From Continuous Specification to Discrete Design

François Siewe[*]and Dang Van Hung[†]
The United Nations University
International Institute for Software Technology
P.O.Box 3058, Macau
{fs,dvh}@iist.unu.edu

November 25, 2002

Abstract

The paper presents a syntactical approach to designing real-time distributed systems that can handle both continuous time and discrete time models in a uniform logical framework. We approximate continuous state variables by discrete ones and formalise the relationship between them. The requirements of a real-time system are specified as a formula over continuous state variables. Then we derive a discrete design of a digital controller as formula over discrete state variables that makes the system to satisfy the continuous specification under some assumptions about the behaviour of the environment and the relationship between continuous state variables and discrete state variables. We provide rules useful for refining and verifying the correctness of a design syntactically.

Keywords: Continuous specification, discrete design, real-time distributed systems, verification.

1 Introduction

A real-time system can be thought of as a plant in permanent interaction with its environment. However, the behaviour of the plant should satisfy specific critical-time constraints to face disturbances from the environment. Very often, a slight failure in the reaction of the plant can cause harm or damage. In this respect formal methods are needed for specifying and reasoning about the design of real-time systems.

The states of the environment and those of the plant can change at any time according to the laws of physics. It appears then that the continuous time model (real numbers) is suitable for specifying their continuous behaviour. However the plant must be controlled to make the system satisfy its real-time requirements.

The state of a digital program changes only at discrete time points at ticks of a computer clock. The discrete time model (natural numbers) should then be considered for the implementation of the system. A question raises as how the two models of time can be combined into the same formalism such that the design and its correctness can accurately be reasoned about in an uniform manner.

This paper presents a syntactical approach for designing real-time systems that can handle both continuous time and discrete time models in an uniform logical framework. Such an approach is suitable for both refinement and verification of the correctness of a design w.r.t. the specification using proof assistant tools.

Duration Calculus (DC for short), introduced in 1991 by Zhou, Hoare and Ravn [3], is a logic for reasoning about *state durations* which is based on Interval Temporal Logic. The model of time is the set of real numbers. A *state* is modelled as a boolean valued function over time, having at most finite number of discontinuity points in any finite interval (finite variability). So, the duration of a state over a time-interval is the integral of the state over the time-interval. DC is powerful enough for specifying and reasoning about the behaviour of real-time systems in continuous time model. It has been used successfully in many case studies as specification language.

Duration Calculus with Iteration (DC^* for short), a logic obtained by extending DC with the iteration operator (*), has been defined in [8] to play the role of interface between DC and *Timed Automata* model [1]. A sound proof system for DC^* has been proposed. The completeness of the proof system is ensured for a restricted class of DC^* formulae called *Simple* DC^* *(*SDC^* *for short) formulae*. A simple DC^* formula denotes exactly a *Timed Regular Expression* (see [8]) and therefore a timed automaton [2]. Moreover, if only rational numbers are al-

[*]On leave from the Dept. of Maths. and Computer Science, University of Dschang, Cameroon, from May to December 1999.

[†]On leave fron the Institute of Information Technology, Nghia Do, Cau Giay, Hanoi, Vietnam.

lowed, SDC^* is decidable, viz we can check if a design is implementable or not.

We use DC as specification language and SDC^* as the language of the detailed design. Our model of real-time hybrid system is a distributed system depicted as in Figure 1. We use *sensors* to sample continuous states and *actuators* to control the plant. The *control program* gets at discrete time points the states of the environment via the sensors, computes a command and imposes it to the plant via the actuators. We assume that the computation time is negligible. As there is no global clock in distributed systems, the concept of discreteness is a local property in our model. That is a state variable might be considered as discrete in one component of the system and as continuous in another as the two components do not have the same clock. Therefore, for each process, we define a subset $dVar$ of the set of state variables $Svar$ which includes the state variables considered to be discrete by the process. The process can access only its discrete state variables. Let $cVar$ denotes the set of states which are considered to be continuous by the process. Then we consider the relationship between continuous states and discrete ones, defining the approximation of them. Our goal is to derive a 'discrete' specification of the control program from the specification of the continuous behaviour of the plant and the environment. The design is correct if the 'discrete' specification implies the specification of the continuous behaviour of the plant under the assumption of the behaviour of the environment and the relationship between continuous states and discrete states.

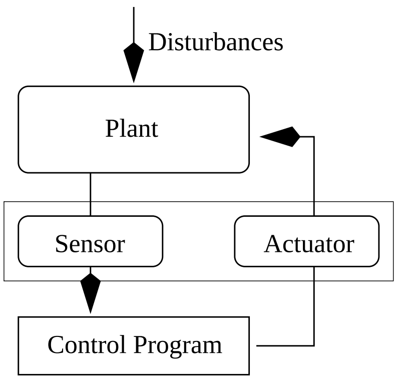

Figure 1: Structure of controlled systems

The design process can be formalised as follows. For a specification S of the requirement of a real-time system, at the first step a design decision must be taken as how the requirement of the real-time system is to be met and formalised as a DC^* formula D over $cSvar$ such that $D \Rightarrow S$. Then we find a SDC^* formula I over $dSvar$ such that $A \vdash I \Rightarrow D$ where A stands for the assumption of the behaviour of the environment and the relationship between continuous state and discrete state variables. The discrete SDC^* formula I is the specification of the control program. We provide rules for deriving I and verifying the correctness of an implementation syntactically.

