Modelling and Refining Hybrid Systems in Event-B and Rodin

Michael Butler, University of Southampton
Jean-Raymond Abrial, Independent consultant
Richard Banach, University of Manchester

April 13, 2015

Abstract

We outline an approach to modelling and reasoning about hybrid systems with the Event-B method supported by the Rodin toolset. The approach uses continuous functions over real intervals to model the evolution of continuous values over time. Nondeterministic interval events are used to specify how continuous variables evolve within an operating mode. Refinement is used to constrain the choice of continuous functions and to decompose a non-deterministic interval event into a series of periodic interval events.

1 Introduction

Event-B is an established formalism that has been applied to a range of systems especially control systems and distributed systems [1]. A key feature of Event-B is the use of abstract modelling to represent the *purpose* of a system and the use of refinement to demonstrate conformance between the abstract models and more detailed models representing the designs that are intended to achieve the desired purpose. An Event-B machine represents a single level of abstraction and consists of state variables, invariants and events (i.e., parameterised guarded atomic actions). The application of Event-B is enabled by the Rodin toolset [2] which provides capabilities for proof obligations generation and automated and interactive proof capabilities.

Event-B has largely been used to represent and reason about discrete models. In this chapter we are interested in hybrid models, that is, models containing a mix of discrete and continuous behaviour. We focus on one kind of continuous property, namely, bounds on a continuous function. A common approach in modelling a hybrid system is to identify a number of discrete modes such that within each mode the evolution of continuous variables is specified through dynamic control laws (e.g., differential equations) [10, 12, 13]. We follow such an approach here though we abstract away from control equations, instead specifying assumptions about continuous functions; for example, we might assume that

a continuous function is monotonically increasing over an interval. At the abstract level, an atomic event is used to specify the continuous behaviour within a mode; such an event nondeterministically chooses a continuous function over a time interval representing the continuous behaviour within that mode. We refer to this as an *interval* event. We use refinement for two purposes. The first is to make the choice of the continuous function more constrained (reduction of nondeterminism). The second use of refinement is to introduce additional discrete steps within a mode to represent a periodic control strategy that determines, at each period, whether to remain in a mode or switch to a different mode.

We illustrate the approach we have adopted using the classic example of a controller for a water tank. The purpose of the water tank controller is to maintain the water level in the tank between a low and a high level. This is specified through bounds on a continuous function representing the evolution of the water level within a mode. In a first refinement we constrain the choice of continuous function further, specifying that it is monotonically increasing (or monotonically decreasing) within a mode. In a second refinement, we introduce periodic control events within the modes.

One objective of the work outlined here was the desire to follow an abstraction/refinement approach, starting with a model of the purpose and refining this towards an implementation strategy. An additional key aim was to understand the extent to which the existing Event-B refinement concepts and the Rodin toolset could be used to achieve the modelling and proofs of the hybrid water tank. A key enabler for mechanising the modelling and proofs in Rodin is the Theory Plug-in for Rodin [7] . The Theory Plug-in allows us to extend the mathematical language of Event-B with theories of real numbers and continuous functions over intervals. The Theory Plug-in also allows us to define new proof rules about reals and continuous functions that can be used by the Rodin proof manager.

The work presented here uses the standard refinement proof obligations of Event-B so that no changes where required in the Rodin tool. The Event-B proof rules are defined in [1]. Proving refinement in Event-B requires gluing invariants that relate abstract and concrete variables. Each event of a refining machine either refines an event of the abstract machine or refines *skip*. For an event that refines an abstract event, the guards of that refined event must entail the guards of the abstract event under the gluing invariants and the actions must maintain the gluing invariants. An event refines *skip* if it maintains the gluing invariants when no change occurs in the abstract variables.

