Deriving Real-Time Action Systems Controllers from Multiscale System Specifications

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

[1] School of Information Technology and Electrical Engineering
The University of Queensland, Australia
[2] Department of Computer Science, The University of Sheffield, UK
B.Dongol@sheffield.ac.uk, Ian.Hayes@itee.uq.edu.au

Abstract. This paper develops a method for deriving controllers for real-time systems in which the components of the system operate at different time granularities. To this end, we incorporate the theory of time bands into action systems, which allows one to structure a system into multiple abstractions of time. The framework includes a logic that facilitates reasoning about different types of sampling errors and transient properties (i.e., properties that only hold for a brief amount of time), and we develop theorems for simplifying proofs of hardware/software interaction. We formalise true concurrency and define refinement for the parallel composition of action systems. Our method of derivation builds on the verify-while-develop paradigm, where the action system code is developed side-by-side with its proof.

1 Introduction

Action systems provide a simple framework within which several theories of program refinement have been developed [3–6]. In its simplest form, an action system consists of a set of actions (i.e., guarded statements) and a loop that at each iteration non-deterministically chooses then executes an enabled action from the set of actions. The loop terminates iff all of the actions are disabled. Typically, an action system includes actions of both the controller and its environment and uses an execution model in which the controller and environment actions are interleaved with each other. To cope with continuous environments, action systems have been extended in several ways. *Continuous action systems* [2, 27] give a semantics using standard action systems but with an added time variable. At the end of each iteration of the main loop, time is incremented to the first time at which some guard of the action system is enabled. *Hybrid action systems* [30] take the approach that the actions of the controller are discrete (and instantaneous) and allow the environment to execute evolution actions, which describe the (continuous) evolution of the state over the interval in which the evolution guard is enabled. A prioritised alternating model of execution is used to ensure that the (discrete) controller actions are able to execute. Hybrid action systems have been extended to *qualitative action systems* [1], but this work

J. Gibbons and P. Nogueira (Eds.): MPC 2012, LNCS 7342, pp. 102–131, 2012.

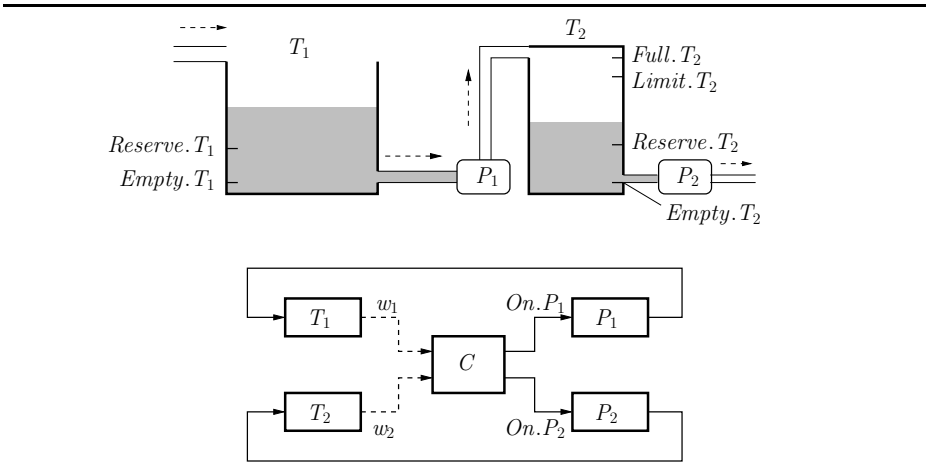

Fig. 1. Two-pump system

is focused on methods for testing real-time systems as opposed to their formal verification/derivation.

Ultimately, the interleaving execution model is problematic in contexts such as real-time and multi-core systems where the environment evolves with the controller in a truly concurrent manner. In such contexts, one must address issues with sampling multiple variables over a time interval [9, 17, 23] and be able to reason about transient properties [14, 17]. Furthermore, as software controllers are increasingly used in complex cyber-physical systems, it becomes important to be able to reason over multiple time granularities [8, 9, 16, 23].

1.1 Motivating Example

We consider a system consisting of two water tanks T_1, T_2 and two pumps P_1, P_2 depicted in Fig. 1 (also see [1]). The environment (of the system) adds water to tank T_1 and does not affect tank T_2. We assume that tank T_1 is allowed to overflow, but T_2 is not. Pump P_1 removes water from tank T_1 and fills tank T_2. Pump P_2 only operates if a button B (not shown in Fig. 1) is pressed and removes water from tank T_2. Aichernig et al [1] describe the following requirements. We have adapted their informal specification to clarify the input/output behaviours of the pump and to better distinguish safety (**S1**, **S2** and **S3**) and progress (**P1**, **P2** and **P3**). Note that a progress property to turn pump P_1 off is not needed because it is implied by safety properties **S1** and **S2**.

S1. If the water level in T_1 is *Empty* or below, then P_1 must be stopped.
S2. If the water level in T_2 is *Full* or above, then P_1 must be stopped.
S3. If the water level in T_2 is *Empty* or below, then P_2 must be stopped.

P1. If the water level in T_2 is definitely below the *Reserve* and water level in T_1 is definitely above the *Reserve*, then turn on P_1.

P2. If B is definitely pressed and the water level in T_2 is definitely above *Reserve*, then turn on P_2.
P3. If B is definitely not pressed, then turn off P_2.

Thus, we must keep track of water levels $Reserve.T_1$ and $Empty.T_1$ in tank T_1 and $Full.T_2$, $Reserve.T_2$, $Empty.T_2$ in tank T_2. For $i \in \{1, 2\}$, we distinguish between signal On_i that starts/stops pump P_i, and $Running_i$ and $Stopped_i$ that hold iff P_i is physically running and stopped, respectively. Note that pump P_i may also be associated with other states such as $Starting_i$, etc. We let w_1 and w_2 denote the water levels in tanks T_1 and T_2, respectively and say *Pressed* holds iff the button B is pressed.

A (digital) controller for pump P_2 must sample both the water level in tank T_2 and the state of the button B, perform some processing, then send on/off signals to pump P_2 if necessary. Each of these phases takes time. Furthermore, the components operate at different time granularities and hence have different notions of precision (the amount of time that may be regarded as instantaneous [8, 9]). For example, w_1 may have a precision of 30 seconds (i.e., there is no significant change in the water level in tank T_1 within 30 seconds) and pump P_2 turns on/off with precision 1 second (i.e., it takes pump P_2 at most 1 second to reach its operating speed or to come to a stop). Formally reasoning about the system in a manner that properly addresses each of these timing aspects is complicated [16, 17, 22]. To reduce the complexity of the reasoning, formal frameworks often simplify specifications by assuming that certain aspects of the system (e.g., sampling) are instantaneous or take a negligible amount of time. However, it is well-known that such simplifications can cause complications during implementation. In particular, the developed specifications become unimplementable because their timing requirements cannot be satisfied by any real system [22, 31].

Properties that use "definitely" are properties that hold over events of a time band. The progress properties are interpreted in the water time band. For example, within **P1**, "T_2 is definitely below the *Reserve*" is interpreted as "T_2 is definitely below the *Reserve* for at least the precision of the water time band".

1.2 Contributions and Overview

In this paper, we use time bands [8, 9] which facilitate reasoning about systems specified over multiple time granularities. Together with a logic of sampling, this allows one to properly address transient properties [14, 17]. We develop a framework using action systems and formalise true concurrency between an action system and its environment as well as between the parallel composition of two or more action systems. We define stream-based refinement of action systems (and their parallel composition) and present our method of derivation using "enforced properties" [12, 14], which allows one to use a verify-while-develop method [11, 18, 19]. We develop high-level theories for reasoning about systems that involve interaction between hardware and software over multiple time bands and as an example, we present the derivation of a pump controller.

Unlike Burns/Hayes [9] whose framework is based on sets of states, we develop our semantics using an interval-based framework. We present the background theory in Section 2, where we define interval predicates, methods of evaluating state predicates over an interval (including via sampling), our chop and iterated chop operators [32], and an LTL-like [26] logic for interval predicates. In Section 3, we present our formalisation of time bands (which includes time-band predicates), formalisation of the syntax and semantics of action systems with time bands and parallel composition of action systems. In Section 4, we present our methods for deriving action systems using enforced properties that builds on our previous refinement theories [12, 14, 17]. Section 5 presents our high-level theorems for reasoning about hardware/software interaction and an example derivation is given in Section 6.

1.3 Related Work

The idea of reasoning about systems using multiple granularities of time is not new. Moszkowski presents a method of abstracting between different time granularities for interval temporal logic using a projection operator for a discrete interval temporal logic [28]. Guelev and Hung present a projection operator for the duration calculus. Although computation is assumed to take time, the time taken is assumed to be negligible [21]. Henzinger presents a theory of timed refinement where sampling events are executed by a separate process [25]. Broy [7] presents a timed refinement framework that formalises the relationships between dense and discrete time where sampling is considered be a discretisation of dense streams.

This paper continues our research into methods for program derivation using the verify-while-develop paradigm. The method of enforced properties [12, 13] has been extended to enable development of action systems in a compositional manner [14]. The logic in [14] considers traces that consist of pre/post state relations, develops a temporal logic on relations and assumes that environment transitions are interleaved with those of an action system. Although the framework facilitates compositional derivation of action systems code, the underlying interleaving semantics assumption could not properly address sampling anomalies and transient properties. Hence, the framework was generalised so that traces consisted of adjoining intervals together with a sampling logic (Section 2.2), which allowed sampling-related issues to be properly addressed [17]. However, the logic in [17] does not adequately handle specifications over multiple time granularities. Instead, hardware is assumed to react and take effect instantaneously, which is unrealistic.

2 Background Theory

2.1 Interval Predicates

We model time using the real numbers, $\mathbb{R}$, and let *Interval* denote the set of all contiguous non-empty subsets of time. An interval may be open or closed at

either end, have a least upper bound ∞ or a greatest lower bound $-\infty$ (i.e., not in $\mathbb{R}$).

We let $glb.\Delta$ and $lub.\Delta$ denote the *greatest lower* and *least upper bounds* of interval Δ, respectively, where '.' denotes function application. For intervals Δ and Δ', we define the *length* of Δ and *adjoins* relation between Δ and Δ' as follows.

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

Given that variable names are taken from the set Var, a *state space* over a set of variables $V \subseteq Var$ is given by $\Sigma_V \mathrel{\widehat{=}} V \to Val$, which is a total function from variables in V to values in Val. A *state* is a member of Σ_V. The (dense) stream of states over V is given by $Stream_V \mathrel{\widehat{=}} \mathbb{R} \to \Sigma_V$, which is a total function from real numbers to states. A *predicate* over a type T is given by $\mathcal{P}\,T \mathrel{\widehat{=}} T \to \mathbb{B}$ (e.g., a *stream predicate* is a member of $\mathcal{P}\,Stream_V$), where $\mathbb{B}$ is the type of a boolean. An interval stream predicate, which we shorten to *interval predicate*, has type $IntvPred_V \mathrel{\widehat{=}} Interval \to \mathcal{P}\,Stream_V$. We write Σ, $Stream$ and $IntvPred$ for Σ_V, $Stream_V$ and $IntvPred_V$, respectively when the set V is clear from the context.

For an interval predicate p and interval Δ we define the following, where the stream is implicit in both sides of the definitions.

$$(prev.p).\Delta \mathrel{\widehat{=}} \exists \Delta' : Interval \bullet (\Delta' \propto \Delta) \wedge p.\Delta'$$
$$(next.p).\Delta \mathrel{\widehat{=}} \exists \Delta' : Interval \bullet (\Delta \propto \Delta') \wedge p.\Delta'$$
$$(\boxplus p).\Delta \mathrel{\widehat{=}} \forall \Delta' : Interval \bullet \Delta' \subseteq \Delta \Rightarrow p.\Delta'$$

Thus $(prev.p).\Delta$ and $(next.p).\Delta$ hold iff p holds in some interval that immediately precedes and follows Δ, respectively and $(\boxplus p).\Delta$ holds iff p holds in each subinterval of Δ.

We assume pointwise 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 $(p_1 \wedge p_2).\Delta.s = (p_1.\Delta.s \wedge p_2.\Delta.s)$. When reasoning about programs and their properties, we must often state that if an interval predicate p_1 holds over an arbitrarily chosen interval Δ and stream s, then an interval predicate p_2 also holds over Δ and s. Hence, we define universal implication over intervals and streams as follows. Operators '$\equiv$' and '$\Leftarrow$' are similarly defined.

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

2.2 Evaluating State Predicates over an Interval

Because there are multiple states of a stream within a non-point interval, there are several possible ways of evaluating a state predicate with respect to a given interval and stream [23].

