Electronic Communications of the EASST
Volume 46 (2011)

Proceedings of the 11th International Workshop on Automated Verification of Critical Systems

Approximating Idealised Real-Time Specifications Using Time Bands

Brijesh Dongol and Ian J. Hayes

16 pages

Guest Editors: Jens Bendisposto, Cliff Jones, Michael Leuschel, Alexander Romanovsky
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Approximating Idealised Real-Time Specifications Using Time Bands

Brijesh Dongol[1] **and Ian J. Hayes** [2*]

[1]brijesh@itee.uq.edu.au [2]Ian.Hayes@itee.uq.edu.au
School of Information Technology and Electrical Engineering,
The University of Queensland

Abstract: Timed specifications are often formalised at an absolute level of precision, which does not reflect the real world that the specifications model, i.e., in the real world, inputs cannot be sampled with absolute precision and physical hardware cannot react instantaneously. As a result the developed specifications can often become unimplementable. In this paper, we consider the time bands model which allows time to be structured into several layers of abstraction and relationships between bands to be formalised. This allows the timed specifications developed under idealised assumptions to be approximated using the time band in which the variables are sampled. We implement the approximated specifications using teleo-reactive programs embedded with time bands.

Keywords: Time bands, Real-time systems, Teleo-reactive programs, Refinement, Cyber-physical systems

1 Introduction

There is an increasing prevalence of *cyber-physical* systems, where software controllers are used to control physical hardware. Formally reasoning about cyber-physical systems is complicated because one must inherently consider real-time properties, parallelism between an agent and its environment and hardware/software interactions. Furthermore, components tend to operate over multiple time granularities (e.g., days, milliseconds) and hence, the reasoning rules must be able to incorporate each of these within a single formalism.

A more difficult task is the development of correct real-time implementations via stepwise refinement. Real-time logics tend to produce specifications that are too precise about their timing requirements. As a result, the assumptions and models that are developed often become unimplementable due to the mismatch between the idealised assumptions of the specification and imprecision of digital clocks, delays in processing and imprecision of physical hardware that are inevitable in the real world. However, development of implementations from idealised assumptions remains an attractive option because it allows one to decouple development so that systems are first developed under idealised timing constraints (which allows one to focus on component interactions). The idealised system can be modified at a later stage of the development so that it takes real-world timing constraints into account.

* This research is supported by Australian Research Council Discovery Grant DP0987452.

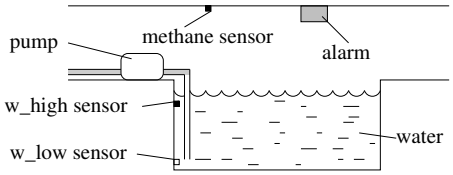

Figure 1: Mine pump example

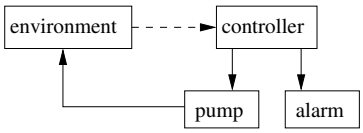

Figure 2: Controller diagram

Modifying idealised systems to handle real-world timing inaccuracies seldom produce refinements [WDMR08]. Thus, in this paper, we propose a method of *approximating* an idealised specification. We use the *time bands* framework [BH10], which includes a logic for reasoning about sampling errors (see Section 4.1) and allows specifications to be associated with notions such as precision and accuracy (see Section 4.2). In particular, we propose a method where we a develop a program for an idealised specification. Then, we produce an approximation of this specification using time bands and consider the modifications that are necessary to the program so that it implements this approximated specification.

Throughout this paper, we consider the safety-critical system in Fig. 1 in which a pump is used to remove water from a mine shaft [BL91]. To prevent an explosion, the pump must not be operating if there is a critically high level of methane in the mine. As depicted in Fig. 2, the controller must sample sensor values from the environment (dashed arrow) and send signals to the pump and alarm to turn them on or off. In turn, the pump may change the water level, which affects the state of the environment.

We implement the controllers using teleo-reactive programming model, which is a high-level approach to developing real-time systems [Nil01, Hay08]. The teleo-reactive paradigm is radically different from frameworks such as action systems, timed automata and TLA$^+$ because actions are considered to be durative (as opposed to instantaneous). Teleo-reactive programs are particularly useful for implementing controllers for autonomous agents that must react robustly to their dynamically changing environments.

2 A real-time framework

We reason about a teleo-reactive program by reasoning over the contiguous time intervals within which it executes. We model time using the real numbers $\mathbb{R}$ and define:

$$\mathit{Interval} \mathrel{\widehat=} \left\{ \Delta \subseteq \mathbb{R} \;\middle|\; \Delta \neq \{\} \wedge \forall t, t' : \Delta \bullet \forall t'' : \mathbb{R} \bullet t < t'' < t' \Rightarrow t'' \in \Delta \right\}$$

Thus, if t and t' are elements of an interval Δ, then all real numbers between t and t' are also in Δ. Note that an interval may be open or closed at either end. We use $glb.S$ and $lub.S$ to refer to the *greatest lower* and *least upper bounds* of a set of numbers S, respectively, where '.' represents function application. For intervals Δ, Δ', we define the *length* of Δ (denoted $\ell.\Delta$) and Δ *adjoins* Δ' (denoted $\Delta \propto \Delta'$) as follows:

$$\begin{aligned} \ell.\Delta &\mathrel{\widehat=} lub.\Delta - glb.\Delta \\ \Delta \propto \Delta' &\mathrel{\widehat=} (lub.\Delta = glb.\Delta') \wedge (\Delta \cup \Delta' \in \mathit{Interval}) \wedge (\Delta \cap \Delta' = \{\}) \end{aligned}$$

The *partitions* of an interval $\Delta \in \mathit{Interval}$ is given by $\Pi.\Delta$, which is defined as follows:

$$\Pi.\Delta \;\widehat{=}\; \{z{:}\,\mathrm{seq}.\mathit{Interval} \mid (\Delta = \bigcup \mathrm{ran}.z) \wedge (\forall i{:}\,\mathrm{dom}.z - \{0\} \bullet z.(i-1) \propto z.i)\}$$

Given that variable names are taken from the set $V \subseteq \mathit{Var}$, a *state space* is given by $\Sigma_V \mathrel{\widehat{=}} V \to \mathit{Val}$, which is a total function mapping each variable in V to a value from Val. We leave out the subscript when the set V is clear from the context. A *state* is a member of Σ_V and a *predicate* over a type X is given by $\mathscr{P}X \mathrel{\widehat{=}} X \to \mathbb{B}$ (e.g., a *state predicate* is a member of $\mathscr{P}\Sigma_V$). The (real-time) trace of environment behaviours is given by $\mathit{Stream}_V \mathrel{\widehat{=}} \mathbb{R} \to \Sigma_V$, which is a total function from time to states representing the evolution of the state of the system over time. To define properties of programs running over a time interval Δ, we use an *interval stream predicate*, which has type $\mathit{IntvPred}_V \mathrel{\widehat{=}} \mathit{Interval} \to \mathscr{P}\mathit{Stream}_V$.

