Basic Research in Computer Science

BRICS RS-96-57 Larsen et al.: Diagnostic Model-Checking for Real-Time Systems

Diagnostic Model-Checking for Real-Time Systems

Kim G. Larsen
Paul Pettersson
Wang Yi

BRICS Report Series **RS-96-57**

ISSN 0909-0878 **December 1996**

See back inner page for a list of recent publications in the BRICS Report Series. Copies may be obtained by contacting:

**BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK - 8000 Aarhus C
Denmark
Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@brics.dk**

BRICS publications are in general accessible through World Wide Web and anonymous FTP:

`http://www.brics.dk/`
`ftp://ftp.brics.dk/`
This document in subdirectory `RS/96/57/`

Diagnostic Model-Checking for Real-Time Systems*

Kim G. Larsen[1] *Paul Pettersson*[2] *Wang Yi*[2]**

[1] BRICS*** , Aalborg University, DENMARK. E-mail: kgl@iesd.auc.dk
[2] Department of Computer Systems, Box 325, Uppsala University, S751 05, Uppsala, Sweden. E-mail: {paupet,yi}@docs.uu.se.

Abstract. UPPAAL is a new tool suit for automatic verification of networks of timed automata. In this paper we describe the diagnostic model-checking feature of UPPAAL and illustrates its usefulness through the debugging of (a version of) the Philips Audio-Control Protocol. Together with a graphical interface of UPPAAL this diagnostic feature allows for a number of errors to be more easily detected and corrected.

1 Introduction

UPPAAL is a new tool for automatic verification of safety and bounded liveness properties of real-time systems modeled as networks of timed automata [10]. The current version of UPPAAL deals with the traditionally encountered state-explosion problem by combining on-the-fly verification with a symbolic technique based on constraint-solving. UPPAAL contains a suit of tools and features including:

- a graphical interface allowing networks of timed automata to be defined by drawing,
- an automatic compilation of the graphical definition into a textual format used by the model-checker, thus supporting the important principle "what you see is what you verify" (WYSIWYV),
- compilation of certain types of hybrid automata into ordinary timed automata (again supporting WYSIWYV), and
- in case verification of a particular real-time system fails (which happens more often than not), a *diagnostic* trace is automatically reported by UPPAAL in order to facilitate debugging. Here the principle supported could be called "What You Don't Verify You Are Explained"(WYDVYAE).

This paper concentrates on describing the diagnostic model-checking feature of UPPAAL, and on demonstrating its usefulness through the debugging of an early version of the Philips Audio-Control Protocol [6].

The paper is organized as follows: In the next section we give a short review of the notions of timed automata and networks; in section 3 the logic for safety and

* This work has been supported by the European Communieties (under CONCUR2 and REACT), NUTEK (Swedish Board for Technical Development) and TFR (Swedish Technical Research Council)

** This author would also like to thank the Chinese NSF and the Hong Kong Wang's Foundation for supporting a visit to the Institute of Software, Chinese Academy of Sciences, in 1995.

*** **B**asic **R**esearch in **C**omputer **S**cience, Centre of the Danish National Research Foundation.

bounded liveness properties is presented. Section 4 describes the diagnostic model-checking procedure; in section 5 we show how these results have been applied in a case study where Philips Audio-Control Protocol was analyzed.

2 Real-Time Systems

We shall use *timed transition systems* as a basic semantical model for real-time systems. The type of systems we are studying will be a particular class of timed transition systems that are syntactically described by *networks of timed automata* [12, 8].

2.1 Timed Transition Systems

A timed transition system is a labeled transition system with two types of labels: atomic actions and delay actions (i.e. positive reals), representing discrete and continuous changes of real-time systems.

Let Act be a finite set of actions and $\mathcal{P}$ be a set of atomic propositions. We use $\mathbf{R}$ to stand for the set of non-negative real numbers, Δ for the set of delay actions $\{\epsilon(d) \mid d \in \mathbf{R}\}$, and L for the union $Act \cup \Delta$.

Definition 1. *A timed transition system over Act and $\mathcal{P}$ is a tuple $\mathcal{S} = \langle S, s_0, \longrightarrow, V\rangle$, where S is a set of states, s_0 is the initial state, $\longrightarrow \subseteq S \times L \times S$ is a transition relation, and $V : S \to 2^{\mathcal{P}}$ is a proposition assignment function.* □

We will need for $\longrightarrow$ to satisfy the following well known properties:

- *(Time Determinism)* $s \xrightarrow{\epsilon(d)} s_1$ and $s \xrightarrow{\epsilon(d)} s_2 \Rightarrow\ s_1 = s_2$.
- *(Time Continuity)* $s \xrightarrow{\epsilon(d+e)} s' \Leftrightarrow \exists s''.s \xrightarrow{\epsilon(d)} s'' \xrightarrow{\epsilon(e)} s'$.

Whenever defined, we will use the notation s^d for the state satisfying $s \xrightarrow{\epsilon(d)} s^d$. Note that the state s^d is unique due to time determinism.