We must often determine the value of a variable at the left and right ends of an interval. Because intervals may be open/infinite at either end, these values are determined using limits. We use $\lim_{x \to a^+} f.x$ and $\lim_{x \to a^-} f.x$ to denote the limit of $f.x$ as x tends to a from above and below, respectively. To ensure that the limits are well defined, we assume all variables are piecewise continuous [20]. For a vector of variables $\mathbf{v}$, interval Δ, stream s, time t, we let $(\mathbf{v}@t).s \mathrel{\widehat{=}} (s.t).\mathbf{v}$ denote the value of $\mathbf{v}$ in state $s.t$ and define:

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

Thus, if Δ is right closed, then the value of $\overrightarrow{\mathbf{v}}$ in Δ is the value of $\mathbf{v}$ at the greatest upper bound of Δ, otherwise (i.e., Δ is right-open), the value of $\overrightarrow{\mathbf{v}}$ is the value of the $\mathbf{v}$ as it approaches $lub.\Delta$ from the right. The interpretation of $\overleftarrow{\mathbf{v}}.\Delta$ is similar. Given that $vec.c$ denotes the vector of all free variables of state predicate c, we define

$$\overleftarrow{c}.\Delta.s \mathrel{\widehat{=}} c[vec.c \setminus (\overleftarrow{vec.c}).\Delta.s] \qquad \overrightarrow{c}.\Delta.s \mathrel{\widehat{=}} c[vec.c \setminus (\overrightarrow{vec.c}).\Delta.s]$$

We must often specify properties on the actual states of a stream within an interval. Thus, we define the *always* and *sometime* operators as follows[1], where $(c@t).s \mathrel{\widehat{=}} c.(s.t)$ for any state predicate c, time t and stream s.

$$(\circledast c).\Delta \equiv \forall t{:}\, \Delta \bullet (c@t) \qquad (\odot c).\Delta \equiv \exists t{:}\, \Delta \bullet (c@t)$$

For a state predicate c, variable v and set of variables V:

$$st.v \mathrel{\widehat{=}} \forall k{:}\, Val \bullet prev.(\overrightarrow{v} = k) \Rightarrow \circledast(v = k) \qquad st.V \mathrel{\widehat{=}} \forall v{:}\, V \bullet st.v$$

Hence, a variable v is stable, denoted $st.v$, iff its value does not change from its value at the right end of some previous interval, and $st.V$ holds iff each variable in V is stable. Such definitions of stability are necessary because adjoining intervals are disjoint, and hence $prev.(\overrightarrow{v} = k)$ does not necessarily imply $\overleftarrow{v} = k$ and vice versa.

Example 1. Consider a variable x such that $(x@0) = 10$ and $(\circledast \widehat{x}).[0,2] = 1$ hold, where $\widehat{x}$ denotes the rate of change of variable x (c.f. [24]). Thus, the value of x at time 0 is 10 and the rate of change of x throughout the closed interval $[0,2]$ is 1. Then for adjoining intervals $[0,1)$ and $[1,2]$, both $(\overrightarrow{x} = 11).[0,1)$ and $(\overleftarrow{x} = 11).[1,2]$ hold. In fact, we can deduce both $(\circledast(x < 11)).[0,1)$ and $(\circledast(x \geq 11)).[1,2]$. However, for adjoining intervals $[0,1]$ and $(1,2]$, $(\circledast(x \leq 11)).[0,1]$ and $(\circledast(x > 11)).(1,2]$ hold.

[1] Our notation follows Burns and Hayes [9] and should not be confused with modal operator 'always' ($\Box$) (and 'next' ($\bigcirc$) later). Instead, we ask the reader to focus on the '$*$' within $\circledast$ (and $\circledcirc$ later), which represents "for all" and '$\cdot$' within $\odot$ (and $\odot$ later) which represents "for some" as used when writing regular expressions.

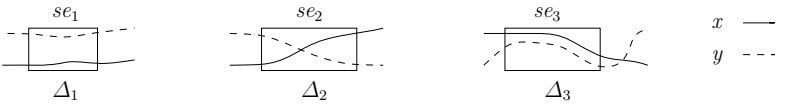

Fig. 2. Sampling events se_1, se_2 and se_3

Real-time controllers often evaluate an expression over an interval by sampling the variables of the expression (once per variable) at different instants within the interval. Hence, reasoning about an expression evaluation that samples two or more variables can be problematic. For example, consider the three sampling events se_1, se_2 and se_3 in Fig. 2, where environment variables x and y are sampled at different times within the interval. Event se_1 will return $x < y$ regardless of when x and y are read within the sampling interval because $x < y$ *definitely* holds for all sampled values of x and y. Event se_2 may return either $x > y$, $x = y$ or $x < y$ because it is *possibly* true that $x > y$, $x = y$ and $x < y$ hold. Event se_3 may have a sampling anomaly. Although $x > y$ holds throughout se_3, because x and y are sampled at different times, it is *possible* for se_3 to return either $x > y$, $x = y$ or $x < y$. If a variable occurs multiple times within an expression, the same sampled value is used for each occurrence of the variable. Hence, expression $x = x$ is guaranteed to evaluate to true regardless of how x changes within the evaluation interval, however, $x > y$ may evaluate to false even if $\boxplus(x > y)$ holds [9, 16, 23] as in se_3 in Figure 2.

We use the set of apparent states of $s \in \mathit{Stream}_V$ within interval Δ (denoted $\mathit{apparent}.\Delta.s$) to reason about sampling-based expression evaluation. We define:

$$\mathit{apparent}.\Delta.s \mathrel{\widehat{=}} \{\sigma{:}\, \Sigma_V \mid (\forall\, v{:}\, V \bullet \sigma.v \in \{t{:}\, \Delta \bullet (s.t).v\})\}$$

where $\{t{:}\, \Delta \bullet (s.t).v\}$ is equivalent to $\{x \in \mathit{Val} \mid \exists t{:}\, \Delta \bullet x = (s.t).v\}$. To generate the apparent states, we first generate $\{t{:}\, \Delta \bullet (s.t).v\}$, the set of possible values of the variables within the interval, then generate the set of all possible states using these values. 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:

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

Hence, $(\circledast c).\Delta.s$ and $(\odot c).\Delta.s$ hold iff c holds in every and in some apparent state of s within the interval Δ, respectively. For example, for Δ_1, Δ_2 and Δ_3 in Fig. 2 we can deduce both $(\circledast(x < y)).\Delta_1$ and $(\odot(x < y) \wedge \odot(x \geq y)).\Delta_2$. For se_3 (the event with a sampling anomaly), $(\odot(x \leq y)).\Delta_3$ holds, despite the fact that $\boxplus(x > y).\Delta_3$ holds. Both $\circledast c \Rrightarrow \boxplus c$ and $\boxdot c \Rrightarrow \odot c$ hold, but the converse of both implications is not necessarily true.

Lemma 2. *For any variable v and constant k, $st.v \wedge \odot(v = k) \Rrightarrow \boxplus(v = k)$.*

We let $vars.c$ denote the free variables of state predicate c. The following lemma states that if all but one variable of c is stable over an interval Δ, then c definitely holds in Δ iff c always holds in Δ and c possibly holds in Δ iff c holds sometime in Δ [23].

Lemma 3. *For any state predicate c and variable v,*
$st.(vars.c \setminus \{v\}) \Rrightarrow (\circledast c = \boxplus c) \wedge (\odot c = \boxdot c)$.

2.3 Chop and Iteration

The *chop* operator ';' is a useful basic operator in interval-based logics [29, 32]. For interval predicates p_1 and p_2 and interval Δ, we define:

$$(p_1 \,;\, p_2).\Delta \mathrel{\widehat{=}} \exists \Delta_1, \Delta_2 \colon \mathit{Interval} \bullet (\Delta = \Delta_1 \cup \Delta_2) \wedge (\Delta_1 \propto \Delta_2) \wedge p_1.\Delta_1 \wedge p_2.\Delta_2$$

Thus $(p_1 \,;\, p_2).\Delta$ holds iff Δ can be split into two adjoining intervals so that p_1 holds for the first interval and p_2 holds for the second. Unlike Moszkowski [29], we have a dense notion of time and unlike the duration calculus [32], our chop operator does not require Δ_1 and Δ_2 to be closed intervals. Thus, for x as given in Example 1, both $(\boxplus(x < 11);\, \boxplus(x \geq 11)).[0, 2]$ and $(\boxplus(x \leq 11);\, \boxplus(x > 11)).[0, 2]$ hold, but $(\boxplus(x < 11) \,;\, \boxplus(x > 11)).[0, 2]$ does not.

Using chop, we define the *weak chop* and *iterated chop* operators as follows:

$$p_1 : p_2 \mathrel{\widehat{=}} p_1 \vee (p_1 \,;\, p_2) \qquad\qquad p^\omega \mathrel{\widehat{=}} \mu\, q \bullet p : q$$

That is, $p_1 : p_2$ holds iff either p_1 holds or the given interval may be chopped so that $p_1 \,;\, p_2$ holds. The iterated chop p^ω is the least fixed point of the weak chop (which defines both finite and infinite iteration of p) assuming that predicates are ordered using universal reverse entailment ($\Lleftarrow$).

Because we have a dense notion of time, there is a possibility for an iteration p^ω to behave in a Zeno-like manner, where p iterates an infinite number of times within a finite interval. We can rule out Zeno-like behaviour in our implementations because there is a physical lower limit on the time taken to perform each iteration and hence a specification that *allows* Zeno-like behaviour can be safely ignored. However, we must be careful not to *require* Zeno-like behaviour, which would cause our specifications to become unimplementable.

Lemma 4 (ω-unfolding). $p^\omega \equiv p \vee (p \,;\, p^\omega)$

Definition 5. *We say an interval predicate p* splits *iff $p \Rrightarrow \boxplus p$ holds and* joins *iff $p^\omega \Rrightarrow p$ holds.*

For example, $\boxplus c$ both splits and joins, $\ell \leq 42$ splits but does not join, $\ell \geq 42$ joins but does not split, and $\ell = 42$ neither splits nor joins. In particular, if $(\ell \leq 42).\Delta$ holds, then $\ell \leq 42$ holds for all subintervals of Δ. On the other hand, if $(\ell \leq 42).\Delta_1$ and $(\ell \leq 42).\Delta_2$ where $\Delta_1 \propto \Delta_2$, we cannot guarantee that $(\ell \leq 42).(\Delta_1 \cup \Delta_2)$ holds. Interval predicates that join allow proofs to be decomposed more easily [16].

2.4 ILTL

To cope with durative behaviour, the traces of an action system are defined using a sequence of adjoining intervals. Thus, we use an interval-based linear temporal logic (ILTL) (as opposed to state-based [26]). If p is an interval predicate, the syntax of basic ILTL formulae is given by

$$F ::= p \mid \Box F \mid \bigcirc F \mid F_1 \,\mathcal{U}\, F_2 \mid \Pi p \mid \neg F$$

Given that $\mathrm{seq}.T$ denotes the possibly infinite sequences of type T we define

$$\mathit{AdjSeq} \mathrel{\widehat{=}} \{z{:}\,\mathrm{seq}.\mathit{Interval} \mid \forall i{:}\,\mathrm{dom}.z \setminus \{0\} \bullet z.(i-1) \propto z.i\}\,.$$

For the rest of this paper, we let z be a variable of type AdjSeq. The semantics of ILTL formulae is given below, where notation $(z,s)_i \vdash F$ states that the ILTL formula F holds for the pair (z,s) starting from index $i \in \mathrm{dom}.z$.

Definition 6. *If p is an interval predicate, F is an ILTL formula, $z \in \mathit{AdjSeq}$, s is a stream and $i \in \mathrm{dom}.z$, and $tr = (z,s)$ then:*

$$\begin{aligned}
tr \vdash F &\mathrel{\widehat{=}} tr_0 \vdash F\\
tr_i \vdash p &\mathrel{\widehat{=}} p.(z.i).s\\
tr_i \vdash \Box F &\mathrel{\widehat{=}} \forall j{:}\,\mathrm{dom}.z \bullet j \geq i \Rightarrow (tr_j \vdash F)\\
tr_i \vdash \bigcirc F &\mathrel{\widehat{=}} i+1 \in \mathrm{dom}.z \Rightarrow (tr_{i+1} \vdash F)\\
tr_i \vdash F_1 \,\mathcal{U}\, F_2 &\mathrel{\widehat{=}} \exists j{:}\,\mathrm{dom}.z \bullet j \geq i \wedge (tr_j \vdash F_2) \wedge \forall k \bullet i \leq k < j \Rightarrow (tr_k \vdash F_1)\\
tr_i \vdash \Pi p &\mathrel{\widehat{=}} p.(\textstyle\bigcup_{j \in \mathrm{dom}.z \wedge i \leq j}(z.j)).s\\
tr_i \vdash \neg F &\mathrel{\widehat{=}} \neg(tr_i \vdash F)
\end{aligned}$$

Thus, $(z,s)_i \vdash p$ holds iff p holds in s within the interval $z.i$. Operators $\Box$ and $\bigcirc$ and $\mathcal{U}$ express *always*, *next* and *until*, respectively, and $(z,s)_i \vdash \Pi p$ states that interval predicate p holds for the interval consisting of the union of all intervals in z from $z.i$ onwards. We define universal implication for temporal formulae F_1 and F_2 as follows. Both $F_1 \equiv F_2$ and $F_1 \Leftarrow F_2$ are similarly defined.

$$F_1 \Rrightarrow F_2 \mathrel{\widehat{=}} \forall z{:}\,\mathit{AdjSeq}, s \in \mathit{Stream} \bullet ((z,s) \vdash F_1) \Rightarrow ((z,s) \vdash F_2)$$

Temporal operators *eventually*, *unless* and *leads-to* are defined as follows:

$$\Diamond F \mathrel{\widehat{=}} \neg\Box\neg F \qquad F_1 \,\mathcal{W}\, F_2 \mathrel{\widehat{=}} \Box F_1 \vee (F_1 \,\mathcal{U}\, F_2) \qquad F_1 \rightsquigarrow F_2 \mathrel{\widehat{=}} \Box(F_1 \Rightarrow \Diamond F_2)$$

Systems often require that a state predicate be maintained unless another property is established.Thus, for state predicates c_1 and c_2, we define a *maintained unless* operator $\mathcal{M}$ as follows:

$$c_1 \,\mathcal{M}\, c_2 \mathrel{\widehat{=}} \Box(\mathit{prev}.\overrightarrow{c_1} \Rightarrow (\circledast c_1 \,\mathcal{W}\, \odot c_2))$$

That is, if $(z,s) \vdash c_1 \,\mathcal{M}\, c_2$ holds, then for any $i \in \mathrm{dom}.z$, if $(\mathit{prev}.\overrightarrow{c_1}).(z.i).s$, then either c_1 definitely holds for all $j \geq i$, or c_2 possibly holds in $z.k$ and c_1 definitely holds for all j such that $k > j \geq i$.