The approach presented here was inspired by previous work on reasoning about hybrid systems using an abstraction/refinement approach. Continuous Action Systems [4] are an extension of the classical Action System approach [3] where variables are time dependent functions, that is, variables are functions over non-negative reals. Hybrid Action Systems [13] provide the ability to specify evolution of continous functions using differential equations. Inspired by Continuous Action Systems, Su et al [14] have developed an approach to modelling hybrid systems in Event-B using a combination of discrete and time varying variables. The approach followed in [14] is to start with discrete models

and introduce continuous behaviour in refinement steps. When mechanising the models and proofs in Rodin, [14] approximated reals using integers (because of the lack of support for reals in Rodin at the time that the work was undertaken). Hybrid Event-B is an extension of Event-B that distinguishes *mode* variables (discrete) and *pliant* variables (continuous) [5]. Hybrid Event-B also distinguishes mode events and pliant events; mode events model instantaneous discrete changes while pliant events model continuous evolution of pliant variables over time intervals. The interval events used in this chapter are essentially intended to mimic the pliant events of Hybrid Event-B. At the time of writing there is no tool support for Hybrid Event-B.

We make a distinction between a control *goal* and a control *strategy*. We outline how both can be modelled and how refinement can be used to prove that a strategy satisfies a goal. In the case of the water tank system the control goals is as follows:

The control goal is to maintain the water level between a high level, H, and a low level, L.

Water may flow out of the tank according to some known maximum rate. To maintain a satisfactory water level, the controller can switch on a pump which causes the water level to increase according to some known maximum rate.

The control strategy is to sense the water level periodically and switch the pump on or off as appropriate.

We find it useful to follow the categorisation of modelling variables given in the Four-variable model of Parnas and Madey [11]. In this model, there are two main groupings of variables, *environment* variables and *controller* variables. Environment variables represent quantities in the environment of the controller. Controller variables represent quantities inside the controller machine. There are two kinds of environment variable as follows:

Monitored variables Environmental quantities whose value is not determined by the controller but that can be monitored. For example, the water level in the tank is a monitored variable.

Controlled variables Environmental quantities whose value is expected to be determined by the controller. For example the pump status (*on* or *off*) is a controlled variable.

The approach we follow is to start with a model of the control goal expressed in terms of monitored variables. We then refine this by a model of the strategy which is expressed in terms of monitored and controlled variables.

2 Reals and continuous functions

We have defined a theory of real arithmetic using the Rodin Theory feature. This introduces a new basic type, $REAL$, defines real versions of the standard

arithmetic operators (addition, subtraction, multiplication and division) and defines the usual total order on reals. With the Theory feature, operators may be defined directly (in terms of previously-defined operators), recursively (for inductive data types) and axiomatically with a collection of axioms on a group of operators. In our case we define the arithmetic operators axiomatically using standard axioms for reals.

We define an interval between two reals as follows[1] :

$$i..j \quad = \quad \{\ k \mid k \in REAL \ \wedge\ i \leq k \ \wedge\ k \leq j\ \}$$

This is an example of a direct definition in the Rodin Theory feature: the interval operator is defined using existing operators (set comprehension) for arguments i and j.

We use a standard definition of continuity. A function f is continuous at point c, written $cts(f, c)$, when it satisfies the following condition:

$$\begin{array}{lll} cts(f,c) & \iff & f \in REAL \pfun REAL \ \wedge \\ & & \forall \epsilon \cdot 0 < \epsilon \ \Rightarrow \\ & & \exists \delta \cdot 0 < \delta \ \wedge \\ & & \quad \forall x \cdot x \in dom(f) \ \wedge \\ & & \qquad c - \delta \ < \ x \ < \ c + \delta \\ & & \qquad \Rightarrow \\ & & \qquad f(c) - \epsilon \ < f(x) \ < \ f(c) + \epsilon \end{array}$$

This states that for every neighbourhood around $f(c)$, defined by $f(c) \pm \epsilon$, there exists a δ that defines a neighbourhood around c, defined by $c \pm \delta$, that yields that neighbourhood around $f(c)$.

