Verification of an Audio Control Protocol

Doeko Bosscher, Indra Polak, Frits Vaandrager

Computer Science/Department of Software Technology

CS-R9445 1994

Verification of an Audio Control Protocol

Doeko Bosscher
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
doeko@cwi.nl

Indra Polak
Department of Philosophy, Utrecht University
P.O.Box 80126,3508 TC Utrecht, The Netherlands
ipolak@phil.ruu.nl

Frits Vaandrager
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
fritsv@cwi.nl
Programming Research Group, University of Amsterdam
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands

Abstract

We analyze a simple version of a protocol developed by Philips for the physical layer of an interface bus that connects the various devices of some stereo equipment (tuner, CD player,...). The protocol, which uses Manchester encoding, has to deal with a significant uncertainty in the timing of events, due to both hardware and software constraints. We present a formal specification of the protocol, and a proof of correctness for the case where the tolerance of the clocks used within the system is less than $\frac{1}{17}$. A counterexample shows that the protocol fails for tolerances greater than or equal to this value. The verification is carried out using a model of linear hybrid systems, which is similar to the phase transition system model of Manna and Pnueli, and the model of linear hybrid automata of Alur, Henzinger and Ho. The semantics of linear hybrid systems is defined via a translation to the timed I/O automata model of Lynch and Vaandrager.

AMS Subject Classification (1991): 68M10, 68Q05, 68Q25, 68Q60, 68Q75.
CR Subject Classification (1991): C.2.2, C.2.5, C3, D.2.4, F.1.1, F.3.1.
Keywords & Phrases: Communication protocols, Manchester encoding, consumer electronics, protocol verification, real-time systems, timed I/O automata, linear hybrid systems.
Note: The first and third author were supported by Esprit BRA 7166 CONCUR 2.

1. Introduction

Hybrid systems are reactive systems that consist of both digital and analog components. The digital components are typically computers or microprocessors controlled by programs, whereas the analog components will be continuously changing environment variables. Due to the rapid development of processor and circuit technology, we see more and more devices, ranging from aircraft and cars to consumer electronics, in which software interacts with physical processes. Consequently, the specification, design and verification of hybrid systems has recently become an active area of research [9].

We have the opinion that the development of formal methods should go hand in hand with the application on realistic case studies. Following this philosophy, we report in this paper on the analysis of a simple protocol for the physical layer of an interface bus that connects the devices of a stereo equipment. The protocol, which is a simple but realistic fragment of what actually runs within Philips equipment that one can buy in the shop, is naturally described as a hybrid system. In the rest of this introduction we will first outline the protocol, and then discuss the model of "linear" hybrid systems that we developed for our analysis.

1.1 The Protocol

In modern audio equipment the various components (amplifier, tuner, CD player, boxes,...) are not only connected by cables that transport the audio signals, but also by a special cable (the "bus") for the exchange of control information. This tiny "local area network" makes it possible for the different devices to talk to each other, and to offer a series of new, useful services to the consumer. A consumer can for instance wake up the whole system by touching a single button: there is no need to switch on the tuner first, then the CD player, then the amplifier, etc. Instead the system will do this job by broadcasting a "wake up" message over the network. Also, it becomes possible to offer a "dubbing service": by pressing a single button the consumer can order the system to transpose a compact disc to a cassette. He/she does no longer have to go through the whole button-pushing protocol that was previously needed to copy music. Instead the system knows the protocol and implements it by using the network.

Of course it is well-known how to build a local-area network. The only reason why it is difficult in this particular case is that it must be cheap: consumers are only willing to pay a tiny bit more for the additional services provided by the network. In fact, the only additional hardware that Philips needs to implement the network consists of a few transistors, resistors, etc., for the bus interface. The software runs on microprocessors that have to be present anyway. Because the clocks of these microprocessors drift, and because sometimes the programs dealing with the network have to wait for other programs that run on the same microprocessors, the network protocol has to deal with a significant uncertainty in the timing of events. In fact, Philips allows for a tolerance of $\pm 5\%$ on all the timing. The goal of the protocol that we have analyzed is to achieve reliable communication between the devices despite this very large timing uncertainty.

The protocol uses the well-known Manchester encoding of bit strings. In this encoding it is assumed that the voltage on the bus is either *high* or *low*. The time axis is divided into *bit slots* of equal length, one bit slot for each bit in the string. In the block-shaped signal that is sent over the bus, a bit "0" is represented by a downgoing edge in the middle of a bit slot, and a bit "1" is represented by an upgoing edge in the middle of a bit slot. If the same bit is sent twice in a row then an additional edge is required, which is placed exactly in between the corresponding two bit slots. An example of the encoding is displayed in Figure 1.1. Besides the $\pm 5\%$ tolerance on the timing, the protocol has to face a number of other difficulties:

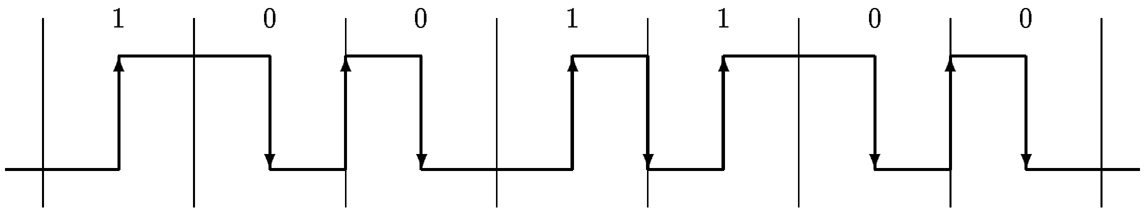

Figure 1.1: Manchester encoding of the string "1001100"

1. Although a receiver knows the length of the bit slots ($888\mu s$), it does not know the time at which the first bit slot begins. This problem is resolved by requiring that the voltage on the bus is low whenever no message is being transmitted, and that the first bit of a string is always "1". Thus, when a receiver sees the first upgoing edge it knows that this occurs in the middle of the first bit slot.
2. A receiver does not know the length of the bit string it is receiving.
3. In reality, the signal on the bus does not have a block shape. In particular, it takes a significant amount of time for the voltage on the wire to change from high to low. This means that it is not possible for the hardware at the receiver side to determine reliably when downgoing edges occur, and for this reason the receiver has to decode the signal without "seeing" the downgoing edges. This is always possible, except that a message ending on "10" cannot be distinguished from the same message ending on "1". To resolve this problem, we require that all messages either end on "00" or have an odd length.
4. Different senders may start sending at approximately the same time, so bus collisions may occur.
5. The message delay in the bus can be significant.

For simplicity we ignore in this paper all the exciting complications that arise from problems 4 and 5, and consider a setting where one sender and one receiver are communicating through a bus in which messages travel with negligible delay. The specific question that we will address is to find the maximal allowable tolerance on the timing for which a specific decoding algorithm developed by Philips is still correct in this setting.

1.2 Linear Hybrid Systems

Abadi and Lamport [1] plead for the use of "old-fashioned recipes" when developing methods for specifying and reasoning about computer systems:

> A new class of systems is often viewed as an opportunity to invent a new semantics. A number of years ago, the new class was distributed systems. More recently, it has been real-time systems. The proliferation of new semantics may be fun for semanticists, but developing a practical method for reasoning about systems is

a lot of work. It would be unfortunate if every new class of systems required inventing new semantics, along with proof rules, languages, and tools. [1, page 1]

Following this philosophy, they show in [1] how real-time systems can be handled in TLA [12], a Temporal Logic of Actions that was originally proposed in the context of untimed systems. In subsequent work Lamport demonstrates how TLA can deal with hybrid systems [13]. We agree with Abadi and Lamport, and would like to carry out their program for the I/O automata model of Lynch and Tuttle [16]. This model has been highly successful in the area of distributed systems (for some examples of recent applications see [14, 15, 10]), and so it seems interesting to investigate whether it can handle hybrid systems as well.

The I/O automata model is based on *labeled transition systems.* In the untimed case the transition labels can be *input* and *output actions*, which model the interaction of a system with its environment, and *internal actions*, which model internal computation steps. In [26, 18] it is shown how real-time systems can be represented as labeled transition systems by adding, as additional labels, *time-passage actions.* In the resulting model of *timed I/O automata*, the continuous progress of real-time is represented by a continuum of discrete time-passage transitions. The new model of *linear hybrid systems* that we will define in this paper can be viewed as a subclass of timed I/O automata.

An important aspect of the I/O automata model is the view on correctness: both a system and its specification are described as I/O automata and correctness amounts to inclusion of traces between these automata. In [26, 18, 7], it is argued that inclusion of *timed traces* provides the "right" notion of implementation for timed I/O automata, and it is shown that the simulation proof techniques that have been developed for the untimed model carry over smoothly to the timed case. In [7], an embedding of the untimed model into the timed model is presented, which makes it possible to view a timed I/O automaton as an implementation of an untimed I/O automaton via the notion of timed trace inclusion. Here, we will also use inclusion of timed traces as the implementation relation between linear hybrid systems.

A final characteristic of the I/O automata approach is the specific syntax that is used for defining systems: by viewing states as valuations of a collection of state variables, transitions can be defined in guarded command style or equivalently via *action predicates* with primed and unprimed variables that refer to the values of the state variables before and after a transition. In our approach the only difference at the syntactic level between the timed and the untimed model is that in the timed case there is an additional action predicate to specify time-passage transitions. Via our notion of a *linear hybrid system* we present syntactic restrictions on this predicate which guarantee that the underlying transition system is a timed I/O automaton, and in particular satisfies the *trajectory axiom.* The trajectory axiom is a fundamental property saying that if there is a time step between two states, there exists a continuum of intermediate states that are also related by time steps. In a linear hybrid system the state variables are partitioned in *discrete* and *continuous* variables, just like in the phase transition systems of Manna and Pnueli [20, 19]. But whereas phase transition systems and other models that have been proposed for hybrid systems contain several additional components (*activities*, *important events*, *invariants*, *rate intervals*, *interval sequences*, *evolutions*, etc.), the operational behavior of a linear hybrid system is defined by action predicates only, just as in the untimed case. Thus all the "old-fashioned" notations and proof techniques for labeled transition systems carry over immediately, and we do not have to redefine and reinvent from scratch. We want to stress however that there exists a close connection between the syntactic restrictions that we impose on the action predicate for time, and concepts that play a role in the phase transition systems of Manna and Pnueli, and in the linear hybrid automata of Alur, Henzinger and Ho [4, 2].

One could argue that the case study presented in this paper is not dealing with hybrid systems in the proper sense. All real-time systems are hybrid systems, and this paper concerns a real-time verification problem because instead of our model of linear hybrid systems we could have used (for instance) the more restricted timed automata model of Alur and Dill [3]. In this model there are continuous entities, called clocks, but these all change with exactly the same rate. The main reason why we use linear hybrid systems in this paper is that they allow for a more natural description of the audio control protocol than the timed automata of [3].

