Now, up to the Controller module all other modules have been described and connected. The already formalized modules build the whole system. This can be built by $\mathcal{S} = (\mathit{Heater} \to \mathcal{S}_1) \to \mathcal{S}_3) \to \mathcal{S}_2$.

Controller. We would like to develop a controller to prevent under any circumstances that the pressure in C1 will become too high, the material in T1 will gelatinize, and there is an overflow in T2. In other words, the Controller module must ensure that the states *gel* in *Solution1*, *pressure2high* in *Pressure*, and *overflow* in *Solution2* are unreachable. One module which meets such requirements is shown in Figure 5.8. In that figure we use a hierarchical representation of states like the one used in statecharts. So, for instance, the box involving the states *normal*, *st1*, *st2*, *st3*, and *st6* is a super state. That means, that each transition occurring from that super state is a transition occurring from each state involved by this super state. The same also holds for transitions coming into the super state. For a symbol a the notation $\hat{\ }a$ represents all symbols of the corresponding alphabet except a.

Transformation. The full specification, complete transformation, and the results of the verification of the evaporator system are given in the full paper [HLL+98]. Here, we only give the transformation of section 4 when applied to the timed condition/event system *Heater*. We obtain the timed automaton shown in Figure 5.7. In that graphical representation we omit the synchronization symbols $\mathbf{0}_{Switch}, \mathbf{0}_{H\_Off}$ and also the set of them $\{\mathbf{0}_{Switch}, \mathbf{0}_{H\_Off}\}$. Urgent transitions are shown as dashed arrows. Note that since both transitions from state *waiting* to state *off* are urgent the automaton cannot stay in *waiting* beyond 5.