An interval function is continuous if it is continuous at every point in its domain so we define the set of continuous functions on the interval $i..j$, written $ctsF(i, j)$, as follows:

$$ctsF(i,j) \quad = \quad \{\ f \mid f \in i..j \to REAL \ \wedge\ \forall c \cdot c \in i..j \Rightarrow cts(f,c)\ \}$$

3 Modelling a Continuous Control Goal

We specify a continuous control goal in terms of monitored variables represented by continuous functions on time intervals. We use a simple form of timed automaton, where the state variables include a clock, clk, and monitored variables, m, specified as continuous functions from time zero up to clk (where $PosREAL = \{r | r \in REAL \wedge r \geq 0\}$):

$$\begin{array}{rcl} clk & \in & PosREAL \\ m & \in & (0..clk) \to REAL \end{array}$$

[1] For readability, we use the usual symbols for real arithmetic ($+$, $-$, $\leq$, etc). These symbols are used for integer arithmetic in Rodin and, since operator overloading is not allowed in Rodin, we use different symbols ($plus$, sub, leq, etc) in our REAL theory.

This gives us two ways of specifying continuous goals in Event-B:

- *Invariants* are used to specify a property satisfied by the entire evolution of continuous variables.
- *Interval events* are used to specify a property on the continuous evolution within a behaviour mode.

Boundary constraints can be specified independently of time, that is, the value of a continuous variable at each point in its interval remains within some fixed boundaries. If we characterise the boundary as the set of values B within the boundary, then satisfaction of the boundary by all points of the continuous variable can be specified by range inclusion:

$$ran(m) \subseteq B$$

Requiring that the water level is always between the low and high marks is an example of a boundary constraint defined by the set $L..H$. Another form of pointwise property could involve the relationship between several continuous variables. For example, in a cruise control system for a car, the speed should be close to the target speed. Assuming *target* and *speed* are continuous functions on interval $i..j$, this can be specified as a pointwise predicate as follows:

$$\forall t \cdot t \in i..j \Rightarrow target(t) - \delta \leq speed(t) \leq target(t) + \delta$$

Other properties, such as monotonicity, smoothness, responsiveness and stability, span an interval and cannot be specified on individual time points. We do not consider a full range of interval properties here, but we do make use of monotonicity. For example, we define the set of *monotonically increasing* interval functions as follows:

$$\begin{aligned} mono\_inc \quad = \quad & \{\ f, i, j \mid f \in ctsF(i,j) \ \wedge \\ & \qquad (\forall k, l \cdot i \leq k \leq l \leq j \ \Rightarrow \ f(k) \leq f(l)) \bullet f\ \} \end{aligned}$$

The control goal for the water tank is a boundary property (defined by constants H and L) that should always be true so we use a boundary constraint on the continuous variable to model this:

inv1 : $clk \in PosREAL$

inv2 : $wl \in ctsF(0, clk)$

inv3 : $ran(wl) \subseteq L..H$

To model the dynamics within a mode, we use a nondeterministic interval event that extends the continuous behaviour for an interval of nondeterministic length. Such an event chooses some future time t and a continuous function f over the future interval and updates the clock and the continuous variables in a way that satisfies a property P. It has the following form:

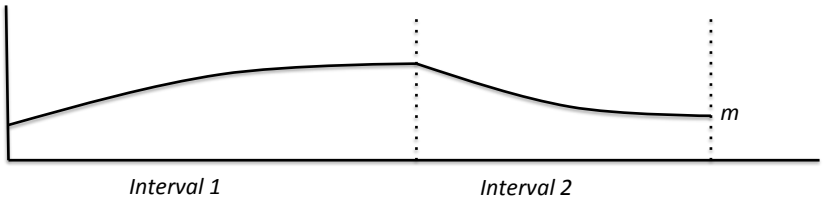

Figure 1: Evolution of continuous function during intervals.