We use $\lim_{x \to a^+} f.x$ and $\lim_{x \to a^-} f.x$ to denote the limit of $f.x$ from the right and left, respectively. To ensure that limits are well defined, we assume all variables are piecewise continuous [GM01]. For a state predicate c, interval Δ and stream s, we define:

$$\overrightarrow{c}.\Delta.s \mathrel{\widehat{=}} \begin{cases} c.(s.(\mathit{lub}.\Delta)) & \text{if } \mathit{lub}.\Delta \in \Delta \\ \lim\limits_{t \to \mathit{lub}.\Delta^-} c.(s.t) & \text{otherwise} \end{cases} \qquad \overleftarrow{c}.\Delta.s \mathrel{\widehat{=}} \begin{cases} c.(s.(\mathit{glb}.\Delta)) & \text{if } \mathit{glb}.\Delta \in \Delta \\ \lim\limits_{t \to \mathit{glb}.\Delta^+} c.(s.t) & \text{otherwise} \end{cases}$$

Thus, $\overrightarrow{c}.\Delta.s$ and $\overleftarrow{c}.\Delta.s$ state that c holds in s at the right and left ends of Δ, respectively. We define the *always* and *sometime* operators as follows.

$$\begin{aligned} (\circledast c).\Delta.s &\;\widehat{=}\; \forall t{:}\,\Delta \bullet c.(s.t) \\ (\odot c).\Delta.s &\;\widehat{=}\; \exists t{:}\,\Delta \bullet c.(s.t) \end{aligned}$$

Thus, $(\circledast c).\Delta.s$ and $(\odot c).\Delta.s$ hold iff c holds in s for all times and some time with Δ, respectively.

Example 1 (Safety requirement for the mine pump.) *A required safety condition of our mine pump example is that in any state of the real-time stream (i.e., at the absolute level of precision), if the methane level, m, is above the critical level, C, then the pump must be stopped. Note that this means that the pump must have physically come to a stop, which we distinguish from the output control signal that turns the pump off. The safety condition Safety is defined as follows:*

$$\mathit{Safety} \;\widehat{=}\; \circledast(m \geq C \Rightarrow \mathit{stopped}) \tag{1}$$

For an interval predicate p and interval Δ, we say p holds in an immediately *previous* interval if $(\mathbf{prev}\,p).\Delta$ holds, which is defined as follows:

$$(\mathbf{prev}\,p).\Delta \;\widehat{=}\; \exists \Delta'{:}\,\mathit{Interval} \bullet \Delta' \propto \Delta \wedge p.\Delta'$$

We use $\circledast$ and **prev** to define *invariance* of a state predicate c and *stability* of a set of variables V as follows:

$$\begin{aligned} \mathbf{inv}\,c &\;\widehat{=}\; (\mathbf{prev}\,\overrightarrow{c} \Rightarrow \circledast c) \\ \mathbf{st}\,V &\;\widehat{=}\; \forall v{:}\,V \bullet \forall k{:}\,\mathit{Val} \bullet \mathbf{inv}(v = k) \end{aligned}$$

Hence, $(\mathbf{inv}\, c).\Delta$ holds iff $\circledast c.\Delta$ holds provided that c holds at the right ends of an interval that precedes Δ (i.e., c is *invariant* within Δ) and $\mathbf{st}\, V$ holds iff the value of each $v \in V$ does not change within Δ (i.e., V is *stable* in Δ). Note that defining $\mathbf{st}\, V$ as $\forall v{:}\, V \bullet \exists k{:}\, Val \bullet \mathbf{inv}(v = k)$ is incorrect because $\exists k{:}\, Val \bullet \mathbf{inv}(v = k)$ can be trivially satisfied by choosing a k such that $\mathbf{prev}(\overrightarrow{v \neq k})$.

We define the following operators to facilitate reasoning about interval predicates. The *chop* operator ';' allows one to split a given interval into two parts, where p_1 holds for the first interval and p_2 holds for the second [ZH04]. The *weak chop* ':' states that either p_1 holds or the chop $p_1 ; p_2$ holds over the given interval.

$$\begin{array}{rcl}(p_1 ; p_2).\Delta & \widehat{=} & \exists \Delta_1, \Delta_2{:}\, Interval \bullet (\Delta = \Delta_1 \cup \Delta_2) \wedge (\Delta_1 \propto \Delta_2) \wedge p_1.\Delta_1 \wedge p_2.\Delta_2 \\ (p_1 : p_2).\Delta & \widehat{=} & p_1.\Delta \vee (p_1 ; p_2).\Delta \end{array}$$

Note that the stream is implicit in both sides of the definitions above. Furthermore, unlike the duration calculus [ZH04], we do not require that the intervals Δ_1 and Δ_2 in the definition of chop are closed.

We assume point-wise lifting of the boolean operators on stream and interval predicates in the normal manner, e.g., if p_1 and p_2 are interval predicates, Δ is an interval and s is a stream, we have $(p_1 \wedge p_2).\Delta.s = (p_1.\Delta.s \wedge p_2.\Delta.s)$. Furthermore, when reasoning about properties of programs, we would like to state that whenever a property p_1 holds over any interval Δ and stream s, a property p_2 also holds over Δ and s. Hence, we define universal implication over intervals and streams as follows. Operators '$\equiv$' and '$\Leftarrow$' are similar.

$$p_1 \Rrightarrow p_2 \mathrel{\widehat{=}} \forall \Delta{:}\, Interval \bullet p_1.\Delta \Rrightarrow p_2.\Delta \qquad p_1.\Delta \Rrightarrow p_2.\Delta \mathrel{\widehat{=}} \forall s{:}\, Stream \bullet p_1.\Delta.s \Rightarrow p_2.\Delta.s$$

3 Idealised teleo-reactive programs

We introduce teleo-reactive programs by considering a program that monitors the methane and controls the mine-pump in Fig. 1.

Example 2 (Teleo-reactive controller for the mine-pump.) *The teleo-reactive program in Fig. 3 implements a controller for the system in Fig. 2. The main program (*mine_pump*) consists of a sequence of two guarded actions. For such a sequence, the first alternative that has a true guard is executed continuously while that guard remains true and no earlier guard in the sequence becomes true. As soon as that guard becomes false or an earlier guard becomes true, the execution switches to the first true guard. For example, if program* mine_pump *is executing* pump_water*, it continues to do so while the methane level, m, is continuously less than the critical level, C, i.e., $m < C$. In doing so,* pump_water *may switch back and forth between its two alternatives, depending on the water level. However, as soon as the methane level reaches its critical level, the execution of* pump_water *is immediately terminated and control is passed to the first alternative of* mine_pump*. That is, within a hierarchical teleo-reactive program, the guards of the top-level program take precedence over an activity within a lower-level program.*

Note that program pump_water *refers to an output variable stopped that is fed back as input $stopped_0$. In this paper, we focus on the interaction between the* mine_pump *and* pump_water *programs, and hence elide discussion of feedback.*

$$\mathsf{mine\_pump} \;\widehat{=}\; \left\langle \begin{array}{rcl} m \geq C & \rightarrow & \mathsf{alarm} \| \mathsf{stop\_pump}\ , \\ \mathit{true} & \rightarrow & \mathsf{pump\_water} \end{array} \right\rangle$$

$$\mathsf{pump\_water} \;\widehat{=}\; \left\langle \begin{array}{rcl} \mathit{w\_high} \vee (\neg \mathit{w\_low} \wedge \neg \mathit{stopped}_0) & \rightarrow & \mathsf{run\_pump}\ , \\ \mathit{true} & \rightarrow & \mathsf{stop\_pump} \end{array} \right\rangle$$