To illustrate our discretization technique, let us consider a classical simple example, Gas Burner, taken from [7]. The critical real-time requirement of the gas burner is that the proportion of time spent in the leak state is not more than one twentieth of the elapsed time, whenever the burner is observed for at least one minute. This requirement can be formalised as a DC formula $S_{gb} \mathrel{\widehat{=}} \Box(\ell \geq 60 \Rightarrow 20 * \int leak \leq \ell)$. We consider the design decision that any occurrence of leak should not last for a period longer than one second, and the distance between two consecutive occurrences of leaks must be at least thirty seconds long. We formalise the design decision as a DC^* formula $D_{gb} \mathrel{\widehat{=}} \Box(\llbracket leak \rrbracket \Rightarrow \ell \leq 1) \wedge \Box(\llbracket leak \rrbracket \frown \llbracket \neg leak \rrbracket \frown \llbracket leak \rrbracket \Rightarrow \ell \geq 30)$. It is proved in [3] that $D_{gb} \Rightarrow S_{gb}$. Let us consider a discrete state variable $leak'$ used by the control program to observe the presence or absence of leak via the sensor. Let us assume that a presence of leak can be observed within δ time units ($\delta < 1$), which could follow the fact that leak is stable for more than δ time units, and the sampling step is at most δ. This implies that $leak'$ approximate $leak$ with the tolerance δ. Let $recover$ be a discrete state which can stop leak within $1 - 2\delta$ time units and keeps the Burner in $nonleak$ state as long as it is true. Then a discrete specification of the control program is defined as

$$I'_{gb} \mathrel{\widehat{=}} \Box(\llbracket leak' \rrbracket \Rightarrow \ell \leq 1 - \delta) \wedge \Box\left(\begin{array}{r} \llbracket leak' \rrbracket \frown \llbracket \neg leak' \rrbracket \frown \llbracket leak' \rrbracket \Rightarrow \\ \ell > 30 + \delta \end{array} \right)$$

I'_{gb} in turn is refined by $I_{gb} \mathrel{\widehat{=}} \mathit{PREF}(\llbracket \neg leak' \rrbracket \frown (\llbracket leak' \wedge recover \rrbracket \wedge 1 - 2\delta \leq \ell \leq 1 - \delta) \frown (\llbracket recover \rrbracket \wedge \ell = 31 + \delta) \frown (\llbracket \neg leak' \rrbracket \vee \ell = 0)^*)$, where for any DC formula F, $\mathit{PREF}(F)$ denote the formula satisfied by all the prefix of any time interval satisfying F. We prove that $I_{gb} \Rightarrow D_{gb}$ under above assumptions about the relationship between $leak$ and $leak'$, $recover$ and $nonleak$, and the assumption about the stability of $leak$. Note that only 'discrete' state variables can occur in I_{gb}, and I_{gb} can be translated to a timed automaton in the obvious way.

The remaining of the paper is organised as follows. In Section 2 we give a brief summary of DC^*. The discretization technique is presented in Section 3. We give some refinement and verification rules in Section 4. we prove the correctness of the design of the gas burner in section 5. In Section 6 we formalise and verify the correctness of the audio control protocol using our discretization technique. Section 7 concludes the paper.

2 Duration Calculus with Iteration

In this section we give a brief summary of DC^*. The readers are referred to [8] for more details on the calculus.

A language for DC* is built starting from the following sets of *symbols*: a set of *constant symbols* $\{a, b, c, \ldots\}$, a set of *individual variables* $\{x, y, z, \ldots\}$, a set of *state variables* $\{P, Q, \ldots\}$, a set of *temporal variables* $\{u, v, \ldots\}$, a set of *function symbols* $\{f, g, \ldots\}$, a set of *relation symbols* $\{R, U, \ldots\}$, and a set of *temporal propositional letters* $\{A, B, \ldots\}$.

A DC* language definition is essentially that of the sets of *state expressions* S, *terms* t and *formulae* φ of the language. These sets can be defined by the following BNFs:

$$\begin{array}{lcl} S & \widehat{=} & \mathbf{0} \mid P \mid \neg S \mid S \vee S \\ t & \widehat{=} & c \mid x \mid u \mid \int S \mid f(t, \ldots, t) \\ \varphi & \widehat{=} & A \mid R(t, \ldots, t) \mid \neg\varphi \mid (\varphi \vee \varphi) \mid (\varphi ^\frown \varphi) \mid \\ & & (\varphi^*) \mid \exists x \varphi \end{array}$$

A state variable P is interpreted as a function $I(P) : I\!R^+ \to \{0, 1\}$ (a state). $I(P)(t) = 1$ means that state P is present at time t, and $I(P)(t) = 0$ means that P is not present at time t. We assume that a state has finite variability in any finite time interval. A state expression is interpreted as a function which is defined by the interpretations for the state variables and boolean operators.