2. The Timed I/O Automata Model

In this section we give a brief account of the model of *timed I/O automata* [26, 18, 7], which is a variant of the I/O automata model of Lynch and Tuttle [16] extended with features to model real-time, but without a notion of fairness. As the time domain we use in this paper the set R of real numbers, with typical elements $d, d_1, \ldots$

2.1 Action Signatures

An *action signature* S is a triple $(in(S), out(S), int(S))$ of three disjoint sets of respectively *input actions*, *output actions* and *internal actions*. We assume that these sets are disjoint with the set R^+ of positive real numbers, elements of which set will play the role of *time-passage actions*. The derived sets of *external actions*, *locally controlled actions* and *actions* of S are defined respectively by

$$\begin{aligned} ext(S) &\triangleq in(S) \cup out(S), \\ local(S) &\triangleq out(S) \cup int(S), \\ acts(S) &\triangleq in(S) \cup out(S) \cup int(S). \end{aligned}$$

An action signature is called *finite* if all three components are finite sets.

2.2 Timed I/O Automata

A *timed I/O automaton* (or just *automaton*) A consists of four components[1]:

- A set $states(A)$ of *states*.
- A nonempty set $start(A) \subseteq states(A)$ of *start states*.
- An action signature $sig(A)$
  (we write $in(A)$ for $in(sig(A))$, $out(A)$ for $out(sig(A))$, etc.).

[1]This definition is a slight variant of the definition from [18]. The difference is just the explicit indication of the amount of elapsed time in the time-passage action instead of using a *.now* function that associates the current time to a state.

- A set $steps(A) \subseteq states(A) \times (acts(A) \cup \mathsf{R}^+) \times states(A)$ of *transitions*.

In this paper $s, s', u, u', \ldots$ range over states, and $a, b, \ldots$ over actions in $acts(A) \cup \mathsf{R}^+$. As usual $s \xrightarrow{a}_A s'$ abbreviates $(s, a, s') \in steps(A)$. The subscript A is frequently omitted when no confusion can arise. An action a of automaton A is said to be *enabled* in a state s if $s \xrightarrow{a} s'$, for some state s'. We require that A satisfies the following axioms:

A1 Each input action is enabled in each state.

A2 If $s \xrightarrow{d} s'$ and $s' \xrightarrow{d'} s''$, then $s \xrightarrow{d+d'} s''$.

To state the last axiom, some auxiliary definitions are needed. An *interval* is a nonempty convex subset of R. Let I be an interval. Then an *I-trajectory* is a function $w : I \rightarrow states(A)$ such that

$$w(d) \xrightarrow{d'-d} w(d') \text{ for all } d, d' \in I \text{ with } d < d'.$$

If I is left-closed, then denote $w(inf(I))$ by $w.fstate$. Similarly if I is right-closed, then denote $w(sup(I))$ by $w.lstate$. If I is closed then w is said to span *from* state $w.fstate$ *to* state $w.lstate$.

A3 If $s \xrightarrow{d} s'$ then there exists an $[0, d]$-trajectory from s to s'.

Axiom **A1** is also referred to as the *input-enabling* requirement. The intuition behind this axiom is that input actions are under control of the environment, and that the system cannot prevent the environment from doing these actions. Axiom **A2** gives a natural property of time, namely that if time can pass in two steps, then it can also pass in a single step. Finally, the *trajectory axiom* **A3** says that if time can pass with an amount d, then it is possible to associate states with all times in the interval $[0, d]$ in a consistent way. For a further discussion of this axiom we refer to [18, 11].

2.3 Composition and Hiding

Intuitively, the composition of two timed I/O automata is their Cartesian product, with the added requirement that automata synchronize on shared actions and on passage of time. The synchronization of shared actions models communication between system components: if a is an output action of A and an input action of B, then the simultaneous performance of a models communication from A to B. Since we do not want synchronization between output actions of different I/O automata, or synchronizations involving internal actions, we require that the I/O automata are *compatible* in the sense that they do not share these actions.

Formally, we say that two action signatures S_1 and S_2 are *compatible* if $out(S_1) \cap out(S_2) = \emptyset$, $int(S_1) \cap acts(S_2) = \emptyset$, and $int(S_2) \cap acts(S_1) = \emptyset$. The *composition* $S_1 \| S_2$ of a pair of compatible action signatures S_1, S_2 is defined to be the action signature S with

$$\begin{aligned} in(S) &= (in(S_1) \cup in(S_2)) - (out(S_1) \cup out(S_2)), \\ out(S) &= out(S_1) \cup out(S_2), \\ int(S) &= int(S_1) \cup int(S_2). \end{aligned}$$

It is easy to see that S is an action signature and that composition of action signatures is commutative and associative. We say that a pair of timed I/O automata is *compatible* if their action signatures are compatible. The *composition* $A_1 \| A_2$ of a pair of compatible timed I/O automata A_1, A_2 is the timed I/O automaton defined by

- $states(A) = states(A_1) \times states(A_2)$,
- $start(A) = start(A_1) \times start(A_2)$,
- $sig(A) = sig(A_1) \| sig(A_2)$,
- $steps(A)$ is the set of triples $((s_1, s_2), a, (s_1', s_2'))$ in $states(A) \times (acts(A) \cup \mathsf{R}^+) \times states(A)$ such that, for $i \in \{1, 2\}$, if $a \in acts(A_i) \cup \mathsf{R}^+$ then $s_i \xrightarrow{a}_{A_i} s_i'$ else $s_i = s_i'$.

The reader can check that A is a timed I/O automaton indeed, and that composition of timed I/O automata is commutative and associative up to isomorphism. Note that, by definition, time is only allowed to pass in the composition if both component automata allow the same amount of time to pass.

If S is an action signature and $H \subseteq out(S)$, then the action signature $\mathsf{HIDE}\ H\ \mathsf{IN}\ S$ is defined as the triple $(in(S), out(S) - H, int(S) \cup H)$. If A is a timed I/O automaton and $H \subseteq out(A)$, then $\mathsf{HIDE}\ H\ \mathsf{IN}\ A$ is the timed I/O automaton obtained from A by replacing $sig(A)$ by $\mathsf{HIDE}\ H\ \mathsf{IN}\ sig(A)$, and leaving all the other components unchanged.

2.4 Timed Traces

Let A be a timed I/O automaton. An *execution fragment* of A is a finite or infinite alternating sequence $s_0 a_1 s_1 a_2 s_2 \cdots$ of states and actions in $acts(A) \cup \mathsf{R}^+$, beginning with a state and, if it is finite, ending with a state, such that for all i, $s_i \xrightarrow{a_{i+1}} s_{i+1}$. An *execution* of A is an execution fragment that begins with a start state. A state s of A is *reachable* if it is the last state of some finite execution of A.

Executions correspond to what are called *sampling computations* in [20]: they give information about a run of a system at a countable number of points in time. In [18] also a notion of *timed execution* is defined for timed automata: these are alternating sequences of trajectories and actions, which correspond to the *super-dense computations* of [20]. It can be argued that timed executions provide a more precise representation of the behavior of real-time systems than (sampling) executions. However, our trajectory axiom **A3** guarantees that for each (sampling) execution of a timed automaton there exists a corresponding timed execution. This means that the full externally visible behavior of timed automata can already be inferred from the technically much simpler (sampling) executions:

Suppose $\alpha = s_0 a_1 s_1 a_2 s_2 \cdots$ is an execution fragment of A. For each index i, let d_i be given by

$$\begin{aligned} d_0 &= 0, \\ d_{i+1} &= \mathsf{if}\ a_i \in \mathsf{R}^+\ \mathsf{then}\ d_i + a_i\ \mathsf{else}\ d_i. \end{aligned}$$

The *limit time* of α, notation $\alpha.ltime$, is the smallest element of $\mathsf{R}^{\geq 0} \cup \{\infty\}$ larger than or equal to (i.e., the supremum of) all the d_i. Execution fragment α is *admissible* if $\alpha.ltime = \infty$, and *Zeno* if it is an infinite sequence but with a finite limit time. The *timed trace* $t\text{-}trace(\alpha)$ associated with α is defined by

$$t\text{-}trace(\alpha) \quad \triangleq \quad (((a_1, d_1)(a_2, d_2)\cdots)\lceil(ext(A) \times \mathsf{R}^{\geq 0}), \alpha.ltime).$$

So $t\text{-}trace(\alpha)$ records the external actions of α paired with their time of occurrence, as well as the limit time of the execution. We write $t\text{-}traces^{\infty}(A)$ for the set of traces of admissible executions of A.

2.5 Implementation

Let A and B be timed I/O automata with the same input actions. Then we say that A *implements* B if $t\text{-}traces^{\infty}(A) \subseteq t\text{-}traces^{\infty}(B)$.

In I/O automata theory, inclusion of (fair, admissible,...) traces is commonly used as implementation relation. Intuitively, one may think of B as defining a set of constraints, which A must obey. Note that A does not need to exhibit *all* of the behaviors in $t\text{-}traces^{\infty}(B)$; merely a subset is sufficient. However, by requiring that A and B have the same input actions, and since input actions must always be enabled, implementations can not be completely trivial. Of course, the requirement of input enabling is not enough to exclude trivial implementations. A convincing definition of what it means for an implementation to be non-trivial is presented in [7], via the notion of *environment-freedom.* Intuitively, a timed I/O automaton is *environment-free* if it has a strategy by which, after any finite execution and with any sequence of input actions, it can generate an execution which is either admissible[2] or *Zeno-tolerant*, i.e., Zeno with infinitely many input actions and finitely many locally controlled actions. We claim that all timed I/O automata that we discuss in this paper are environment-free in the sense of [7].

2.6 Simulations

In the literature, a whole menagerie of so-called simulation techniques has been proposed to prove that the set of (finite, fair, timed,...) traces of one automaton is included in that of another. We refer to [17, 18] for an overview and further references. In this paper we only need one simple type of simulation, which is the *weak timed forward simulation* of [18].

Suppose A and B are timed I/O automata. A *weak timed forward simulation* from A to B is a relation $R \subseteq states(A) \times states(B)$ that satisfies the following two conditions[3]:

1. If $s \in start(A)$ then there exists a state $u \in start(B)$ with $s \, R \, u$.
2. If $s \xrightarrow{a}_A s'$, $s \, R \, u$, and s and u are reachable, then there exists a state u' of B such that $u \overset{p}{\leadsto}_B u'$ and $s' \, R \, u'$, where $p = t\text{-}trace(s \; a \; s')$.

Here we write $u \overset{p}{\leadsto}_B u'$ if B has a finite execution fragment α with first state u, final state u' and timed trace p.

[2]Actually, the definition of [7] is presented in the more general setting of *live* timed I/O automata. Our automata can be viewed as live timed I/O automata by taking the collection of admissible executions as the liveness condition.

[3]For simplicity, this definition is slightly less general than the one presented in [18].