In order to study compositionality problems we introduce a parallel composition between timed transition systems. Following [7] we use synchronization functions that generalize a large range of existing notions of parallel compositions. A *synchronization function* f is a partial function $(Act \cup \{0\}) \times (Act \cup \{0\}) \hookrightarrow Act$, where 0 denotes a distinguished no-action symbol[4]. Now, let $\mathcal{S}_i = \langle S_i, s_{i,0}, \longrightarrow_i, V_i\rangle$, $i = 1, 2$, be two timed transition systems and let f be a synchronization function. Then the *parallel composition* $\mathcal{S}_1 \mid_f \mathcal{S}_2$ is the timed transition system $\langle S, s_0, \longrightarrow, V\rangle$, where $s_1 \mid_f s_2 \in S$ whenever $s_1 \in S_1$ and $s_2 \in S_2$, $s_0 = s_{1,0} \mid_f s_{2,0}$, $\longrightarrow$ is inductively defined as follows:

- $s_1 \mid_f s_2 \xrightarrow{c} s_1' \mid_f s_2'$ if $s_1 \xrightarrow{a}_1 s_1'$, $s_2 \xrightarrow{b}_2 s_2'$ and $f(a, b) = c$
- $s_1 \mid_f s_2 \xrightarrow{\epsilon(d)} s_1' \mid_f s_2'$ if $s_1 \xrightarrow{\epsilon(d)}_1 s_1'$ and $s_2 \xrightarrow{\epsilon(d)}_2 s_2'$

[4] We extend the transition relation of a timed transition system such that $s \xrightarrow{0} s'$ iff $s = s'$.

and finally, the proposition assignment function V is defined by $V(s_1 \mid_f s_2) = V_1(s_1) \cup V_2(s_2)$.

We now introduce the notion of a *trace*. A trace σ of a timed transition system is a finite alternating sequence of the form:

$$\sigma = s_0 \xrightarrow{\epsilon(d_0)} s'_0 \xrightarrow{a_1} s_1 \xrightarrow{\epsilon(d_1)} s'_1 \xrightarrow{a_2} s_2 \xrightarrow{\epsilon(d_2)} \ldots \xrightarrow{a_n} s_n \xrightarrow{\epsilon(d_n)} s'_n$$

where $d_i \in \mathbf{R}$. A position π of a trace σ is a pair $\pi = (i, d)$ where $i \in 0 \ldots n$ and $0 \leq d \leq d_i$. We use $\Delta(\sigma, \pi)$ to stand for the accumulated delay of the trace σ before the position π, i.e. $\Delta(\sigma, \pi) = \sum_{j<i} d_j + d$ and $\sigma(\pi)$ for the suffix of σ starting from π, i.e.

$$\sigma(\pi) = s_i^d \xrightarrow{\epsilon(d_i - d)} s'_i \xrightarrow{a_{i+1}} s_{i+1} \xrightarrow{\epsilon(d_{i+1})} \ldots \xrightarrow{a_n} s_n \xrightarrow{\epsilon(d_n)} s'_n$$

Whenever $s \xrightarrow{l} s_0$ ($l \in L$) we shall denote by $s \xrightarrow{l} \sigma$ the trace obtained by extending σ[5]. We order positions lexicographically, denoted $\pi < \pi'$. Finally, we write $V(\sigma)$ for the set $V(s_0)$.

2.2 Networks of Timed Automata

A timed automaton [1] is a standard finite-state automaton extended with a finite collection of real-valued clocks. The clocks are assumed to proceed at the same rate and their values may be tested (compared with natural numbers) and reset (assigned to 0).

Definition Clock Constraints *Let C be a set of real-valued clocks. We use $\mathcal{B}(C)$ to stand for the set of formulas ranged over by g, generated by the following syntax: $g ::= c \mid g \wedge g$, where c is an atomic constraint of the form: $x \sim n$ or $x - y \sim n$ for $x, y \in C$, $\sim \in \{\leq, \geq, =, <, >\}$ and n being a natural number. We shall call $\mathcal{B}(C)$ clock constraints or clock constraint systems over C.* □

We shall use $\mathtt{tt}$ to stand for a constraint like $x \geq 0$ which is always true, and $\mathtt{ff}$ for a constraint $x < 0$ which is always false as clocks can only have non-negative values.

Definition 2. *A timed automaton A over actions Act, atomic propositions $\mathcal{P}$ and clocks C is a tuple $\langle N, l_0, E, I, V \rangle$, where N is a finite set of nodes (control-nodes), l_0 is the initial node, and $E \subseteq N \times \mathcal{B}(C) \times Act \times 2^C \times N$ corresponds to the set of edges. In the case, $\langle l, g, a, r, l' \rangle \in E$ we shall write, $l \xrightarrow{g,a,r} l'$. $I : N \to \mathcal{B}(C)$ is a function which for each node assigns an invariant condition, and $V : N \to 2^{\mathcal{P}}$ is a proposition assignment function which for each node gives a set of atomic propositions true in the node.* □

A *state* of an automaton A is a pair (l, u) where l is a node of A and u a *clock assignment* for C, mapping each clock in C to a value in $\mathbf{R}$. The initial state of A is (l_0, u_0) where u_0 is the initial clock assignment mapping all clocks in C to 0. The semantics of A is given by the timed transition system $\mathcal{S}_A = \langle S, s_0, \longrightarrow, V \rangle$, where S is the set of states of A, s_0 is the initial state (l_0, u_0), $\longrightarrow$ is the transition relation defined as follows:

[5] In order to keep the extended trace alternating we might have to apply the time continuity property to avoid two neighboring delay-transitions and we might have to insert 0-delay transition in order to avoid neighboring action-transitions.

– $(l,u) \xrightarrow{a} (l',u')$ if there exist r, g such that $l \xrightarrow{g,a,r} l'$, g is satisfied by u and $u' = r(u)$[6],
– $(l,u) \xrightarrow{\epsilon(d)} (l',u')$ if $(l = l')$, $u' = u + d$[7] and $I(l')$ is satisfied by u',

and V is extended to S simply by $V(l,u) = V(l)$. We denote by $Tr(A)$ all traces of $\mathcal{S}_A$ starting from the initial state (l_0, u_0).