Like LTL [26], it is difficult to prove general ILTL formulae directly. However, certain forms of LTL formulae may be transformed into formulae of the form $\Box p$ (for an interval predicate p), which is simpler to prove [17].

Lemma 7. *For any interval predicate p and state predicates c_1 and c_2,*

(a) *if p joins, then $\Box p \Rrightarrow \amalg p$,*
(b) *$\amalg \boxplus p \Rrightarrow \Box p$, and*
(c) *$(c_1 \mathcal{M} c_2) \equiv \Box(prev.\overrightarrow{c_1} \Rightarrow \circledast c_1 \vee \odot c_2)$.*

3 Action Systems with Time Bands

3.1 Time Bands

Like Burns, we assume that the set of all time bands is given by the primitive type *TimeBand* [8, 9]. Each time band may be associated with *events* that execute within the *precision* of the time band. We use ρ: *TimeBand* $\to \mathbb{R}^{>0}$ to denote the precision of the given time band.

To simplify the specification of the behaviour of an event in a time band, we define the type of a *time band predicate* as $TBPred_V$: *TimeBand* $\to IntvPred_V$, which for a given time band returns an interval predicate. As with interval predicates, we assume time band predicates are lifted pointwise over boolean operators and for time band predicates tp_1 and tp_2, we define $tp_1 \Rrightarrow tp_2 \mathrel{\widehat{=}} \forall \beta$: *TimeBand* $\bullet$ $tp_1.\beta \Rrightarrow tp_2.\beta$ (and similarly $\Lleftarrow$ and $\equiv$).

We define the following interval predicates, which are useful for reasoning about sampling events, where c is a state predicate and n is a real-valued constant.

$$\circledast_n c \mathrel{\widehat{=}} (\ell \leq n) \Rightarrow \circledast c \qquad \odot_n c \mathrel{\widehat{=}} (\ell \leq n) \wedge \odot c$$

Hence, $(\circledast_n c).\Delta$ holds iff c definitely holds within Δ provided that the length of Δ is at most n. Similarly, $(\odot_n c).\Delta$ holds iff c possibly holds within Δ and the length of Δ is at most n. Note that $\neg\circledast_n c \equiv \odot_n \neg c$.

Because sampling approximates the true value of an environment variable, we must reason about how the value of a variable changes within an interval [16]. For a real-valued variable v, the maximum difference to v in stream s within Δ is given by $(\mathit{diff}.v).\Delta.s$, where:

$$(\mathit{diff}.v).\Delta.s \mathrel{\widehat{=}} \mathsf{let}\ vs = \{t{:}\,\Delta \bullet (s.t).v\}\ \mathsf{in}\ lub.vs - glb.vs$$

Note that for any real-valued variable v, $st.v \Rrightarrow (\mathit{diff}.v = 0)$.

Sampled real-valued variables in a time band β are related to their true values within an event of β using the *accuracy* of the variable in β [16]. In particular, we let $\mathsf{acc}.v \in$ *TimeBand* $\to \mathbb{R}^{\geq 0}$ denote the accuracy of variable v in a given time band. The maximum change to v within an event of time band β is an assumption on the environment. To enable this assumption to be stated more succinctly, we define a time band predicate:

$$\mathit{DIFF}.v.\beta \mathrel{\widehat{=}} \boxplus(\ell \leq \rho.\beta \Rightarrow \mathit{diff}.v \leq \mathsf{acc}.v.\beta)$$

$$\text{II}\circledast(w_1 \leq \mathit{Empty}.T_1 \vee w_2 \geq \mathit{Full}.T_2 \Rightarrow \mathit{Stopped}_1) \quad (1)$$

$$\text{II}\circledast(w_2 \leq \mathit{Empty}.T_2 \Rightarrow \mathit{Stopped}_2) \quad (2)$$

$$\text{II} \boxplus (\circledast(w_2 \leq \mathit{Reserve}.T_2 \wedge w_1 \geq \mathit{Reserve}.T_1) \wedge \ell \geq \rho.\Gamma \Rightarrow \boxdot \mathit{On}_1 \vee \mathit{next}.(\boxdot \mathit{On}_1)) \quad (3)$$

$$\text{II} \boxplus (\circledast(w_2 > \mathit{Reserve}.T_2 \wedge \mathit{Pressed}) \wedge \ell \geq \rho.\Gamma \Rightarrow \boxdot \mathit{On}_2 \vee \mathit{next}.(\boxdot \mathit{On}_2)) \quad (4)$$

$$\text{II} \boxplus (\circledast\neg \mathit{Pressed} \wedge \ell \geq \rho.\Gamma \Rightarrow \boxdot\neg \mathit{On}_2 \vee \mathit{next}.(\boxdot \mathit{On}_2)) \quad (5)$$

Fig. 3. Formalisation of the two-pump system requirements

The lemma below allows one to relate a sampled variable to its values in the environment based on its accuracy and generalises the result in [16].

Lemma 8. *If x and y are real-valued variables and $\gg \in \{\geq, >\}$ then* $\mathit{DIFF}.x \wedge \mathit{DIFF}.y \wedge \odot_\rho(x - \mathsf{acc}.x \gg y + \mathsf{acc}.y) \Rrightarrow \circledast(x \gg y)$.

Corollary 9. *If x and y are real-valued variables and $\gg \in \{\geq, >\}$ then* $\mathit{DIFF}.x \wedge \mathit{st}.y \wedge \odot_\rho(x - \mathsf{acc}.x \gg y) \Rrightarrow \boxplus(x \gg y)$.

Example 10. The informal requirements of the two-pump system in Section 1 are formalised using the ILTL formulae in Fig. 3.

Safety. We combine **S1** and **S2** as (1) and formalise **S3** as (2). By (1), over the interval in which the program is executing, in all actual (as opposed to apparent) states of the stream, if w_1 (the water level in tank T_1) is below $\mathit{Empty}.T_1$ or w_2 (the water level in tank T_2) is above $\mathit{Full}.T_2$, then pump P_1 must be stopped. Note that the consequent of (1) states that the P_1 has physically come to a stop, which we distinguish from the signal $\neg \mathit{On}_1$ that causes P_1 to stop. Condition (2) is similar.

Progress. Progress properties **P1**, **P2** and **P3** translate into properties (3), (4) and (5). Each of the progress properties involve time bands of the water. For simplicity, we assume that the time bands of the water in both tanks is Γ. Thus, condition (3) states that over the (infinite) interval corresponding to the execution of the program, in any subinterval say Δ of the interval, if it is definitely the case that w_2 is less than or equal to $\mathit{Reserve}.T_2$ and w_1 is greater than or equal to $\mathit{Reserve}.T_1$ for at least the precision of the water time band, then the pump must be turned on either within Δ or some interval that follows Δ. Conditions (4) and (5) are similar.

3.2 Actions

The syntax and semantics of actions are given in Definition 11 and Definition 12, respectively.

Definition 11. *Suppose b is a state predicate, d is a label, $\mathbf{y}$ is a vector of output variables, $\mathbf{E}$ is a vector of expressions that has the same length as $\mathbf{y}$, β is a time band, F is a set of output variables and p is an interval predicate. The abstract syntax of an* action *A is given by:*

$$A ::= d{:}\, S \mid A_1 \,[\!]\, A_2 \mid A \dagger \beta$$
$$S ::= b \to \mathsf{idle} \mid b \to \mathbf{y} := \mathbf{E} \mid b \to [\![F \cdot p]\!]$$

Definition 12. *For the syntax of actions defined in Definition 11 and a set of output variables V, the function $beh_V{:}\, A \to IntvPred$ is defined inductively as follows:*

$$beh_V.(b \to \mathsf{idle}) \mathrel{\widehat{=}} (\odot b \,;\, true) \wedge st.V \tag{6}$$
$$beh_V.(b \to \mathbf{y} := \mathbf{E}) \mathrel{\widehat{=}} (\exists \mathbf{k} \bullet (\odot(b \wedge (\mathbf{k} = \mathbf{E})) \wedge st.\mathbf{y}); \overrightarrow{\mathbf{y}} = \mathbf{k}) \wedge st.(V \backslash \mathbf{y}) \tag{7}$$
$$beh_V.(b \to [\![F \cdot p]\!]) \mathrel{\widehat{=}} (\odot b \wedge st.V) \,;\, (p \wedge st.(V \setminus F)) \tag{8}$$
$$beh_V.(d{:}\, S) \mathrel{\widehat{=}} beh_V.S \wedge \circledast(\xi = d) \tag{9}$$
$$beh_V.(A_1 \,[\!]\, A_2) \mathrel{\widehat{=}} beh_V.A_1 \vee beh_V.A_2 \tag{10}$$
$$beh_V.(A \dagger \beta) \mathrel{\widehat{=}} \ell \leq \rho.\beta \wedge beh_V.A \tag{11}$$

The primitive idle is a statement that does nothing but may take time to execute, $\mathbf{y} := \mathbf{E}$ is the *assignment* statement and $[\![F \cdot p]\!]$ is a *specification* statement. Action $d{:}\, b \to S$ is a *guarded statement* consisting of statement S with guard b and label d. Action $A_1 \,[\!]\, A_2$ consists of the non-deterministic choice between A_1 and A_2, and $A \dagger \beta$ defines an action A within time band β. Note that action $d{:}\, b \to [\![F \cdot p]\!]$ is not directly executable, but needs to be refined to an executable implementation. Further note that we do not allow nested actions, i.e., each guarded action consists of a guard followed by a statement.

We define a function *grd* and shorthand **else** as follows:

$$grd.(d{:}\, b \to S) \mathrel{\widehat{=}} b$$
$$grd.(A_1 \,[\!]\, A_2) \mathrel{\widehat{=}} grd.A_1 \vee grd.A_2$$
$$grd.(A \dagger \beta) \mathrel{\widehat{=}} grd.A$$
$$A \textbf{ else } d{:}\, S \mathrel{\widehat{=}} A \,[\!]\, (d{:}\, \neg grd.A \to S)$$

We let $labels.A$ denote the set of all labels within action A. Action $((d{:}\, b \to S) \dagger \beta) \,[\!]\, A$ is only well defined if $d \notin labels.A$, i.e., the label of each guarded statement within a non-deterministic choice is unique. If $d \in labels.A$, we let A_d denote the guarded statement labelled d in action A. We reserve a "program counter" variable ξ whose value is the label of the guarded statement that is currently executing [12].

The following lemma allows one to simplify the behaviour of guarded actions.

Lemma 13. $beh_V.((b \to S) \dagger \beta) \Rrightarrow \odot_{\rho.\beta} b$

Proof. The proof holds because $beh_V.((b \to S) \dagger \beta) \Rrightarrow \ell \leq \rho.\beta \wedge (\odot b \,;\, true)$ and $(\odot b \,;\, true) \Rrightarrow \odot b$. □

Definition 14. *If V is a set of variables, an action A is* refined *by an action C, denoted $A \sqsubseteq_V C$, iff $beh_V.C \Rrightarrow beh_V.A$ holds. If $A \sqsubseteq_V C$ and $C \sqsubseteq_V A$, we write $A \sqsupseteq_V C$.*

A refinement may reduce the non-determinism or strengthen the guard of an action.

3.3 Action Systems

Definition 15. *If I is an interval predicate and A is an action such that $grd.A$ holds,* action system $\mathscr{A} \mathrel{\widehat{=}} \mathbf{init}\ I \bullet \mathbf{do}\, A\, \mathbf{od}$ *consists of an* initial property I *followed by an infinite loop that executes action A.*

We use $in.\mathscr{A} \subseteq Var$ and $out.\mathscr{A} \subseteq Var$ to distinguish the input and output variables of action system $\mathscr{A}$ and use $vars.\mathscr{A} \mathrel{\widehat{=}} in.\mathscr{A} \cup out.\mathscr{A}$ for the variables of $\mathscr{A}$. Because we assume true concurrency between an action system $\mathscr{A}$ and its environment, we require that $in.\mathscr{A}$ and $out.\mathscr{A}$ are disjoint. We use $\mathscr{A} \dagger \beta$ as shorthand for the action system whose action executes in time band β, i.e., $\mathscr{A} \dagger \beta \mathrel{\widehat{=}} \mathbf{init}\ I \bullet \mathbf{do}\, A \dagger \beta\, \mathbf{od}$.