For an arbitrary state expression S, its duration is denoted by $\int S$. Given an interpretation I of the state variables and an interval, duration $\int S$ is interpreted as the accumulated length of time within the interval at which S is present. So for any interval $[t, t']$, the interpretation $I(\int S)([t, t'])$ is defined as $\int_t^{t'} I(S)(t)dt$.

A formula φ is satisfied by an interpretation in an interval $[t, t']$ when it evaluates to true for that interpretation over that time interval. This is written as $I, [t, t'] \models \varphi$.

Given an interpretation I, a formula $\varphi ^\frown \phi$ is true for $[t, t'']$ if there exists a t' such that $t \le t' \le t''$ and φ and ϕ are true for $[t, t']$ and $[t', t'']$ respectively.

We consider the following abbreviations: $\ell \mathrel{\widehat{=}} \int 1$, $\llbracket S \rrbracket \mathrel{\widehat{=}} (\int P = \ell) \wedge (\ell > 0)$, $\Diamond\varphi \mathrel{\widehat{=}} true ^\frown \varphi ^\frown true$, and $\Box\varphi \mathrel{\widehat{=}} \neg\Diamond\neg\varphi$.

The proof system for DC* consists of a complete Hilbert-style proof system for first order logic (cf. e.g. [10]), axioms and rules for interval logic (cf. e.g. [5]), Duration Calculus axioms and rules ([6]) and axioms about iteration ([8]). We only recall here some axioms and rules of the proof system of DC^*.

$$\begin{array}{ll} (DC1) & \int \mathbf{0} = 0 \\ (DC2) & \int \mathbf{1} = \ell \\ (DC3) & \int S \ge 0 \\ (DC4) & \int S_1 + \int S_2 = \int (S_1 \vee S_2) + \int (S_1 \wedge S_2) \\ (DC5) & (\int S = x ^\frown \int S = y) \Rightarrow \int S = x + y \\ (DC6) & \int S_1 = \int S_2 \text{ if } S_1 \Leftrightarrow S_2 \text{ in} \\ & \text{propositional calculus.} \end{array}$$

$$(IR_1) \quad \frac{\begin{array}{c}[\ell = 0/A]\varphi \quad \varphi \Rightarrow [A ^\frown \llbracket S \rrbracket/A]\varphi \\ \varphi \Rightarrow [A ^\frown \llbracket \neg S \rrbracket/A]\varphi\end{array}}{[true/A]\varphi}$$

$$(IR_2) \quad \frac{\begin{array}{c}[\ell = 0/A]\varphi \quad \varphi \Rightarrow [\llbracket S \rrbracket ^\frown A/A]\varphi \\ \varphi \Rightarrow [\llbracket \neg S \rrbracket ^\frown A/A]\varphi\end{array}}{[true/A]\varphi}$$

$$(\omega) \quad \frac{\forall k < \omega \; [(\llbracket S \rrbracket \vee \llbracket \neg S \rrbracket)^k/A]\varphi}{[true/A]\varphi}$$

$$\begin{array}{ll} (DC_1^*) & \ell = 0 \Rightarrow \varphi^* \\ (DC_2^*) & (\varphi^* {}^\frown \varphi) \Rightarrow \varphi^* \\ (DC_3^*) & (\varphi^* \wedge \psi ^\frown true) \Rightarrow (\psi \wedge \ell = 0 ^\frown true) \vee \\ & \quad (((\varphi^* \wedge \neg\psi ^\frown \varphi) \wedge \psi) ^\frown true). \end{array}$$

The proof system of DC^* is complete for sentences where iteration is allowed only for a restricted class of formulae called *simple*.

Definition 1 *A Simple DC^* formula is defined inductively by:*

$$\begin{array}{lcl} \varphi & \widehat{=} & \ell = 0 \mid \llbracket S \rrbracket \mid a \le \ell \mid \ell \le a \mid (\varphi \vee \varphi) \mid \\ & & (\varphi \wedge \varphi) \mid (\varphi ^\frown \varphi) \mid \varphi^* \end{array}$$

For simplifying the presentation in the next section, we introduce the notion *PREF*(D) for any simple formula D. Intuitively, *PREF*(D) is a simple formula that holds for all prefixes of an interval that validates D.

Definition 2 *Given a simple DC* formula φ, we define a simple DC* formula PREF(φ) as follows.*

1. *PREF*$(\llbracket S \rrbracket) \mathrel{\widehat{=}} \llbracket S \rrbracket^*$
2. *PREF*$(a \le \ell) \mathrel{\widehat{=}} \ell \ge 0$
3. *PREF*$(\ell \le a) \mathrel{\widehat{=}} \ell \le a$
4. *PREF*$(\varphi \vee \varphi') \mathrel{\widehat{=}}$ *PREF*$(\varphi) \vee$ *PREF*(φ')
5. *PREF*$(\varphi \wedge \varphi') \mathrel{\widehat{=}}$ *PREF*$(\varphi) \wedge$ *PREF*(φ')
6. *PREF*$(\varphi ^\frown \varphi') \mathrel{\widehat{=}}$ *PREF*$(\varphi) ^\frown$*PREF*(φ')
7. *PREF*$(\varphi^*) \mathrel{\widehat{=}} \varphi^* {}^\frown$*PREF*$(\varphi)$

It follows immediately from the definition that

Proposition 1 $\varphi \Rightarrow \neg(\neg$*PREF*$(\varphi) ^\frown true)$.