Lemma 2.1 *If there exists a weak timed forward simulation from A to B, then $t\text{-}traces^{\infty}(A) \subseteq t\text{-}traces^{\infty}(B)$.*

The converse implication does not hold, i.e., there exist timed I/O automata A and B such that $t\text{-}traces^{\infty}(A) \subseteq t\text{-}traces^{\infty}(B)$, but no weak timed forward simulation from A to B can be given. In those cases one has to use other, more general simulations. Typically, weak timed forward simulations fail if B contains some form of nondeterminism.

3. Linear Hybrid Systems

In this section we define the notion of a linear hybrid system, and describe the semantics of these systems in terms of timed I/O automata.

3.1 Terms and Formulas

We start from a many sorted signature Σ containing a collection of types and predicate, function and constant symbols over those types. We assume that Σ contains equality predicates for each type, and a special type **Real** with constant symbols $0, 1$, a binary predicate symbol $\leq$, binary function symbols $+$ and $\cdot$, and unary function symbols $-$ and $(.)^{-1}$, i.e., the signature of an ordered field.

Next there is a set $\mathcal{V}$ of *variables* containing infinitely many variables for each type. $\mathcal{V}$ is partitioned into two sets V and V' of *unprimed* and *primed* variables, respectively, such that V' is a copy of V that contains for each variable $v \in V$ a corresponding variable v' of the same type. We will use a primed variable v' to denote the new value of an (unprimed) state variable v after a transition. We extend priming of variables to the whole of $\mathcal{V}$ such that $v'' = v$, and let $z, \ldots$ range over $\mathcal{V}$ and $v, w, x, \ldots$ over V.

To describe properties, we use a first-order language over signature Σ and variables $\mathcal{V}$. We use symbols $e, f, \ldots$ to range over terms, and symbols $\phi, \ldots$ to range over formulas. We adopt the usual notational conventions for real valued terms. Thus we write $\frac{e}{f}$ for $e \cdot f^{-1}$, $e > f$ for $\neg(e \leq f)$, 2 for $1 + 1$, etc. With V_e and V_ϕ we denote the set of (free) variables in some term e resp. formula ϕ. For X a term, formula, set of variables, etc, we denote by X' the term, formula, set of variables, etc, obtained by replacing all unprimed variables in X by their primed version.

3.2 Semantics

We assume a Σ-algebra $\mathcal{A}$ which gives meaning to the function and constant symbols of Σ. The type **Real** has the usual interpretation in $\mathcal{A}$ as the set R of real numbers, and also the constant and function symbols for this type are all interpreted as the usual corresponding constants and functions (we set 0^{-1}=1 to keep all functions total).[4] An $\mathcal{A}$-*valuation* is a function ξ that takes every variable $v \in \mathcal{V}$ into an element of its domain in $\mathcal{A}$. If e is a term, then we write $[\![e]\!]^{\xi}_{\mathcal{A}}$ for the evaluation under ξ of e in $\mathcal{A}$, and if ϕ is a formula then we write $\mathcal{A}, \xi \models \phi$ if ϕ holds under ξ.

[4]In fact, for the purposes of this paper any interpretation of **Real** as an ordered field would do: the only properties of reals that we use are the axioms for ordered fields.

3.3 Special Types of Formulas

Let D, C be disjoint sets of variables, with all variables in C of type **Real**. Then the sets *Convex*(D, C) and *Halves*(D, C) consist of the formulas Γ and Δ, respectively, defined by the following grammar

$$\begin{aligned}\Gamma &::= \sum_i e_i x_i \Box e \mid \Gamma \wedge \Gamma \mid \phi \mid \textsf{if } \phi \textsf{ then } \Gamma \textsf{ else } \Gamma \mid \phi \vee \Gamma \mid \Gamma \vee \phi,\\ \Delta &::= \sum_i e_i x_i \Box e \mid \phi \mid \textsf{if } \phi \textsf{ then } \Delta \textsf{ else } \Delta \mid \phi \wedge \Delta \mid \Delta \wedge \phi \mid \phi \vee \Delta \mid \Delta \vee \phi,\end{aligned}$$

where $\Box \in \{<, \leq, \geq, >\}$, the e, e_i are terms of type **Real** with variables in D, the x_i are in C, and ϕ is a formula with free variables in D.

Note that *Halves*$(D, C) \subseteq$ *Convex*(D, C). The key property of formulas in *Convex*(D, C) is that, for fixed interpretation of the variables in D, they denote convex polyhedra in the space spanned by the variables from C. Similarly, the formulas in *Halves*(D, C) denote, for fixed interpretation of the variables in D, the set of points on one side of a hyperplane.

For $W \subseteq V$ a finite set of variables, ϕ an unprimed formula, and $x \in V$, we define

$$\begin{aligned}&\mathit{Unchanged}(W) &&\triangleq \textstyle\bigwedge_{w \in W} w' = w\\ &\mathit{Stable}(\phi) &&\triangleq \phi \rightarrow \phi'\\ &\mathit{Below}(x, \phi) &&\triangleq \exists x' : x \leq x' \wedge \phi[x'/x].\end{aligned}$$

3.4 Syntax of Linear Hybrid Systems

A *linear hybrid system* (or just *system*) P consists of the following components:

- A finite set $V_P \subseteq V$ of *state variables*. The set $V_P = D_P \cup C_P$ is partitioned into D_P, the set of *discrete variables*, and C_P, the set of *continuous variables*. All continuous variables have type **Real**.
- A satisfiable formula *init*(P) with variables from V_P.
- A finite action signature *sig*(P).
- For each $a \in$ *acts*$(P) \cup \{$*TIME*$\}$, a finite list *param*(P, a) of terms, the *action parameters*, and a formula *pred*(P, a), the *action predicate*, such that
  - if $a \in$ *in*(P) then the terms in *param*(P, a) do not contain variables from $V_P \cup V'_P$, and *pred*(P, a) is of the form

    $$\psi \wedge \bigwedge_{v \in V_P} v' = f_v$$

    where ψ does not contain variables from $V_P \cup V'_P$, and the f_v do not contain variables from V'_P,
  - if $a =$ *TIME* then *param*$(P, a) = (t)$, for some variable $t \notin V_P \cup V'_P$, of type **Real**, and *pred*(P, a) is of the form[5]:

[5]Here and elsewhere we use Lamport's list notation for conjunction.

$$\begin{array}{ll}\wedge & t > 0 \\ \wedge & \mathit{Unchanged}(D_P) \\ \wedge & \psi[\sigma] \\ \wedge & \bigwedge_{j \in J} \mathit{Stable}(\psi_j)\end{array}$$

where ψ is in $\mathit{Convex}(D_P, C_P)$, σ is the substitution that replaces each variable $x \in C_P$ by $\frac{x'-x}{t}$, J is a finite index set, and all the ψ_j are in $\mathit{Halves}(D_P, C_P)$.

The action predicate for *TIME* consists of four parts, which intuitively state that: (1) time always passes with a positive amount, (2) time steps leave the discrete variables unchanged, (3) the rates of the n continuous variables are contained in a convex polyhedron over R^n, and (4) time steps preserve a number of properties. Part 1 is obvious. Part 2 says that our model adopts the *two-phase functioning* principle of [22]: a phase where instantaneous (input, output and internal) actions cause discrete changes of the state space, is followed by a phase in which the discrete part of the state remains unchanged and the continuous part is transformed according to a law depending on time progress. This law is given in part 3, and forms a generalization of the notion of *rate intervals* of [4]: it gives lower and upper bounds on the rate of change of linear combinations of the continuous variables with respect to time. Finally, part 4 allows us to state that certain actions other than time passage actions *must* occur before or at some point in time. This part plays the same role as the *invariants* of [4, 2] and the *important events* of [20]. Note that although the "rate" formula ψ can be any formula in $\mathit{Convex}(D_P, C_P)$, we require that the "stable" formulas ψ_j are taken from the smaller set $\mathit{Halves}(D_P, C_P)$. If we would allow for arbitrary conjunctions within a stable formula, then the underlying timed I/O automaton would in general not satisfy the trajectory axiom.

3.5 Semantics of Linear Hybrid Systems

Let P be a linear hybrid system. Define $\mathit{aut}(P)$ to be the 4-tuple A consisting of:

- The set $\mathit{states}(A)$ of valuations of the variables V_P (just these) in their domains.
- The set $\mathit{start}(A)$ of states in $\mathit{states}(A)$ that satisfy formula $\mathit{init}(P)$.
- For each input, output or internal action $a \in \mathit{sig}(P)$ and for each valuation ξ such that $\mathcal{A}, \xi \models \mathit{pred}(P, a)$, $\mathit{sig}(A)$ contains an input, output respectively internal action $a[\![\mathit{param}(P, a)]\!]^{\xi}_{\mathcal{A}}$. Here the result of applying $[\![.]\!]^{\xi}_{\mathcal{A}}$ on a list of terms is defined as the list obtained by applying $[\![.]\!]^{\xi}_{\mathcal{A}}$ on each of these terms separately.
- The set $\mathit{steps}(A)$ of triples (s, b, s') in $\mathit{states}(A) \times (\mathit{acts}(A) \cup \mathsf{R}^+) \times \mathit{states}(A)$ for which there is some $a \in \mathit{acts}(P) \cup \{\mathit{TIME}\}$ and an $\mathcal{A}$-valuation ξ such that
  - $\mathcal{A}, \xi \models \mathit{pred}(P, a)$,
  - $\forall v \in V_P : s(v) = \xi(v)$,
  - $b = a[\![\mathit{param}(P, a)]\!]^{\xi}_{\mathcal{A}}$,
  - $\forall v \in V_P : s'(v) = \xi(v')$.

  Here we identify actions $\mathit{TIME}(d)$ and d. Valuation ξ is called a *witness* for (s, b, s').

Theorem 3.1 *Let P be a linear hybrid system. Then $aut(P)$ is a timed I/O automaton.*

Proof: See Appendix 1. ■

For each trajectory w and for each continuous variable x, define $w_x \triangleq \lambda t.w(t)(x)$ to be the function that gives the value of x along w. The functions w_x are not necessarily differentiable, but if they are then for each state on the trajectory formula $\psi[\rho]$ holds, where ψ is the "rate" formula in the action predicate for *TIME* and ρ is the substitution that replaces each continuous variable x by the first derivative $\dot{w}_x$. From the proof of Theorem 3.1 it follows that whenever there is a trajectory between two states, there is also a "straight line" trajectory l between these states for which the functions l_x are trivially differentiable. If "rate" formula ψ denotes a *bounded* polyhedron for each valuation of the discrete variables, then the functions w_x will be continuous for *each* trajectory w, although not necessarily differentiable.

3.6 Precondition/Effect Notation

When describing systems we will often use guarded commands to specify the action predicates. In the case of "deterministic" action predicates, i.e., ones where the value of the primed variables is determined by that of the unprimed variables and the action parameters, guarded commands allow us to give slightly more concise specifications, since we do not have to state explicitly that certain variables remain unchanged.