Event *UpdateMonitored* $\widehat{=}$

any

t,f

where

grd1 : $t \geq clk + \epsilon$
grd2 : $f \in ctsF(clk, t)$
grd3 : $f(clk) = m(clk)$
grd4 : $P(f)$

then

act1 : $m := m \cup f$
act2 : $clk := t$

end

Possible continuous behaviour allowed by this event is illustrated by the graph in FIGURE 1. In the graph, the behaviour within each interval is defined by a continuous function over that interval and continuity is maintained between intervals. The function is monotonically increasing in *Interval 1* and monotonically decreasing in *Interval 2*.

Each of the guards in the *UpdateMonitored* event is essential:

- Guard *grd1* requires the next interval to have a duration of at least ϵ where ϵ is a constant. This is to prevent zeno behaviour where the time interval continually gets smaller and smaller.
- Guard *grd2* requires the next interval function f to be continuous.
- Guard *grd3* requires the endpoint of the existing interval function m and the starting point of the next interval function f to agree. This is to ensure that continuity is preserved when the interval function m is extended in action *act2*.
- Guard *grd4* requires the next interval function f to satisfy the interval property P.

We require that interval events always preserve continuity of the continuous variables (so *grd2* and *grd3* are essential). Some interval properties P are preserved under extension, that is, if the existing interval function m satisfies P and the next interval f satisfies P, that the extended interval function $m \cup f$ also satisfies P. It is easy to show that boundary properties are preserved when extending an interval function since they are time independent. Monotonicity is also reserved by interval extension, e.g., if m is monotonically increasing and f is monotonically increasing, then $m \cup f$ is monotonically increasing (provided $m \cup f$ is continuous). Clearly there are interval extensions that are not property preserving, e.g., extending a monotonically increasing function with a monotonically decreasing function does not preserve monotonicity.

In the water tank example, we use the following nondeterministic interval event to represent the required evolution of the water level during a mode:

Event *WaterLevelInterval* $\widehat{=}$

any

t,f

where

grd1 : $t \geq clk + \epsilon$
grd2 : $f \in ctsF(clk, t)$
grd3 : $f(clk) = wl(clk)$
grd4 : $ran(f) \subseteq L..H$

then

act1 : $wl := wl \cup f$
act2 : $clk := t$

end

The interval property (*grd4*) for this event is a boundary property, and as just discussed, is preserved by interval extension, i.e., this event preserves the boundary invariant on the water level.

4 Distinguishing modes

Our simple water tank system has two modes of operation: when the pump is on, the water level is monotonically increasing and when the pump is off, it is monotonically decreasing. In both cases the boundary property must be maintained. The interval event *WaterLevelInterval* of the previous section is an abstraction of both of these modes of operation. We construct a refined model of the water tank machine with two interval events, one for increasing the water level and the other for decreasing the water level. We require both of these to be refinements of *WaterLevelInterval* and thus they need to preserve the boundary property.

The specification of the interval event for increasing the water level is as follows:

Event *IncreaseWaterLevelInterval* $\widehat{=}$

refines *WaterLevelInterval*

any

t,f

where

grd1 : $t \geq clk + \epsilon$
grd2 : $f \in ctsF(clk, t)$
grd3 : $f(clk) = wl(clk)$
grd4 : $f \in mono\_inc$
grd5 : $f(t) \in L..H$

then

act1 : $wl := wl \cup f$
act2 : $clk := t$

end

Here we replace the abstract interval property ($ran(f) \subseteq L..H$) with refined interval properties stating that f is monotonically increasing and that the endpoint of f is within the boundaries (the start-point is also bounded because of $grd3$ and the invariants specifying that wl is bounded). The refinement is valid because, if f is monotonic and its endpoints are bounded, then f is bounded at all points. This is capture by the follow inference rule (here $mono = mono\_inc \cup mono\_dec$):

$$\frac{\begin{array}{c} f \in cstF(i,j), \quad f \in mono, \\ f(i) \in L..H, \quad f(j) \in L..H \end{array}}{ran(f) \subseteq L..H}$$

The interval event for decreasing the water level can be defined in a similar way. Note that we have not yet introduced a controller variable representing the status of the pump . At this level of refinement we remain focused on the monitored variable representing the water level.