Figure 3: Teleo-reactive controller for the mine pump

The first branch of mine_pump *executes within any interval in which* $m \geq C$ *continuously holds. Execution of* alarm||stop_pump *consists of the parallel execution of primitive actions* alarm *and* stop_pump. *Note that the parallel actions execute in a truly concurrent manner (as opposed to via an interleaving semantics).*

Definition 1 For a state predicate c, set of variables F and interval predicates r and p, the syntax of a *teleo-reactive program* is given by P where:

$$\begin{array}{rcl} P & ::= & F{:}[\![p]\!] \mid \mathsf{seq}.GP \mid P \overrightarrow{\|} P \mid \mathsf{rely}\, r \bullet P \mid \mathsf{init}\, c \bullet P \\ GP & ::= & c \rightarrow P \end{array}$$

Here $F{:}[\![p]\!]$ denotes a *primitive action* with outputs F that behaves as described by the interval predicate p. A *guarded program* $c \rightarrow P$ consists of a state predicate c that guards a teleo-reactive program. A program may either be a primitive action, a sequence of guarded programs, the parallel composition of two programs, a program with a rely condition r or a program with an initialisation c. We follow the convention of using Z for a teleo-reactive program and S for a sequence of guarded programs. Sequences are written within brackets '$\langle$' and '$\rangle$' and '$\frown$' is used for sequence concatenation. We let $\mathit{vars}.p$ and $\mathit{vars}.c$ denote the set of all variables that occur free in interval predicate p and state predicate c, respectively. The set of variables of Z is given by $\mathit{vars}.\mathsf{Z}$, where

$$\begin{array}{rcl} \mathit{vars}.(F{:}[\![p]\!]) & \widehat{=} & \mathit{vars}.p \cup F \\ \mathit{vars}.\langle\rangle & \widehat{=} & \{\} \\ \mathit{vars}.(\langle c \rightarrow \mathsf{Z}\rangle \frown S) & \widehat{=} & \mathit{vars}.c \cup \mathit{vars}.\mathsf{Z} \cup \mathit{vars}.S \end{array} \qquad \begin{array}{rcl} \mathit{vars}.(\mathsf{Z}_1 \overrightarrow{\|} \mathsf{Z}_2) & \widehat{=} & \mathit{vars}.\mathsf{Z}_1 \cup \mathit{vars}.\mathsf{Z}_2 \\ \mathit{vars}.(\mathsf{rely}\, r \bullet \mathsf{Z}) & \widehat{=} & \mathit{vars}.r \cup \mathit{vars}.\mathsf{Z} \\ \mathit{vars}.(\mathsf{init}\, c \bullet \mathsf{Z}) & \widehat{=} & \mathit{vars}.c \cup \mathit{vars}.\mathsf{Z} \end{array}$$

The functions $\mathit{in}, \mathit{out}{:}\, P \rightarrow \mathbb{P}\, \mathit{Var}$ return the input and output variables of a teleo-reactive program, respectively, where:

$$\begin{array}{rcl} \mathit{out}.(F{:}[\![p]\!]) & \widehat{=} & F \\ \mathit{out}.\langle\rangle & \widehat{=} & \{\} \\ \mathit{out}.(\langle c \rightarrow \mathsf{Z}\rangle \frown S) & \widehat{=} & \mathit{out}.\mathsf{Z} \cup \mathit{out}.S \end{array} \qquad \begin{array}{rcl} \mathit{out}.(\mathsf{Z}_1 \overrightarrow{\|} \mathsf{Z}_2) & \widehat{=} & \mathit{out}.\mathsf{Z}_1 \cup \mathit{out}.\mathsf{Z}_2 \\ \mathit{out}.(\mathsf{rely}\, r \bullet \mathsf{Z}) & \widehat{=} & \mathit{out}.\mathsf{Z} \\ \mathit{out}.(\mathsf{init}\, c \bullet \mathsf{Z}) & \widehat{=} & \mathit{out}.\mathsf{Z} \end{array}$$

We define the set of input variables of Z as $\mathit{in}.\mathsf{Z} \,\widehat{=}\, \mathit{vars}.\mathsf{Z} \backslash \mathit{out}.\mathsf{Z}$.

Because the semantics of teleo-reactive programs is truly concurrent (as opposed to an interleaving semantics), two concurrent programs $\mathsf{Z}_1 \overrightarrow{\|} \mathsf{Z}_2$ may not modify the same variable, i.e., we require that $\mathit{out}.\mathsf{Z}_1 \cap \mathit{out}.\mathsf{Z}_2 = \{\}$. Furthermore, in program $\mathsf{Z}_1 \overrightarrow{\|} \mathsf{Z}_2$ the outputs of Z_1 may be

Suppose F is a set of variables, p is an interval predicate, Z, Z_1 and Z_2 are teleo-reactive programs, and S and $T \mathrel{\widehat{=}} \langle c \to \mathsf{Z} \rangle \frown S$ are sequences of guarded programs and r is a rely condition of Z. If V is an output context of each of the programs below, we define:

$$beh_V.(F{:}[\![p]\!]) \mathrel{\widehat{=}} p \wedge \mathbf{st}(V \backslash F) \quad (2)$$

$$beh_V.\langle\rangle \mathrel{\widehat{=}} true \quad (3)$$

$$beh_V.T \mathrel{\widehat{=}} ((\circledast c \wedge beh_V.\mathsf{Z}) : (\overleftarrow{\neg c} \wedge beh_V.T)) \vee ((\circledast \neg c \wedge beh_V.S) : (\overleftarrow{c} \wedge beh_V.T)) \quad (4)$$

$$beh_V.(\mathsf{Z}_1 \overrightarrow{\parallel} \mathsf{Z}_2) \mathrel{\widehat{=}} beh_{V \backslash out.\mathsf{Z}_2}.\mathsf{Z}_1 \wedge beh_{V \backslash out.\mathsf{Z}_1}.\mathsf{Z}_2 \quad (5)$$

$$beh_V.(\mathsf{rely}\, r \bullet \mathsf{Z}) \mathrel{\widehat{=}} r \Rightarrow beh_V.\mathsf{Z} \quad (6)$$

$$beh_V.(\mathsf{init}\, c \bullet \mathsf{Z}) \mathrel{\widehat{=}} \mathbf{prev}\, \overrightarrow{c} \wedge beh_V.\mathsf{Z} \quad (7)$$

Figure 4: *beh* function

used as inputs to Z_2, thus parallel composition is not necessarily commutative. We define *simple parallelism* $\mathsf{Z}_1 \| \mathsf{Z}_2$ as a special case of parallel composition in which the inputs of Z_1 and Z_2 are disjoint with the outputs of Z_2 and Z_1, respectively. Unlike $\mathsf{Z}_1 \overrightarrow{\parallel} \mathsf{Z}_2$, the programs under simple parallelism commute.

Teleo-reactive programs are often only required to execute correctly under certain environment assumptions; these assumptions may be formalised within a rely condition.

Definition 2 We say interval predicate r is a *rely condition* of teleo-reactive program Z iff $vars.r \cap out.\mathsf{Z} = \{\}$.

Definition 3 A set of variables V is an *output context* of a teleo-reactive program Z iff $V \supseteq out.\mathsf{Z}$ and $V \cap in.\mathsf{Z} = \{\}$.