The class of simple DC^* formulae plays an important role in our design technique. The following section presents our discretization technique.

3 Discrete Interface

In this section we defined three concepts for formalising the relationship between continuous state variables and discrete ones.

Definition 3 (Stability) *Given a state variable s and a positive real number δ, we say s is δ-*stable *iff the following formula is satisfied by any interval*

$$\delta\text{-}stable(s) \mathrel{\hat{=}} \Box(\llbracket\neg s\rrbracket^\frown\llbracket s\rrbracket^\frown\llbracket\neg s\rrbracket \Rightarrow \llbracket\neg s\rrbracket^\frown(\llbracket s\rrbracket \wedge \ell > \delta)^\frown\llbracket\neg s\rrbracket$$

The stability means that a state should not change quickly. This property is required for the states of the environment to be observable.

Definition 4 (Control state) *Given two state variables r and s, and a non-negative real number δ, we say r δ-*controls *s iff the following formula is satisfied by any interval*

$$r \rhd_\delta s \mathrel{\hat{=}} \Box(\llbracket r\rrbracket \wedge \ell > \delta \Rightarrow (\ell \le \delta)^\frown\llbracket s\rrbracket)$$

This formula asserts that whenever r lasts for a period longer than δ time units, s must hold within δ time units and last until at least r changes. The relation ship between $recover$ and $nonleak$ in the example in the introduction of the paper is expressed by $recover \rhd_\delta nonleak$. The concept of control state is used for formalising the behaviour of actuators.

Finally we define the concept of *observation* as depicted in Figure 2, for formalising the behaviour of sensors.

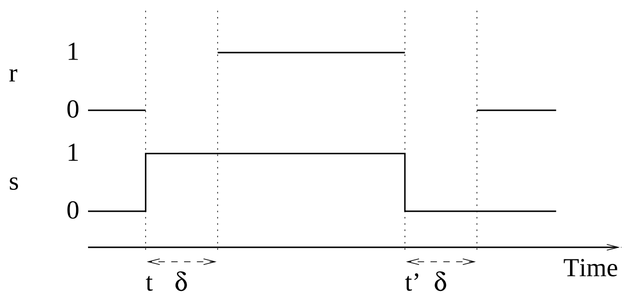

Figure 2: The state r δ-observes the state s

Definition 5 (Observation state) *Given two state variables r and s, and a non-negative real number δ, we say r δ-*observes *s iff the following formula is satisfied by any interval*

$$r \overset{\rightarrow}{\approx}_\delta s \mathrel{\hat{=}} (s \rhd_\delta r) \wedge (\neg s \rhd_\delta \neg r)$$

This formula says that any change in s, stable for at least δ time units, occurs in r within at most δ time units, and r keeps the new value until at least s changes again, i.e. state r approximate state s with the tolerance δ if s and $\neg s$ are δ-stable. In figure 2 the behaviour of r is undetermined but should be unchanged in the intervals $[t, t+\delta]$ and $[t', t'+\delta]$, i.e when s changes. Obviously, $r\overset{\rightarrow}{\approx}_\delta s$ iff $\neg r\overset{\rightarrow}{\approx}_\delta \neg s$. Note that the definition say nothing about unstable change of s.

We will assume that continuous state variables are stable enough, otherwise there is no way to observe them in discrete time.

For formalising the discrete interface, for any continuous state variable s, we consider a discrete state variable s' used by the control program to observe s via the sensors. The relationship between s and its sampling s' is formalised by $s'\overset{\rightarrow}{\approx}_\delta s$ for some non-negative real number δ. For instance, in the example in the introduction of the paper the states $leak'$ is a sampling state of $leak$.

Similarly, for any state t, say $nonleak$, of the plant we consider a command t'', a discrete state, say $recover$, for requesting (via the actuators) the plant getting into state t. The relationship between t and t'' is formalised by $t'' \rhd_\tau t$ for some non-negative real number τ.

4 Refinement and Verification Rules

In this section we give some rules useful for both the refinement and the verification. For a rule, the formula above the line is a refinement of the formula under the line, while the formula under the tine is the consequence of the formula above the line. The proofs of the rules and more details are given in [4].

Transitivity rules

Rule 1 $$\frac{(r \rhd_\delta s) \quad (s \rhd_\tau t)}{r \rhd_{(\delta+\tau)} t}$$

Rule 2 $$\frac{(r\overset{\rightarrow}{\approx}_\delta s) \quad (s\overset{\rightarrow}{\approx}_\tau t)}{r\overset{\rightarrow}{\approx}_{(\delta+\tau)} t}$$

These rules say that the accuracy is deteriorated through sequential samplings of a state. They are helpful for the design of distributed systems comprising many sensors, as well as how to use the sensors efficiently.

Observation rules

Rule 3 $$\frac{r\overset{\rightarrow}{\approx}_\delta s}{\begin{array}{c}(\llbracket s\rrbracket \wedge \ell > \delta)^\frown(\llbracket\neg s\rrbracket \wedge \ell > \delta) \Rightarrow \\ \ell \le \delta^\frown\llbracket r\rrbracket^\frown\llbracket\neg r\rrbracket^\frown true\end{array}}$$

Rule 3 allows to capture the change of state (edge) from 0 to 1 or from 1 to 0 by observation.