5 Modelling the Control Strategy

To model the control strategy we introduce controlled variables whose value is modified at discrete steps. We make the assumption that the value of a controlled variable may influence the value of a monitored variable. For example, in the water tank we introduce a *pump* variable representing the status of the pump (*on* or *off*). If the pump is on, we assume that the water level increases monotonically while if the pump is off, then the water level decreases monotonically. For the water tank, we assume a periodic controller with a fixed period of length T that uses the following strategy at each control period:

C1 If the level is below a low threshold LT, the pump is switched on.

C2 If the level is above a high threshold HT, the pump is switched off.

C2 If the level is between LT and HT, the pump status does not change.

We assume that constants LT and HT lie in between L and T as follows:

axm1 : $L < LT < HT < H$

Furthermore we assume that the rate of increase in the water level is bounded by constant RI, that is, the water level can increase by a maximum of $RI \times T$ during one fixed-length period. Similarly we assume the rate of decrease of the water level is bounded by constant RD. The values for LT and HT are chosen such that if the water level is within $LT..HT$ at a periodic control point, then the water level cannot go outside $L..H$ by the next control point, i.e.,

axm2 : $L \ \leq \ LT - (RD \times T)$

axm3 : $HT + (RI \times T) \ \leq \ H$

The interval events introduced in the previous modelling level represent the evolution of a continuous variable during an entire control mode throughout which the controlled variables remain unchanged. For example, the *IncreaseWaterLevelInterval* event represents the water level increasing while the pump is on. We refer to these as *big step* events. When introducing the periodic controller in a refinement, the effect of a single interval event at the abstract level, e.g., *IncreaseWaterLevelInterval*, will be achieved by multiple sequential fixed-length periodic intervals. To model this we represent the evolution of the continuous variables *during* a mode using more fine-grained periodic interval events. The periodic events extend the continuous variables by fixed length intervals, each of size T. We refer to these as *small step* events. In addition to the controlled variables, we introduce additional variables in the refinement to represent the changes made by the periodic controller events: a clock variable to represent the periodic steps of time during a single mode, and continuous variables to represent the periodic evolution of the continuous variables during the mode. For the water tank example, we introduce two variables, one representing the periodic steps in time within a pump mode (clk_m), and another representing the periodic evolution of the water level during a mode (wl_m). This is illustrated in FIGURE 2: the left hand graph represents a series of periodic evolutions of the newly introduced wl_m variable within a mode; once sufficient periodic intervals have been defined by small step events, the big step event extends wl for a full mode interval using the periodic interval functions accumulated in wl_m.

We introduce invariants representing properties of the newly introduced variables in the refinement. For the water tank example we have the following invariants:

inv11 : $clk \leq clk_m$

inv12 : $wl_m \in ctsF(clk, clk_m)$

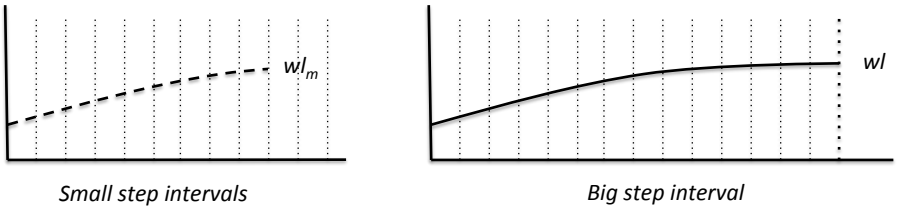

Figure 2: Small step and big step intervals.

inv13 : $wl_m(clk) = wl(clk)$

inv14 : $wl_m(clk_m) \in L..H$

inv15 : $pump = on \;\; \Rightarrow \;\; wl_m \in mono\_inc$

inv16 : $pump = off \;\; \Rightarrow \;\; wl_m \in mono\_dec$

In interpreting these invariants, it is important to understand that the new variables clk_m and wl_m will be updated within a mode by the periodic controller events (small step events), while the original variables (which remain part of the refined model) will be updated by the refinements of the abstract nondeterministic interval events (big step events). The invariants specify that clk_m is never behind clk ($inv11$) and that wl_m is a continuous function over the interval $clk..clk_m$ ($inv12$) whose start point is fixed ($inv13$) and whose end point is bounded ($inv14$). Depending on whether the pump is *on* or *off*, determines whether wl_m is monotonically increasing or monotonically decreasing ($inv15$, $inv16$).