The behaviour of a teleo-reactive program Z in a possibly wider output context V is given by $beh_V.\mathsf{Z}$ as defined in Fig. 4. The behaviour of a specification $F{:}[\![p]\!]$ with respect to the set V is given by (2), where in addition to behaving as specified by p, the variables of V that are not in F are guaranteed to be stable. An empty sequence of programs (3), is chaotic and allows any behaviour. The behaviour of a non-empty sequence of guarded programs, (4), is defined recursively. There are two disjuncts corresponding to either c or $\neg c$ holding initially on the interval. If c holds initially, either $\circledast c \wedge beh_V.\mathsf{Z}$ holds for the whole interval or the interval may be split into an initial interval in which $\circledast c \wedge beh_V.\mathsf{Z}$ holds, followed by an interval in which $\neg c$ holds initially and $beh_V.T$ holds (recursively) for the second interval. The other disjunct is similar. Note that each chopped interval must be a maximal interval over which either $\circledast c$ or $\circledast \neg c$ holds. The behaviour of the parallel composition between Z_1 and Z_2 is given by (5) and is defined to be the conjunction of the two behaviours with the outputs of each branch removed from the output context of the other. The behaviour of a program Z with rely condition r is denoted $\mathsf{rely}\, r \bullet \mathsf{Z}$ and its behaviour is given by (6). Thus, the program executes as defined by $beh_V.\mathsf{Z}$ provided the rely condition r holds. The initialisation of Z specifies a condition that holds prior to execution of Z and hence, the behaviour of $\mathsf{init}\, c \bullet \mathsf{Z}$ is given by (7).

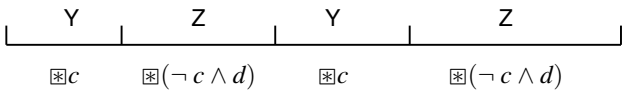

Figure 5: Execution of program $\langle c \to \mathsf{Y}, d \to \mathsf{Z} \rangle$

Example 3 *Consider the execution of program* $\langle c \to \mathsf{Y}, d \to \mathsf{Z} \rangle$ *in Fig. 5, where program* Y *executes over any interval in which* $\circledast c$ *holds and* Z *executes over any interval in which* $\circledast(\neg c \wedge d)$ *holds. Note that both programs* Y *and* Z *may themselves be teleo-reactive programs, and hence, each interval shown in Fig. 5 may further split into smaller sub-intervals for the branches of* Y *and* Z*. Because* $\circledast c$ *and* $\circledast(\neg c \wedge d)$ *are guaranteed to hold during the execution of* Y *and* Z*, respectively, we may assume* $\circledast c$ *and* $\circledast(\neg c \wedge d)$ *when reasoning about the subprograms within* Y *and* Z*, respectively.*

Definition 4 For teleo-reactive programs Z and Z′, we say Z *is refined by* Z′ (denoted $\mathsf{Z} \sqsubseteq \mathsf{Z}'$) iff $out.\mathsf{Z}' \subseteq out.\mathsf{Z}$ and $beh_V.\mathsf{Z}' \Rrightarrow beh_V.\mathsf{Z}$ for any V that is an output context of both Z and Z′.

Lemma 1 *If* F, F' *are a sets of variables and* p, p' *are interval predicates, then*

$$(F' \subseteq F) \wedge (vars.p' \subseteq vars.p) \wedge (\mathbf{st}(F \backslash F') \wedge p' \Rrightarrow p) \Rightarrow (F{:}[\![p]\!] \sqsubseteq F'{:}[\![p']\!])\,.$$

Definition 5 For an interval predicate p, we say p *splits* iff $p.\Delta \Rrightarrow \forall \delta{:}\Pi.\Delta \bullet \forall i{:}\mathrm{dom}.\delta \bullet p.(\delta.i)$, and p *joins* iff $(\exists \delta{:}\Pi.\Delta \bullet \forall i{:}\mathrm{dom}.\delta \bullet p.(\delta.i)) \Rrightarrow p.\Delta$.

For example, interval predicate $\ell < 3$ splits but does not join, $\boxdot c$ and $\ell \geq 3$ join but do not split and $\circledast c$ both splits and joins.

The next theorem presents a method for decomposing refinements for programs that are sequences of guarded programs.

Theorem 1 *For a teleo-reactive program* $T \mathrel{\widehat{=}} \langle c \to \mathsf{Z} \rangle \frown S$*, if* r *is a rely condition of* T *that splits and* g *is an interval predicate that joins, then* $\mathsf{rely}\, r \bullet F{:}[\![g]\!] \sqsubseteq T$ *holds provided that both:*

$$\mathsf{rely}\, r \bullet F{:}[\![\circledast c \Rightarrow g]\!] \quad \sqsubseteq \quad \mathsf{Z} \tag{8}$$

$$\mathsf{rely}\, r \bullet F{:}[\![\circledast \neg c \Rightarrow g]\!] \quad \sqsubseteq \quad S \tag{9}$$

Example 4 (Mine-pump in Fig. 3 satisfies *Safety*.) *Note that the program in Fig. 3 is idealised, i.e., we assume that the guards are continuously evaluated and that the pump reacts instantaneously. By defining* $MO \mathrel{\widehat{=}} out.\mathsf{mine\_pump}$ *and*

$$\mathsf{stop\_pump} \quad \mathrel{\widehat{=}} \quad \{stopped\}{:}[\![\circledast stopped]\!] \tag{10}$$

the proof of $MO{:}[\![Safety]\!] \sqsubseteq \mathsf{mine\_pump}$ *is straightforward, which proves that the mine pump program in Fig. 3 implements the safety requirement Safety. In particular, we have:*

$MO{:}[\![Safety]\!] \sqsubseteq \mathsf{mine\_pump}$
$\Leftarrow$ *Theorem 1, Safety joins*

$MO: [\![\circledast (m \geq C) \Rightarrow \mathit{Safety}]\!] \sqsubseteq (\mathsf{alarm} \| \mathsf{stop\_pump}) \wedge$
$MO: [\![\circledast (m < C) \Rightarrow \mathit{Safety}]\!] \sqsubseteq \mathsf{pump\_water}$
$\Leftarrow$ *first conjunct, LHS: Lemma 1, definition of Safety, RHS: definition of* $\|$
second conjunct: definition of Safety
$(MO: [\![\circledast \mathit{stopped}]\!] \sqsubseteq \mathsf{stop\_pump}) \wedge$
$(MO: [\![\mathit{true}]\!] \sqsubseteq \mathsf{pump\_water})$
$\Leftarrow$ *first conjunct: (10) and Lemma 1, second conjunct: Lemma 1 and* $\mathit{out}.\mathsf{pump\_water} \subseteq MO$
true

The ideal execution of a teleo-reactive program would *continuously* evaluate its guards all the time. Of course, continuous evaluation is not feasible and it has to be approximated by repeated sampling and evaluation. One of the main issues addressed in this paper is handling the imprecision introduced by such implementations by making use of Burns' time band framework [BB06, BH10].