Execution of an action system starts in an interval for which the initial property holds for an immediately preceding interval. Then each successive interval of the trace is generated by the behaviour of some guarded action.

Definition 16. *Given $AS \mathrel{\widehat{=}} AdjSeq \times Stream$, the set of all* complete traces *of action system $\mathscr{A}$ with outputs $V \mathrel{\widehat{=}} out.\mathscr{A}$ is given by $\mathsf{Tr}.\mathscr{A}$, where:*

$$\mathsf{Tr}.\mathscr{A} \mathrel{\widehat{=}} \{(z, s) : AS \mid \mathrm{dom}.z = \mathbb{N} \wedge ((z, s) \vdash prev.I \wedge \Box(beh_V.A))\}$$

Thus, for each $(z, s) \in \mathsf{Tr}.\mathscr{A}$, it is assumed that I holds for some interval that precedes $z.0$ and some action executes in each interval $z.i$. Furthermore, because the action systems we consider are non-terminating, z is an infinite sequence.

Given any action system $\mathscr{A}$ and variable $v \in out.\mathscr{A}$, we require a healthiness condition:

$$\mathscr{A} \models \Box(prev.\overrightarrow{v} = \overleftarrow{v}) \tag{12}$$

i.e., the value of v does not change over the boundary between adjoining intervals.

Lemma 17. *If boolean variable x is an output variable, then*

$$beh_V.(x \rightarrow S) \Rrightarrow prev.\overrightarrow{x} \tag{13}$$

$$(beh_V.(x \rightarrow S) \Rrightarrow \neg\overrightarrow{x}) \Rrightarrow ((beh_V.(x \rightarrow S))^{\omega} = beh_V.(x \rightarrow S)) \tag{14}$$

Proof (13). We first show that $beh_V.(x \rightarrow S) \Rrightarrow (\odot x \wedge st.x)\,;\ true$. The proofs for $S \in \{\mathsf{idle}, [\![F \cdot p]\!]\}$ are trivial because $st.V$ splits and ';' is monotonic. For $S = (\mathbf{y} := \mathbf{E})$, we have:

$beh_V.(x \to \mathbf{y} := \mathbf{E})$
$\Rrightarrow$ definition of beh_V, $st.V$ splits, '; ' is monotonic
$\exists \mathbf{k} \bullet \odot(x \wedge st.V)\ ;\ true$
$\Rrightarrow$ logic
$(\odot x \wedge st.x)\ ;\ true$

Thus, we have:

$beh_V.(x \to S)$
$\Rrightarrow$ proof above
$(\odot x \wedge st.x)\ ;\ true$
$\Rrightarrow$ by Lemma 2 $\circledast(x = true)$ (because x is boolean) and for any c, $\circledast c \Rrightarrow \overleftarrow{c}$ $\overleftarrow{x}$
$\Rrightarrow$ healthiness condition (12)
$prev.\overrightarrow{x}$

Proof (14). $(beh_V.(x \to S))^\omega \Lleftarrow beh_V.(x \to S)$ trivially holds by Lemma 4 (ω-unfolding). Assuming $beh_V.(x \to S) \Rightarrow \neg\overrightarrow{x}$, we have

$(beh_V.(x \to S))^\omega$
$\equiv$ Lemma 4 (ω-unfolding)
$beh_V.(x \to S) \vee (beh_V.(x \to S)\ ;\ (beh_V.(x \to S))^\omega)$
$\Rrightarrow$ assumption $beh_V.(x \to S) \Rightarrow \neg\overrightarrow{x}$ and (13)
$beh_V.(x \to S) \vee (\neg\overrightarrow{x}\ ;\ (prev.\overrightarrow{x})^\omega)$
$\equiv$ $(prev.\overrightarrow{c})^\omega \Rrightarrow (prev.\overrightarrow{c})$ and $\neg(\neg\overrightarrow{c}\ ;\ prev.\overrightarrow{c})$
$beh_V.(x \to S)$ □

Definition 18. *We say that an action system $\mathscr{A}$* satisfies *an ILTL formula F (denoted $\mathscr{A} \models F$) iff $\forall tr{:}\ \mathsf{Tr}.\mathscr{A} \bullet tr \vdash F$ holds.*

To show that an action system $\mathscr{A}$ with output context V satisfies $\Box p$, one may show that, $beh_V.A \Rrightarrow p$, i.e., the execution of each guarded action of $\mathscr{A}$ satisfies p.

Theorem 19. *$\mathscr{A} \models \Box p$ if $beh_{out.\mathscr{A}}.A \Rrightarrow p$.*

Proof. The proof follows by definitions 16 and 18 and the definition of $\Box$.

Reactive systems are often structured so that a controller sends signals to the environment, then becomes idle while changes occur in the environment based on the controller signals. Using Theorem 19 to prove $\mathscr{A} \models \Box p$ can be difficult when p includes properties of the environment. Instead, we often use Theorem 21 below which allows one to consider the iterated execution of an action (usually idle) and the properties that held before the action started executing. We first prove a preliminary lemma.

Lemma 20. *If $(z, s) \in \mathsf{Tr}.\mathscr{A}$ and $i \in \mathrm{dom}.z$, there exists a $j \in \mathrm{dom}.z$, where $j \leq i$ and a $d \in label.A$ such that $(prev.(I \vee (\exists e{:}\ label.A \setminus \{d\} \bullet beh_{out.\mathscr{A}}.A_e))).(z.j).s$ and for all $j \leq k \leq i$, $(beh_{out.\mathscr{A}}.A_d).(z.k).s$.*

Proof. The proof is trivial for $i = 0$. For any $i \in \mathrm{dom}.z \setminus \{0\}$, because $(z, s) \in \mathsf{Tr}.\mathscr{A}$, there exists a $d \in \mathit{label}.\mathscr{A}$ such that $(\mathit{beh}_{\mathit{out}.\mathscr{A}}.A_d).(z.i).s$. Then, either

- $(\mathit{prev}.(I \vee (\exists e{:}\, \mathit{label}.A \setminus \{d\} \bullet \mathit{beh}_{\mathit{out}.\mathscr{A}}.A_e))).(z.i).s$ holds, in which case the proof is trivial, or
- $(\mathit{prev}.(\mathit{beh}_{\mathit{out}.\mathscr{A}}.A_d)).(z.i).s$ holds, i.e., $(\mathit{beh}_{\mathit{out}.\mathscr{A}}.A_d).(z.(i-1)).s$ holds and the proof follows by induction. □

For a label $d \in \mathit{label}.\mathscr{A}$, we define

$$\mathit{iterate}.\mathscr{A}.d \mathrel{\widehat{=}} (\mathit{beh}_{\mathit{out}.\mathscr{A}}.A_d)^{\omega} \wedge \mathit{prev}.(I \vee \exists e{:}\, \mathit{label}.\mathscr{A} \setminus \{d\} \bullet \mathit{beh}_{\mathit{out}.\mathscr{A}}.A_e)$$

which holds over an interval Δ iff the action of $\mathscr{A}$ labelled d iterates over Δ and in some interval that precedes Δ, either the initialisation I of $\mathscr{A}$ holds or some action different from e executes.

Theorem 21. *If p splits, then $\mathscr{A} \models \Box p$ holds provided that*

$$\mathscr{A} \models \mathrm{II} \boxplus (\forall d{:}\, \mathit{label}.\mathscr{A} \bullet \mathit{iterate}.\mathscr{A}.d \Rightarrow p) \tag{15}$$

Proof. By Lemma 20, for any $(z, s) \in \mathsf{Tr}.\mathscr{A}$ and $i \in \mathrm{dom}.z$,

$$\begin{array}{l} \exists j{:}\, \mathrm{dom}.z, d{:}\, \mathit{label}.A \bullet \\ \quad j \leq i \wedge (\mathit{prev}.(I \vee (\exists e{:}\, \mathit{label}.A \setminus \{d\} \bullet \mathit{beh}_{\mathit{out}.\mathscr{A}}.A_e))).(z.j).s \wedge \\ \quad (\forall k \bullet j \leq k \leq i \Rightarrow (\mathit{beh}_{\mathit{out}.\mathscr{A}}.A_d).(z.k).s) \end{array}$$

Hence,

$$(\mathit{iterate}.\mathscr{A}.d).(\textstyle\bigcup_{j \leq k \leq i} z.k).s$$

holds and therefore by (15), $p.(\bigcup_{j \leq k \leq i} z.k).s$ holds. Because p splits, $p.(z.i).s$ holds and because i was arbitrarily chosen, $(z, s) \vdash \Box p$ holds. □

Like safety properties, it is often simpler to first translate progress properties into '$\rightsquigarrow$' formulae.

Lemma 22. *For any state predicates c_1 and c_2, If $\mathscr{A} \dagger \beta \models \circledast c_1 \rightsquigarrow \boxdot c_2$ then $\mathscr{A} \dagger \beta \models \mathrm{II} \boxplus (\circledast c_1 \wedge \ell \geq 2\rho.\beta \Rightarrow \boxdot c_2 \vee \mathit{next}.\boxdot c_2)$.*

Proof. For any $(z, s) \in \mathsf{Tr}.(\mathscr{A} \dagger \beta)$, suppose $\Delta = \bigcup \mathrm{ran}.z$ and $\Delta' \subseteq \Delta$ such that $(\circledast c_1 \wedge \ell \geq 2\rho.\beta).\Delta'$. We have:

$$\begin{array}{ll} & (z, s) \in \mathsf{Tr}.(\mathscr{A} \dagger \beta) \wedge (\Delta = \bigcup \mathrm{ran}.z) \wedge (\Delta' \subseteq \Delta) \wedge (\circledast c_1 \wedge \ell \geq 2\rho.\beta).\Delta' \\ \Rightarrow & \quad \text{definition of } \mathscr{A} \dagger \beta \\ & (\forall i{:}\, \mathrm{dom}.z \bullet (\ell \leq \rho.\beta).(z.i)) \wedge (\Delta = \bigcup \mathrm{ran}.z) \wedge \\ & \qquad\qquad (\Delta' \subseteq \Delta) \wedge (\circledast c_1 \wedge \ell \geq 2\rho.\beta).\Delta' \\ \Rightarrow & \quad \text{logic} \\ & \exists i{:}\, \mathrm{dom}.z \bullet (\circledast c_1).(z.i) \wedge z.i \subseteq \Delta' \\ \Rightarrow & \quad \text{assumption } \mathscr{A} \dagger \beta \models \circledast c_1 \rightsquigarrow \boxdot c_2 \\ & \exists i{:}\, \mathrm{dom}.z \bullet z.i \subseteq \Delta' \wedge \exists j{:}\, \mathrm{dom}.z \bullet i \geq j \wedge (\boxdot c_2).(z.j) \\ \Rightarrow & \quad \text{logic} \\ & (\boxdot c_2).\Delta' \vee (\mathit{next}.(\boxdot c_2)).\Delta' \end{array}$$

□

3.4 Parallel Composition

We use $\mathscr{A} \overrightarrow{\parallel} \mathscr{B}$ to denote the parallel composition of action systems $\mathscr{A}$ and $\mathscr{B}$. For the program $\mathscr{A} \overrightarrow{\parallel} \mathscr{B}$ to be well formed, we require:

$$(in.\mathscr{A} \cup out.\mathscr{A}) \cap out.\mathscr{B} = \{\} \tag{16}$$

i.e., $\mathscr{B}$ cannot modify the inputs and outputs of $\mathscr{A}$ but the outputs of $\mathscr{A}$ may be used as inputs to $\mathscr{B}$ and furthermore $\mathscr{A}$ and $\mathscr{B}$ may share inputs. Hence, $\mathscr{A} \overrightarrow{\parallel} \mathscr{B}$ is not necessarily equivalent to $\mathscr{B} \overrightarrow{\parallel} \mathscr{A}$.

Within $\mathscr{A} \overrightarrow{\parallel} \mathscr{B}$, action systems $\mathscr{A}$ and $\mathscr{B}$ execute in a truly concurrent manner. Furthermore, $\mathscr{A}$ and $\mathscr{B}$ may execute in different time bands. Hence, the adjoining intervals defined by the execution of $\mathscr{A}$ and $\mathscr{B}$ are unrelated and we cannot use $\mathsf{Tr}.\mathscr{A}$ and $\mathsf{Tr}.\mathscr{B}$ to define the traces of $\mathscr{A} \overrightarrow{\parallel} \mathscr{B}$. Instead, we consider the whole interval over which the action systems execute.