The periodic events for a mode will be continually executed, once per control period, while it is ok to remain within that mode. For the water tank, the following periodic event specifies the system behaviour during a single period while the pump is in the *on* mode:

Event *PeriodicIntervalIncrease* $\widehat{=}$

any

f

where

grd1 : $pump = on$
grd2 : $wl_m(clk_m) \leq HT$
grd3 : $f \in ctsF(clk_m, clk_m + T)$
grd4 : $f(clk_m) = wl_m(clk_m)$
grd5 : $f(clk_m + T) \;\leq\; wl_m(clk_m) + (RI \times T)$
grd6 : $f \in mono\_inc$

then

act1 : $wl_m := wl_m \cup f$

act2 : $clk_m := clk_m + T$

end

Here the first two guards specify an enabling condition for the event, i.e., that the pump is on and that the current water level has not exceeded HT. The remaining guards specify constraints on the choice of interval function used to extend wl_m, i.e., f is an interval of length T starting from the current time ($grd3$), its starting value is the current water level $grd4$, its ending value is bounded by the rate of increase ($grd5$) and it is monotonically increasing ($grd6$). As well as extending wl_m by f, the actions of the event increase clk_m by a fixed amount T representing the fixed duration of a period.

There are two key reasons why the *PeriodicIntervalIncrease* event maintains the invariants on wl_m:

- The level at the end of the interval, $f(clk_m + T)$ is bounded above by $HT + (RI \times T)$ (from $grd2$, $grd5$) which means it is bounded above by H ($axm3$).
- The continuous composition of two monotonically increasing function is monotonically increasing:

$$\frac{\begin{array}{c} f \in cstF(i,j), \quad f \in mono\_inc, \\ g \in cstF(j,k), \quad g \in mono\_inc, \\ f(j) = g(j) \end{array}}{f \cup g \ \in \ mono\_inc}$$

The periodic event for decreasing the water level is defined and verified in a similar way.

The events that represent the end of a mode are specified as refinements of the abstract nondeterministic interval events. The events are made deterministic by providing witness for the nondeterministic parameters of the abstract events and the witnesses are provided by the variables introduced to represent the evolution of time and of the continuous variables during a mode. For example, the nondeterministic interval event representing the monotonically increasing mode is refined as follows:

Event *IncreaseWaterLevelInterval* $\widehat{=}$

refines *IncreaseWaterLevelInterval*

where

grd1 : $pump = on$

grd2 : $wl_m(clk_m) > HT$

with

t : $t = clk_m$

f : $f = wl_m$

then

act1 : $wl := wl \cup wl_m$

act2 : $clk := clk_m$

act3 : $wl_m := \{clk_m\} \lhd wl_m$

act4 : $pump := off$

end

The refined event is enabled when the pump is on ($grd1$) and the water level exceeds HT ($grd2$). In the abstraction of this event, t and f are nondeterministically chosen parameters. In the refinement they are eliminated as parameters and their values are represented by deterministic witness predicates (**with** clause). The invariants of this refined model ensure that the witness values satisfy the constraints on the choice of values for the parameters in the more abstract model. The original actions of the abstract event are retained in the refinement event, and additional actions are added to reset wl_m to be a point interval on the current time and change the pump status to *off*.