State Distance

Rule 4a $$\frac{r\overset{\rightarrow}{\approx}_\delta s \quad \delta - stable(s)}{(\delta+\tau) - stable(r) \Rightarrow \tau - stable(s)}$$

Rule 4b $$\frac{r \gtrapprox_\delta s \quad \delta - stable(r)}{(\delta+\tau) - stable(s) \Rightarrow \tau - stable(r)}$$

This rule defines a necessary condition for the stability of a continuous state, which is the stability of its sampling and vice-versa. Namely, the observation of a stable state is also a stable state. The smaller the sampling step is, the less different the continuous state and its sampling. It is useful for refinement.

State Occurrence

Rule 5 $$\frac{r \gtrapprox_\delta s}{\Box(\llbracket r \rrbracket \Rightarrow \ell \le \tau) \Rightarrow \Box(\llbracket s \rrbracket \Rightarrow \ell \le \delta + \tau)}$$

This rule is helpful for both refinement and verification. For example, if we can prove that $I_{gb} \Rightarrow \Box(\llbracket Leak' \rrbracket \Rightarrow \ell \le 1 - \delta)$ then we can deduce by this rule that $I_{gb} \Rightarrow \Box(\llbracket Leak \rrbracket \Rightarrow \ell \le 1)$. This rule also defines how fast the control program should be to satisfy a time constraint about the occurrence of state.

Duration of state

Rule 6

$$\begin{aligned}&\mathit{PREF}(\llbracket r \rrbracket^* {}^\frown (\llbracket \neg r \rrbracket {}^\frown (\llbracket r \rrbracket \wedge \ell > \delta))^* {}^\frown \llbracket \neg r \rrbracket)\\ &\quad \Rightarrow \delta - \text{stable}(r)\end{aligned}$$

Rule 7

$$\begin{aligned}&\mathit{PREF}(\llbracket \neg r \rrbracket^* {}^\frown (\llbracket r \rrbracket \wedge \ell \le \delta) {}^\frown \llbracket \neg r \rrbracket)^* {}^\frown\\ &\quad (\llbracket r \rrbracket \wedge \ell \le \delta))\\ &\qquad \Rightarrow \Box(\llbracket r \rrbracket \Rightarrow \ell \le \delta)\end{aligned}$$

Rule 6 is useful for verifying that a SDC^* formula satisfies a time constraint about the distance of state. For example, we can use this rule to prove that $I_{gb} \Rightarrow (30 + \delta) -$stable$(\neg Leak')$.

Rule 7 is useful for verifying that a SDC^* formula satisfies a time constraint about the occurrence of state. For example, we can use this rule to prove that $I_{gb} \Rightarrow \Box(\llbracket Leak' \rrbracket \Rightarrow \ell \le 1 - \delta)$.

The following Rules are useful for proving safety properties.

Invariant Rule for loop

Rule 8 $$\frac{\begin{array}{c}\varphi \Rightarrow \neg(true^\frown \neg\alpha) \;\; \ell = 0 \Rightarrow \alpha\\ \varphi \Rightarrow \neg(\neg\beta^\frown true) \;\; \ell = 0 \Rightarrow \beta\\ \alpha^\frown \varphi^{*\frown} \beta \Rightarrow \chi\end{array}}{\varphi^* \Rightarrow \Box\chi}$$

Invariant Rule for sequential concatenation

Rule 9 $$\frac{\begin{array}{c}\psi \Rightarrow \Box\chi \;\; \varphi \Rightarrow \Box\chi \;\; \alpha^\frown\beta \Rightarrow \chi\\ \psi \Rightarrow \neg(\neg\beta^\frown true) \;\; \varphi \Rightarrow \neg(true^\frown \neg\alpha)\end{array}}{\varphi^\frown\psi \Rightarrow \Box\chi}$$

Trivial parallel composition

Rule 10 $$\frac{A \Rightarrow \Box\psi \quad B \Rightarrow \Box\varphi}{A \wedge B \Rightarrow \Box(\psi \wedge \varphi)}$$

5 An example

We consider our example in the introduction, Gas burner, and formalise the assumptions about the behaviour of the sensor and the stability of $Leak$ as

$$\begin{aligned} A_{gb} \quad \widehat{=} \quad & (Leak' \gtrapprox_\delta Leak) \wedge 2\delta\text{-stable}(Leak) \wedge \\ & \delta < 1/2 \wedge recover \rhd_{(1-2\delta)} \neg Leak \wedge \\ & \delta\text{-stable}(Leak') \wedge \delta\text{-stable}(\neg Leak')\end{aligned}$$

where $Leak'$ is the sampling of $Leak$ via the sensor. So it is reasonable that $Leak'$ should be stable for one sampling step. Here we prove that the implementation I_{gb} satisfies the design decision D_{gb} under assumption A_{gb}, provided $\delta < 1$.