Definition 23. *For an action system $\mathscr{A}$, interval Δ and stream s, we say $\mathscr{A}$ executes over Δ in s iff $(exec.\mathscr{A}).\Delta.s$ holds, where:*

$$(exec.\mathscr{A}).\Delta.s \mathrel{\widehat{=}} \exists z \bullet (z, s) : \mathsf{Tr}.\mathscr{A} \wedge \Delta = \bigcup \mathrm{ran}.z \tag{17}$$

Definition 24. *Suppose $\mathscr{A}$ and $\mathscr{B}$ are action systems such that (16) holds. Then,*

$$exec.(\mathscr{A} \overrightarrow{\parallel} \mathscr{B}) \mathrel{\widehat{=}} exec.\mathscr{A} \wedge exec.\mathscr{B}$$

Thus, for any interval Δ and stream s, $(exec.(\mathscr{A} \overrightarrow{\parallel} \mathscr{B})).\Delta.s$ holds iff there exist traces $(z_1, s) \in \mathsf{Tr}.\mathscr{A}$ and $(z_2, s) \in \mathsf{Tr}.\mathscr{B}$ such that $\Delta = \bigcup \mathrm{ran}.z_1 = \bigcup \mathrm{ran}.z_2$, i.e., it is possible for $\mathscr{A}$ and $\mathscr{B}$ execute in the same overall interval and stream.

A special case of parallel composition is *simple parallelism*, denoted $\mathscr{A} \| \mathscr{B}$, where no output of $\mathscr{A}$ is an input to $\mathscr{B}$ and vice versa, i.e., $\mathscr{A} \| \mathscr{B}$ is defined iff (16) $\wedge$ $(out.\mathscr{A} \cap in.\mathscr{B} = \{\})$ holds. Note that $in.\mathscr{A} \cap in.\mathscr{B}$ may be non-empty, i.e., $\mathscr{A}$ and $\mathscr{B}$ may share inputs. Unlike $\mathscr{A} \overrightarrow{\parallel} \mathscr{B}$, $\mathscr{A} \| \mathscr{B}$ is equivalent to $\mathscr{B} \| \mathscr{A}$.

4 Deriving Action System Controllers

4.1 Enforced Properties

Our derivation method uses enforced properties [12, 14], which are ILTL formulae that restrict the traces of an action system to those that satisfy the formulae. Enforced properties are temporal formulae and hence may be used to state general properties on the traces, e.g., we may formalise fairness assumptions on the scheduler [12]. We first present enforced properties on actions, which allows finer-grained control over the execution of an action system.

Definition 25. *An action A with* enforced property $p \in IntvPred$*, is an action $A \,!\, p$ and its behaviour in an output context $V \subseteq Var$ is given by $beh_V.(A \,!\, p) \mathrel{\widehat{=}} beh_V.A \wedge p$.*

init $I_1 \bullet$
do d_0: $true \rightarrow [\![On_1 \cdot true]\!]$
od $\dagger\tau_1 \,?\, SP_1 \,?\, DA.w_1 \wedge DA.w_2$

Fig. 4. Initial action system for P_1

init $I_2 \bullet$
do e_0: $true \rightarrow [\![On_2 \cdot true]\!]$
od $\dagger\tau_2 \,?\, SP_2 \,?\, DA.w_2$

Fig. 5. Initial action system for P_2

Thus, when executing $A \,!\, p$, in addition to behaving as specified by $beh_V.A$, the interval predicate p must also hold. We may represent time bands on actions using enforced properties.

Lemma 26. *For an action A, time band β and $V \subseteq Var$, $A \dagger \beta \sqsubseteq_V A!(\ell \leq \rho.\beta)$*

We obtain straightforward lemmas on actions with enforced properties [17].

Lemma 27. *For an action A, interval predicates p and q and set of variables V,*

(a) $A \,!\, (p \wedge q) \sqsubseteq_V (A \,!\, p) \,!\, q$,
(b) $A \,!\, (p \vee q) \sqsubseteq_V (A \,!\, p) \,[\!]\, (A \,!\, q)$ *and*
(c) *if* $beh_V.A \Rrightarrow p$ *then* $A \,!\, p \sqsubseteq_V A$.

Note that it is possible to enforce unimplementable behaviour, e.g., $beh_V.(A \,!\, false)$. Hence, we typically introduce or strengthen an enforced property to the weakest possible predicate to allow greater flexibility in an implementation.

We extend the concept of enforced properties on actions to enforced properties on action systems, which are specified using ILTL formula.

Definition 28. *If F is an ILTL formula then action system $\mathscr{A}$ with* enforced property *F is denoted $\mathscr{A} \,?\, F$, and its traces are given by* $\mathsf{Tr}.(\mathscr{A} \,?\, F) \mathrel{\widehat{=}} \{tr{:}\, \mathsf{Tr}.\mathscr{A} \mid tr \vdash F\}$.

Thus, although $\mathsf{Tr}.\mathscr{A}$ may contain traces that do not satisfy F, by definition, $\mathscr{A} \,?\, F$ is guaranteed to satisfy F. We have used enforced properties to develop theories of refinement, where the enforced properties are LTL formulae [12], relational LTL formulae [14] and ILTL formulae [17].

Example 29. We specify an initial action system controller for the two-pump system in Section 1. The initial actions are liberal and allow arbitrary modification of signals On_1 and On_2. However, execution of the action systems are constrained by their enforced properties, which ensure that the programs are correct with respect to the given properties. We develop the system as the simple parallel composition between the controllers for pumps P_1 and P_2, which allows the pumps to be controlled independently (see Fig. 4 and Fig. 5). We assume that the time band of pump P_i is ϕ_i and recall that the time band of the controller for P_i is τ_i. Thus, each iteration of the **do** loop of the controller for pump P_1 can be completed within an interval of length $\rho.\tau_1$ and events of P_1 take at most $\rho.\phi_1$ time (similarly for P_2). The action for the initial version of the controller of P_1 is d_0: $true \rightarrow [\![On_1 \cdot true]\!]$, where d_0 is a label, guard *true*

never blocks the action from executing and $[\![On_1 \cdot true]\!]$ allows the output On_1 to be set to true or false non-deterministically. We collate the properties on P_1 and P_2 as ILTL formulae SP_1 and SP_2, respectively:

$$SP_1 \mathrel{\widehat{=}} (1) \wedge (3)$$
$$SP_2 \mathrel{\widehat{=}} (2) \wedge (4) \wedge (5)$$

Both SP_1 and SP_2 are introduced to the program in Fig. 4 and Fig. 5 as enforced properties, which guarantees that the programs are correct with respect to these requirements. That is, although the program without the enforced properties is able to arbitrarily change the values of On_1 and On_2, by including the enforced properties, the traces of the controllers are guaranteed to satisfy the required properties SP_1 and SP_2, respectively.

Enforced properties can also be used to specify assumptions about the behaviour of the environment. Here, we use enforced properties to ensure that the accuracies of w_1 and w_2 bound the maximum possible change to w_1 and w_2 in any time band. For $i \in \{1, 2\}$, we define:

$$DA.w_i \mathrel{\widehat{=}} \Pi(DIFF.w_i.\phi_i \wedge DIFF.w_i.\tau_i)$$

Hence, the maximum difference between two values of w_1 in events of time bands ϕ_1 and τ_1 are bounded by the accuracy of w_1 in ϕ_1 and τ_1, respectively. By using Π in the formula above, we are stating that $DIFF.w_i.\phi_i \wedge DIFF.w_i.\tau_i$ holds over the whole interval in which the action systems executes.

The controllers for P_1 and P_2 are only partially developed and actions d_0 and e_0 are not yet executable. Hence, we perform a series of refinements to obtain an implementation that can be executed.

4.2 Action System Refinement

Our method of derivation allows programs to be developed in an incremental manner. In particular, we calculate the effect of (partially) developed actions on the enforced properties, which generates new properties and actions. However unlike Dijkstra [11], Feijen/van Gasteren [19] and Dongol/Mooij [18], we disallow arbitrary modifications to the program; each change must be justified using a lemma/theorem that ensures that the previous version is refined [14, 17].

Definition 30. *Action system* $\mathscr{C}$ refines $\mathscr{A}$ *(denoted* $\mathscr{A} \sqsubseteq \mathscr{C}$*) iff* $exec.\mathscr{C} \Rrightarrow exec.\mathscr{A}$.

Thus, for any interval Δ and stream s, if it is possible to execute $\mathscr{C}$ within Δ in s, then it must be possible to execute $\mathscr{A}$ within Δ in s. The lemma below provides a sufficient condition for action system refinement [17].

Lemma 31. *Suppose* $\mathscr{A} \mathrel{\widehat{=}}$ **init** I_A; **do** A **od** *and* $\mathscr{C} \mathrel{\widehat{=}}$ **init** I_C; **do** C **od**. *Then* $\mathscr{A} \sqsubseteq \mathscr{C}$ *holds if* $out.\mathscr{C} \subseteq out.\mathscr{A}$ *and* $(I_C \Rightarrow I_A) \wedge (A \sqsubseteq_{out.\mathscr{C}} C)$.

Lemma 32. *If* $\mathscr{A} \mathrel{\widehat{=}}$ **init** $I \bullet$ **do** A **od**, *then* $\mathscr{A} \sqsupseteq$ **init** $I \bullet$ **do** $A \,[\!]\, B$ **od** *if* $A \sqsubseteq_V B$.

The following lemma allows trace refinement of action systems with enforced properties [17].

Lemma 33. *For action systems $\mathscr{A}$ and $\mathscr{C}$ and ILTL formulae F and F',*

(a) *$\mathscr{A} \sqsubseteq \mathscr{A} \,?\, true$ holds,*
(b) *$\mathscr{A} \,?\, F \sqsubseteq \mathscr{A} \,?\, F'$ holds if $F' \Rrightarrow F$,*
(c) *$\mathscr{A} \,?\, F \sqsubseteq \mathscr{C} \,?\, F$ holds if $\mathscr{A} \sqsubseteq \mathscr{C}$ and*
(d) *$\mathscr{A} \,?(F \wedge F') \sqsubseteq \mathscr{A} \,?\, F$ holds if $\mathscr{A} \,?\, F \models_V F'$.*

Thus, introducing *true* or strengthening existing enforced properties results in a refinement. Furthermore, if an action system without an enforced property refines another, then the refinement holds with the enforced property included. An enforced property $F \wedge F'$ may be simplified to F if the action system $\mathscr{A} \,?\, F$ satisfies F'.

Lemma 34. *For an action system $\mathscr{A}$ and time band β, $\mathscr{A} \dagger \beta \sqsubseteq \mathscr{A} \,?\, \Box(\ell \leq \rho.\beta)$.*

Example 35. Using Lemma 22, given the following relationship between Γ and the controller time bands τ_1 and τ_2,

$$\left(\rho.\tau_1 \leq \frac{\rho.\Gamma}{2}\right) \wedge \left(\rho.\tau_2 \leq \frac{\rho.\Gamma}{2}\right) \tag{18}$$

progress properties (3), (4) and (5) are implied by leads-to properties (19), (20) and (21), respectively (see below).

$$\circledast(w_2 \leq \mathit{Reserve}.T_2 \wedge w_1 \geq \mathit{Reserve}.T_1) \leadsto \boxdot \mathit{On}_1 \tag{19}$$

$$\circledast(w_2 > \mathit{Reserve}.T_2 \wedge \mathit{Pressed}) \leadsto \boxdot \mathit{On}_2 \tag{20}$$

$$\circledast \neg \mathit{Pressed} \leadsto \boxdot \neg \mathit{On}_2 \tag{21}$$

By (19), if it is definitely the case that the water level in tank T_2 is below $\mathit{Reserve}.T_2$ and the water level in tank T_1 is above $\mathit{Reserve}.T_1$, then pump P_1 must eventually be turned on. Conditions (20) and (21) are similar.

Unlike Aichernig et al, we specify maintenance properties to ensure that the pump does not arbitrarily change its state.

M1. If P_1 has been turned on, then it must remain on unless w_1 is possibly below $\mathit{Reserve}.T_1$ or w_2 is possibly above $\mathit{Limit}.T_2$.
M2. If P_1 has been turned off, then it must remain off unless w_1 is possibly above $\mathit{Reserve}.T_1$ and w_2 is possibly below $\mathit{Limit}.T_2$.
M3. If P_2 has been turned on, then it must remain on unless w_2 is possibly below $\mathit{Reserve}.T_2$ or the button B is possibly released.
M4. If P_2 has been turned on, then it must remain off unless B is possibly pressed and w_2 is possibly above $\mathit{Reserve}.T_2$.

Note that condition **M1** must allow pump P_1 to be turned off before the water level drops to empty because by **S1**, the pump must (physically) be off if the

init I_1 •	**init** I_2 •
do d_0: $true \rightarrow [\![On_1 \cdot true]\!]$	**do** e_0: $true \rightarrow [\![On_2 \cdot true]\!]$
od $\dagger\tau_1 \,?\, SPM_1 \,?\, DA.w_1 \wedge DA.w_2$	**od** $\dagger\tau_2 \,?\, SPM_2 \,?\, DA.w_2$

Fig. 6. Refined action system for P_1 **Fig. 7.** Refined action system for P_2

water level ever reaches $Empty.T_1$. The maintenance properties are interpreted in the controller time band, e.g., within **M1**, "w_1 is possibly below $Reserve.T_1$" is interpreted as "w_1 is possibly below $Reserve.T_1$ for an event of the controller time band" (see Example 10).