Parallel composition may now be extended to timed automata in the obvious way: for two timed automata A and B and a synchronization function f, the parallel composition $A \mid_f B$ denotes the timed transition system $\mathcal{S}_A \mid_f \mathcal{S}_B$.

3 A Logic for Safety and Bounded Liveness Properties

It has been pointed out [4, 12], that the practical goal of verification of real-time systems, is to verify simple safety properties such as deadlock-freeness and mutual exclusion. Our previous work [12, 10] shows that such properties can be verified on-the-fly by simple reachability analysis which avoids to construct the whole reachable state-space of systems.

We consider a timed modal logic to specify safety and bounded liveness properties (sometimes called bounded response time properties). The logic may be seen as a fragment of the timed μ-calculus presented in [5], and also studied in [9][8].

Definition 3. *(Syntax) Assume K is a finite set of clocks. Then formulas over K is defined by the following abstract syntax:*

$$\varphi ::= a \mid \varphi_1 \wedge \varphi_2 \mid a \vee \varphi \mid Inv(\varphi) \mid \varphi \; Until_r \; a$$

where $r \subseteq K$ and $a ::= c \mid p$ where c is an atomic clock constraint over K and $p \in \mathcal{P}$ □

Intuitively, for $Inv(\varphi)$ to be satisfied all reachable states must satisfy φ. $\varphi \; Until_r \; a$ is a weak until-property expressing that φ must either hold invariantly or until a. The use of the clock set r allows for *bounded* liveness properties to be expressed, e.g. $(x < 5) \; Until_{\{x\}} \; a$ insists that a must hold within 5 time units. We interpret a formula φ with respect to a trace σ relative to a time assignment v over formula clocks K. We use $\sigma \models_v \varphi$ to mean that σ satisfies φ under u. The interpretation is defined on the structure of φ in Table 1. Naturally, if all the traces of an automaton satisfy a formula, we say that the automaton satisfies the formula.

Definition 4. *Let $Tr(\varphi) = \{\sigma \mid \sigma \models_{v_o} \varphi\}$ where v_0 is the initial time assignment. For a timed automaton A and a formula φ we write $A \models \varphi$ when $Tr(A) \subseteq Tr(\varphi)$. If there exists a trace σ s.t. $\sigma \in Tr(A) \setminus Tr(\varphi)$, we write $A \not\models \varphi$ and in this case, σ is called a diagnostic trace of A w.r.t. φ.* □

[6] $r(u)$ is the assignment s.t. $r(u)(x) = 0$ if $x \in r$ and $r(u)(x) = u(x)$ otherwise.
[7] $(u + d)$ is the assignment s.t. $(u + d)(x) = u(x) + d$.
[8] The connectives of our logic are expressible as derived operators w.r.t. those of [9].

$$
\begin{aligned}
\sigma \models_v c \text{ iff } & c(v) \\
\sigma \models_v p \text{ iff } & p \in V(\sigma) \\
\sigma \models_v \varphi_1 \wedge \varphi_2 \text{ iff } & \sigma \models_v \varphi_1 \text{ and } \sigma \models_v \varphi_2 \\
\sigma \models_v a \vee \varphi \text{ iff } & \sigma \models_v a \text{ or } \sigma \models_v \varphi \\
\sigma \models_v Inv(\varphi) \text{ iff } & \forall \pi : \sigma(\pi) \models_{v+\Delta(\sigma,\pi)} \varphi \\
\sigma \models_v \varphi \; Until_r \; a \text{ iff } & \begin{cases} \forall \pi : \sigma(\pi) \models_{r(v)+\Delta(\sigma,\pi)} \varphi, \\ \text{or} \\ \exists \pi : \Big(\sigma(\pi) \models_{r(v)+\Delta(\sigma,\pi)} a \;\wedge\; \forall \pi' < \pi : \sigma(\pi') \models_{r(v)+\Delta(\sigma,\pi')} \varphi \Big) \end{cases}
\end{aligned}
$$

Table 1. Definition of satisfiability.

4 Diagnostic Model-Checking

Given a network of timed automata A and a formula φ in the logic specifying a property, the so-called model-checking problem is to check if the formula is satisfied by the system. We will take an opposite point of view and check for $A \not\models \varphi$ instead of $A \models \varphi$. From a proof of $A \not\models \varphi$ we will then be able to synthesize a diagnostic trace which may prove useful in subsequent debugging. However, if we fail to prove $A \not\models \varphi$ we can assert that $A \models \varphi$.

4.1 Operations on Clock Constraints

To develop the diagnostic model-checking algorithm, we need a few operations to manipulate clock constraints. Given a clock constraint D, we shall call the set of clock assignments satisfying D, the *solution set* of D.

Definition 5. *Let A and A' be the solution sets of clock constraints $D, D' \in \mathcal{B}(C \cup K)$. We define*

$$
\begin{aligned}
A^{\uparrow} &= \{w + d \mid w \in A \text{ and } d \in \mathbf{R}\} \\
\{x\}A &= \{\{x\}w \mid w \in A\} \\
A \wedge A' &= \{w \mid w \in A \text{ and } w \in A'\} \\
A{\downarrow}C &= \{w{\downarrow}C \mid w \in A\}
\end{aligned}
$$

where $w{\downarrow}C$ denotes the restriction of w to the clock set C. □

First, note that $A \wedge A'$ is simply the intersection of the two sets. Intuitively, $A^{\uparrow}$ is the set of time assignments that may be reached from A by some delay. We extend the projection operator $\{x\}A$ to sets of clocks. Let $r = \{x_1 ... x_n\}$ be a set of clocks. We define $r(A)$ recursively by $\{\}(A) = A$ and $\{x_1 ... x_n\}(A) = \{x_1\}(\{x_2 ... x_n\}A)$. The following Proposition establishes that the class of clock constraints $\mathcal{B}(C \cup K)$ is closed under the four operations defined above.