Theorem 1 $(\delta < 1) \wedge A_{gb} \vdash I_{gb} \Rightarrow D_{gb}$

Proof:

1 $\delta < 1$
{assumption}

2 A_{gb}
{assumption}

3 $I_{gb} \Rightarrow$
$\mathit{PREF}(\llbracket \neg leak' \rrbracket^\frown$
$((\llbracket leak' \rrbracket \wedge 1 - 2\delta \le \ell \le 1 - \delta) \wedge$
$(\ell \le 1 - 2\delta^\frown \llbracket \neg Leak \rrbracket))^\frown$
$(\llbracket \neg leak \rrbracket \wedge \ell = 31 + \delta)^\frown$
$(\llbracket \neg leak' \rrbracket \vee \ell = 0)^*)$
{DC*}

4 $I_{gb} \Rightarrow$
$\mathit{PREF}(\llbracket \neg leak' \rrbracket^\frown$
$(\llbracket leak' \rrbracket \wedge 1 - 2\delta \le \ell \le 1 - \delta)^\frown$
$(\llbracket \neg leak' \rrbracket \wedge \ell \ge 30 + \delta)^\frown$
$(\llbracket \neg leak' \rrbracket \vee \ell = 0)^*)$
{DC*, A_{gb}}

5 $I_{gb} \Rightarrow \Box(\llbracket Leak' \rrbracket \Rightarrow \ell \le 1 - \delta)$
{by 4,Rule 7, DC^*, PC}

6 $\Box(\llbracket Leak' \rrbracket \Rightarrow \ell \le 1 - \delta) \Rightarrow \Box(\llbracket Leak \rrbracket \Rightarrow \ell \le 1)$
{by 1, 2, Rule 5}

7 $I_{gb} \Rightarrow \Box(\llbracket Leak \rrbracket \Rightarrow \ell \le 1)$
{by 5, 6, PC}

8 $I_{gb} \Rightarrow D_{gb}$
{by 2, 7, PC, arithmetic}

6 Audio Control Protocol

In this section, we use our approach to specify and verify the correctness of the audio control protocol. The protocol uses the well-known Manchester encoding of bit strings. Each bit is encoded as a cell consisting of $4Q$ cycles of the receiver clock. Bit 1 is encoded as an upgoing edge at the middle of the cell, and bit 0 is encoded as a downgoing edge at the middle of the cell. 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 cells (see Figure 4).

The receiver scans only upgoing edges using the Philips decoding. It has to decode the signal without seeing the downgoing edges. This is always possible except that a message ending with '10' cannot be distinguished from the same message ending with '1'. The problem is solved by requiring that all messages either end with '00' or have an odd length. In addition, all messages should begin with '1' for the receiver to detect the beginning of the messages.

Let s be a state variable modelling the signal sent by the sender and r a state variable modelling the signal received by the receiver. We consider the following assumptions about the physical laws and the digitisation. If the sender changes the signal from low to high or from high to low, the receiver's signal is unreliable for δ cycles of the sender clock, $\delta \geq 1$, $\delta \in I\!N$ during which it can be any value chosen nondeterministically among 0 and 1. However if the sender keeps the signal stable then the same signal will be stable for the receiver. The relationship between the signal sent s and the signal received r is formalised by

$$r \overset{\rightarrow}{\approx}_{\delta} s$$

and is illustrated in Figure 3.

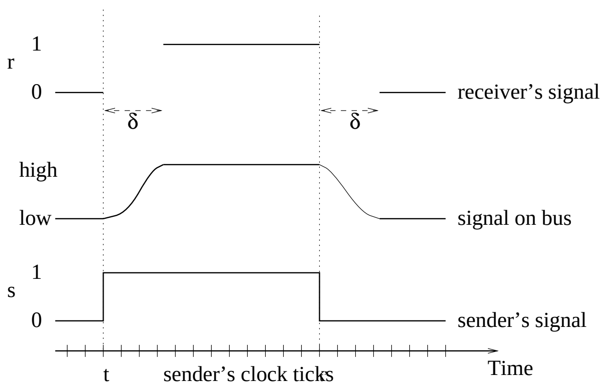

Figure 3: A model of communication at physical layer

Since a state can be specified by a DC formula, a communication protocol can be modelled as consisting of a language $\mathcal{L}_s$ whose elements are DC formulae over state variable s, a language $\mathcal{L}_r$ whose elements are DC formulae over state variable r, a coding function $f : \{0,1\}^+ \rightarrow \mathcal{L}_s$ and a decoding function $g : \mathcal{L}_r \rightarrow \{0,1\}^+$. The protocol is correct if and only if for all $w \in dom(f)$ there exists one and only one formula $D \in \mathcal{L}_r$ such that $f(w) \Rightarrow D$ and $g(D) = w$.

Now, we consider how the audio control protocol is formalised in this way. For sake of simplicity we consider the following abbreviations, where a and b are positive integers and h a positive real number:
$Hi\_Lo_s \mathrel{\widehat{=}} (\llbracket s \rrbracket \wedge \ell = a)^\frown(\llbracket \neg s \rrbracket \wedge \ell = b)$ representing an downgoing edge of the sender's signal,
$Lo\_Hi_s \mathrel{\widehat{=}} (\llbracket \neg s \rrbracket \wedge \ell = a)^\frown(\llbracket s \rrbracket \wedge \ell = b)$ representing a upgoing edge of the sender's signal and similarly for the receiver's signal $Hi\_Lo_r \mathrel{\widehat{=}} (\llbracket r \rrbracket \wedge \ell > \delta)^\frown(\llbracket \neg r \rrbracket \wedge \ell = h)$ and $Lo\_Hi_r \mathrel{\widehat{=}} (\llbracket \neg r \rrbracket \wedge \ell > \delta)^\frown(\llbracket r \rrbracket \wedge \ell = h)$.

