Pliant Modalities in Hybrid Event-B

Richard Banach[1]

[1]School of Computer Science, University of Manchester,
Oxford Road, Manchester, M13 9PL, U.K.
banach@cs.man.ac.uk

Abstract. Hybrid Event-B includes provision for continuously varying behaviour as well as the usual discrete changes of state in the context of Event-B. As well as being able to specify hybrid behaviour in the usual way, using differential equations or continuous assignments for the continuous parts of the behaviour, looser ways of specifying behaviour at higher levels of abstraction are extremely useful. Although the need for such looser specification can be met using the logic of the formalism, certain metaphors (or patterns) occur so often, and are so useful in practice, that it is valuable to introduce special machinery into the specification language, to allow these frequently occurring patterns to be readily referred to. This paper introduces such machinery into Hybrid Event-B.

1 Introduction

In today's ever-increasing interaction between digital devices and the physical world, formalisms are needed to express the more complex behaviours that this allows. Furthermore, these days, it is no longer sufficient to focus on isolated systems, as it is more and more the case that families of such systems are coupled together using communication networks, and can thus influence each others' working. So today *Cyber-Physical Systems* [19, 22, 1, 11] are the primary focus of attention. It is gratifying to note on the occasion of Jifeng He's festschrift, that the present author's own interest in such systems was sparked during cooperation with Prof. He's group at ECNU.

These new kinds of system throw up novel challenges in terms of design technique, as it is proving more and more difficult to ignore the continuous characteristics in their behaviours. Specifically, such technical challenges are being increasingly felt in the context of the B-Method [2, 3], where an increasing number of applications involve continuous behaviour of some sort in an essential way. Hybrid Event-B [10] has been introduced to bring new capabilities to traditional discrete Event-B [3], in order to address the challenges referred to.

Hybrid Event-B is a formalism that aims to provide a 'minimally invasive' extension of traditional discrete Event-B, capable of dealing with continuous behaviour as a first class citizen. As described in the next section, traditional discrete Event-B events serve as the 'mode events' that interleave the 'pliant events' of Hybrid Event-B. The latter express the continuously varying behaviour of a hybrid formalism including both kinds of event. In this manner, a rigorous link can be made between continuous and discrete update, as needed in these contemporary applications.

Since Event-B may be seen as related to the Action Systems formalism of Back and co-workers [5, 6], so Hybrid Event-B may be seen as related to the Hybrid Action Systems that extend the Action Systems formalism [7, 9, 8]. However, there are some crucial differences. The most important of these is the fact that in Hybrid Action Systems, (pieces of) continuous behviours are packaged up into lambda abstractions (with time as the lambda variable), whereas discrete updates are handled as usual (i.e. by manipulating free variables). Although the approach is mathematically feasible, it introduces a discrepancy between the way that discrete and continuous updates are handled — and in fact, continuous updates are processed in discrete lumps, at occurrences of discrete updates, where the lambda abstractions are updated monolithically.[1]

From an applications perspective, it can be argued that this distinction is aesthetically jarring, but it also has technical repercussions. Most importantly, the mechanical processing of lambda abstractions, in practice, typically has much less power (in terms of the inferences that can be made automatically) than the mechanical processing of expressions predominantly featuring free variables. So automation will typically be less effective using such an approach. Hybrid Event-B treats the continuous behaviours via free variables, which does not in itself dilute the potential for mechanical processing.

Although Hybrid Event-B is a fully expressive formalism, based on the general theory of first order ordinary differential equations (ODEs) for the continuous part of the formalism and thus is capable of describing all the kinds of behaviour needed for arbitrary hybrid systems, it is nevertheless the case that in the continuous context there are metaphors that arise so commonly, that it is worth optimising the formalism to enable their even more convenient use. It is the aim of this paper to describe a representative selection of such optimisations, called here pliant modalities, and to show how their use can be integrated into the existing formalism of Hybrid Event-B.

The rest of the paper is as follows. Section 2 gives a brief description of Hybrid Event-B that is sufficient for the remainder. Section 3 discusses pliant modalities in general. The subsequent sections discuss specific pliant modalities in detail. Section 4 discusses the CONTINUOUS and similar modalities. Section 5 discusses the CONSTant modality. Section 6 discusses the PLiant ENVelope modality and related modalities. Sections 7 and 8 discuss monotonic modalities and convexity and concavity. Section 9 covers a small case study. Section 10 concludes.