Properties **M1**, **M2**, **M3** and **M4** translate directly to maintained unless properties (22), (23), (24) and (25), respectively (see below).

$$On_1 \,\mathcal{M}\, (w_1 \leq Reserve.T_1 \vee w_2 \geq Limit.T_2) \tag{22}$$

$$\neg On_1 \,\mathcal{M}\, (w_1 > Reserve.T_1 \wedge w_2 < Limit.T_2) \tag{23}$$

$$On_2 \,\mathcal{M}\, (\neg Pressed \vee w_2 \leq Reserve.T_2) \tag{24}$$

$$\neg On_2 \,\mathcal{M}\, (Pressed \wedge w_2 > Reserve.T_2) \tag{25}$$

By (22) if signal On_1 holds at the end of a preceding interval at any point during the program's execution, then On_1 must continue to hold unless there is an interval in which $w_1 \leq Reserve.T_1$ or $w_2 \geq Full.T_2$ is possibly true. Conditions (23), (24) and (25) are similar.

We define:

$$SPM_1 \mathrel{\widehat{=}} (1) \wedge (19) \wedge (22) \wedge (23)$$
$$SPM_2 \mathrel{\widehat{=}} (2) \wedge (20) \wedge (21) \wedge (24) \wedge (25)$$

Then using Lemma 33, we replace SP_1 in Fig. 4 by SPM_1 to obtain the refined action system in Fig. 6. Similarly, the action system in Fig. 5 is refined by the one in Fig. 7.

Note that using a sampling logic over intervals allows one to reason about transient behaviour and hence avoid formalisation of unimplementable specifications. We say a state predicate is *transient* in a stream if the predicate only holds for a brief (e.g., an attosecond) amount of time, whereby it is not possible to reliably detect that the predicate held [14, 17]. For example, property (20) without using a sampling logic would be stated using LTL [26] as $w > Reserve.T_2 \wedge Pressed \rightsquigarrow On_2$ [10, 14]. Such a property is unimplementable if the state predicate $w > Reserve.T_2 \wedge Pressed$ on the left of $\rightsquigarrow$ is transient (which can happen if the button is quickly pressed then released). In this paper, because we use $\circledast$ on the left of '$\rightsquigarrow$' within (19), (20) and (21), the corresponding state predicate must hold for all *apparent* states, i.e., the state predicate on the left of '$\rightsquigarrow$' is guaranteed to be detected by the controller provided that the variables are sampled within the sampling interval. For example, in (20), we can guarantee that $w_2 > Reserve.T_2 \wedge Pressed$ is detected by the controller

regardless of when the variables w_2 and *Pressed* are sampled within the sampling interval. If $\odot(w \leq \mathit{Reserve}.T_2 \vee \neg \mathit{Pressed})$ holds over a sampling interval, i.e., it is possible for the controller not to detect $w > \mathit{Reserve}.T_2 \wedge \mathit{Pressed}$, then the controller is not required to turn P_2 on.

We define trace refinement of a parallel composition of action systems as follows.

Definition 36. *For action systems $\mathscr{A}$, $\mathscr{A}'$, $\mathscr{B}$ and $\mathscr{B}'$, such that $\mathscr{A} \overrightarrow{\parallel} \mathscr{B}$ and $\mathscr{A}' \overrightarrow{\parallel} \mathscr{B}'$ are well formed, we say $\mathscr{A} \overrightarrow{\parallel} \mathscr{B}$ is* trace refined *by $\mathscr{A}' \overrightarrow{\parallel} \mathscr{B}'$ (i.e., $\mathscr{A} \overrightarrow{\parallel} \mathscr{B} \sqsubseteq \mathscr{A}' \overrightarrow{\parallel} \mathscr{B}'$) iff $\mathit{exec}.(\mathscr{A}' \overrightarrow{\parallel} \mathscr{B}') \Rrightarrow \mathit{exec}.(\mathscr{A} \overrightarrow{\parallel} \mathscr{B})$.*

Lemma 37. *For action systems $\mathscr{A}$, $\mathscr{A}'$, $\mathscr{B}$ and $\mathscr{B}'$, such that $\mathscr{A} \overrightarrow{\parallel} \mathscr{B}$ and $\mathscr{A}' \overrightarrow{\parallel} \mathscr{B}'$ are well formed, $\mathscr{A} \overrightarrow{\parallel} \mathscr{B} \sqsubseteq \mathscr{A}' \overrightarrow{\parallel} \mathscr{B}'$ holds iff both $\mathscr{A} \sqsubseteq \mathscr{A}'$ and $\mathscr{B} \sqsubseteq \mathscr{B}'$.*

5 Hardware/Software Interaction

One of the benefits of using time bands is that it simplifies reasoning about the interaction between hardware and software. Namely, we may formalise delays in turning signals on/off within a digital controller, and the effect of the signal in the physical world. In this section, we present some high-level interval predicates for expressing properties of hardware/software interaction and theorems for reasoning about such systems.

We assume that the controller sends a boolean signal *sig* to achieve a boolean effect *eff* in the environment, where *eff* occurs if *sig* holds continuously over a long enough interval (i.e., not instantaneously). There are often delays in the controller setting *sig* and in the environment reacting to *sig* to achieve *eff*. Furthermore, the time bands of the controller and environment may differ (e.g., if the environment consists of physical hardware). Thus, we define a time band predicate signal that formalises the behaviour of the controller setting the signal *sig* (in the time band of the controller) and a time band predicate effect that formalises the relationship between *sig* and *eff* (in the time band of the environment). Within the controller, we may distinguish between actions that set and maintain *sig*, where the *sig* is set to true due to a trigger c. Signal *sig* is set to true by a single action, but is maintained by the iterated execution of several "maintenance" actions (e.g., idle). Hence, given time bands β and γ, we define:

$$\mathsf{signal}(c, \mathit{sig}).\beta \mathrel{\widehat{=}} \mathsf{if}\ \mathit{prev}.\overrightarrow{\neg \mathit{sig}}\ \mathsf{then}(\ell \leq \rho.\beta \wedge \circledast c \wedge \overrightarrow{\mathit{sig}})\ \mathsf{else}\ (\mathit{prev}.\overrightarrow{c} \wedge \circledast \mathit{sig})$$

$$\mathsf{effect}(\mathit{sig}, \mathit{eff}).\gamma \mathrel{\widehat{=}} \circledast \mathit{sig} \Rightarrow \mathsf{if}\ \mathit{prev}.\overrightarrow{\mathit{eff}}\ \mathsf{then}\ \circledast \mathit{eff}\ \mathsf{else}(\ell \leq \rho.\gamma : \circledast \mathit{eff})$$

which describe relationships between the signal predicate *sig* and the corresponding effect predicate *eff*. If $\mathsf{signal}(c, \mathit{sig}).\beta$ holds, then if *sig* does not hold, then $\circledast c$ is guaranteed to hold and *sig* is guaranteed to be established within the precision of β, otherwise c must hold initially and *sig* must hold continuously in the interval. In essence, this ensures that the precision of setting *sig* to true is $\rho.\beta$. For example, the controller for P_2 sends a signal On_2 (i.e., *sig* is On_2)

and has effect $Stopped_2$ (i.e., *eff* is $Stopped_2$). The action that sets On_2 to true may guarantee a condition c that actions that maintain On_2 can rely on holding initially. We calculate c for specific problem instances using Theorem 39 below.

If $\mathsf{effect}(sig, eff).\gamma$ holds, then provided that *sig* is continuously true over an interval, *eff* continues to hold if it held at the right end of the previous interval, otherwise there is a delay of at most $\rho.\gamma$ before $\circledast eff$ holds. This models the fact that the effect takes place within the precision of the given time band.

Real-time controllers must often sample a variable in the environment and control a physical mechanism so that the effect of the mechanism occurs before the value of the variable in the environment drops below a critical level. The following lemma provides conditions that allow one to show that in all states of the stream, either the value of a variable v is above a critical level C, or the physical effect *eff* has taken place.

Lemma 38. *Suppose v is a real-valued variable, $\gg \in \{>, \geq\}$, sig and eff are state predicates, and $T, C \in \mathbb{R}$ are constants. Then*

$$(T \geq C + \mathsf{acc}.v) \wedge DIFF.v \wedge \mathsf{effect}(sig, eff) \wedge \overleftarrow{v \gg T} \wedge \circledast sig \Rrightarrow \circledast(v \gg C \vee eff)$$

Proof.
$$\begin{array}{ll} & (T \geq C + \mathsf{acc}.v) \wedge DIFF.v \wedge \mathsf{effect}(sig, eff) \wedge \overleftarrow{v \gg T} \wedge \circledast sig \\ \Rrightarrow & \text{definition of } \mathsf{effect}(sig, eff) \text{ using } \circledast sig \\ & (T \geq C + \mathsf{acc}.v) \wedge DIFF.v \wedge \overleftarrow{v \gg T} \wedge (\circledast eff \vee (\ell \leq \rho : \circledast eff)) \\ \Rrightarrow & \text{Corollary 9} \\ & \circledast eff \vee (\circledast(v \gg C) : \circledast eff) \\ \Rrightarrow & (\circledast c_1 : \circledast c_2) \Rrightarrow \circledast(c_1 \vee c_2) \\ & \circledast(v \gg C \vee eff) \end{array}$$
□

We use Lemma 38 in the proof of the theorem below, which defines a relationship between signals and their effects in the context of an action system controller. In particular, it provides conditions necessary for a property of the form $\Box\circledast(v \gg C \vee eff)$ to be established.

Theorem 39. *If v is a continuous variable, β and γ are time bands, $\gg \in \{>, \geq\}$, sig and eff are state predicates, $O \mathrel{\widehat{=}} out.\mathscr{A}$ and $T, C \in \mathbb{R}$ are constants such that $T \geq C + max(\mathsf{acc}.v.\beta, \mathsf{acc}.v.\gamma)$, then $\mathscr{A} \dagger \beta \models \Box\circledast(v \gg C \vee eff)$ holds if:*

$$\mathscr{A} \dagger \beta \models \mathrm{II} \boxplus (\mathsf{effect}(sig, eff)).\gamma \tag{26}$$

$$\mathscr{A} \dagger \beta \models \mathrm{II} \boxplus \left(\begin{array}{r} (\forall d{:}\, label.\mathscr{A} \bullet iterate.\mathscr{A}.d \Rightarrow \\ (\odot_{\rho.\beta}(v \gg C + \mathsf{acc}.v.\beta))^\omega \vee (\mathsf{signal}(v \gg T, sig)).\beta \end{array} \right) \tag{27}$$

$$\mathscr{A} \dagger \beta \models \mathrm{II}(DIFF.v.\beta \wedge DIFF.v.\gamma) \tag{28}$$

Proof. By Theorem 21, because $\circledast c$ splits, $\mathscr{A} \dagger \beta \models \Box\circledast(v \gg C \vee eff)$ holds if

$$\mathscr{A} \dagger \beta \models \mathrm{II} \boxplus (\forall d{:}\, label.\mathscr{A} \bullet iterate.\mathscr{A}.d \Rightarrow \circledast(v \gg C \vee eff))$$

Using (27) and transitivity, the condition above holds if we prove

$$\mathscr{A} \dagger \beta \models \mathrm{II} \boxplus \left(\begin{array}{r} ((\odot_{\rho.\beta}(v \gg C + \mathsf{acc}.v.\beta))^\omega \vee (\mathsf{signal}(v \gg T, sig)).\beta \Rightarrow \\ \circledast(v \gg C \vee eff) \end{array} \right) \tag{29}$$

We have

$$
\begin{array}{ll}
 & (29) \\
= & \quad \text{logic} \\
 & \mathscr{A} \dagger \beta \models \mathrm{II} \boxplus ((\odot_{\rho.\beta}(v \gg C + \mathsf{acc}.v.\beta))^{\omega} \Rightarrow \circledast(v \gg C \vee \mathit{eff})) \wedge \\
 & \mathscr{A} \dagger \beta \models \mathrm{II} \boxplus ((\mathsf{signal}(v \gg T, \mathit{sig})).\beta \Rightarrow \circledast(v \gg C \vee \mathit{eff})) \\
\Leftarrow & \quad \text{Corollary 9 and (28)} \\
 & \mathscr{A} \dagger \beta \models \mathrm{II} \boxplus ((\mathsf{signal}(v \gg T, \mathit{sig})).\beta \Rightarrow \circledast(v \gg C \vee \mathit{eff})) \\
= & \quad \mathit{sig} \text{ is a boolean, logic} \\
 & \mathscr{A} \dagger \beta \models \mathrm{II} \boxplus ((\mathsf{signal}(v \gg T, \mathit{sig})).\beta \wedge \mathit{prev}.\overrightarrow{\mathit{sig}} \Rightarrow \circledast(v \gg C \vee \mathit{eff})) \wedge \\
 & \mathscr{A} \dagger \beta \models \mathrm{II} \boxplus ((\mathsf{signal}(v \gg T, \mathit{sig})).\beta \wedge \mathit{prev}.\overrightarrow{\neg \mathit{sig}} \Rightarrow \circledast(v \gg C \vee \mathit{eff})) \\
\Leftarrow & \quad \text{definition of } \mathsf{signal} \\
 & \mathscr{A} \dagger \beta \models \mathrm{II} \boxplus (\mathit{prev}.(\overrightarrow{v} \gg T) \wedge \circledast \mathit{sig} \Rightarrow \circledast(v \gg C \vee \mathit{eff})) \wedge \\
 & \mathscr{A} \dagger \beta \models \mathrm{II} \boxplus (\ell \leq \rho.\beta \wedge \circledast(v \gg T) \wedge \overrightarrow{\mathit{sig}} \Rightarrow \circledast(v \gg C \vee \mathit{eff})) \\
\Leftarrow & \quad \text{first conjunct: (26), (28) and } T \geq C + \mathit{max}(\mathsf{acc}.v.\beta, \mathsf{acc}.v.\gamma) \\
 & \quad \text{second conjunct: } \circledast(v \gg T) \Rightarrow \circledast(v \gg C) \\
 & \mathit{true}
\end{array}
$$

□

By (26) the effect predicate holds between *sig* and *eff* within time band γ. By (27), either it is possible to sample that the value of v is above $C + \mathsf{acc}.v.\beta$ in each sampling interval, or *sig* is set to true. By (28) the difference between two values of v within events of time bands β and γ does not exceed the accuracy of v in β and γ, respectively. Conditions (26) and (28) are usually properties of the environment of the action system and hence are introduced as enforced properties. On the other hand, properties such as (27) must be guaranteed by the actions of the action system.