4 Teleo-reactive programs with sampling

4.1 Sampling

A reactive controller uses (discrete) *sampling events* to determine the state of its (continuous) environment. Although a sampling event is viewed as instantaneous in the time band of the controller, sampling events actually take time. Thus, sampling events are prone to *timing precision errors* (where there is a range of possible sampled values due to imprecise timing of when the sample is taken) and *sampling anomalies*, i.e., sampling two or more sensors causes a non-existent state to be returned. Sampling is also prone to *sensor errors* (where the sensors have inaccuracies in measuring the environment), however, sampling errors are not the focus of this paper, and hence are elided.

We use a logic that assumes each environment variable in an expression is read exactly once within a sampling event and that this value is used for each occurrence of the variable in the expression. However, different variables may be read at different times within an interval, which makes it possible for a sampling event to return a state that does not actually exist [BH10]. Given a set of states $SS \subseteq \Sigma_V$, we define:

$$\mathit{values}.SS \mathrel{\widehat{=}} \lambda\, v{:}V \bullet \{\sigma{:}SS \bullet \sigma.v\} \qquad \mathit{apparent}.SS \mathrel{\widehat{=}} \{\sigma{:}\Sigma \mid (\forall v{:}V \bullet \sigma.v \in \mathit{values}.SS)\}$$

where the notation $\{\sigma{:}SS \bullet \sigma.v\}$ is equivalent to $\{x \mid (\exists \sigma{:}SS \bullet x = \sigma.v)\}$. That is, $\mathit{values}.SS$ returns a state that maps each variable v to the set of values that v may have in SS and $\mathit{apparent}.SS$ generates the set of all states in which each variable is one of its values in a state in SS, but different states could be used for different variables.

To reason about sampling anomalies, we define functions *states* and *av* that return the sets of *all states* and *all apparent states* within an interval of a stream, respectively. That is, if Δ is an interval and s is a stream, we define:

$$\mathit{states}.\Delta.s \mathrel{\widehat{=}} \{t{:}\Delta \bullet s.t\} \qquad \mathit{av}.\Delta.s \mathrel{\widehat{=}} \mathit{apparent}.(\mathit{states}.\Delta.s)$$

Using functions *states* and *av*, we formalise state predicates that are *definitely* true (denoted $\circledast$) and *possibly* true (denoted $\odot$) over a given interval Δ and stream s as follows:

$$(\circledast c).\Delta.s \mathrel{\widehat{=}} \forall \sigma{:}\, av.\Delta.s \bullet c.\sigma \qquad\qquad (\odot c).\Delta.s \mathrel{\widehat{=}} \exists \sigma{:}\, av.\Delta.s \bullet c.\sigma$$

If $(\circledast c).\Delta.s$ holds, then c holds for each apparent state in the interval Δ and if $\odot c$ holds then c holds in some apparent state. Note that because $states.\Delta.s \subseteq apparent.\Delta.s$ holds for any interval Δ and stream s, both $\circledast c \Rrightarrow \boxplus c$ and $\boxdot c \Rrightarrow \odot c$ hold. There are several relationships between $\circledast$ and $\odot$ [BH10]. In this paper, we find the following lemma to be useful [HBDJ11].

Lemma 2 *For a state predicate c and variable v,* $\mathbf{st}(vars.c \backslash \{v\}) \Rrightarrow (\circledast c = \boxplus c) \wedge (\odot c = \boxdot c)$.

That is, if at most one variable of c is non-stable within an interval then c definitely holds iff c holds everywhere (and similarly c possibly holds).

4.2 Time bands

The set of all time bands is given by the primitive type *TimeBand*, which defines a unit of time, e.g., seconds, days, years. The precision of a time band is given by $\rho{:}\, TimeBand \to \mathbb{R}^{>0}$. We define a time band $b_1 + b_2$ to be a band such that $\rho.(b_1 + b_2) \mathrel{\widehat{=}} \rho.b_1 + \rho.b_2$. For a real-valued variable v, interval Δ and stream s, we define:

$$diff.v.\Delta.s \quad \mathrel{\widehat{=}} \quad \mathsf{let}\ vs = \{\sigma{:}\, states.\Delta.s \bullet \sigma.v\}\ \mathsf{in}\ lub.vs - glb.vs$$

Thus, *diff* returns the difference between the maximum and minimum values of the given variable in the given interval and stream. To this end, we define the *accuracy* of a variable v in time band b using $accuracy.v.b$, which limits the maximum change to the variable within events of time band b. For any variable v and time band b, we implicitly assume the following rely condition:

$$\ell.\Delta \le \rho.b \quad \Rrightarrow \quad diff.v.\Delta \le accuracy.v.b \tag{11}$$

Lemma 3 *For a variable v, constant k, and time band b,*

$$\ell \le \rho.b \wedge \boxdot(v < k - accuracy.v.b) \Rrightarrow \boxplus(v < k)\,.$$

Proof. The proof is trivial by the assumption (11) on *accuracy*. □

4.3 Extended syntax and semantics

For a guarded sequence of actions T and a time band b, we use $T \dagger b$ to denote that the guards of T are repeatedly evaluated within the precision of time band b. We use functions $grd.(c \to \mathsf{Z}) \mathrel{\widehat{=}} c$ and $body.(c \to \mathsf{Z}) \mathrel{\widehat{=}} \mathsf{Z}$ to denote the guard and body of the guarded program $c \to \mathsf{Z}$, respectively.

Definition 6 For any $i \in \mathrm{dom}.T$, the *effective guard* of $T.i$ (denoted $eff.(T.i)$) is given by $grd.(T.i) \wedge \bigwedge_{j:0..i-1} \neg grd.(T.j)$.

That is, the effective guard of $T.i$ is the actual guard of $T.i$ in conjunction with the negations of all guards that precede i in T.

Execution of program $T \dagger b$ consists of evaluation of all guards within intervals of size $\rho.b$ or less. Then, branch $T.j$ is executed over interval Δ if there is a partition δ of Δ such that the effective guard of $T.j$ possibly holds in each $\delta.i$, and furthermore, the body of $T.j$ executes as defined by the behaviour function over Δ. For a state predicate c, time band b and interval Δ, we define the following:

$$\odot_b c \;\widehat{=}\; \ell \leq \rho.b \wedge \odot c \tag{12}$$

$$(\odot_b^+ c).\Delta \;\widehat{=}\; \exists \delta : \Pi.\Delta \bullet \forall i : \mathrm{dom}.\delta \bullet (\odot_b c).(\delta.i) \tag{13}$$

Lemma 4 *If c is a state predicate, b is a time band, p is an interval predicate that joins, s is a stream and $\forall \Omega : Interval \bullet (\odot_b c \Rightarrow p).\Omega.s$ holds, then $\forall \Delta : Interval \bullet (\odot_b^+ c \Rightarrow p).\Delta.s$.*

Proof. For any interval Δ, we have the following calculation:

$$\begin{array}{ll} & (\odot_b^+ c \Rightarrow p).\Delta.s \\ = & \quad \text{point-wise lifting, definition of } \odot_b^+ c \\ & (\exists \delta : \Pi.\Delta \bullet \forall i : \mathrm{dom}.\delta \bullet (\odot_b c).(\delta.i).s) \Rightarrow p.\Delta.s \\ = & \quad \text{logic} \\ & \forall \delta : \Pi.\Delta \bullet \forall i : \mathrm{dom}.\delta \bullet (\odot_b c).(\delta.i).s \Rightarrow p.\Delta.s \\ \Leftarrow & \quad p \text{ joins} \\ & \forall \delta : \Pi.\Delta \bullet \forall i : \mathrm{dom}.\delta \bullet (\odot_b c).(\delta.i).s \Rightarrow p.(\delta.i).s \\ \Leftarrow & \quad \text{logic} \\ & \forall \Omega : Interval \bullet (\odot_b c \Rightarrow p).\Omega.s \end{array}$$

□

We prove the following lemma that relates a sampled value of a variable to its true value in the environment using the accuracy of the variable.