Let the set *Stat* of *statements* be defined by the following grammar:

$$Stat \ ::= \ x := e \mid Stat_1 \ Stat_2 \mid \mathsf{if}\ \phi\ \mathsf{then}\ Stat \mid \mathsf{case}\ \phi_1 \Rightarrow Stat_1 \cdots \phi_n \Rightarrow Stat_n,$$

where $x \in V$, e a term and ϕ, ϕ_i formulas, all with unprimed variables only. To each statement *Stat* we associate a formula *Form*(*Stat*) and a finite collection of "modified" variables *Mod*(*Stat*) as follows:

$$\begin{array}{lcl}
Form(x := e) & \triangleq & x'=e \\
Form(Stat_1\ Stat_2) & \triangleq & Form(Stat_1) \wedge Form(Stat_2) \\
Form(\mathsf{if}\ \phi\ \mathsf{then}\ Stat) & \triangleq & \mathsf{if}\ \phi\ \mathsf{then}\ Form(Stat)\ \mathsf{else} \\
 & & \quad Unchanged(Mod(Stat)) \\
Form(\mathsf{case}\ \phi_1 \Rightarrow Stat_1 \cdots \phi_n \Rightarrow Stat_n) & \triangleq & \mathsf{if}\ \phi_1\ \mathsf{then}\ Stat_1\ \mathsf{else} \\
 & & \quad \mathsf{if}\ \phi_2\ \mathsf{then}\ Stat_2\ \mathsf{else} \\
 & & \qquad \vdots \\
 & & \qquad \mathsf{if}\ \phi_{n-1}\ \mathsf{then}\ Stat_{n-1}\ \mathsf{else}\ Stat_n \\
Mod(x := e) & \triangleq & \{x\} \\
Mod(Stat_1\ Stat_2) & \triangleq & Mod(Stat_1) \cup Mod(Stat_2) \\
Mod(\mathsf{if}\ \phi\ \mathsf{then}\ Stat) & \triangleq & Mod(Stat) \\
Mod(\mathsf{case}\ \phi_1 \Rightarrow Stat_1 \cdots \phi_n \Rightarrow Stat_n) & \triangleq & Mod(Stat_1) \cup \cdots \cup Mod(Stat_n)
\end{array}$$

In the definition of a system P an expression

$a(e_1, \ldots, e_n)$

Precondition
ϕ
Effect
Stat

will be interpreted as a definition of the action parameter and predicate of an action a:

$$\begin{array}{lcl} param(P,a) & \triangleq & (e_1,\ldots,e_n), \\ pred(P,a) & \triangleq & \phi \wedge Form(Stat) \wedge Unchanged(V_P - Mod(Stat)). \end{array}$$

In this case, we write $prec(P,a)$ for the formula ϕ.

4. Protocol Specification

In this section, we present the formal specification of the protocol. Following a brief description of the many-sorted algebra that we use, we will first give linear hybrid systems for each of the components of the protocol, and then define the full protocol as the composition of the automata denoted by these systems. At the end of this section we will moreover present the definition of the system that specifies the allowed external behavior of the protocol.

4.1 Data Types

We start the specification of the protocol with a description of the various data types that we will need. We assume a many-sorted signature Σ and a Σ-algebra $\mathcal{A}$ which, besides the ingredients listed in Section 3.1, consist of the following components:

- a type **Nat** of natural numbers, with a constant symbol zero, a successor function symbol succ, and a predicate symbol odd, all with the usual interpretation. Also, there is an embedding $\iota : \mathbf{Nat} \to \mathbf{Real}$ of the natural numbers into the reals. We will suppress ι's in terms.

- a type **Bit** of bits that the protocol has to transmit, with constants symbols 0 and 1. Again there is an embedding $\iota : \mathbf{Bit} \to \mathbf{Real}$, which we will suppress in terms.

- a type **List**, with as domain the collection of finite lists of bits. There is a constant symbol ϵ for the empty list, an embedding $\langle . \rangle : \mathbf{Bit} \to \mathbf{List}$, and a binary function symbol $\hat{}$, denoting concatenation of lists. Besides these constructors, there are function symbols

  $$\begin{array}{lll} \mathsf{head} : \mathbf{List} \to \mathbf{Bit} & \mathsf{last} : \mathbf{List} \to \mathbf{Bit} & \mathsf{length} : \mathbf{List} \to \mathbf{Nat} \\ \mathsf{tail} : \mathbf{List} \to \mathbf{List} & \mathsf{last\_two} : \mathbf{List} \to \mathbf{List} & \end{array}$$

  head takes the first element of a list (defined arbitrarily as 0 in case of the empty list), tail returns the remainder of a list after removal of the first element, last gives the last element of a list, last_two gives the last two elements of a list, and length returns the length of a list. These operations are fully characterized by the axioms (here m is a variable of type **List**, and d, e are variables of type **Bit**):

$$\begin{array}{lcl@{\qquad}lcl}
\mathsf{head}(\epsilon) & = & 0 & \mathsf{last\_two}(\epsilon) & = & \epsilon \\
\mathsf{head}(\langle d\rangle\hat{\ }m) & = & d & \mathsf{last\_two}(\langle d\rangle) & = & \langle d\rangle \\
\mathsf{tail}(\epsilon) & = & \epsilon & \mathsf{last\_two}(m\hat{\ }\langle de\rangle) & = & \langle de\rangle \\
\mathsf{tail}(\langle d\rangle\hat{\ }m) & = & m & \mathsf{length}(\epsilon) & = & \mathsf{zero} \\
\mathsf{last}(\epsilon) & = & 0 & \mathsf{length}(\langle d\rangle\hat{\ }m) & = & \mathsf{succ}(\mathsf{length}(m)) \\
\mathsf{last}(m\hat{\ }\langle d\rangle) & = & d & & &
\end{array}$$

Here (and elsewhere) we write $\langle de\rangle$ for $\langle d\rangle\hat{\ }\langle e\rangle$. Finally, we need an operation finalize : **List** $\rightarrow$ **List** defined by:

$$\text{if } \mathsf{last}(m){=}1 \wedge \mathsf{odd}(\mathsf{length}(m)) \text{ then } \mathsf{finalize}(m){=}m \text{ else } \mathsf{finalize}(m){=}m\hat{\ }\langle 0\rangle.$$

- a type **Bool** of booleans with constant symbols true and false. We view boolean valued terms as formulas and use b as an abbreviation of b=true.
- a function symbol min : **Real** $\times$ **Real** $\rightarrow$ **Real**, with the obvious interpretation, and two constant symbols Q and T of type **Real**:
  - Q denotes one quarter of the length of a bit slot in the Manchester encoding. In the Philips specifications Q equals $222\mu s$.
  - T gives the tolerance on the timing of the sender and receiver in the protocol. Philips allows a maximum tolerance of $\pm\frac{1}{20}$.

In this paper we will assume $\mathsf{Q} > 0$ and $0 \leq \mathsf{T} < 1$.

4.2 The Sender

We now define the system S, which models the sender of the protocol. The discrete variables of S are a variable *list*, which records the bit string still to be transmitted, a boolean *wire_high* to keep track of the voltage on the wire, and a boolean *transmitting* which records whether the sender is busy transmitting. There is also a continuous variable x which represents a drifting clock with tolerance T that is reset in the middle of each bit slot. The input action $IN(m)$ corresponds to a request by the environment to transmit a bit string m. Upon the occurrence of such an action in the initial state, S immediately does an *UP*-action, which represents an upgoing edge on the bus. Depending on whether the second bit in the string is absent, 0 or 1, a *DOWN*-action occurs 2Q or 4Q time units after the first *UP*, according to the local clock of S. An action *DOWN* of course represents a downgoing edge on the bus. Subsequent *UP*'s and *DOWN*'s are generated as required by the Manchester encoding, and when the transmission is finished the protocol returns to its initial state. *IN*-actions that occur before the transmission finishes are ignored.

Inputs	*IN* : **List**		
Outputs	*UP*		
Internals	*DOWN*		
Discrete	*transmitting* : **Bool**	**Init**	$\wedge\ \neg$*transmitting*
	wire_high : **Bool**		$\wedge\ \neg$*wire_high*
	list : **List**		$\wedge$ *list*=ϵ
Continuous	x : **Real**		

```
IN(m)
  Precondition
    ∧ head(m)=1
    ∧ (odd(length(m)) ∨ last_two(m)=⟨00⟩)
  Effect
    if ¬transmitting ∧ ¬wire_high ∧ list=ϵ then  [list := m
                                                  x := 0]
```

```
UP
  Precondition
    ∧ ¬wire_high
    ∧ list≠ϵ
    ∧ if transmitting then (if head(list)=1 then x=4Q else x=2Q) else x=0
  Effect
    transmitting := true
    wire_high := true
    if head(list)=1 then  [list := tail(list)
                           x := 0]
```

```
DOWN
  Precondition
    ∧ wire_high
    ∧ if list≠ϵ ∧ head(list)=0 then x=4Q else x=2Q
  Effect
    if list=ϵ ∨ list=⟨0⟩ then  [transmitting := false]
    wire_high := false
    if list≠ϵ ∧ head(list)=0 then  [list := tail(list)
                                    x := 0]
```

```
TIME(t)
  Action formula
    ∧ t > 0
    ∧ Unchanged(D_S)
    ∧ 1 − T ≤ (x′−x)/t ≤ 1 + T
    ∧ Stable(Below(x, prec(S, UP)))
    ∧ Stable(Below(x, prec(S, DOWN)))
```

4.3 The Receiver

Next we define system R, which models the receiver of the protocol. System R has only two state variables: a discrete variable *list*, which gives the bit string received thus far, and a continuous variable x, which represents a drifting clock with tolerance T that is reset whenever an upgoing edge is detected. There are two actions: an action *UP* that corresponds to the detection of an upgoing edge, and an action *OUT* by which the receiver passes a received string on to the environment. The action predicates for *UP* and *OUT* are straightforward formalizations of the informal specifications by Philips of the algorithm for the receiver.