Proposition 6. *Let $D, D' \in \mathcal{B}(C \cup K)$ with solution sets A and A', and $x \in C \cup K$. Then there exist $D_1, D_2, D_3, D_4 \in \mathcal{B}(C \cup K)$ with solution sets $A^{\uparrow}$, $\{x\}A$, $A \wedge A'$ and $A{\downarrow}C$ respectively.* □

In fact, the resulted constraints D_i's can be effectively constructed from D and D' [11, 10]. In order to save notation, from now on, we shall simply use $D^{\uparrow}$, $\{x\}D$ $D \wedge D'$ and $D{\downarrow}C$ to denote the clock constraints which are guaranteed to exist due to the above proposition. We will use $D^{\uparrow l}$ to denote $(D \wedge I(l))^{\uparrow} \wedge I(l)$ where $I(l)$ is the invariant condition of node l.

We will also need a few *predicates* over clock constraints for the diagnostic model-checking procedure. We write $D \subseteq D'$ to mean that the solution set of D is included in the solution set of D', $D = \emptyset$ to mean that the solution set of D is empty and $u \in D$ to denote that the time assignment u belongs to the solution set of D[9].

4.2 Model-Checking with Diagnostic Synthesis

Note that the definition $A \not\models \varphi$ means that there exists a trace σ of A s.t. $\sigma \notin Tr(\varphi)$. Intuitively, σ is a possible execution of A that does not meet the requirement φ, and therefore it may be used as diagnostic information for subsequent debugging. In order to effectively construct diagnostic traces, we define a relation $\not\vdash$ of the following type: $\sigma \not\vdash [l, D] : \varphi$, where σ is a trace of automaton A over the automata clocks C, l is a node of A, D is a constraint system over $C \cup K$ and φ is a formula over K. Now $\not\vdash$ is the smallest relation satisfying the rules of Table 2.

We use the third invariant rule to exemplify the intuitive explanation of the inference rules. The assertion $(l, u) \xrightarrow{a} (l', u') \longrightarrow \cdots \not\vdash [l, D] : Inv(\varphi)$ can be justified if any of the symbolic states, reachable using an edge $l \xrightarrow{g,a,r} l'$ from the symbolic state $[l, D]$, does not satisfy the invariant property $Inv(\varphi)$. The clock assignments in this resulting symbolic state is restricted to the (non-empty) constraint system $r(g \wedge D)$. The premise of the rule assumes the existence of a diagnostic trace for $[l', r(g \wedge D)] : Inv(\varphi)$ and the side-condition of the rule provides information as to how one may extend this trace (obviously with an a-transition) in order to obtain a diagnostic trace for $[l, D] : Inv(\varphi)$.

The rules in Table 2 are sound and complete in the following sense:

Theorem 7. *Let A be a timed transition system with initial node l_0. Then*

1. *Whenever $\sigma \not\vdash [l_0, D_0] : \varphi$ then $\sigma \in Tr(A)$ and $\sigma \notin Tr(\varphi)$.*
2. *Whenever $A \not\models \varphi$ then $\sigma \not\vdash [l_0, D_0] : \varphi$ for some $\sigma \in Tr(A)$.* □

4.3 Obtaining an Algorithm

Given a symbolic state $[l, D]$ of the automata A and a property φ it is decidable whether there exists a diagnostic trace σ such that $\sigma \not\vdash [l, D] : \varphi$. We obtain an algorithm by using the rules in Table 2 in two phases, In Phase 1 a goal directed search, starting in the symbolic state $[l, D]$, searching for a violating symbolic state, is performed by using the inference rules in Table 2. We have the following two termination criteria for the symbolic state $[l_n, D_n]$ and the property φ_n:

- (Success) c or p axiom can be applied,
- (Fail) for some i, $l_n = l_i$, $D_n \subseteq D_i$ and $\varphi_n = \varphi_i$.

[9] We will also write $u \in D$ to mean the operation of computing a time assignment u given a constraint system D.