6 Merging big and small step variables

In the refinement just outlined, we retain the variable updated by the big step events (wl) and we introduced a new variable that is updated by the small step events (wl_m). Variable wl represents the history of the water level from time zero up to the most recent big step interval while wl_m represents the recent history from the most recent big step interval up to the most recent small step interval. It is possible in a further refinement to merge the big step and small step variables into a single variable wl_s representing the full history from time zero to the most recent small step interval. This merge means that the big step clock is not required since it is no longer used in any guards and can be eliminated. The elimination and merge is characterised by the following simple invariant:

inv21 : $wl_s = wl \cup wl_m$

This merge leads to a simplification of the actions of the *IncreaseWaterLevelInterval* event: the update of *clk* is eliminated and the simultaneous update of wl and wl_m is realised by a *skip* action on wl_s (thus no change to wl_s is required). The only remaining action is the update to the pump status. We also take the opportunity to rename this event to *PumpOff* to indicate the control purpose that it now represents. The simplified event is specified as follows:

Event *PumpOff* $\widehat{=}$

refines *IncreaseWaterLevelInterval*

where

grd1 : $pump = on$

grd2 : $wl_s(clk_m) > HT$

then

act1 : $pump := off$

end

Through this merging of continuous variables, the *PumpOff* event has become an *instantaneous* event whereas previously it was an interval event. We say it is now instantaneous since it does not update a clock and it does not extend a continuous variable. We are able to simplify *PumpOff* to be instantaneous because the overall effect of the abstraction of this event is achieved by a sequence of periodic interval events. Once sufficiently many small step events have been executed to accumulate the recent history of the water level within the mode, the instantaneous mode change event is enabled and the effect specified by the abstraction of that instantaneous event will have been achieved by the accumulation of small step interval since the most recent big step events.

7 Derivatives

In control systems it is common to specify properties of interval functions in terms of properties of derivatives of those functions. For example, a function is monotonically increasing if its derivative is always positive, a function is linear if its derivative is a constant. Rather than defining derivatives exactly, we can characterise them axiomatically. Since not all functions have a derivative, we capture the set of differentiable functions over interval $i..j$ with a set $diff(i, j)$. We write $der(f)$ for the derivative of f. We assume (axiomatically) that all differentiable functions are also continuous and that the derivative of a differentiable function f is a continuous interval function over the same interval as f:

axm : $\forall i, j \cdot diff(i, j) \subseteq ctsF(i, j)$

axm : $\forall f, i, j \cdot f \in diff(i, j) \Rightarrow der(f) \in ctsF(i, j)$

We can capture the property that functions with positive derivatives are monotonically increasing through the following axiom:

axm : $\forall f, i, j \cdot f \in diff(i, j) \wedge ran(der(f)) \subseteq PosREAL \Rightarrow f \in mono\_inc$

This axiom allows us to refine an event guard $f \in mono\_inc$ by a guard requiring f to be a differentiable function with positive derivative.

8 Concluding

We summarise our approach as follows: Continuous behaviour is specified through nondeterministic big step interval events that constrain the shape of continuous interval functions during a mode. We can refine these nondeterministic events by further constraining the shape of the continuous interval functions, e.g., we refined an interval function, that is bounded at every point, to a monotonically increasing function, that is bounded at its end points. We can also introduce periodic small step events within a mode as new events in a refinement to represent the strategy to be followed by a control system. Preservation of properties of continuous functions is key to ensuring the correctness of the refinements. For example, the most abstract interval event for the water tank preserves boundary invariants, the small step events in the water tank preserve monotonicity.

We need to be very careful in how we allow the clocks and continuous functions to be modified in order to reflect assumptions about the progression of time: time must move forwards, not backwards, and zeno behaviour must be avoided; continuous functions are extended in a forward direction only. It should be possible to enforce these idioms through a syntactic layer and this is one of the features of Hybrid Event-B [5].

Our approach fits with the abstraction/refinement approach of Event-B and the water tank development is supported by the Rodin toolset through the use of theories to define operators and proof rules for continuous functions. What is less clear is how well the approach scales to the case of high degrees of concurrency with multiple continuous functions operating to different mode intervals. This is the subject of future work.