Inputs	*UP*		
Outputs	*OUT* : **List**		
Discrete	*list* : **List**	**Init**	$list=\epsilon$
Continuous	x : **Real**		

```
UP
  Precondition
    true
  Effect
    case
       list=ϵ        ⇒ list := ⟨1⟩
```

```
        last($list$)=0 $\Rightarrow$ case
                                   $x < 3\mathsf{Q}$              $\Rightarrow list := \epsilon$
                                   $3\mathsf{Q} \le x < 5\mathsf{Q}$  $\Rightarrow list := list\hat{}\langle 0\rangle$
                                   $5\mathsf{Q} \le x$              $\Rightarrow list := list\hat{}\langle 01\rangle$
        last($list$)=1 $\Rightarrow$ case
                                   $x < 3\mathsf{Q}$              $\Rightarrow list := \epsilon$
                                   $3\mathsf{Q} \le x < 5\mathsf{Q}$  $\Rightarrow list := list\hat{}\langle 1\rangle$
                                   $5\mathsf{Q} \le x < 7\mathsf{Q}$  $\Rightarrow list := list\hat{}\langle 0\rangle$
                                   $7\mathsf{Q} \le x$              $\Rightarrow list := list\hat{}\langle 01\rangle$
    $x := 0$
```

```
$OUT$(finalize($list$))
    Precondition
        $\wedge\ list \neq \epsilon$
        $\wedge$ if last($list$)=0 then $x$=7Q else $x$=9Q
    Effect
        $list := \epsilon$
```

```
$TIME(t)$
    Action formula
        $\wedge\ t > 0$
        $\wedge\ Unchanged(D_R)$
        $\wedge\ 1 - \mathsf{T} \le \frac{x'-x}{t} \le 1 + \mathsf{T}$
        $\wedge\ Stable(Below(x, prec(R, OUT)))$
```

4.4 The Full Protocol

The full protocol can now be defined as the composition of automata S and R, with communication between these components hidden:

$$Impl \;\triangleq\; \mathsf{HIDE}\ \{UP\}\ \mathsf{IN}\ (S \| R)$$

4.5 The Correctness Criterion

System P defines the collection of allowed behaviors of *Impl*. It has the same input and output actions as *Impl*, but no internal actions. In P each action $IN(m)$ is followed by an action $OUT(m)$ within time

$$\frac{(4\,\mathsf{length}(m) + 5)\,\mathsf{Q}}{1 - \mathsf{T}}$$

However, if the environment offers another IN-action before the system has generated a corresponding OUT-action, P moves to a state of chaos in which anything is possible. This means that in such a situation any behavior of *Impl* is allowed. In the next section we will prove that the *Impl* is indeed a correct implementation of P.

Inputs	IN : **List**		
Outputs	OUT : **List**		
Discrete	$list$: **List**	**Init**	$\wedge\ list=\epsilon$
	$chaos$: **Bool**		$\wedge\ \neg chaos$
Continuous	x : **Real**		

```
$IN(m)$
    Precondition
        $\wedge$ head($m$)=1
        $\wedge$ (odd(length($m$)) $\vee$ last_two($m$)=$\langle 00\rangle$)
    Effect
        if $list=\epsilon$ then  [$list := m$
                                  $x := 0$]
        if $list\neq\epsilon$ then  [$chaos :=$ true]
```

```
$OUT(list)$
    Precondition
        $\vee\ (list\neq\epsilon \wedge (1-\mathsf{T})x \le (4\,\mathsf{length}(list) + 5)\,\mathsf{Q})$
        $\vee\ chaos$
    Effect
        $list := \epsilon$
```

$TIME(t)$

Action formula

$\wedge\ t > 0$
$\wedge\ Unchanged(D_P)$
$\wedge\ x' - x = t$
$\wedge\ Stable(prec(P, OUT))$

5. Correctness Proof

In this section we will establish that there exists a weak timed forward simulation from the implementation to the specification. We first gain insight into the reachable states by presenting a number of *invariants*, i.e., properties that hold initially and that are preserved by the transitions. We have omitted all proofs, which are mostly routine and tedious; the creative part is finding the right invariants and the order in which to prove them.

From now on, we will assume that tolerance T is *less than* $\frac{1}{17}$. The following scenario shows what goes wrong if $\mathsf{T} \geq \frac{1}{17}$. Assume that the sender's clock progresses *maximally slow* and the receiver's clock *maximally fast.* Now the sender and receiver are at rest and the message to be sent is "101". Immediately after the *IN* of this message the sender will output an *UP* to the receiver. Both clocks are (re)set to 0, the buffer of the sender contains "01" and that of the receiver "1". The receiver can output "1" at 9Q local receiver's time, before the last *UP* arrives at 8Q local sender's time, if

$$9\mathsf{Q} \cdot \frac{1-\mathsf{T}}{1+\mathsf{T}} \leq 8\mathsf{Q}.$$

And this is, as the reader can verify, when $\mathsf{T} \geq \frac{1}{17}$.

The variables in the invariants are prefixed with their origin, e.g., $S.x$ for sender's clock x. Besides the variables present in the sender and the receiver, we add to *Impl* a (discrete) boolean history variable *error* that indicates whether *Impl* is in an erroneous or chaotic state. We will need this variable to express that a premature input has occurred. Variable *error* is defined by adding a clause $\neg error$ to the initialization condition of *Impl*, and a clause

$\mathsf{if}\ R.list \neq \epsilon\ \mathsf{then}\ error := \mathsf{true}$

to the effect of *IN* in *Impl*. All the other actions, including *TIME*, leave *error* unchanged. With [23, 18] we know that this is a harmless extension by which, as one can easily verify, the set of timed traces of *Impl* is not changed.

We start with a few invariants about the state space of the sender. The first invariant reflects the observation that the sender is always busy (i.e., transmitting) if the bus is high.

Lemma 5.1 *The following property holds for all reachable states of Impl :*

$$S.wire\_high \quad \rightarrow \quad S.transmitting.$$

The second invariant gives upper bounds for the the various stages of progress of the sender: in the first conjunct the sender is at rest and ready to accept any input, in the second conjunct the sender has received its message but has not yet begun to transmit, in the third conjunct it is waiting to send the next "1" etc.

Lemma 5.2 *The following property holds for all reachable states of Impl :*

$$\begin{array}{ll}\vee & init(S)\\ \vee & \neg S.wire\_high \wedge S.list{\neq}\epsilon \wedge \neg S.transmitting \wedge S.x{=}0\\ \vee & \neg S.wire\_high \wedge S.list{\neq}\epsilon \wedge S.transmitting \wedge \mathsf{head}(S.list){=}1 \wedge S.x \leq 4\mathsf{Q}\\ \vee & \neg S.wire\_high \wedge S.list{\neq}\epsilon \wedge S.transmitting \wedge \mathsf{head}(S.list){=}0 \wedge S.x \leq 2\mathsf{Q}\\ \vee & S.wire\_high \wedge S.list{\neq}\epsilon \wedge \mathsf{head}(S.list){=}0 \wedge S.x \leq 4\mathsf{Q}\\ \vee & S.wire\_high \wedge (S.list{=}\epsilon \vee \mathsf{head}(S.list){=}1) \wedge S.x \leq 2\mathsf{Q}.\end{array}$$

The next invariant gives an upper bound for the clock of the receiver.

Lemma 5.3 *The following property holds for all reachable states of Impl :*

$$R.list{=}\epsilon \vee \mathsf{if}\ \mathsf{last}(R.list){=}0\ \mathsf{then}\ R.x \leq 7\mathsf{Q}\ \mathsf{else}\ R.x \leq 9\mathsf{Q}.$$

We now give invariants for relations between the states of the sender and the receiver. The next invariant tells us that for a good working of the implementation an input of a new message can only happen when the receiver is at rest.

Lemma 5.4 *The following property holds for all reachable states of Impl :*

$$\neg S.wire\_high \wedge S.list{\neq}\epsilon \wedge \neg S.transmitting \rightarrow R.list{=}\epsilon.$$

We want to reason about the two clocks in the implementation as if they were not drifting, but precise. For this reason we introduce a new symbol $\approx$ with an intended meaning of "almost equal". With this we abstract from the amount of drifting.

Notation 5.5 *Let e and f be expressions of type* **Real**. *We define:*

$$e \approx f \quad \triangleq \quad \frac{1-\mathsf{T}}{1+\mathsf{T}}e \leq f \leq \frac{1+\mathsf{T}}{1-\mathsf{T}}e.$$

Notice that if $\mathsf{T}{=}0$ we can read $\approx$ again as $=$. Also note that $e \approx f$ if and only if $f \approx e$.

Lemma 5.6 *Let s, s' be states of Impl and e, f terms of type* **Real** *in which no continuous variables occur. Suppose $s \models S.x + e \approx R.x + f$, and suppose that $s \xrightarrow{d} s'$, for some $d \in \mathsf{R}^+$. Then $s' \models S.x + e \approx R.x + f$.*

The most important observations about the implementation are those in which the distance between the clocks is related to the contents of the buffers of the sender and the receiver. We start with the possible distances and then give a more detailed description.

Lemma 5.7 *The following property holds for all reachable states of Impl :*

$$\begin{array}{lll} \wedge\ S.transmitting \wedge \neg S.wire\_high & \rightarrow & \vee\ R.x \approx S.x + 4\mathsf{Q} \\ & & \vee\ R.x \approx S.x + 2\mathsf{Q} \\ & & \vee\ R.x \approx S.x \wedge \mathsf{head}(S.list){=}1 \\ \wedge\ S.transmitting \wedge S.wire\_high & \rightarrow & \vee\ R.x \approx S.x \\ & & \vee\ R.x \approx S.x - 2\mathsf{Q} \wedge S.list{\neq}\epsilon \wedge \mathsf{head}(S.list){=}0. \end{array}$$

Lemma 5.8 *The following property holds for all reachable states of Impl :*

$$\begin{array}{lll} \wedge\ S.transmitting \wedge R.list{\neq}\epsilon & \rightarrow & \vee\ \mathsf{last}(R.list){=}1 \wedge R.x \leq \frac{1+\mathsf{T}}{1-\mathsf{T}}4\mathsf{Q} \wedge R.x \approx S.x \\ & & \vee\ \mathsf{last}(R.list){=}1 \wedge \frac{1-\mathsf{T}}{1+\mathsf{T}}4\mathsf{Q} \leq R.x \leq \frac{1+\mathsf{T}}{1-\mathsf{T}}8\mathsf{Q} \wedge R.x \approx S.x + 4\mathsf{Q} \\ & & \vee\ \mathsf{last}(R.list){=}0 \wedge R.x \leq \frac{1+\mathsf{T}}{1-\mathsf{T}}2\mathsf{Q} \wedge R.x \approx S.x - 2\mathsf{Q} \\ & & \vee\ \mathsf{last}(R.list){=}0 \wedge \frac{1-\mathsf{T}}{1+\mathsf{T}}2\mathsf{Q} \leq R.x \leq \frac{1+\mathsf{T}}{1-\mathsf{T}}6\mathsf{Q} \wedge R.x \approx S.x + 2\mathsf{Q}) \\ \wedge\ init(S) \wedge R.list{\neq}\epsilon & \rightarrow & \vee\ \mathsf{last}(R.list){=}1 \wedge R.x \leq 9\mathsf{Q} \wedge R.x \approx S.x \\ & & \vee\ \mathsf{last}(R.list){=}1 \wedge \frac{1-\mathsf{T}}{1+\mathsf{T}}4\mathsf{Q} \leq R.x \leq 9\mathsf{Q} \wedge R.x \approx S.x + 4\mathsf{Q} \\ & & \vee\ \mathsf{last}(R.list){=}0 \wedge \frac{1-\mathsf{T}}{1+\mathsf{T}}2\mathsf{Q} \leq R.x \leq 7\mathsf{Q} \wedge R.x \approx S.x + 2\mathsf{Q} \\ \wedge\ R.list{=}\epsilon & \rightarrow & \neg S.transmitting \wedge \neg S.wire\_high. \end{array}$$

The following invariant implies that, with our additional assumption that $\mathsf{T} < \frac{1}{17}$, the above defective scenario is not possible: an output of a message by the receiver cannot happen when the sender is still busy.