c	$\dfrac{}{(l,u) \not\Vdash [l,D] : c} \quad w \in D \wedge \neg c, u = w{\downarrow}C$
p	$\dfrac{}{(l,u) \not\Vdash [l,D] : p} \quad \begin{array}{l} w \in D, u = w{\downarrow}C, \\ p \notin V(l) \end{array}$
$\varphi_1 \wedge \varphi_2$	$\dfrac{\sigma \not\Vdash [l,D] : \varphi_i}{\sigma \not\Vdash [l,D] : \varphi_1 \wedge \varphi_2} \quad i = 1 \text{ or } i = 2$
$a \vee \varphi$	$\dfrac{\sigma \not\Vdash [l, D \wedge \neg c] : \varphi}{\sigma \not\Vdash [l,D] : c \vee \varphi} \qquad \dfrac{\sigma \not\Vdash [l,D] : \varphi}{\sigma \not\Vdash [l,D] : p \vee \varphi} \quad p \notin V(\sigma)$
$Inv(\varphi)$	$\dfrac{\sigma \not\Vdash [l,D] : \varphi}{\sigma \not\Vdash [l,D] : Inv(\varphi)} \qquad \dfrac{(l,u') \longrightarrow \cdots \not\Vdash [l, D^{\uparrow l}] : Inv(\varphi)}{(l,u) \xrightarrow{\epsilon(d)} (l,u') \cdots \not\Vdash [l,D] : Inv(\varphi)} \quad \begin{array}{l} u \in D{\downarrow}C, \\ u' = u + d \end{array}$ $\dfrac{(l',u') \longrightarrow \cdots \not\Vdash [l', r(g \wedge D)] : Inv(\varphi)}{(l,u) \xrightarrow{a} (l',u') \longrightarrow \cdots \not\Vdash [l,D] : Inv(\varphi)} \quad \begin{array}{l} l \xrightarrow{g,a,r} l',\ u' = r(u), \\ u \in (g \wedge D){\downarrow}C \end{array}$
$\varphi\ Until_r\ a$	$\dfrac{\sigma \not\Vdash [l, r(D)] : \varphi Until_\emptyset\ a}{\sigma \not\Vdash [l,D] : \varphi Until_r\ a}$
$\varphi\ Until_\emptyset\ c$	$\dfrac{\sigma \not\Vdash [l, D \wedge \neg c] : \varphi}{\sigma \not\Vdash [l,D] : \varphi\ Until_\emptyset\ c} \qquad \dfrac{(l,u') \longrightarrow \cdots \not\Vdash [l, (D \wedge \neg c)^{\uparrow l}] : (\varphi\ Until_\emptyset\ c)}{(l,u) \xrightarrow{\epsilon(d)} (l,u') \longrightarrow \cdots \not\Vdash [l,D] : (\varphi\ Until_\emptyset\ c)}$ $\dfrac{(l',u') \longrightarrow \cdots \not\Vdash [l, r(g \wedge D \wedge \neg c)] : (\varphi\ Until_\emptyset\ c)}{(l,u) \xrightarrow{a} (l',u') \longrightarrow \cdots \not\Vdash [l,D] : (\varphi\ Until_\emptyset\ c)} \quad \begin{array}{l} l \xrightarrow{g,a,r} l',\ u' = r(u), \\ u \in (D \wedge g \wedge \neg c){\downarrow}C \end{array}$
$\varphi\ Until_\emptyset\ p$	$\dfrac{\sigma \not\Vdash [l,D] : \varphi}{\sigma \not\Vdash [l,D] : \varphi\ Until_\emptyset\ p} \qquad \dfrac{(l,u') \longrightarrow \cdots \not\Vdash [l, D^{\uparrow l}] : (\varphi\ Until_\emptyset\ p)}{(l,u) \xrightarrow{\epsilon(d)} (l,u') \longrightarrow \cdots \not\Vdash [l,D] : (\varphi\ Until_\emptyset\ p)}$ $\dfrac{(l',u') \longrightarrow \cdots \not\Vdash [l, r(g \wedge D)] : (\varphi\ Until_\emptyset\ p)}{(l,u) \xrightarrow{a} (l',u') \longrightarrow \cdots \not\Vdash [l,D] : (\varphi\ Until_\emptyset\ p)} \quad \begin{array}{l} l \xrightarrow{g,a,r} l',\ u' = r(u), \\ u \in (D \wedge g){\downarrow}C \end{array}$

Table 2. Inference rules for $\not\Vdash$.

The search will be terminated on the Fail criterion if all the possibilities of backtracking have been exhausted. It can then be asserted that the automaton A in any state complying with $[l, D]$ satisfies φ. However, if Phase 1 terminates on the Success criterion it follows that $\sigma \not\Vdash [l, D] : \varphi$. The rules in Table 2 provide a way to synthesize the diagnostic trace of the conclusion from a diagnostic trace of the premise, constituting Phase 2. If the search in Phase 1 is performed using a breadth-first strategy, a resulting trace will be a shortest diagnostic trace.

The implementation of both phases relies on efficient implementation of the operations and predicates on clock constraint systems discussed in Section 4.1. In fact, they can be efficiently implemented by representing constraint systems as weighted directed graphs [11, 10].