Manchester encoding

$f : \{0,1\}^+ \rightarrow \mathcal{L}_s$ in this case is a partial function only, $dom(f) \mathrel{\widehat{=}} 1(00 \oplus 01 \oplus 10 \oplus 11)^* \oplus 1(0 \oplus 1)^*00$ and $f(w) \mathrel{\widehat{=}} \llbracket \neg s \rrbracket^\frown f'(w)^\frown(\llbracket \neg s \rrbracket \wedge \ell \geq c)$ where $c \geq 1$ is a natural number and f' a homomorphism defined by

$$\begin{array}{lcl} f'(1) & \widehat{=} & Lo\_Hi_s \\ f'(0) & \widehat{=} & Hi\_Lo_s \\ f'(wx) & \widehat{=} & f'(w)^\frown f'(x), \text{ for } x = 0, 1. \end{array}$$

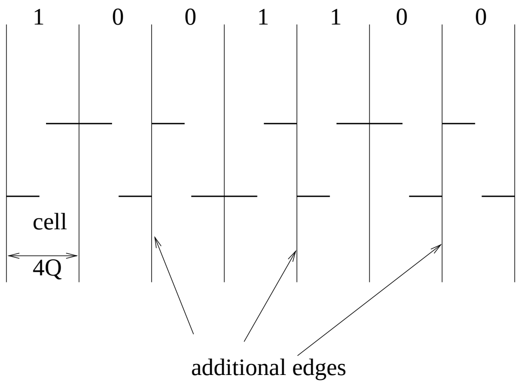

Figure 4: Manchester encoding

Philips decoding

We consider the following abbreviation:

$$\begin{array}{lcl} FourH & \widehat{=} & ((\neg\Diamond Lo\_Hi_r)^\frown Lo\_Hi_r)\wedge \\ & & 3Q \leq \ell < 5Q, \\ SixH & \widehat{=} & ((\neg\Diamond Lo\_Hi_r)^\frown Lo\_Hi_r)\wedge \\ & & 5Q \leq \ell < 7Q, \\ EightH & \widehat{=} & ((\neg\Diamond Lo\_Hi_r)^\frown Lo\_Hi_r) \wedge 7Q \leq \ell, \\ NineH & \widehat{=} & (\neg\Diamond Lo\_Hi_r) \wedge 9Q \leq \ell. \end{array}$$

Let $\mathcal{L}_r \,\widehat{=}\, (\neg\Diamond Lo\_Hi_r)^\frown Lo\_Hi_r(^\frown(FourH \oplus SixH \oplus EightH))^{*\frown} NineH$. Then, the decoding function $g : \mathcal{L}_r \to \{0,1\}^+$ is defined by $g(D) \,\widehat{=}\, Just(g''(D))$ where the functions $Just$ and g'' are defined as follows.

$Just : \{0,1\}^* \to \{0,1\}^*$
if $w = x1$ and $|w|$ is odd the $Just(w) = w$,
if $w = x1$ and $|w|$ is even the $Just(w) = w0$,
if $w = x0$ and $|w|$ is odd the $Just(w) = w$,
if $w = x0$ and $|w|$ is even then
if $x = y0$ then $Just(w) = w$,
if $x = y1$ then $Just(w) = w0$.

$g''(D^\frown NineH) \,\widehat{=}\, g'(D)$
where g' is a mapping from $(\neg\Diamond Lo\_Hi_r)^\frown Lo\_Hi_r(^\frown(FourH \oplus SixH \oplus EightH))^*$ to $\{0,1\}^+$ defined by

$$\begin{array}{ll} g'((\neg\Diamond Lo\_Hi_r)^\frown Lo\_Hi_r) \,\widehat{=}\, 1. & \\ g'(D^\frown FourH) \,\widehat{=}\, w00 & \text{if } g'(D) = w0, \\ g'(D^\frown SixH) \,\widehat{=}\, w01 & \text{if } g'(D) = w0, \\ g'(D^\frown FourH) \,\widehat{=}\, w11 & \text{if } g'(D) = w1, \\ g'(D^\frown SixH) \,\widehat{=}\, w100 & \text{if } g'(D) = w1, \\ g'(D^\frown EightH) \,\widehat{=}\, w101 & \text{if } g'(D) = w1, \end{array}$$

For example, suppose that the sender wants to send the sequence of bits 10010. The sequence is encoded as a DC formula $f(10010)$,

$$\begin{aligned} f(10010) = \;& [\![\neg s]\!]^\frown Lo\_Hi_s{}^\frown Hi\_Lo_s{}^\frown Hi\_Lo_s{}^\frown \\ & Lo\_Hi_s{}^\frown Hi\_Lo_s{}^\frown([\![\neg s]\!] \wedge \ell \geq c). \end{aligned}$$

If we can prove that $f(10010) \Rightarrow D$, where $D \,\widehat{=}\, (\neg\Diamond Lo\_Hi_r)^\frown Lo\_Hi_r{}^\frown SixH^\frown SixH^\frown NineH$, then the same sequence 10010 will be received by the receiver, since $g(D) = 10010$.