Lemma 5.9 *The following property holds for all reachable states of Impl :*

$$S.list{\neq}\epsilon \wedge ((R.list{\neq}\epsilon \wedge \mathsf{last}(R.list){=}0 \wedge R.x = 7\mathsf{Q}) \vee (\mathsf{last}(R.list){=}1 \wedge R.x = 9\mathsf{Q})) \rightarrow error.$$

The last invariant gives an obvious property of the specification automaton.

Lemma 5.10 *The following property holds for all reachable states of P:*

$$P.list{=}\epsilon \vee (\mathsf{head}(P.list) = 1 \wedge (\mathsf{odd}(P.list) \vee \mathsf{last\_two}(P.list) = \langle 00\rangle).$$

We have now collected enough invariants to establish a weak timed forward simulation from the implementation to the specification. Besides a part needed to deal with premature inputs, the simulation consists of two parts: a part relating the buffers of the sender and the receiver to the buffer of the specification and a part relating the clocks of the protocol to the single precise clock of the specification. As in most verifications of data link protocols it is essential to realize at what moment which part of the message is in transit between the sender and the receiver. In our case this comes down to establishing when there is a "0" in transit that is about to be accepted by the receiver.

Theorem 5.11 *The relation determined by the following formula over the state variables of Impl and P is a weak timed forward simulation from Impl to P:*

$$
\begin{array}{lll}
SIM & \triangleq & \textsf{if } error \textsf{ then } P.chaos \textsf{ else} \\
& & \textsf{if } R.list{=}\epsilon \textsf{ then } P.list{=}S.list \wedge (S.list{=}\epsilon \vee P.x{=}0) \textsf{ else} \\
& & \textsf{if } R.x \approx S.x + 2\mathsf{Q}(\mathsf{last}(R.list) - 1) \\
& & \quad \textsf{then} \;\; \wedge\ P.list{=}R.list\hat{\ }S.list \\
& & \qquad\qquad \wedge\ (1-\mathsf{T})P.x \leq 4\mathsf{Q}\mathsf{length}(R.list) - 2\mathsf{Q}(1 + \mathsf{last}(R.list)) \\
& & \qquad\qquad\qquad + \mathsf{min}(R.x, S.x + 2\mathsf{Q}(\mathsf{last}(R.list) - 1)) \\
& & \quad \textsf{else} \;\; \wedge\ P.list{=}R.list\hat{\ }\langle 0\rangle\hat{\ }S.list \\
& & \qquad\qquad \wedge\ (1-\mathsf{T})P.x \leq 4\mathsf{Q}\mathsf{length}(R.list) - 2\mathsf{Q}(1 + \mathsf{last}(R.list)) \\
& & \qquad\qquad\qquad + \mathsf{min}(R.x, S.x + 2\mathsf{Q}(\mathsf{last}(R.list) + 1)).
\end{array}
$$

6. Conclusions and Future Work

A first main conclusion that can be drawn from this work is that the "old recipes" from the I/O automata model carry over smoothly to the setting of linear hybrid systems. The size of the invariant and simulation proofs in this paper is substantial, but in our opinion justified by the complexity of the protocol and comparable to the proof size for similar untimed protocols. A next step will be to see whether more general types of hybrid systems can be handled in the context of the I/O automata model.

A second main conclusion, of course, is that the audio control protocol is correct, provided that the tolerance is less than 5.88% ($\frac{1}{17} = 5.882\cdots$). This value is larger than the maximum tolerance of ±5% that is allowed by Philips.

Communication protocols based on Manchester encoding are widely used in applications, for instance in the Ethernet [25]. It is therefore surprising that, as far as we know, there is almost no work on the rigorous analysis of the tolerance of asynchrony within this or related protocols. A notable exception is a recent paper by Moore [21], who mechanically verifies a biphase mark protocol. The protocol and model of Moore are slightly different from ours (for instance, clock jitter is ignored in the model) but despite these differences he surprisingly arrives at a maximal tolerance of 5%, which is very close to our result. Clearly, there are many interesting open questions left concerning this type of protocols. Because they live at the boundary between continuous physical phenomena (e.g., voltage on communication lines, and physical clocks) and discrete logical phenomena (e.g., microprocessors controlled by programs) their formal analysis is an ideal application area for the theory of hybrid systems.

We think that the audio control protocol that we have analyzed in this paper is a rather nice example that can play a role as a benchmark for other researchers to test their methods on, just as the *Cat and Mouse* example of [19] and the *Gas Burner* example of [24]. In particular it would be interesting to see whether automatic verification methods such as [4] can handle this protocol. Another challenge is to redo the verification of this paper within a process algebraic setting such as [26]. It is worthwhile to note that Wang Yi's [27] axiom of *time determinism*, which says that if time passes with an amount d the resulting state is uniquely determined, is not valid for linear hybrid systems: due to the timing uncertainty, time nondeterministic transition systems arise naturally in our setting. We think that a "natural" process algebraic description of the protocol requires an operator that adds timing uncertainty to a system.

The second author has checked most of the verification of this paper using the Coq proof development system [6]. We believe that computer support will become very important or even indispensable when dealing with larger protocols.

David Griffioen [8] has analyzed an extension of the protocol with multiple senders that has to deal with bus collisions. As a next step we also want to take the message delay on the bus into account. Our analysis has already helped to clarify some ambiguities in the original description of the protocol, and will hopefully lead to additional confidence in the protocol, orthogonal to that obtained via testing.

Acknowledgements
The authors thank Erik Kaashoek and Rudy Vanherck of N.V. Philips Industrial Activities in Leuven, Belgium, for providing the problem and answering our questions about the protocol. Thanks also to Ron Koymans from Philips Research in Eindhoven for his involvement in the project, to the referees for their comments, and to David Griffioen, Steven Klusener and Peter Polak for proofreading. The third author would like to thank the organizers of the AMAST International Workshop on Real-Time Systems in Iowa, November 1993, for giving him an excellent opportunity to present this work in an early stage.

References

1. M. Abadi and L. Lamport. An old-fashioned recipe for real time. In de Bakker et al. [5], pages 1–27.
2. R. Alur, C. Courcoubetis, T.A. Henzinger, and P.-H. Ho. Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems. In Grossman et al. [9], pages 209–229.
3. R. Alur and D.L. Dill. Automata for modeling real-time systems. In M. Paterson, editor, *Proceedings* 17th *ICALP,* Warwick, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer-Verlag, July 1990.
4. R. Alur, T.A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. In *Proceedings of the 14th Annual IEEE Real-time Systems Symposium*, 1993.
5. J.W. de Bakker, C. Huizing, W.P. de Roever, and G. Rozenberg, editors. *Proceedings REX Workshop on Real-Time: Theory in Practice,* Mook, The Netherlands, June 1991, volume 600 of *Lecture Notes in Computer Science.* Springer-Verlag, 1992.
6. G. Dowek, A. Felty, H. Herbelin, G.P. Huet, C. Murthy, C. Parent, C. Paulin-Mohring, and B. Werner. The Coq proof assistant user's guide. Version 5.8. Technical report, INRIA – Rocquencourt, May 1993.
7. R. Gawlick, R. Segala, J. Søgaard-Andersen, and N.A. Lynch. Liveness in timed and untimed systems. Technical Report MIT/LCS/TR-587, Laboratory for Computer Science, MIT, Cambridge, MA, December 1993. An extended abstract will appear in *Proceedings ICALP'94.*
8. W.O.D. Griffioen. *Analysis of an Audio Control Protocol with Bus Collision.* Master thesis, Programming Research Group, University of Amsterdam, 1994.
9. R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors. *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science.* Springer-Verlag, 1993.

10. L. Helmink, M.P.A. Sellink, and F.W. Vaandrager. Proof-checking a data link protocol. In H. Barendregt and T. Nipkow, editors, *Proceedings International Workshop TYPES'93,* Nijmegen, The Netherlands, May 1993, number 806 in Lecture Notes in Computer Science, pages 127–165. Springer-Verlag, 1994. Full version available as Report CS-R9420, CWI, Amsterdam, March 1994.

11. A.S.A. Jeffrey, S.A. Schneider, and F.W. Vaandrager. A comparison of additivity axioms in timed transition systems. Report CS-R9366, CWI, Amsterdam, November 1993.

12. L. Lamport. A temporal logic of actions. Research Report 79, Digital Equipment Corporation, Systems Research Center, December 1991.

13. L. Lamport. Hybrid systems in TLA$^+$. In Grossman et al. [9], pages 77–102.

14. B.W. Lampson, N.A. Lynch, and J.F. Søgaard-Andersen. Correctness of at-most-once message delivery protocols. In *FORTE'93 - Sixth International Conference on Formal Description Techniques,* Boston, MA, October 1993, pages 387–402, 1993.

15. N.A. Lynch, M. Merritt, W. Weihl, and A. Fekete. *Atomic Transactions.* Morgan Kaufmann Publishers, 1994.

16. N.A. Lynch and M.R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing*, pages 137–151, August 1987. A full version is available as MIT Technical Report MIT/LCS/TR-387.

17. N.A. Lynch and F.W. Vaandrager. Forward and backward simulations – part I: Untimed systems. Report CS-R9313, CWI, Amsterdam, March 1993. Also, MIT/LCS/TM-486, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA.

18. N.A. Lynch and F.W. Vaandrager. Forward and backward simulations – part II: Timing-based systems. Report CS-R9314, CWI, Amsterdam, March 1993. Also, MIT/LCS/TM-487.b, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA.

19. O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In de Bakker et al. [5], pages 447–484.

20. Z. Manna and A. Pnueli. Verifying hybrid systems. In Grossman et al. [9], pages 4–35.

21. J.S. Moore. A formal model of asynchronous communication and its use in mechanically verifying a biphase mark protocol. *Journal of Formal Aspects of Computing Science*, 6(1):60–91, 1994.

22. X. Nicollin and J. Sifakis. An overview and synthesis on timed process algebras. In K.G. Larsen and A. Skou, editors, *Proceedings of the 3rd International Workshop on Computer Aided Verification,* Aalborg, Denmark, volume 575 of *Lecture Notes in Computer Science*, pages 376–398. Springer-Verlag, 1992.

23. S. Owicki and D. Gries. An axiomatic proof technique for parallel programs. *Acta Informatica*, 6(4):319–340, 1976.

24. A.P. Ravn, H. Rischel, and K.M. Hansen. Specifying and verifying requirements of real-time systems. *IEEE Transactions on Software Engineering*, 19(1):41–55, January 1993.

25. R.S. Roden. *Digital Communication Systems Design.* Prentice Hall, 1988.

26. F.W. Vaandrager and N.A. Lynch. Action transducers and timed automata. In W.R. Cleaveland, editor, *Proceedings CONCUR 92,* Stony Brook, NY, USA, volume 630 of *Lecture Notes in Computer Science*, pages 436–455. Springer-Verlag, 1992.

27. Wang Yi. Real-time behaviour of asynchronous agents. In J.C.M. Baeten and J.W. Klop, editors, *Proceedings CONCUR 90,* Amsterdam, volume 458 of *Lecture Notes in Computer Science*, pages 502–520. Springer-Verlag, 1990.