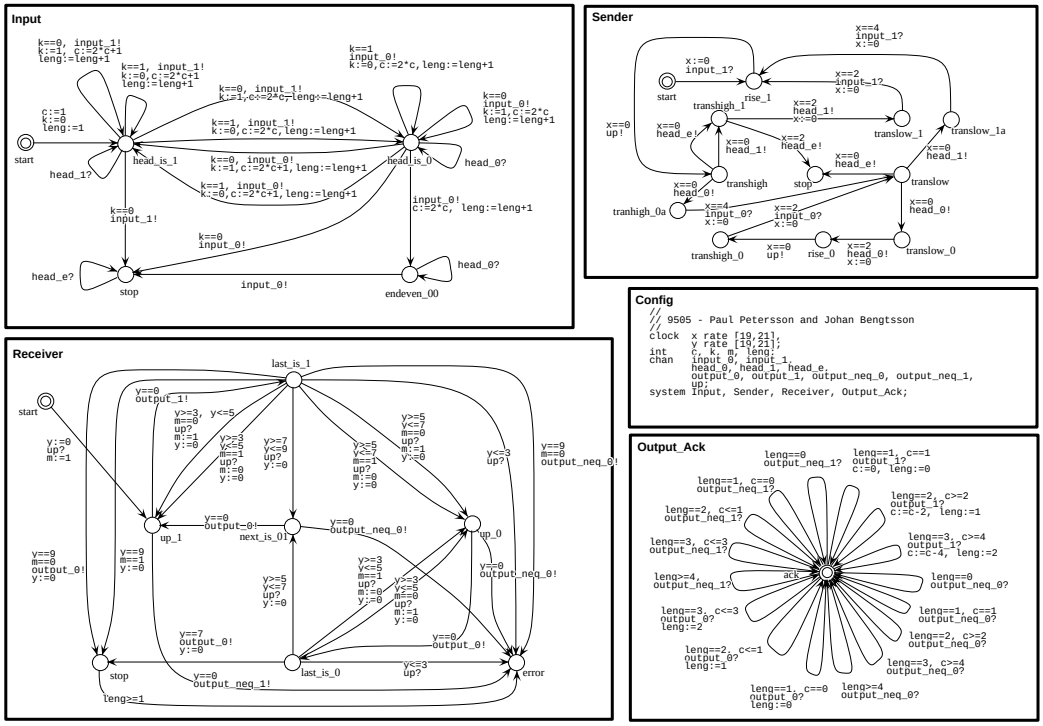

Fig. 1. Philips's Audio-Control Protocol — Final Version.

5 Applications

The techniques presented in previous sections have been implemented in the verification tool UPPAAL. The tool has been used in a case study, where Philips Audio-Control Protocol was verified. We demonstrate the usefulness of the diagnostic model-checking feature of UPPAAL by debugging an early description of the protocol. For detailed information about the tool UPPAAL, see [2] in this volume.

5.1 Philips Audio-Control Protocol

This protocol by Philips was first verified by Bosscher et al [3] and recently using verification tools [6]. The protocol is used for exchanging control information in tiny local area networks between components in modern audio equipment. Bit streams are encoded using the well-known Manchester encoding that relies on timing delay between signals. The protocol uses bit slots of four time units, a 1 bit is encoded by raising the voltage from low to high in the middle of the bit slot. A 0 bit is encoded in the opposite way. The goal of the protocol is to guarantee reliable communication with a tolerance of $\pm 5\%$ on all the timing. The communication is further complicated since the voltage changing from high to low can not be reliably detected. The decoding has to be done using only the changing from low to high. A linear hybrid automaton network description of the protocol is shown in Figure 1.

To perform experiments on the protocol we used an early draft version of a description by Wong–Toi and Ho [6][10]. In their work they automatically verifies the audio-control protocol using the tool HyTech (The Cornell Hybrid Technology Tool is a symbolic model checker for linear hybrid systems). By reusing their description we avoid the difficult and time-consuming work of modelling the protocol. The protocol is modeled as a parallel composition of four processes described below. Several integer variables are used for recording information: `leng` for recording the number of bits generated by the input automaton but not yet acknowledged as being received; `c` for representing the binary encoding of these bits; `k` for recording the parity of the number of bits generated; and `m` for recording the parity of the number of bits received. The four parallel processes are:

- Input. The Input automaton nondeterministically generates valid bit sequences for the Sender automaton. Valid bit sequences are restricted to either odd length or ending in two 0 bits. The values of the integer variables `k`, `c` and `leng` are also updated appropriately. The Input automaton is also used by the Sender automaton to decide the next input bit.
- Sender. This automaton encodes the bit sequences by reading the value of the next bit from the Input automaton and determine the time delay for the next high voltage, modeled as an `up!`-action.
- Receiver. The Receiver automaton decodes the bit stream by measuring the time delay between two subsequent `up?`-actions received from the Sender. The decoded bits are then acknowledged by synchronizing on the `output_1` or `output_0` port with the output-acknowledgment automaton. The Receiver also records the parity of the received number of bits by updating `m`.
- Output_Ack. The output-acknowledgment automaton checks the current number of unacknowledges bits (`leng`) together with their binary encoding (`c`) and acknowledges the bits decoded by the receiver. It also updates the values of the variables `leng` and `c`.

The way the protocol has been modeled enables correctness of the received bits to be verified by reachability analysis. By introducing the edge $\mathsf{stop} \xrightarrow{\texttt{leng}\geq 1} \mathsf{error}$ in the receiver automaton, the received bit stream is guaranteed to be identical to the sent bit stream precisely when the system satisfies the property $Inv(\neg \mathsf{at}(\mathsf{error}))$.

First Version. The first version was an adjusted version of the description in [6]. The adjustments were necessary due to differences in HyTech and Uppaal. This step comprised: transforming the invariant conditions of the original description into enabling conditions of the model in Uppaal; introducing complementary synchronization actions; adding the edge $\mathsf{stop} \xrightarrow{\texttt{leng}\geq 1} \mathsf{error}$ in the receiver automaton; and model the modulo-2 counters `m` and `k` as integer variables. Modulo-2 addition $\oplus$ was modeled as a conditional value assignment on integers (e.g. `m==0, m:=1` or `m==1, m:=0`)[11]. This first version was also free from some obvious typing errors found in the original description of the system.

[10] Available, at that time, from the Web server at Cornell University (`http://www.cs.cornell.edu/`).

[11] Alternatively, modulo-2 addition $\oplus$ can be modeled using the integer assignment `m:=-m+1`.