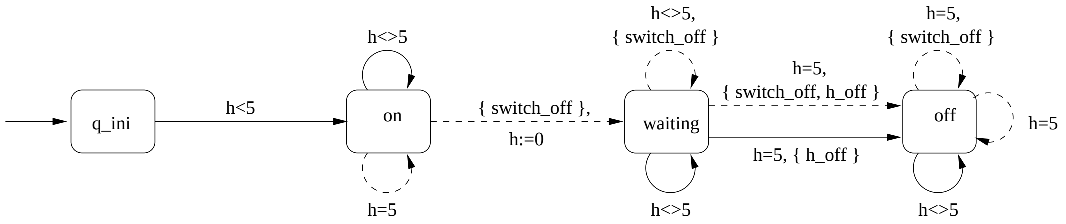

FIG. 5.7. *Timed Automaton Transformation of Module Heater*

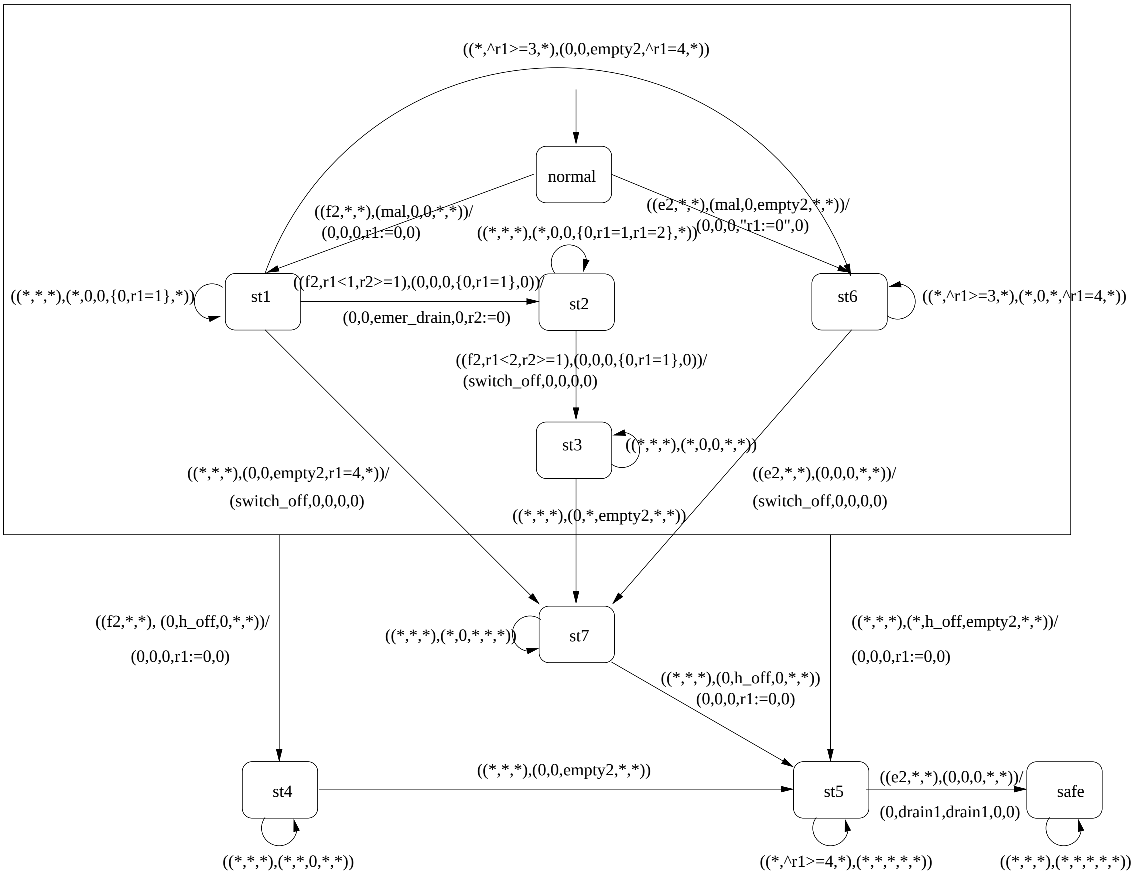

FIG. 5.8. *Controller*

6. Conclusion. In this paper we showed that timed condition/event systems can be efficiently transformed into timed automata allowing this way for automatic verification using tools as [OY93, HHWT95, BLL⁺96]. Currently, we are developing a tool which includes a graphical interface for the description of block diagrams of timed condition/event systems. This tool uses the transformation given in the present paper to transform the described system into a timed automaton which is then analyzed by one of the tools [OY93, HHWT95, BLL⁺96]. We are also studying a compositional approach for the verification of timed condition/event systems.

REFERENCES

[ACD90] R. Alur, C. Courcoubetis, and D.L. Dill. Model Checking for Real–Time Systems. In *LICS'90*, pages 414–425, 1990.

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

[AFH96] R. Alur, T. Feder, and T.A. Henzinger. The Benefits of Relaxing Puntuality. *Journal of the ACM*, 43:116–146, 1996.

[BL95] A. Bouajjani and Y. Lakhnech. Temporal Logic + Timed Automata: Expressiveness and Decidability. In *CONCUR'95*, LNCS 962, pages 531–546. Springer–Verlag, 1995.

[BLL+96] J. Bengtsson, K.G. Larsen, F. Larsson, P. Pettersson, and W. Yi. UPPAAL — A Tool Suite for Automatic Verification of Real–Time Systems. In *Hybrid Systems III*, LNCS 1066, pages 232–243. Springer–Verlag, 1996.

[BLY96] A. Bouajjani, Y. Lakhnech, and S. Yovine. Model Checking for Extended Timed Temporal Logics. In *FTRTFT'96*, LNCS 1135, pages 306–326. Springer–Verlag, 1996.

[EH96] S. Engell and I. Hoffmann. Modular Hierarchical Models of Hybrid Systems. In *35th IEEE Conf. on Decision and Control*, Kobe, Japan, 1996.

[EKKP95] S. Engell, S. Kowalewski, B. Krogh, and J. Preußig. Condition/Event Systems: A Powerful Paradigm for Timed and Untimed Discrete Models of Technical Systems. In *EUROSIM'95*, 1995.