1. Appendix: Proof of Theorem 3.1

Lemma 1.1 *Let P be a lineair hybrid system, ϕ a formula with variables in V_P, s, s' states of $aut(P)$, and ξ an $\mathcal{A}$-valuation with, for all $v \in V_P$, $s(v) = \xi(v)$ and $s'(v) = \xi(v')$. Then*

$$\mathcal{A}, \xi \models \mathit{Stable}(\phi) \quad \Leftrightarrow \quad (s \models \phi \ \Rightarrow \ s' \models \phi).$$

Proof: Easy. ■

Proof of Theorem 3.1 Let P be a lineair hybrid system, and let $A = aut(P)$. We must prove that $aut(P)$ is a timed I/O automaton, and for this it is enough to show that A satisfies axioms **A1**, **A2** and **A3**.

For axiom **A1**, let $s \in states(A)$ and $b \in in(A)$. Then there exist $a \in in(P)$ and an $\mathcal{A}$-valuation ξ such that $b = a[\![param(P,a)]\!]^{\xi}_{\mathcal{A}}$ and $\mathcal{A}, \xi \models pred(P,a)$. Since $a \in in(P)$, $pred(P,a)$ is of the form $\psi \wedge \bigwedge_{v \in V_P} v'{=}f_v$ where ψ does not contain variables from $V_P \cup V'_P$, and the f_v do not contain variables from V'_P.

Let ξ', ξ'' be the $\mathcal{A}$-valuations given by

$$\xi'(z) = \begin{cases} s(z) & \text{if } z \in V_P \\ \xi(z) & \text{otherwise} \end{cases} \qquad \xi''(z) = \begin{cases} [\![f_{z'}]\!]^{\xi'}_{\mathcal{A}} & \text{if } z \in V'_P \\ \xi'(z) & \text{otherwise} \end{cases}$$

and let s' be the state of A with, for $v \in V_P$, $s'(v) = \xi''(v')$. We claim that ξ'' is a witness for (s, b, s'). Proof:

- $\mathcal{A}, \xi'' \models pred(P,a)$. This follows from two observations
  1. $\mathcal{A}, \xi'' \models \psi$. This follows since $\mathcal{A}, \xi \models \psi$, ψ does not contain variables from $V_P \cup V'_P$, and ξ and ξ'' agree on variables outside $V_P \cup V'_P$.
  2. For $v \in V_P$, $\mathcal{A}, \xi'' \models v'{=}f_v$. This follows since by definition of ξ'', $[\![v']\!]^{\xi''}_{\mathcal{A}} = [\![f_v]\!]^{\xi'}_{\mathcal{A}}$, f_v does not contain variables from V'_P, and ξ' and ξ'' agree on variables outside V'_P.
- $\forall v \in V_P : s(v) = \xi''(v)$. This follows since, by definition of ξ', $s(v) = \xi'(v)$ for all $v \in V_P$, and ξ' and ξ'' agree on variables outside V'_P.
- $b = a[\![param(P,a)]\!]^{\xi''}_{\mathcal{A}}$. This follows since, $b = a[\![param(P,a)]\!]^{\xi}_{\mathcal{A}}$, the terms in $param(P,a)$ contain no variables from $V_P \cup V'_P$, and ξ and ξ'' agree on variables outside $V_P \cup V'_P$.
- $\forall v \in V_P : s'(v) = \xi''(v')$. Immediate from the definition of s'.

For axiom **A2**, suppose that $steps(A)$ contains transitions $s \xrightarrow{d} s'$ and $s' \xrightarrow{d'} s''$. Let ξ, ξ' be witnesses for transitions $s \xrightarrow{d} s'$ and $s' \xrightarrow{d'} s''$, respectively. We must prove that $s \xrightarrow{d+d'} s''$ is in $steps(A)$, and for this it is enough to find a witness for $s \xrightarrow{d+d'} s''$. Define $\mathcal{A}$-valuation ξ'' by

$$\xi''(z) = \begin{cases} s''(z) & \text{if } z \in V'_P \\ d + d' & \text{if } z = t \\ \xi(z) & \text{otherwise} \end{cases}$$

We claim that ξ'' is a witness for $s \overset{d+d'}{\rightarrow} s''$. By definition of ξ'' and because ξ is a witness for $s \overset{d}{\rightarrow} s'$, we have

- $\forall v \in V_P : s(v) = \xi''(v)$
- $[\![t]\!]^{\xi''}_{\mathcal{A}} = d + d'$
- $\forall v \in V_P : s''(v) = \xi''(v')$

It remains to prove that $\mathcal{A}, \xi'' \models \mathit{pred}(P, \mathit{TIME})$. We know that $\mathit{pred}(P, \mathit{TIME})$ is of the form

$$t > 0 \wedge \mathit{Unchanged}(D_P) \wedge \psi[\sigma] \wedge \bigwedge_{i=1}^{k} \mathit{Stable}(\psi_i)$$

where all ψ, ψ_i are in $\mathit{Convex}(D_P, C_P)$, and σ maps each variable $x \in C_P$ to $\frac{x'-x}{t}$. We prove that each of the conjuncts holds under ξ''.

- Since ξ is a witness for $s \overset{d}{\rightarrow} s'$, $\mathcal{A}, \xi \models t > 0$ and $[\![t]\!]^{\xi}_{\mathcal{A}} = d$, and thus $d > 0$. Similarly, it follows that $d' > 0$. Thus $\xi''(t) = d + d' > 0$, which implies $\mathcal{A}, \xi'' \models t > 0$.

- Since ξ, ξ' are witnesses for $s \overset{d}{\rightarrow} s'$ and $s' \overset{d'}{\rightarrow} s''$, respectively, we have, for each variable $v \in D_P$, $s'(v) = s(v)$ and $s''(v) = s'(v)$. Thus $s''(v) = s(v)$, for each $v \in D_P$, and hence, by definition of ξ'', $\mathcal{A}, \xi'' \models \mathit{Unchanged}(D_P)$.

- Let $\Gamma \in \mathit{Convex}(D_P, C_P)$. We claim that

  $$\mathcal{A}, \xi \models \Gamma[\sigma] \text{ and } \mathcal{A}, \xi' \models \Gamma[\sigma] \quad \Rightarrow \quad \mathcal{A}, \xi'' \models \Gamma[\sigma] \tag{1.1}$$

  The proof is by induction on the structure of Γ. We present here the case where Γ is of the form $\sum_i e_i x_i \Box e$. The other cases are easy and left to the reader.

  So suppose $\Gamma = \sum_i e_i x_i \Box e$, where $\Box \in \{<, \leq, \geq, >\}$, the e, e_i are terms of type **Real** with variables in D_P, and the x_i are variables in C_P. Assume $\mathcal{A}, \xi \models \Gamma[\sigma]$ and $\mathcal{A}, \xi' \models \Gamma[\sigma]$.

  $\mathcal{A}, \xi \models \Gamma[\sigma] \ \Rightarrow$

  $\mathcal{A}, \xi \models (\sum_i e_i x_i \Box e)[\sigma] \ \Rightarrow$ $\quad$ {definition σ, and e_i, e contain no variables from C_P}

  $\mathcal{A}, \xi \models \sum_i e_i \left(\frac{x_i' - x_i}{t}\right) \Box e \ \Rightarrow$ $\quad$ {definition $\models$}

  $\sum_i [\![e_i]\!]^{\xi}_{\mathcal{A}} \left(\frac{\xi(x_i') - \xi(x_i)}{\xi(t)}\right) \Box [\![e]\!]^{\xi}_{\mathcal{A}} \ \Rightarrow$ $\quad$ {ξ witness for $s \overset{d}{\rightarrow} s'$}

  $\sum_i [\![e_i]\!]^{\xi}_{\mathcal{A}} \left(\frac{s'(x_i) - s(x_i)}{d}\right) \Box [\![e]\!]^{\xi}_{\mathcal{A}} \ \Rightarrow$ $\quad$ {ξ and ξ'' agree on variables in D_P}

  $\sum_i [\![e_i]\!]^{\xi''}_{\mathcal{A}} \left(\frac{s'(x_i) - s(x_i)}{d}\right) \Box [\![e]\!]^{\xi''}_{\mathcal{A}}$

Similarly, we derive

$$\sum_i [\![e_i]\!]_{\mathcal{A}}^{\xi''} \left(\frac{s''(x_i) - s'(x_i)}{d'}\right) \Box [\![e]\!]_{\mathcal{A}}^{\xi''}$$

Therefore

$$\begin{aligned}
\sum_i [\![e_i]\!]_{\mathcal{A}}^{\xi''} \left(\frac{s''(x_i) - s(x_i)}{d + d'}\right) &= \\
&= \sum_i [\![e_i]\!]_{\mathcal{A}}^{\xi''} \left(\frac{s''(x_i) - s'(x_i) + s'(x_i) - s(x_i)}{d + d'}\right) \\
&= \sum_i [\![e_i]\!]_{\mathcal{A}}^{\xi''} \left[\left(\frac{s''(x_i) - s'(x_i)}{d'}\right)\left(\frac{d'}{d+d'}\right) + \left(\frac{s'(x_i) - s(x_i)}{d}\right)\left(\frac{d}{d+d'}\right)\right] \\
&= \left(\frac{d'}{d+d'}\right) \sum_i [\![e_i]\!]_{\mathcal{A}}^{\xi''} \left(\frac{s''(x_i) - s'(x_i)}{d'}\right) + \left(\frac{d}{d+d'}\right) \sum_i [\![e_i]\!]_{\mathcal{A}}^{\xi''} \left(\frac{s'(x_i) - s(x_i)}{d}\right) \\
&\Box \left(\frac{d'}{d+d'}\right) [\![e]\!]_{\mathcal{A}}^{\xi''} + \left(\frac{d}{d+d'}\right) [\![e]\!]_{\mathcal{A}}^{\xi''} \\
&= [\![e]\!]_{\mathcal{A}}^{\xi''}
\end{aligned}$$

By definition of ξ'', this is equivalent to

$$\sum_i [\![e_i]\!]_{\mathcal{A}}^{\xi''} \left(\frac{\xi''(x_i') - \xi''(x_i)}{\xi''(t)}\right) \Box [\![e]\!]_{\mathcal{A}}^{\xi''}$$

from which it follows that $\mathcal{A}, \xi'' \models \Gamma[\sigma]$. This completes the proof of the induction step. Since $\psi \in \mathit{Convex}(D_P, C_P)$, $\mathcal{A}, \xi \models \psi[\sigma]$ and $\mathcal{A}, \xi' \models \psi[\sigma]$, implication 1.1 gives $\mathcal{A}, \xi'' \models \psi[\sigma]$.