Lemma 5 *For a variable v, constant k and time band b, $\odot_b^+ (v < k - accuracy.v.b) \Rrightarrow \circledast(v < k)$.*

Proof. For any interval Δ and stream s, we have

$$\begin{array}{ll} & (\odot_b^+ (v < k - accuracy.v.b) \Rightarrow \circledast(v < k)).\Delta.s \\ \Leftarrow & \quad \text{Lemma 4, } \circledast c \text{ joins} \\ & \forall \Omega : Interval \bullet (\odot_b (v < k - accuracy.v.b) \Rightarrow \circledast(v < k)).\Omega.s \\ \Leftarrow & \quad \text{Lemma 3 and Lemma 2} \\ & true \end{array}$$

□

If $j \in \mathrm{dom}.T$, we define the execution of guarded program $T.j$ within time band b as follows:

$$exec_V.(T.j).b.\Delta \;\widehat{=}\; (\odot_b^+ eff.(T.j)).\Delta \wedge beh_V.(body.(T.j)).\Delta$$

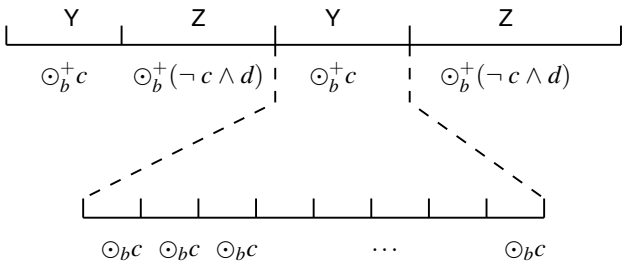

Figure 6: Execution of program $\langle c \to \mathsf{Y}, d \to \mathsf{Z} \rangle \dagger b$ with guard approximation

$$\left\langle \begin{array}{rcl} \boxed{m \geq D} & \to & \mathsf{alarm} \| \mathsf{stop\_pump}, \\ \mathit{true} & \to & \mathsf{pump\_water} \end{array} \right\rangle \boxed{\dagger M}$$

Figure 7: Top-level program with methane time band

Thus, there must exist a partition of Δ, δ, such that for each index $i \in \mathrm{dom}.\delta$, the length of the interval $\delta.i$ is at most the precision, $\rho.b$, of time band b and the effective guard of $T.j$ possibly holds within $\delta.i$. Furthermore, the behaviour of $body.(T.j)$ holds over Δ. We say a teleo-reactive program T is *well-formed* iff $last.T = true \to \mathsf{Z}$ for some program Z.

Definition 7 For a well-formed teleo-reactive program $T \dagger b$, we define

$$\begin{array}{rcl} beh_V.(T \dagger b).\Delta & \widehat{=} & \exists \delta : \Pi.\Delta \bullet \exists act : (\mathrm{dom}.\delta \to \mathrm{dom}.T) \bullet \forall i : \mathrm{dom}.act \bullet \\ & & ((exec_V.(T.(act.i)).b).(\delta.i) \wedge ((i > 0) \Rightarrow act.i \neq act.(i-1)) \end{array}$$

Thus, we say $beh_V.(T \dagger b).\Delta$ holds iff there is a partition, δ, of Δ and a mapping, act, from the domain of δ to the domain of T such that for every $i \in \mathrm{dom}.act$, execution of $T.(act.i)$ holds in $\delta.i$ and furthermore, consecutive intervals of δ are mapped to different elements of T. Note that $\mathrm{dom}.act = \mathrm{dom}.\delta$. Definition 7 allows both Zeno and non-Zeno executions of T, however, we can only implement non-Zeno behaviour. This is not problematic because the definition does not require Zeno behaviour, i.e., it allows non-Zeno behaviour.

Example 5 *Execution of the program $\langle c \to \mathsf{Y}, d \to \mathsf{Z} \rangle \dagger b$ is given in Fig. 6, where the guard $\circledast c$ is approximated as $\odot_b^+ c$. Note that $\odot_b c$ holds iteratively over the interval, whereas Y executes over the whole interval in which $\odot_b^+ c$ holds.*

A version of the top-level mine_pump program from Fig. 3 that does not make idealised assumptions is given in Fig. 7, where the changes are identified within the boxes. In particular, we have introduced a time band $M \in \mathit{TimeBand}$ which represents the time band of the methane. Introduction of M within the program defines the minimum rate at which the methane is sampled. Because the program approximates the value of m using a sampling event, the guard $m \geq C$ has been replaced by $m \geq D$. We calculate the relationship between C and D that is necessary for proving *Safety* in Section 5.1.

5 Approximating specifications with time bands

We have developed a method of refining timed specifications and a formal semantics for both idealised and time-banded teleo-reactive programs. We are able to prove that the idealised teleo-reactive programs implement the (ideal) specifications. However, as with robust automata, proving that the requirements that are specified at the absolute level of precision are implemented by the time-banded teleo-reactive programs is, in general, difficult.

5.1 Incorporating the methane time band

We have a decomposition theorem for time-banded teleo-reactive programs that is similar to Theorem 1 for idealised programs.

Theorem 2 *Suppose* $T \mathrel{\widehat{=}} (\langle c \to \mathsf{Z}\rangle \frown S) \dagger b$ *is a time-banded teleo-reactive program and* F *is a set of variables. If* r *is a rely condition of* T *that splits and* g *is an interval predicate that joins, then* $\mathsf{rely}\, r \bullet F{:}[\![g]\!] \sqsubseteq T$ *holds provided that both*

$$\mathsf{rely}\, r \bullet F{:}\left[\!\left[\odot_b^+ c \Rightarrow g\right]\!\right] \sqsubseteq \mathsf{Z} \tag{14}$$

$$\mathsf{rely}\, r \bullet F{:}\left[\!\left[\odot_b^+ \neg c \Rightarrow g\right]\!\right] \sqsubseteq S \tag{15}$$

Example 6 (Program in Fig. 7 satisfies *Safety*.) *Applying Theorem 2 to prove* $MO{:}[\![Safety]\!]$ *gives us the following proof obligations.*

$$MO{:}\left[\!\left[\odot_M^+(m \geq D) \Rightarrow Safety\right]\!\right] \sqsubseteq \mathsf{alarm} \| \mathsf{stop\_pump} \tag{16}$$

$$MO{:}\left[\!\left[\odot_M^+(m < D) \Rightarrow Safety\right]\!\right] \sqsubseteq \mathsf{pump\_water} \tag{17}$$

We let $D \leq C - accuracy.m.M$ *and have the following calculations:*

$$\begin{array}{ll} & \odot_M^+(m \geq D) \Rightarrow Safety \\ \Leftarrow & \quad \textit{logic} \\ & \circledast stopped \end{array}$$

$$\begin{array}{ll} & \odot_M^+(m < D) \Rightarrow Safety \\ \Leftarrow & \quad \textit{Lemma 5, } D \leq C - accuracy.m.M \\ & \circledast(m < C) \Rightarrow Safety \\ \equiv & \quad \textit{antecedent of Safety is false} \\ & true \end{array}$$