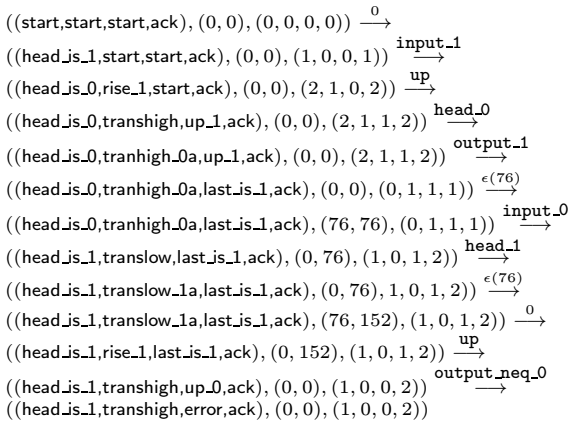

Fig. 2. Diagnostic Trace from the First Version of the Protocol.

The protocol was then attempted verified but found erroneous[12]. Using the diagnostic trace shown in Figure 2[13][14], automatically synthesized by UPPAAL, the system was further improved. The trace indicates errors in several ways. First recall that the existence of a trace implies that the correctness property is not satisfied. This particular trace is wrong since a `head_1`-action is followed by a subsequent `up`-action without an interjacent `input_1`-action. Also, from the diagnostic trace in Figure 2, it was revealed that the action labels `output_neq_1?` and `output_neq_0?` was swapped in the Output_Ack automaton. This must be the case since `c` = 1 and `leng` = 2 implies that the next output should be 0 while `output_neq_0?` is signaled to acknowledge that the next output can *not* be 0.

Improved Version no.1. In the first improved version, missing actions `input_1?` on the edges translow_1 ⟶ rise_1 and translow_1a ⟶ rise_1 in the Sender automaton was added. Furthermore, the action labels `output_neq_0?` and `output_neq_1?` was swapped in the Output_Ack automaton.

Once again, we attempted to verify the system; the systems was found erroneous. From the diagnostic trace shown in Figure 3, a timing error was discovered. In the control-state (endeven_00,transhigh_0,up_1,ack) the Receiver automaton has decoded a 1 bit, but this is not the bit sent by the sender. The disagreement is monitored by the Output_Ack automaton that makes the system violating the correctness property by offering an `output_neq_1?`-action. The reason for this error was found on the edges last_is_1 ⟶ next_is_01 and last_is_1 ⟶ up_0 where the enabling conditions on clock `y` was swapped.

[12] UPPAAL, installed on a SparcStation 10, performs the attempted verification and reports a diagnostic trace in 2.2 seconds.

[13] The states are shown in this trace as triples, where the first component is the control-node, the second component is the clock assignment for the clocks `x` and `y`, and the third component is the value assignment for the auxiliary variables `c`, `k`, `m` and `leng`.

[14] This is a trace of the transformed version of the description, where the non-zero linear hybrid automata have been compiled into timed automata.

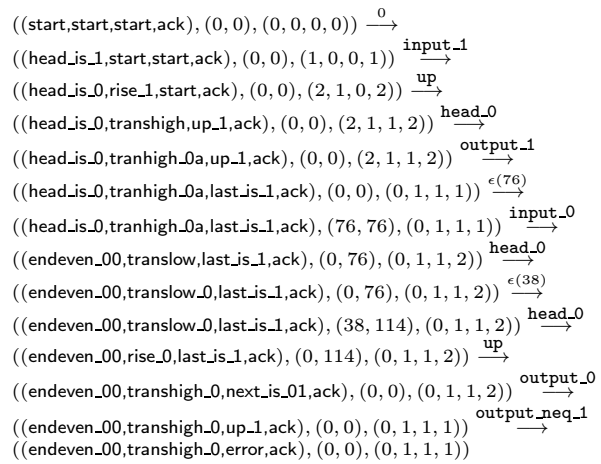

Fig. 3. Diagnostic Trace from the First Improved Version of the Protocol.

Improved Version no.2. An even further improved version was made by swapping the enabling conditions on clock y between the edges last_is_1 ⟶ next_is_01 and last_is_1 ⟶ up_0 in the receiver automaton.

Once again a diagnostic trace was produced. The error was found by inspection of the action sequence. In the control-node (head_is_0,tranhigh_0a,last_is_1,ack) three output_1-actions and one output_0 has been performed but the value of m indicates an odd parity of the accumulated output bit stream. We concluded that some update operation of m was wrong or missing.

Final Version. When the modulo-2 addition on the variable m was removed from the edges last_is_1 ⟶ next_is_01 and last_is_0 ⟶ next_is_01 in the receiver automaton we got the final version of the protocol (Figure 1). By adjusting the rate of the senders and the receivers clocks (i.e. x and y) it can be confirmed that the correctness property is not satisfied if the tolerance is equal to $\pm\frac{1}{17}$.

6 Conclusion and Future Work

In this paper we have presented a diagnostic model-check procedure for real-time systems, capable of, not only deciding if a property is satisfied by a model, but also providing a violating trace whenever the property is not satisfied. Such a trace may be considered as diagnostic information of the error, useful during the subsequent debugging of the model. This principle could be called WYDVYAE.

The presented techniques have been implemented in the new verification tool UPPAAL. Besides a diagnostic model-checker for networks of timed automata, the UPPAAL tool kit have a graphical interface (Autograph), allowing system descriptions to be defined by drawing and thereby allowing the user to see what is verified, i.e. WYSIWYV. In this way, a number of errors can be avoided. In a case study where UPPAAL was used to verify (a version of) Philips Audio-Control Protocol, both the graphical interface and

the automatically generated diagnostic traces proved useful for detecting and correcting several errors in the description of the protocol.