The approach outlined here is influenced by the approach of [6] to the Steam Boiler Problem. In that paper, actions systems (the basis of Event-B) are used to construct a model of the system that includes the monitored variables and the controlled variables. Although [6] uses a very simple model of discrete time whereby the environment actions model the update to monitored variables in one complete control cycle, it does encourage a *system-level* approach. By this we mean that rather than modelling the environment and controller separately, the abstract model captures the overall system and refinement is used to introduce more distinction between the environment and controller. In fact in our approach we take the abstraction one level further than in [6] by focusing on monitored variables (water level) in the abstraction and only introducing the controlled variable (the pump) through refinement.

Besides monotonicity, we have focused on expression of control goals that refer to individual time points, i.e., the monitored variables are required to satisfy some property at each time point. Treatment of properties over time intervals, as expressible in the Duration Calculus [8] or the approach of Hayes, Jackson and Jones [9], merits further investigation. We have yet to include the treatment of faults (e.g., pump or sensor failure) and fault tolerance in our approach though we believe it is sufficiently flexible to support instantaneous (e.g., sensor failure) and continuous faults (e.g., water level is decreasing when

it should be increasing).

References

[1] Jean-Raymond Abrial. *Modeling in Event-B - System and Software Engineering*. Cambridge University Press, 2010.

[2] Jean-Raymond Abrial, Michael Butler, Stefan Hallerstede, Thai Son Hoang, Farhad Mehta, and Laurent Voisin. Rodin: an open toolset for modelling and reasoning in Event-B. *STTT*, 12(6):447–466, 2010.

[3] Ralph-Johan Back and Reino Kurki-Suonio. Distributed cooperation with action systems. *ACM Trans. Program. Lang. Syst.*, 10(4):513–554, 1988.

[4] Ralph-Johan Back, Luigia Petre, and Ivan Porres. Continuous action systems as a model for hybrid systems. *Nord. J. Comput.*, 8(1):2–21, 2001.

[5] Richard Banach, Huibiao Zhu, Wen Su, and Runlei Huang. Continuous kaos, asm, and formal control system design across the continuous/discrete modeling interface: a simple train stopping application. *Formal Asp. Comput.*, 26(2):319–366, 2014.

[6] M. J. Butler, E. Sekerinski, and K. Sere. An action system approach to the steam boiler problem. In J.-R. Abrial, E. Börger, and H. Langmaack, editors, *Formal Methods for Industrial Applications – Specifying and Programming the Steam Boiler Control, LNCS 1165*, pages 129–148. Springer-Verlag, Berlin, 1996.

[7] Michael Butler and Issam Maamria. Practical theory extension in event-b. In *Theories of Programming and Formal Methods - Essays Dedicated to Jifeng He on the Occasion of His 70th Birthday*, volume 8051 of *Lecture Notes in Computer Science*, pages 67–81. Springer, 2013.

[8] Zhou Chaochen and Michael Hansen. *Duration Calculus: A Formal Approach to Real-Time Systems*. Springer, Heidelberg, 2004.

[9] Ian Hayes, Michael Jackson, and Cliff Jones. Determining the specification of a control system from that of its environment. In Keijiro Araki, Stefani Gnesi, and Dino Mandrioli, editors, *FME 2003: Formal Methods, LNCS 2805*. Springer-Verlag, Berlin, 2003.

[10] Thomas A. Henzinger. The theory of hybrid automata. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA, July 27-30, 1996*, pages 278–292. IEEE Computer Society, 1996.

[11] David Lorge Parnas and Jan Madey. Functional documents for computer systems. *Sci. Comput. Program.*, 25(1):4161, 1995.

[12] André Platzer. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, Heidelberg, 2010.

[13] Mauno Ronnko, Anders P. Ravn, and Kaisa Sere. Hybrid action systems. *Theor. Comp. Sci.*, 290(1):937–973, 2003.

[14] Wen Su, Jean-Raymond Abrial, and Huibiao Zhu. Formalizing hybrid systems with event-b and the rodin platform. *Sci. Comput. Program.*, 94:164–202, 2014.