- Suppose $1 \leq i \leq k$. Since ξ, ξ' are witnesses for $s \xrightarrow{d} s'$ and $s' \xrightarrow{d'} s''$, respectively, $\mathcal{A}, \xi \models \mathit{Stable}(\psi_i)$ and $\mathcal{A}, \xi' \models \mathit{Stable}(\psi_i)$. Hence, by implication "$\Rightarrow$" of Lemma 1.1, $s \models \psi_i$ implies $s' \models \psi_i$, and $s' \models \psi_i$ implies $s'' \models \psi_i$. Thus $s \models \psi_i$ implies $s'' \models \psi_i$, and we can apply implication "$\Leftarrow$" of Lemma 1.1 to obtain $\mathcal{A}, \xi'' \models \mathit{Stable}(\psi_i)$. Since i was chosen arbitrarily, it follows in fact that $\mathcal{A}, \xi'' \models \bigwedge_{i=1}^{k} \mathit{Stable}(\psi_i)$.

For axiom **A3**, suppose that $\mathit{steps}(A)$ contains a transition $s \xrightarrow{d} s'$, for some $d \in \mathsf{R}^+$. Define the function $w : [0, d] \to \mathit{states}(A)$ by

$$w(c)(v) = \begin{cases} s(v) + c\left(\frac{s'(v) - s(v)}{d}\right) & \text{if } v \in C_P \\ s(v) & \text{if } v \in D_P \end{cases}$$

We must prove that w, which is just the straight line from s to s', is a $[0,d]$-trajectory from s to s'. Clearly, $w(0) = s$. Let ξ be a witness for $s \overset{d}{\rightarrow} s'$. Since $\mathcal{A}, \xi \models \mathit{Unchanged}(D_P)$, $s(v) = s'(v)$, for all $v \in D_P$, and it follows that $w(d) = s'$. Thus w spans from s to s'. Suppose $0 \leq c < c' \leq d$. We show that $w(c) \overset{c'-c}{\rightarrow} w(c')$. Define $\mathcal{A}$-valuation ξ' by

$$\xi'(z) = \begin{cases} w(c)(z) & \text{if } z \in V_P \\ w(c')(z') & \text{if } z \in V'_P \\ c' - c & \text{if } z = t \\ \xi(z) & \text{otherwise} \end{cases}$$

We claim that ξ' is a witness for $w(c) \overset{c'-c}{\rightarrow} w(c')$. The only nontrivial thing to prove here is that $\mathcal{A}, \xi' \models \mathit{pred}(P, \mathit{TIME})$. We prove that each of the conjuncts of ϕ holds under ξ'.

- $\mathcal{A}, \xi' \models t > 0$. By definition of ξ'.
- $\mathcal{A}, \xi' \models \mathit{Unchanged}(D_P)$. By definition of ξ' and w.
- Let $\Gamma \in \mathit{Convex}(D_P, C_P)$. We claim that

$$\mathcal{A}, \xi \models \Gamma[\sigma] \qquad \Rightarrow \qquad \mathcal{A}, \xi' \models \Gamma[\sigma] \tag{1.2}$$

The proof is by induction on the structure of Γ. We present here the case where Γ is of the form $\sum_i e_i x_i \Box e$. The other cases are easy and left to the reader.

So suppose $\Gamma = \sum_i e_i x_i \Box e$, where $\Box \in \{<, \leq, \geq, >\}$, the e, e_i are terms of type **Real** with variables in D_P, and the x_i are variables in C_P. Assume $\mathcal{A}, \xi \models \Gamma[\sigma]$. We derive

$$\begin{aligned}
&[\![\sum_i e_i \left(\frac{x'_i - x_i}{t}\right)]\!]^{\xi'}_{\mathcal{A}} = \\
&= \sum_i [\![e_i]\!]^{\xi'}_{\mathcal{A}} \left(\frac{\xi'(x'_i) - \xi'(x_i)}{\xi'(t)}\right) \\
&= \sum_i [\![e_i]\!]^{\xi}_{\mathcal{A}} \left(\frac{w(c')(x_i) - w(c)(x_i)}{c' - c}\right) \\
&= \sum_i [\![e_i]\!]^{\xi}_{\mathcal{A}} \left(\frac{\left[s(x_i) + c'\left(\frac{s'(x_i) - s(x_i)}{d}\right)\right] - \left[s(x_i) + c\left(\frac{s'(x_i) - s(x_i)}{d}\right)\right]}{c' - c}\right) \\
&= \sum_i [\![e_i]\!]^{\xi}_{\mathcal{A}} \left(\frac{s'(x_i) - s(x_i)}{d}\right) \\
&= \sum_i [\![e_i]\!]^{\xi}_{\mathcal{A}} \left(\frac{\xi(x'_i) - \xi(x_i)}{\xi(t)}\right)
\end{aligned}$$

$$
\begin{aligned}
&= [\![\sum_i e_i \left(\frac{x_i' - x_i}{t}\right)]\!]^{\xi}_{\mathcal{A}} \\
&\square\ [\![e]\!]^{\xi}_{\mathcal{A}} \quad \text{(here we used that } \mathcal{A}, \xi \models \Gamma[\sigma]) \\
&= [\![e]\!]^{\xi'}_{\mathcal{A}}
\end{aligned}
$$

It follows that $\mathcal{A}, \xi' \models \Gamma[\sigma]$. This completes the proof of the induction step.

Since $\psi \in \mathit{Convex}(D_P, C_P)$ and $\mathcal{A}, \xi \models \psi[\sigma]$, implication 1.2 gives $\mathcal{A}, \xi' \models \psi[\sigma]$.

- Let $\Delta \in \mathit{Halves}(D_P, C_P)$. We claim that

$$
\mathcal{A}, \xi \models \mathit{Stable}(\Delta) \quad \Rightarrow \quad \mathcal{A}, \xi' \models \mathit{Stable}(\Delta) \tag{1.3}
$$

The proof is by induction on the structure of Δ. We present here the case where Δ is of the form $\sum_i e_i x_i \square e$. The other cases are easy and left to the reader.

So suppose $\Delta = \sum_i e_i x_i \square e$, where $\square \in \{<, \leq, \geq, >\}$, the e, e_i are terms of type **Real** with variables in D_P, and the x_i are variables in C_P. Assume $\mathcal{A}, \xi \models \mathit{Stable}(\Delta)$. We distinguish between two cases:

1. Assume $\mathcal{A}, \xi \models \Delta$. Then $\mathcal{A}, \xi \models \Delta \wedge \Delta'$. We derive

$$
[\![\sum_i e_i x_i]\!]^{\xi'}_{\mathcal{A}} = \sum_i [\![e_i]\!]^{\xi'}_{\mathcal{A}} \xi'(x_i) = \sum_i [\![e_i]\!]^{\xi}_{\mathcal{A}} w(c)(x_i) =
$$

$$
\begin{aligned}
&= \sum_i [\![e_i]\!]^{\xi}_{\mathcal{A}} \left(s(x_i) + c\left(\frac{s'(x_i) - s(x_i)}{d}\right)\right) \\
&= \left(\frac{c}{d}\right)\left(\sum_i [\![e_i]\!]^{\xi}_{\mathcal{A}} s'(x_i)\right) + \left(\frac{d-c}{d}\right)\left(\sum_i [\![e_i]\!]^{\xi}_{\mathcal{A}} s(x_i)\right) \\
&= \left(\frac{c}{d}\right) [\![\sum_i e_i x_i']\!]^{\xi}_{\mathcal{A}} + \left(\frac{d-c}{d}\right) [\![\sum_i e_i x_i]\!]^{\xi}_{\mathcal{A}} \\
&\square\ \left(\frac{c}{d}\right) [\![e]\!]^{\xi}_{\mathcal{A}} + \left(\frac{d-c}{d}\right) [\![e]\!]^{\xi}_{\mathcal{A}} \quad \text{(here we used that } \mathcal{A}, \xi \models \Delta \wedge \Delta') \\
&= [\![e]\!]^{\xi}_{\mathcal{A}} = [\![e]\!]^{\xi'}_{\mathcal{A}}
\end{aligned}
$$

   It follows that $\mathcal{A}, \xi' \models \Delta$. Completely analogously we can derive $\mathcal{A}, \xi' \models \Delta'$. Hence $\mathcal{A}, \xi' \models \mathit{Stable}(\Delta)$.

2. Assume $\mathcal{A}, \xi \models \neg\Delta$.
   Assume $\mathcal{A}, \xi' \models \neg\Delta'$. Let

$$
\bigcirc = \begin{cases} \geq & \text{if } \square = < \\ > & \text{if } \square = \leq \\ \leq & \text{if } \square = > \\ < & \text{if } \square = \geq \end{cases}
$$

   We derive

$$
\begin{aligned}
[\![\sum_i e_i x_i]\!]_{\mathcal{A}}^{\xi'} = \cdots = & \\
= \quad & \sum_i [\![e_i]\!]_{\mathcal{A}}^{\xi} \left(s(x_i) + c \left(\frac{s'(x_i) - s(x_i)}{d} \right) \right) \\
= \quad & \sum_i [\![e_i]\!]_{\mathcal{A}}^{\xi} \left(\left(\frac{c' - c}{c'} \right) s(x_i) + \left(\frac{c}{c'} \right) s(x_i) + \left(\frac{c}{c'} \right) c' \left(\frac{s'(x_i) - s(x_i)}{d} \right) \right) \\
= \quad & \left(\frac{c' - c}{c'} \right) \sum_i [\![e_i]\!]_{\mathcal{A}}^{\xi} s(x_i) + \left(\frac{c}{c'} \right) \sum_i [\![e_i]\!]_{\mathcal{A}}^{\xi} \left(s(x_i) + c' \left(\frac{s'(x_i) - s(x_i)}{d} \right) \right) \\
= \quad & \left(\frac{c' - c}{c'} \right) \sum_i [\![e_i x_i]\!]_{\mathcal{A}}^{\xi} + \left(\frac{c}{c'} \right) \sum_i [\![e_i x_i']\!]_{\mathcal{A}}^{\xi'} \\
\bigcirc \quad & \left(\frac{c' - c}{c'} \right) [\![e]\!]_{\mathcal{A}}^{\xi} + \left(\frac{c}{c'} \right) [\![e]\!]_{\mathcal{A}}^{\xi'} \quad \text{(here we used } \mathcal{A}, \xi \models \neg\Delta \text{ and } \mathcal{A}, \xi' \models \neg\Delta') \\
= \quad & [\![e]\!]_{\mathcal{A}}^{\xi'}
\end{aligned}
$$

This means that $\mathcal{A}, \xi' \models \neg\Delta$ and hence $\mathcal{A}, \xi' \models \mathit{Stable}(\Delta)$.

Thus $\mathcal{A}, \xi \models \mathit{Stable}(\Delta) \Rightarrow \mathcal{A}, \xi' \models \mathit{Stable}(\Delta)$, and we have completed the proof of the induction step.

Since, for all i, $\psi_i \in \mathit{Halves}(D_P, C_P)$ and $\mathcal{A}, \xi \models \mathit{Stable}(\psi_i)$, implication 1.3 gives $\mathcal{A}, \xi' \models \bigwedge_{i=1}^{n} \mathit{Stable}(\psi_i)$.