A diagnostic trace, generated by the current version of UPPAAL, is sometimes unnecessarily long. Thus, future work includes implementing synthesis of a *shortest* diagnostic trace. Another future extensions will follow the principle of WYSIWYV. Whenever needed, clock assignments of a diagnostic trace will be transformed back into values in accordance with the original description. This is sometimes needed since UPPAAL is able to compile descriptions of certain types of hybrid systems into timed automata.

Acknowledgment

The UPPAAL tool has been implemented in large parts by Johan Bengtsson and Fredrik Larsson. The authors would like to thank them for their excellent work and also for several discussions concerning the verification of the Audio Control Protocol.

References

1. R. Alur and D. Dill. Automata for Modelling Real-Time Systems. *Theoretical Computer Science*, 126(2):183–236, April 1994.
2. Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. UPPAAL— a Tool Suite for Automatic Verification of Real–Time Systems. In *Proc. of the 4th DIMACS Workshop on Verification and Control of Hybrid Systems*, Lecture Notes in Computer Science, October 1995.
3. D. Bosscher, I. Polak, and F. Vaandrager. Verification of an Audio-Control Protocol. In *Proc. of FTRTFT'94*, volume 863 of *Lecture Notes in Computer Science*, 1993.
4. Nicolas Halbwachs. Delay Analysis in Synchronous Programs. *Lecture Notes in Computer Science*, 697, 1993. In Proc. of CAV'93.
5. T. A. Henzinger, Z. Nicollin, J. Sifakis, and S. Yovine. Symbolic Model Checking for Real-Time Systems. In *Logic in Computer Science*, 1992.
6. Pei-Hsin Ho and Howard Wong-Toi. Automated Analysis of an Audio Control Protocol. In *Proc. of CAV'95*, volume 939 of *Lecture Notes in Computer Science*. Springer Verlag, 1995.
7. H. Hüttel and K. G. Larsen. The use of static constructs in a modal process logic. *Lecture Notes in Computer Science, Springer Verlag*, 363, 1989.
8. F. Laroussinie and K.G. Larsen. Compositional Model Checking of Real Time Systems. In *Proc. of CONCUR'95*, Lecture Notes in Computer Science. Springer Verlag, 1995.
9. F. Laroussinie and K.G. Larsen. From Timed Automata to Logic — and Back. In *Proc. of MFCS'95*, Lecture Notes in Computer Sciencie, 1995. Also BRICS report series RS–95–2.
10. K.G. Larsen, P. Pettersson, and W. Yi. Compositional and Symbolic Model-Checking of Real-Time Systems. To appear in *Proc. of the 16th IEEE Real-Time Systems Symposium*, December 1995.
11. Mihalis Yannakakis and David Lee. An efficient algorithm for minimizing real–time transition systems. In *Proceedings of CAV'93*, volume 697 of *Lecture Notes in Computer Science*, pages 210–224, 1993.
12. Wang Yi, Paul Pettersson, and Mats Daniels. Autfomatic Verification of Real-Time Communicating Systems By Constraint-Solving. In *Proc. of the 7th International Conference on Formal Description Techniques*, 1994.

This article was processed using the LaTeX macro package with LLNCS style

Recent Publications in the BRICS Report Series

RS-96-57 Kim G. Larsen, Paul Pettersson, and Wang Yi. *Diagnostic Model-Checking for Real-Time Systems*. December 1996. 12 pp. Appears in Alur, Henzinger and Sontag, editors, *DIMACS Workshop on Verification and Control of Hybrid Systems*, HYBRID '96 Proceedings, LNCS 1066, 1996, pages 575–586.

RS-96-56 Zine-El-Abidine Benaissa, Pierre Lescanne, and Kristoffer H. Rose. *Modeling Sharing and Recursion for Weak Reduction Strategies using Explicit Substitution*. December 1996. 35 pp. Appears in Kuchen and Swierstra, editors, *8th International Symposium on Programming Languages, Implementations, Logics, and Programs*, PLILP '96 Proceedings, LNCS 1140, 1996, pages 393–407.

RS-96-55 Kåre J. Kristoffersen, François Laroussinie, Kim G. Larsen, Paul Pettersson, and Wang Yi. *A Compositional Proof of a Real-Time Mutual Exclusion Protocol*. December 1996. 14 pp. To appear in Dauchet and Bidoit, editors, *Theory and Practice of Software Development. 7th International Joint Conference CAAP/FASE*, TAPSOFT '97 Proceedings, LNCS, 1997.

RS-96-54 Igor Walukiewicz. *Pushdown Processes: Games and Model Checking*. December 1996. 31 pp. Appears in Alur and Henzinger, editors, *8th International Conference on Computer-Aided Verification*, CAV '96 Proceedings, LNCS 1102, 1996, pages 62–74.

RS-96-53 Peter D. Mosses. *Theory and Practice of Action Semantics*. December 1996. 26 pp. Appears in Penczek and Szalas, editors, *Mathematical Foundations of Computer Science: 21st International Symposium*, MFCS '96 Proceedings, LNCS 1113, 1996, pages 37–61.

RS-96-52 Claus Hintermeier, Hélène Kirchner, and Peter D. Mosses. *Combining Algebraic and Set-Theoretic Specifications (Extended Version)*. December 1996. 26 pp. Appears in Haveraaen, Owe and Dahl, editors, *Recent Trends in Data Type Specification: 11th Workshop on Specification of Abstract Data Types, joint with 8th COMPASS Workshop*, Selected Papers, LNCS 1130, 1996, pages 255–274.