Hence, we have:

$$\begin{array}{ll} & (16) \\ \Leftarrow & \quad \textit{calculation above, definition of } \| \\ & MO{:}[\![\circledast stopped]\!] \sqsubseteq \mathsf{stop\_pump} \\ \Leftarrow & \quad (10) \textit{ definition of } \mathsf{stop\_pump} \\ & true \end{array}$$

$$\begin{array}{ll} & (17) \\ \Leftarrow & \quad \textit{calculation above} \\ & MO{:}[\![true]\!] \sqsubseteq \mathsf{pump\_water} \\ \Leftarrow & \quad out.\mathsf{pump\_water} \subseteq MO \\ & true \end{array}$$

We have considered the time taken to sample the methane into account and established a relationship between the sampled and real value of the methane, which we use to prove *Safety*. However, the program in Fig. 7 is not realistic because it assumes that the pump is stopped instantaneously. In fact, the specification requires the pump to be stopped from the beginning of the interval over which the methane is sampled to be high, even before the first high sample

is taken. In the next section, we describe how the program may be modified so that the safety condition holds for a pump that is guaranteed to stop in its own time band.

5.2 Incorporating the pump time band

We consider the program in Fig. 8, where the program begins executing after initialisation *stopped*, the guard for stopping the pump has been modified to $m \geq E$ and $\mathsf{stop\_pump_P}$ is used to stop the pump. Given that $P \in \mathit{TimeBand}$ is the time band of the pump, we define:

$$\mathsf{stop\_pump_P} \;\widehat{=}\; \{\mathit{stopped}\} : [\![\mathbf{inv}\, \mathit{stopped} \wedge ((\ell \leq \rho.P) : (\circledast \mathit{stopped}))]\!] \tag{18}$$

i.e., if the pump is already stopped it will remain so and otherwise, execution of $\mathsf{stop\_pump_P}$ over any interval of length $\rho.P$ or greater (i.e., the precision of the pump) is guaranteed to stop the pump. Note that this specification does not limit the deceleration of the pump, i.e., there may be several possible implementations of this specification at higher precision time bands. Each implementation must only guarantee that $\overrightarrow{\mathit{stopped}}$ holds within an interval of the precision $\rho.P$. Note that the bands within the program in Fig. 8 serve slightly different purposes; band P restricts the precision of the pump events, whereas band M restricts the rate at which the methane is sampled.

Lemma 6 *For a continuous variable, m, in time band M, and constant E,*

$$\mathbf{prev}\, \odot^{+}_{M}(m < E) \Rrightarrow \overleftarrow{m \leq E + \mathit{accuracy}.m.M}$$

Proof. The proof makes use of the accuracy of m within time band M.

$\mathbf{prev}(\odot^{+}_{M}(m < E))$
$\Rrightarrow$ Lemma 3
$\mathbf{prev}(\circledast(m < E + \mathit{accuracy}.m.M))$
$\Rrightarrow$ continuity of m
$\mathbf{prev}(\overrightarrow{m \leq E + \mathit{accuracy}.m.M})$
$\Rrightarrow$ continuity of m
$\overleftarrow{m \leq E + \mathit{accuracy}.m.M}$

□

Lemma 7 *For a continuous variable m in time band P, and constant K,*

$$\overleftarrow{m \leq K} \wedge \ell < \rho.P \Rrightarrow \circledast(m < K + \mathit{accuracy}.m.P)\,.$$

Proof. Because m is continuous and no greater than K at the left limit of an interval that is of length bounded by $\rho.P$, m cannot increase by more than its accuracy in band P. □

We present a third decomposition theorem for proving refinements where the given program executes after some initialisation.

$$\boxed{\text{init } \mathit{stopped} \bullet} \left\langle \begin{array}{rcl} \boxed{m \geq E} & \rightarrow & \text{alarm} \parallel \boxed{\text{stop\_pump}_{\mathsf{P}}}, \\ \mathit{true} & \rightarrow & \text{pump\_water} \end{array} \right\rangle \dagger M$$

Figure 8: Top-level program with methane and pump time bands

Theorem 3 *Suppose* $T \mathrel{\widehat{=}} (\langle c \rightarrow \mathsf{Z} \rangle \frown S) \dagger b$ *is a time-banded teleo-reactive program,* F *is a set of variables, and* d *is an state predicate. If* r *is a rely condition of* T *that splits and* g *is an interval predicate that joins, then* $\text{rely}\, r \bullet F{:}\,[\![g]\!] \sqsubseteq \text{init}\, d \bullet T$ *holds provided:*

$$\text{rely}\, r \bullet F{:}\left[\!\!\left[\odot_b^+ c \wedge \mathbf{prev}(\odot_b^+ \neg c \vee \overrightarrow{d}) \Rightarrow g \right]\!\!\right] \sqsubseteq \mathsf{Z} \tag{19}$$

$$\text{rely}\, r \bullet F{:}\left[\!\!\left[\odot_b^+ \neg c \wedge \mathbf{prev}(\odot_b^+ c \vee \overrightarrow{d}) \Rightarrow g \right]\!\!\right] \sqsubseteq S \tag{20}$$

Note that Theorem 2 can be derived from Theorem 3 if the predicate d is just true.

Example 7 (Program in Fig. 8 satisfies *Safety*.) *Applying Theorem 3 to prove* $MO{:}\,[\![\mathit{Safety}]\!] \sqsubseteq \text{init}\, \mathit{stopped} \bullet \text{mine\_pump}$ *gives us:*

$$MO{:}\left[\!\!\left[\odot_M^+ (m \geq E) \wedge \mathbf{prev}(\odot_M^+ (m < E) \vee \overrightarrow{\mathit{stopped}}) \Rightarrow \mathit{Safety} \right]\!\!\right] \sqsubseteq \text{alarm}\|\text{stop\_pump}_{\mathsf{P}} \tag{21}$$

$$MO{:}\left[\!\!\left[\odot_M^+ (m < E) \wedge \mathbf{prev}(\odot_M^+ (m \geq E) \vee \overrightarrow{\mathit{stopped}}) \Rightarrow \mathit{Safety} \right]\!\!\right] \sqsubseteq \text{pump\_water} \tag{22}$$