We now verify the correctness of the protocol. The value c is not part of the protocol but only represents the gap between messages. It should be chosen consequently by the engineers to reduce the transmission time of a sequence of messages. The parameter h is a real number according to the sender's clock. We only need $h = 1$ according to the receiver's clock so that the receiver can detect an edge.

Verification

We have to find the values of a, b, δ, and prove that for those a, b, δ

1. For all $D, D' \in \mathcal{L}_r, D \neq D'$ implies that $D \wedge D' \Leftrightarrow false$, i.e at any time the receiver can detect at most one string in its domain.

2. For all $w \in dom(f)$, there is a formula $D \in \mathcal{L}_r$ such that $f(w) \Rightarrow D$ and $g(D) = w$, i.e for any input string in the domain of the sender the receiver can detect at least one string in its domain.

The proof of the correctness is given in [4] using the rules given in the previous section. Because of the space limit, it is omitted here.

Reasoning about clock rates of the sender and the reader

Our design technique is more useful as it can allow to reason about the parameters of the system. We have shown that the protocol is correct provided $a > 2\delta$, $Q > \delta$ and $7Q + \delta \leq 4a \leq 9Q - \delta$. We have considered the sender's clock as reference time. Therefore the numbers a, δ and Q are defined according to the sender's clock. However Q should be given according to the receiver's clock as it is used to decode the received signal. Let us denote by ϵ the ratio of the sender's clock rate and receiver's clock rate. If Q' is the value of Q according to the receiver's clock, then $Q = Q'\epsilon$ according to the sender's clock. Given a, δ and Q', we can determine the ratio of the rates of the sender's and receiver's clock for which the protocol is correct. However if

$$\frac{4a + \delta}{9Q'} \leq \epsilon \leq \frac{4a - \delta}{7Q'}$$

then the inequalities $7Q + \delta \leq 4a \leq 9Q - \delta$ are satisfied and the protocol is correct provided $a > 2\delta$ and $Q > \delta$.

For example let $a = 7$, $\delta = 3$ and $Q' = 4$. Then if the ratio of the rates of the sender's and receiver's clock is between $31/36$ and $25/28$ then the protocol is still correct.

7 Conclusion

We have proposed a syntactical approach for designing real-time systems that can handle continuous time and discrete time models in an uniform logical framework. The main idea of our approach is to model the discretization at the state level by introducing the discrete states approximating the continuous ones, and then derive a specification of the program over discrete states.

We have used our discretization technique to specify and verify the correctness of the audio control protocol. The protocol has been studied in [9] using a digitised extension of DC. A special formula int has been introduced to formalise the digitization of signal using the receiver's clock as reference time. As a result, the relationship between the signal of the sender and the signal of the receiver has been made weaker.

In this approach we do not consider any extension of the original DC. We only consider an approximation of continuous states by discrete ones in a natural way, formalising the causality relationship between continuous states variable and their samplings (see Section 3). We do consider as reference time the sender's clock, instead. So we can determine more precisely in term of sender's clock cycle the period in which the signal is unreliable. We then verify the correctness of the protocol easily using our verification rules of Section 4.

References

[1] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[2] Eugene Asarin, Paul Caspi, and Oded Maler. A kleene theorem for timed automata. In G. Winskel, editor, *IEEE International Symposium on Logics in Computer Science LICS'97*, pages 160–171, 1997.

[3] Zhou Chaochen, C.A.R. Hoare, and Anders P. Ravn. A calculus of duration. *Information Processing Letters*, 40(5):269–276, 1991.

[4] François Siewe and Dang Van Hung. From continuous specification to discrete design. Technical Report 182, UNU/IIST, P.O. Box 3058, Macau, December 1999.

[5] B. Dutertre. On First Order Interval Temporal Logic . Technical Report CSD-TR94-3, Department of Computer science, Royal holloway, university of London, Egham, Surrey TW20 0EX, England, 1995.

[6] M. R. Hansen and Zhou Chaochen. Semantics and Completeness of Duration Calculus. In *Real-Time: Theory and Practice*, LNCS 600, pages 209–225. Springer-Verlag, 1992.

[7] Michael R. Hansen and Zhou Chaochen. Duration calculus: Logical foundations. *Formal Aspects of Computing*, 9:283–330, 1997.

[8] Dang Van Hung and Dimitar P. Guelev. Completeness of a fragment of Duration Calculus with Iteration. In *Proceedings of Asian Computing Science Conference (ASIAN'99)*. Phuket, Thailand, 1999.

[9] Dang Van Hung and Ko Kwang Il. Verification via Digitazed Models of Real-Time Hybrid Systems. In *Proceedings of Asia-Pacific Software Engineering Conference (APSEC'96)*, pages 4–15. IEEE Computer Society Press, 1996.

[10] J. Shoenfield. *Mathematical logic*. Addison-Wesley, Reading, Massachusetts, 1967.