[HHWT95] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech : The Next Generation. In *RTSS'95*, pages 56–65, 1995.

[HKPS96] I. Hoffmann, S. Kowalewski, J. Preußig, and O. Stursberg. Towards Systematic Derivation of Timed and Hybrid Automata from Continuous Models. In *Hybrid Systems III*, volume 1066 of *LNCS*. Springer-Verlag, 1996.

[HLL+98] R. Huuck, Y. Lakhnech, B. Lukoschus, L. Urbina, S. Engell, S. Kowalewski, and J. Preußig. Integrating Timed Condition/Event Systems and Timed Automata to the Verification of Hybrid Systems. Technical report, Institut für Informatik, Christian–Albrechts Universität zu Kiel, 1998.

[HNSY94] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic Model Checking for Real–Time Systems. *Information and Computation*, 111(2):193–244, 1994.

[Hol96] L.E. Holloway. Time Measures and State Maintainability for a Class of Composed Systems. In *WODES'96*, pages 24–30, Edinburgh, UK, 1996.

[HR95] H.-M. Hanisch and M. Rausch. Synthesis of Supervisory Controllers Based on a Novel Representation of Condition/Event Systems. In *IEEE Int. Conf. on Systems, Man., and Cybernetics*, Vancouver, Canada, 1995.

[KEF+95] S. Kowalewski, S. Engell, M. Fritz, R. Gesthuisen, G. Regner, and M. Stobbe. Modular Discrete Modelling of Batch Processes by Means of Condition/Event Systems. In *Analysis and Design of Event–Driven Operations in Process Systems, London*, 1995.

[KEP+97] S. Kowalewski, S. Engell, J. Preußig, R. Huuck, Y. Lakhnech, and L. Urbina. Analyse zeitbewerteter Bedingungs/Ereignis–Systeme mittels Echtzeitautomaten–Tools. In *EKA'97*, Braunschweig, Germany, 1997.

[Kow95] S. Kowalewski. *Modulare diskrete Anlagenmodellierung zum systematischen Steuerungsentwurf.* PhD thesis, Fachbereich Chemietechnik, Universität Dortmund, 1995.

[KP96] S. Kowalewski and J. Preußig. Timed Condition/Event Systems: A Framework for Modular Discrete Models of Chemical Plants and Verification of their Real–Time Discrete Control. In *LNCS 1055*, pages 225–240. Springer–Verlag, 1996.

[Lak96] Y. Lakhnech. *Specification and Verification of Hybrid and Real-Time Systems.* PhD thesis, Institut für Informatik und Praktische Mathematik, Christian-Albrechts-Universität zu Kiel, 1996.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[MPS95] O. Maler, A. Pnueli, and J. Sifakis. On the Synthesis of Discrete Controllers for Timed Systems. In E.W. Mayr and C. Puech, editors, *LNCS 900*, pages 229–242. Springer Verlag, 1995.

[OY93] A. Olivero and S. Yovine. *KRONOS: A Tool for Verifying Real–Time Systems. User's Guide and Reference Manual.* VERIMAG, Grenoble, France, 1993.

[Par81] D. Park. Concurrency and Automata on Infinite Sequences. In *5th GI Conference*, LNCS 104. Springer-Verlag, 1981.

[SK91] R.S. Sreenivas and B.H. Krogh. On Condition/Event Systems with Discrete State Realizations. In *Discrete Event Dynamic Systems: Theory and Applications 1*, pages 209–236. Kluwer Academic, Boston, USA, 1991.

[Wil94] T. Wilke. *Automaten und Logiken zur Beschreibung zeitabhängiger Systeme.* PhD thesis, Institut für Informatik und Praktische Informatik der Universität zu Kiel, 1994.

Edited By: Dr.Lonnie R Welch and Dr.Dieter K Hammer
Date of First Submission: December 1996
Date of First Revision: April 1997
Date of Final Acceptance: December 1997