Example 40. We demonstrate the use of Theorem 39 and derive the necessary conditions on the control signal On_2 and the state of pump P_2 to prove (2). Because $\circledast c$ joins for any state predicate c, using Lemma 7, condition (2) holds if

$$\Box\circledast(w_2 \leq \mathit{Empty}.T_2 \Rightarrow \mathit{Stopped}_2)$$

which by logic is equivalent to

$$\Box\circledast(w_2 > \mathit{Empty}.T_2 \vee \mathit{Stopped}_2)$$

Using Theorem 39 this holds if for some constant T, (30) $\wedge$ (31) $\wedge$ $\Box$(32) holds (see Fig. 8). Condition (28) is satisfied by enforced property DA_{w_2} in the program in Fig. 7. Condition (30) establishes a relationship between T and $\mathit{Empty}.T_2$ based on the accuracy of the water in time bands τ_2 and ϕ_2. By (31), in any subinterval of the action system's execution, the pump must stop if the length of the subinterval is $\rho.\phi_2$ or greater and signal $\neg On_2$ holds continuously. Condition (32) states that either it is possible to iteratively sample $w_2 > \mathit{Empty}.T_2 + \mathsf{acc}.w_2.\tau_2$ or the signal predicate holds, which ensures that $\neg On_2$ holds within an interval of length $\rho.\tau_2$ and $\neg On_2$ is maintained if it already holds.

$$T \geq Empty.T_2 \ + \ max(\mathsf{acc}.w_2.\tau_2, \mathsf{acc}.w_2.\phi_2) \tag{30}$$

$$\amalg \boxplus (\mathsf{effect}(\neg On_2, Stopped_2)).\phi_2 \tag{31}$$

$$\amalg \boxplus \left(\begin{array}{rl} \forall d{:}\, label.\mathscr{A} \bullet iterate.\mathscr{A}.d & \Rrightarrow \ (\odot_{\rho.\tau_2}(w_2 > Empty.T_2 + \mathsf{acc}.w_2.\tau_2)) \vee \\ & \quad (\mathsf{signal}(w_2 \geq T, \neg On_2)).\tau_2 \end{array} \right) \tag{32}$$

$$\circledast(w_2 > Reserve.T_2 \wedge Pressed) \ \Rightarrow \ \overrightarrow{On_2} \tag{33}$$

$$\circledast \neg Pressed \ \Rightarrow \ \overrightarrow{\neg On_2} \tag{34}$$

$$prev.\overrightarrow{On_2} \Rightarrow \circledast On_2 \ \vee \ \odot(\neg Pressed \vee w_2 \leq Reserve.T_2) \tag{35}$$

$$prev.\overrightarrow{\neg On_2} \Rightarrow \circledast \neg On_2 \ \vee \ \odot(Pressed \wedge w_2 > Reserve.T_2) \tag{36}$$

Fig. 8. Transformed properties

6 Example: Two-Tank Pump System

Due to Lemma 37, we may refine the system by refining the controllers of each pump separately. In this paper, we focus on the controller for pump P_2; the controller for P_1 may be derived in a similar manner.

6.1 Formulae Transformation

We first transform the general ILTL formula SPM_2, into formulae with conjuncts of the form $\Box p$ where p is an interval predicate [14, 17]. We may prove that the program satisfies $\Box p$ using Theorem 21.

Safety. The transformation to satisfy safety condition (2) is given in Example 40.

Progress. We may ensure the right hand side of $\rightsquigarrow$ in (20) and (21) holds immediately, i.e., without any intermediate intervals. Hence, we obtain (33) and (34), where (20) and (21) hold if $\Box$(33) and $\Box$(34) hold, respectively. By (33) if it is definitely the case that the water in tank T_2 is above $Reserve.T_2$ and the button is pressed, then signal On_2 must be set to true. Condition (34) is similar.

Maintenance. Using Lemma 7, (24) and (25) hold if $\Box$(35) and $\Box$(36) hold, respectively. By (35), if On_2 holds at the end of the previous interval, then it must hold throughout the current interval, or it must be possible detect that $Pressed$ does not hold or the water in tank T_2 is below $Reserve.T_2$. Condition (36) is similar. We distinguish between the properties of the controller of P_2 and those of the its environment and define:

$$RefSPM_2 \mathrel{\widehat{=}} (32) \wedge (33) \wedge (34) \wedge (35) \wedge (36)$$

Using Lemma 33, we replace enforced property in SPM_2 within the controller for P_2 in Fig. 7 by $\Box RefSPM_2 \wedge$ (31) and we obtain the program in Fig. 9.

do e_0: $true \rightarrow [\![On_2 \cdot true]\!]$
od $\dagger\tau_2 \,?(\Box RefSPM_2 \wedge (31))\,?\, DA.w_2$

Fig. 9. Replace SPM_2

6.2 Action Calculation

Using Theorem 21, for each action a that we introduce, we check that $(beh_V.a)^\omega \Rrightarrow RefSPM_2$ holds.

Turning Pump P_2 Off. The controller achieves this by setting the output signal On_2 to false under a guard b_1 and enforced property p_1. Thus, we check the effect of action $(e_1\colon b_1 \rightarrow On_2 := false)\,!\,p_1$, where e_1 is a fresh label, b_1 is a state predicate and p_1 is an interval predicate. Both b_1 and p_1 are instantiated below when calculating the conditions for e_1 to establish $RefSPM_2$. To prove (32), we assert $b_1 \Rightarrow On_2$ and obtain:

$(beh_V.(e_1\,!\,p_1))^\omega \Rrightarrow (32)$
$\Leftarrow$ (14) of Lemma 17 because $b_1 \Rightarrow On_2$
$beh_V.(e_1\,!\,p_1) \Rrightarrow (32)$
$\Leftarrow$ (13) of Lemma 17 using $b_1 \Rightarrow On_2$, strengthen consequent
$beh_V.e_1 \wedge p_1 \wedge prev.\overrightarrow{On_2} \Rrightarrow (\mathsf{signal}(w_2 \geq T, \neg On_2)).\tau_2$
$\Leftarrow$ definition of signal, use $prev.\overrightarrow{On_2}$
$beh_V.e_1 \wedge p_1 \Rrightarrow \ell \leq \rho.\tau_2 \wedge \circledast(w_2 \geq T) \wedge \overrightarrow{\neg On_2}$
$\Leftarrow$ definition of beh_V, $\ell \leq \rho.\tau_2$ is implicit by $\dagger\tau_2$
$\overrightarrow{\neg On_2} \wedge p_1 \Rrightarrow \circledast(w_2 \geq T) \wedge \overrightarrow{\neg On_2}$
$\Leftarrow$ logic, weaken antecedent
$p_1 \Rrightarrow \circledast(w_2 \geq T)$

By asserting $b_1 \Rightarrow w_2 \leq Reserve.T_2 \vee \neg Pressed$, condition (33) may be discharged using Lemma 13. Condition (34) is trivial because $beh_V.e_1 \Rrightarrow \overrightarrow{\neg On_2}$. Condition (35) is trivial by $b_1 \Rightarrow w_2 \leq Reserve.T_2 \vee \neg Pressed$ from above and Lemma 13, $\odot(w_2 \leq Reserve.T_2 \vee \neg Pressed)$ holds. Condition (36) holds because $b_1 \Rightarrow On_2$ holds, which by (13) of Lemma 17 implies $prev.\overrightarrow{\neg On_2}$. Thus, our derived action is

$$(e_1\colon On_2 \wedge (\neg Pressed \vee w_2 \leq Reserve.T_2) \rightarrow On_2 := false)\,!\,\circledast(w_2 \geq T)$$

This action contains an enforced property $\circledast(w_2 \geq T)$, which we discharge at a later stage of the derivation.

Turning Pump P_2 On. As with action e_1 above, the template for turning the pump on is $(e_2\colon b_2 \rightarrow On_2 := true)\,!\,p_2$. Because the calculations for e_2 to satisfy $RefSPM_2$ are similar to those for e_1 above, we only briefly describe the necessary modifications. For (32), we assert $b_2 \Rightarrow w_2 > Empty.T_2 + \mathsf{acc}.w_2.\tau_2$, condition (33) is trivially discharged because $\overrightarrow{On_2}$ holds, condition (34) is satisfied by asserting $b_2 \Rightarrow Pressed$, which by Lemma 13 ensures $\odot Pressed$ and (35) is

satisfied by asserting $b_2 \Rightarrow \neg On_2$, which by (13) of Lemma 17 implies $prev.\overrightarrow{On_2}$. Finally, for (36), we assert $b_2 \Rightarrow w_2 > Reserve.T_2$ (because $b_2 \Rightarrow Pressed$ already holds) and by assuming

$$Reserve.T_2 \geq Empty.T_2 + \mathsf{acc}.w_2.\tau_2 \tag{37}$$

we obtain action:

$$e_2{:}\,\neg On_2 \wedge Pressed \wedge w_2 > Reserve.T_2 \rightarrow On_2 := true$$

idle Action. Because we are developing a reactive system, **idle** is executed when both e_1 and e_2 are disabled, i.e., when $\neg grd.e_1 \wedge \neg grd.e_2$ holds. We may simplify the (iterated) behaviour of the **idle** action as follows:

$$\begin{array}{ll}
 & (beh_V.(e_3{:}\,\neg grd.e_1 \wedge \neg grd.e_2 \rightarrow \textbf{idle}))^\omega \\
\Rrightarrow & \quad \text{definition of } beh_V,\ (p_1 \wedge p_2)^\omega \Rightarrow p_1^\omega \wedge p_2^\omega \\
 & \left(\odot_{\rho.\tau_2} \begin{pmatrix} (\neg On_2 \vee (w_2 > Reserve.T_2 \wedge Pressed) \\ \wedge\ (w_2 \leq Reserve.T_2 \vee On_2 \vee \neg Pressed) \end{pmatrix}\right)^\omega \wedge (st.On_2)^\omega \\
\Rrightarrow & \quad (st.On_2)^\omega \equiv st.On_2 \text{ and case analysis on } prev.\overrightarrow{On_2}, \text{ use } st.On_2, \\
 & \text{then simplify} \\
 & \left(\left(\odot_{\rho.\tau_2}\left(w_2 > Reserve.T_2 \wedge Pressed\right)\right)^\omega \wedge prev.\overrightarrow{On_2} \wedge \circledast On_2\right) \vee \\
 & \left(\left(\odot_{\rho.\tau_2}\left(w_2 \leq Reserve.T_2 \vee \neg Pressed\right)\right)^\omega \wedge prev.\overrightarrow{\neg On_2} \wedge \circledast \neg On_2\right)
\end{array}$$

Hence, we can prove that $(beh_V.(e_3{:}\,\neg grd.e_1 \wedge \neg grd.e_2 \rightarrow \textbf{idle}))^\omega$ satisfies $RefSPM_2$ by proving that both of the interval predicates below satisfy $RefSPM_2$:

$$(\odot_{\rho.\tau_2}(w_2 > Reserve.T_2 \wedge Pressed))^\omega \wedge prev.\overrightarrow{On_2} \wedge \circledast On_2 \tag{38}$$

$$(\odot_{\rho.\tau_2}(w_2 \leq Reserve.T_2 \vee \neg Pressed))^\omega \wedge prev.\overrightarrow{\neg On_2} \wedge \circledast \neg On_2 \tag{39}$$

Both (38) and (39) trivially satisfy (33), (34), (35) and (36). Condition (38) trivially satisfies (32) because by (37), $Reserve.T_2 \geq Empty.T_2 + \mathsf{acc}.w_2.\tau_2$ holds. To show that (39) satisfies (32), because (39) implies that $\circledast \neg On_2$ holds, we use Lemma 33 to introduce the following enforced property to the program:

$$\Box(\circledast(\xi = e_3) \wedge prev.(\overrightarrow{\neg On_2}) \Rightarrow prev.(\overrightarrow{w_2 \geq T})) \tag{40}$$

which states that if e_3 is being executed and signal On_2 is disabled at the end of the preceding interval, then the water level in tank T_2 must be above T at the end of the preceding interval. This ensures that (39) satisfies $(\mathsf{signal}(w_2 \geq T, \neg On_2)).\tau_2$.