By assuming $E \leq C - \mathit{accuracy.m.M}$*, using Lemma 5, the proof of (22) is straightforward because the left-hand-side of* $\sqsubseteq$ *in (22) reduces to* $MO{:}\,[\![\mathit{true}]\!]$*. For (21), we strengthen the assumption to* $E < C - \mathit{accuracy.m.}(M+P)$ *and perform the following calculation:*

$$\begin{array}{ll} & \odot_M^+ (m \geq E) \wedge \mathbf{prev}(\odot_M^+ (m < E) \vee \overrightarrow{\mathit{stopped}}) \Rightarrow \mathit{Safety} \\ \Leftarrow & \quad \textit{weaken antecedent, definition of } \mathbf{prev} \\ & (\mathbf{prev}\, \odot_M^+ (m < E)) \Rightarrow \mathit{Safety}) \wedge ((\mathbf{prev}\, \overrightarrow{\mathit{stopped}}) \Rightarrow \mathit{Safety}) \\ \Leftarrow & \quad \textit{strengthen consequents} \\ & ((\mathbf{prev}\, \odot_M^+ (m < E)) \Rightarrow ((\ell \leq \rho.P) : (\circledast \mathit{stopped})) \wedge \mathit{Safety}) \wedge ((\mathbf{prev}\, \overrightarrow{\mathit{stopped}}) \Rightarrow \circledast \mathit{stopped}) \\ \Leftarrow & \quad \textit{Safety joins, definition of } \mathit{inv} \\ & ((\mathbf{prev}\, \odot_M^+ (m < E)) \Rightarrow ((\ell \leq \rho.P \wedge \mathit{Safety}) : (\circledast \mathit{stopped} \wedge \mathit{Safety}))) \wedge \mathbf{inv}\, \mathit{stopped} \\ \Leftarrow & \quad \textit{Lemma 6; definition of } \mathit{Safety} \\ & (\overleftarrow{m \leq E + \mathit{accuracy.m.M}} \Rightarrow ((\ell \leq \rho.P \wedge \boxplus (m < C)) : \boxplus \mathit{stopped})) \wedge \mathbf{inv}\, \mathit{stopped} \\ \Leftarrow & \quad \textit{Lemma 7 and assumption } E < C - \mathit{accuracy.m.}(M+P) \\ & (\overleftarrow{m \leq E + \mathit{accuracy.m.M}} \Rightarrow ((\ell \leq \rho.P) : \boxplus \mathit{stopped})) \wedge \mathbf{inv}\, \mathit{stopped} \\ \Leftarrow & \quad \textit{logic} \\ & ((\ell \leq \rho.P) : \boxplus \mathit{stopped}) \wedge \mathbf{inv}\, \mathit{stopped} \end{array}$$

Using Lemma 1 and the definition of $\|$, it is straightforward to verify that the specification $MO{:}\,[\![((\ell \leq \rho.P) : \boxplus \mathit{stopped}) \wedge \mathbf{inv}\, \mathit{stopped}]\!]$ is refined by $\text{alarm}\|\text{stop\_pump}_{\mathsf{P}}$, which completes the proof of *Safety*.

6 Related work and conclusions

In the context of timed-automata, researchers have developed *robust timed automata* [GHJ97], which weaken the specification of the original automata to accept more traces. Implementation of robust automata is known to be problematic because the original specification is weakened. Algorithms for developing robust automata from idealised automata so that safety properties are preserved are currently impractical and preservation of general temporal logic properties is currently not possible [WDMR08].

Alur et al have considered *perturbed timed automata*, which focus on clock errors (or perturbations) [ALM05]. However in their own words:

> Thus, checking equivalence of timed circuits composed of components with imperfect clocks, in terms of timed languages over inputs and outputs, remains an interesting open problem. [ALM05, pg84]

In the context of refinement, Boiten and Derrick have proposed "approximating refinements" [BD05], where metrics are used to develop implementations for situations in which refinement is not possible. The argument is that realistic implementations are limited by physical resources such as memory, which place restrictions on the ideal specifications. Our work differs from this in that we are concerned with approximating idealised timing specifications.

Henzinger presents a theory of timed refinement where sampling events are executed by a separate process [HQR99]. Moszkowski presents a method of abstracting between different time granularities for interval temporal logic, however the model uses a discrete framework of time [Mos95] as opposed to our continuous model. The formalisms above do not consider the possibility of sampling anomalies. Broy [Bro01] presents a refinement framework that formalises the relationships between different models of time. This includes abstraction techniques from dense to discrete models of time using sampling. However, the sampling theory is not well developed and the techniques only consider discretisation of dense streams.

We have presented a model in which specifications defined over an absolute level of precision may be approximated using the time bands in which the input variables are sampled. This approximating process loosens the specification and the values of variables interpreted in a time band, say b, are taken to be the values of the variable within the precision of b. We have presented lemmas for relating the sampled and real values of a variable. We have also described how the behaviour of output variables may be formalised over a time band, which allows one to take delays in both signalling and hardware into account. Implementation of specifications is defined with respect to a refinement relation on interval predicates, which ensures that each real-time behaviour of the implementation is an behaviour of the abstract specification.

Future work includes proofs of progress properties and methods for approximating specifications in the presence of feedback. It is also important to consider more complicated systems in which the teleo-reactive controller samples multiple variables, where it is possible for sampling events to suffer from sampling anomalies.

Bibliography

[ALM05] R. Alur, S. La Torre, P. Madhusudan. Perturbed Timed Automata. In Morari and Thiele (eds.), *Hybrid Systems: Computation and Control*. LNCS 3414, pp. 70–85. Springer Berlin / Heidelberg, 2005.

[BB06] A. Burns, G. Baxter. Time bands in systems structure. In *Structure for Dependability: Computer-Based Systems from an Interdisciplinary Perspective*. Pp. 74–88. Springer-Verlag, 2006.

[BD05] E. A. Boiten, J. Derrick. Formal Program Development with Approximations. In Treharne et al. (eds.), *ZB*. LNCS 3455, pp. 374–392. Springer, 2005.

[BH10] A. Burns, I. J. Hayes. A Timeband Framework for Modelling Real-Time Systems. *Real-Time Systems* 45(1):106–142, 2010.

[BL91] A. Burns, A. M. Lister. A Framework for Building Dependable Systems. *Comput. J.* 34(2):173–181, 1991.

[Bro01] M. Broy. Refinement of time. *Theor. Comput. Sci.* 253(1):3–26, 2001.

[GHJ97] V. Gupta, T. A. Henzinger, R. Jagadeesan. Robust Timed Automata. In Maler (ed.), *HART*. LNCS 1201, pp. 331–345. Springer, 1997.

[GM01] A. Gargantini, A. Morzenti. Automated deductive requirements analysis of critical systems. *ACM Trans. Softw. Eng. Methodol.* 10:255–307, July 2001.

[Hay08] I. J. Hayes. Towards reasoning about teleo-reactive programs for robust real-time systems. In *SERENE '08*. Pp. 87–94. ACM, New York, NY, USA, 2008.

[HBDJ11] I. J. Hayes, A. Burns, B. Dongol, C. Jones. Comparing Models of Nondeterministic Expression Evaluation. Technical report CS-TR-1273, Newcastle University, 2011.

[HQR99] T. A. Henzinger, S. Qadeer, S. K. Rajamani. Assume-Guarantee Refinement Between Different Time Scales. In Halbwachs and Peled (eds.), *CAV*. LNCS 1633, pp. 208–221. Springer, 1999.

[Mos95] B. C. Moszkowski. Compositional reasoning about projected and infinite time. In *ICECCS*. Pp. 238–245. IEEE Computer Society, 1995.

[Nil01] N. J. Nilsson. Teleo-reactive programs and the triple-tower architecture. *Electronic Transactions on Artificial Intelligence* 5:99–110, 2001.

[WDMR08] M. Wulf, L. Doyen, N. Markey, J.-F. Raskin. Robust safety of timed automata. *Form. Methods Syst. Des.* 33:45–84, December 2008.

[ZH04] C. Zhou, M. R. Hansen. *Duration Calculus: A Formal Approach to Real-Time Systems*. EATCS: Monographs in Theoretical Computer Science. Springer, 2004.