We prove (40) using Theorem 21, which gives us the following proof obligation:

$$\begin{array}{r}
(beh_V.e_3)^\omega \wedge \circledast(\xi = e_3) \wedge \\
prev.(\overrightarrow{I} \vee (\exists k{:}\,\{e_1, e_2\} \bullet beh_V.k))
\end{array} \Rrightarrow prev.(\overrightarrow{On_2 \vee w_2 \geq T}) \tag{41}$$

For $\overrightarrow{I}$, we require that $I \Rightarrow w_2 \geq T$, for $k = e_1$, the proof holds by enforced property $\overrightarrow{w_2 \geq T}$ in e_1 and for $k = e_2$ the proof holds because $beh_V.e_2$ implies $\overrightarrow{On_2}$.

$$
\begin{array}{ll}
\textbf{do}\ e_0\colon & \mathit{true} \rightarrow [\![\mathit{On}_2 \cdot \mathit{true}]\!] \\
[\!]\ e_1\colon & \mathit{On}_2 \wedge (\neg \mathit{Pressed} \vee w_2 \leq \mathit{Reserve}.T_2) \rightarrow \mathit{On}_2 := \mathit{false}\,!\,\circledast(w_2 \geq T) \\
[\!]\ e_2\colon & \neg \mathit{On}_2 \wedge \mathit{Pressed} \wedge w_2 > \mathit{Reserve}.T_2 \rightarrow \mathit{On}_2 := \mathit{true} \\
[\!]\ e_3\colon & \neg \mathit{grd}.e_1 \wedge \neg \mathit{grd}.e_2 \rightarrow \textbf{idle} \\
\textbf{od}\ \dagger\tau_2\ ?((31) \wedge \mathit{DA}.w_2) &
\end{array}
$$

Fig. 10. Introduce actions

We introduce actions e_1, e_2 and e_3 to the program using Lemma 32, which gives us the program in Fig. 10. Note that we do not remove e_0 at this stage to enable future modifications to the actions to be made more easily.

6.3 Discharge Enforced Properties on Actions

We now make modifications to remove the enforced property within e_1 in Fig. 10. We refrain from modifying the guards to avoid reproving conditions that have already been established. Instead, noting that $\mathit{beh}_V.e_1 \Rrightarrow \mathit{prev}.\overrightarrow{On_2}$ (by (13) of Lemma 17) and that $\mathit{prev}.(\overrightarrow{w_2 \geq T + \mathsf{acc}.w_2.\tau_2}) \wedge \ell \leq \rho.\tau_2 \Rrightarrow \circledast(w_2 \geq T)$ (by Corollary 9), $\circledast(w_2 \geq T)$ within e_1 holds if the program satisfies

$$\Box(\overrightarrow{On_2} \Rightarrow \odot(w_2 \geq T + \mathsf{acc}.w_2.\tau_2)) \tag{42}$$

Hence, using Lemma 33, we introduce (42) as an enforced property to the program. Condition (42) and Lemma 27 allows us to remove the enforced property $\circledast(w_2 \geq T)$ within e_1, however, we must now consider the changes necessary ensure it holds.

We use Theorem 21, which allows (42) to be proved by showing that for each $i \in \{1, 2, 3\}$, if On_2 holds at the end of the interval corresponding the execution of e_i, then $w_2 \geq T + \mathsf{acc}.w_2.\tau_2$ must also hold. Action e_1 is trivial because $\mathit{beh}_V.e_1 \Rrightarrow \overrightarrow{\neg On_2}$ holds. For e_2, we let

$$\mathit{Reserve}.T_2 \geq T + 2\mathsf{acc}.w_2.\tau_2 \tag{43}$$

which by Corollary 9 ensures $\mathit{beh}_V.e_2 \Rrightarrow \odot(w_2 \geq T + \mathsf{acc}.w_2.\tau_2)$. For e_3, we have:

$$
\begin{array}{ll}
 & (38) \vee (39) \Rrightarrow \overrightarrow{\neg On_2} \vee \odot(w_2 \geq T + \mathsf{acc}.w_2.\tau_2) \\
\Leftarrow & \quad (39) \Rrightarrow \overrightarrow{\neg On_2}, \text{ strengthen consequent} \\
 & (38) \Rrightarrow \odot(w_2 \geq T + \mathsf{acc}.w_2.\tau_2) \\
\Leftarrow & \quad \text{Corollary 9} \\
 & (43)
\end{array}
$$

Because $\mathit{grd}.e_1 \vee \mathit{grd}.e_2 \vee \mathit{grd}.e_3$ holds, we may use Lemma 31 to strengthen the guard of e_0 to *false* without strengthening the guard of the action system. Then using Lemma 32, we may remove action e_0 from the program. Thus, we obtain the final controller in Fig. 11.

$$
\begin{array}{ll}
\textbf{do} & e_1\colon On_2 \wedge (\neg Pressed \vee w_2 \leq Reserve.T_2) \rightarrow On_2 := false \\
[\!] & e_2\colon \ \neg On_2 \wedge Pressed \wedge w_2 > Reserve.T_2 \rightarrow On_2 := true \\
\textbf{else} & e_3\colon \textbf{idle} \\
\textbf{od} & \dagger\tau_2\,?((31) \wedge DA.w_2)
\end{array}
$$

Fig. 11. Final controller

The program in Fig. 11, behaves in time band τ_2 and the environment operates as specified by (31) and $DA.w_2$. Property (31) ensures that $Stopped_2$ holds within an interval of length $\rho.\phi_2$ in which $\circledast\neg On_2$ holds and $DA.w_2$ ensures that that maximum difference of w_2 in any time bands τ_2 and ϕ_2 are within the accuracies of w_2 in τ_2 and ϕ_2, respectively. Combining (30), (37) and (43), we derive a necessary relationship:

$$Reserve.T_2 \geq Empty.T_2 + max(\mathsf{acc}.w_2.\tau_2, \mathsf{acc}.w_2.\phi_2) + 2\mathsf{acc}.w_2.\tau_2$$

on the values of $Reserve.T_2$ and $Empty.T_2$.

7 Conclusions

This paper incorporates a time bands theory [9] into action systems and we develop an interval-based semantics for reasoning about sampling over continuous environments. We use ILTL [17], a temporal logic for sequences of adjoining intervals, and develop a refinement theory using enforced properties specified by ILTL formulae. We have developed high-level methods that use time bands to simplify reasoning about hardware/software interaction. As an example, we have derived an action systems controller for a real-time pump. Notable in our derivation is the development of side conditions that formalise the assumptions on the environment and the derivation of relationships between threshold and critical levels based on the (different) time bands of the controller and pump.

As part of future work, we aim to further develop the theories for parallel composition of action systems by developing (compositional) rely/guarantee-style methods. We also aim to explore the links between action systems and teleo-reactive programs [15, 16]. In particular it will be interesting to consider a development method that starts with a teleo-reactive program (whose semantics are closer to abstract specifications) and refining the teleo-reactive program to an action system.

Acknowledgements. This research is supported by Australian Research Council Discovery Grant DP0987452 and EPSRC Grant EP/J003727/1. We thank our anonymous reviewers for their insightful comments that have helped improve this paper.

References

1. Aichernig, B.K., Brandl, H., Krenn, W.: Qualitative Action Systems. In: Breitman, K., Cavalcanti, A. (eds.) ICFEM 2009. LNCS, vol. 5885, pp. 206–225. Springer, Heidelberg (2009)
2. Back, R.-J., Petre, L., Porres, I.: Generalizing Action Systems to Hybrid Systems. In: Joseph, M. (ed.) FTRTFT 2000. LNCS, vol. 1926, pp. 202–213. Springer, Heidelberg (2000)
3. Back, R.-J.R., Sere, K.: Stepwise refinement of action systems. Structured Programming 12(1), 17–30 (1991)
4. Back, R.-J.R., von Wright, J.: Trace Refinement of Action Systems. In: Jonsson, B., Parrow, J. (eds.) CONCUR 1994. LNCS, vol. 836, pp. 367–384. Springer, Heidelberg (1994)
5. Back, R.-J.R., von Wright, J.: Refinement Calculus: A Systematic Introduction. Springer-Verlag New York, Inc., Secaucus (1998)
6. Back, R.-J.R., von Wright, J.: Compositional action system refinement. Formal Asp. Comput. 15(2-3), 103–117 (2003)
7. Broy, M.: Refinement of time. Theor. Comput. Sci. 253(1), 3–26 (2001)
8. Burns, A., Baxter, G.: Time bands in systems structure. In: Structure for Dependability: Computer-Based Systems from an Interdisciplinary Perspective, pp. 74–88. Springer (2006)
9. Burns, A., Hayes, I.J.: A timeband framework for modelling real-time systems. Real-Time Systems 45(1), 106–142 (2010)
10. Chandy, K.M., Misra, J.: Parallel Program Design: A Foundation. Addison-Wesley Longman Publishing Co., Inc. (1988)
11. Dijkstra, E.W.: Guarded commands, nondeterminacy and formal derivation of programs. Commun. ACM 18(8), 453–457 (1975)
12. Dongol, B.: Progress-based verification and derivation of concurrent programs. PhD thesis, The University of Queensland (2009)
13. Dongol, B., Hayes, I.J.: Enforcing safety and progress properties: An approach to concurrent program derivation. In: 20th ASWEC, pp. 3–12. IEEE Computer Society (2009)
14. Dongol, B., Hayes, I.J.: Compositional Action System Derivation Using Enforced Properties. In: Bolduc, C., Desharnais, J., Ktari, B. (eds.) MPC 2010. LNCS, vol. 6120, pp. 119–139. Springer, Heidelberg (2010)
15. Dongol, B., Hayes, I.J.: Reasoning about teleo-reactive programs under parallel composition. Technical Report SSE-2011-01, The University of Queensland (April 2011)
16. Dongol, B., Hayes, I.J.: Approximating idealised real-time specifications using time bands. In: AVoCS 2011. ECEASST, vol. 46, pp. 1–16. EASST (2012)
17. Dongol, B., Hayes, I.J.: Deriving real-time action systems in a sampling logic. Sci. Comput. Program. (2012); accepted October 17, 2011
18. Dongol, B., Mooij, A.J.: Streamlining progress-based derivations of concurrent programs. Formal Aspects of Computing 20(2), 141–160 (2008)
19. Feijen, W.H.J., van Gasteren, A.J.M.: On a Method of Multiprogramming. Springer (1999)
20. Gargantini, A., Morzenti, A.: Automated deductive requirements analysis of critical systems. ACM Trans. Softw. Eng. Methodol. 10, 255–307 (2001)
21. Guelev, D.P., Hung, D.V.: Prefix and projection onto state in duration calculus. Electr. Notes Theor. Comput. Sci. 65(6), 101–119 (2002)

22. Gupta, V., Henzinger, T.A., Jagadeesan, R.: Robust Timed Automata. In: Maler, O. (ed.) HART 1997. LNCS, vol. 1201, pp. 331–345. Springer, Heidelberg (1997)
23. Hayes, I.J., Burns, A., Dongol, B., Jones, C.B.: Comparing models of nondeterministic expression evaluation. Technical Report CS-TR-1273, Newcastle University (2011)
24. Henzinger, T.A.: The theory of hybrid automata. In: LICS 1996, pp. 278–292. IEEE Computer Society, Washington, DC (1996)
25. Henzinger, T.A., Qadeer, S., Rajamani, S.K.: Assume-Guarantee Refinement Between Different Time Scales. In: Halbwachs, N., Peled, D.A. (eds.) CAV 1999. LNCS, vol. 1633, pp. 208–221. Springer, Heidelberg (1999)
26. Manna, Z., Pnueli, A.: Temporal Verification of Reactive and Concurrent Systems: Specification. Springer-Verlag New York, Inc. (1992)
27. Meinicke, L.A., Hayes, I.J.: Continuous Action System Refinement. In: Yu, H.-J. (ed.) MPC 2006. LNCS, vol. 4014, pp. 316–337. Springer, Heidelberg (2006)
28. Moszkowski, B.C.: Compositional reasoning about projected and infinite time. In: ICECCS, pp. 238–245. IEEE Computer Society (1995)
29. Moszkowski, B.C.: A complete axiomatization of interval temporal logic with infinite time. In: LICS, pp. 241–252 (2000)
30. Rönkkö, M., Ravn, A.P., Sere, K.: Hybrid action systems. Theoretical Computer Science 290(1), 937–973 (2003)
31. Wulf, M., Doyen, L., Markey, N., Raskin, J.-F.: Robust safety of timed automata. Form. Methods Syst. Des. 33, 45–84 (2008)
32. Zhou, C., Hansen, M.R.: Duration Calculus: A Formal Approach to Real-Time Systems. EATCS: Monographs in Theoretical Computer Science. Springer (2004)