2 Hybrid Event-B, a Sketch

In Fig. 1 we see a bare bones Hybrid Event-B machine, *HyEvBMch*. It starts with declarations of time and of a clock. In Hybrid Event-B time is a first class citizen in that all variables are functions of time, whether explicitly or implicitly. However time is special, being read-only, never being assigned, since time cannot be controlled by any human-designed engineering process. Clocks allow a bit more flexibility, since they are assumed to increase their value at the same rate that time does, but may be (re)set during mode events (see below).

[1] The discrete analogue of this would be to express every variable in conventional Event-B as a lambda function of the (normally implicit) indexing variable of a runtime trace of the system.

```
MACHINE HyEvBMch
TIME t
CLOCK clk
PLIANT x, y
VARIABLES u
INVARIANTS
  $x \in \mathbb{R}$
  $y \in \mathbb{R}$
  $u \in \mathbb{N}$
EVENTS
  INITIALISATION
    STATUS ordinary
    WHEN
      $t = 0$
    THEN
      $clk := 1$
      $x := x_0$
      $y := y_0$
      $u := u_0$
    END
...   ...
```

```
...   ...
    MoEv
      STATUS ordinary
      ANY $i?, l, o!$
      WHERE $grd(x, y, u, i?, l, t, clk)$
      THEN
        $x, y, u, clk, o! :| BApred(x, y, u,$
          $i?, l, o!, t, clk, x', y', u', clk')$
      END
    PliEv
      STATUS pliant
      INIT $iv(x, y, t, clk)$
      WHERE $grd(u)$
      ANY $i?, l, o!$
      COMPLY
        $BDApred(x, y, u, i?, l, o!, t, clk)$
      SOLVE
        $\mathcal{D}\, x = \phi(x, y, u, i?, l, o!, t, clk)$
        $y, o! := E(x, u, i?, l, t, clk)$
      END
  END
```

Fig. 1. A schematic Hybrid Event-B machine.

Variables are of two kinds. There are mode variables (like u, declared as usual) which take their values in discrete sets and change their values via discontinuous assignment in mode events. There are also pliant variables (such as x, y), declared in the PLIANT clause, which take their values in topologically dense sets (normally $\mathbb{R}$) and which are allowed to change continuously, such change being specified via pliant events (see below).

Next are the invariants. These resemble invariants in discrete Event-B, in that the types of the variables are asserted to be the sets from which the variables' values *at any given moment of time* are drawn. More complex invariants are similarly predicates that are required to hold *at all moments of time* during a run.

Then we get to the events. The *INITIALISATION* has a guard that synchronises time with the start of any run, while all other variables are assigned their initial values in the usual way. As hinted above, in Hybrid Event-B, there are two kinds of event: mode events and pliant events.

Mode events are direct analogues of events in discrete Event-B. They can assign all machine variables (except time itself). In the schematic *MoEv* of Fig. 1, we see three parameters $i?, l, o!$, (an input, a local parameter, and an output respectively), and a guard *grd* which can depend on all the machine variables. We also see the generic after-value assignment specified by the before-after predicate *BApred*, which can specify how the after-values of all variables (except time, inputs and locals) are to be determined.

Pliant events are new. They specify the continuous evolution of the pliant variables over an interval of time. The schematic pliant event *PliEv* of Fig. 1 shows the structure. There are two guards: there is *iv*, for specifying enabling conditions on the pliant variables, clocks, and time; and there is *grd*, for specifying enabling conditions on the mode variables. The separation between the two is motivated by considerations connected with refinement.

The body of a pliant event contains three parameters $i?, l, o!$, (once more an input, a local parameter, and an output respectively) which are functions of time, defined over the duration of the pliant event. The behviour of the event is defined by the COMPLY and SOLVE clauses. The SOLVE clause specifies behaviour fairly directly. For example the behaviour of pliant variable y and output $o!$ is given by a direct assignment to the (time dependent) value of the expression $E(\ldots)$. Alternatively, the behaviour of pliant variable x is given by the solution of the first order ordinary differential equation (ODE) $\mathcal{D}\, x = \phi(\ldots)$, where $\mathcal{D}$ indicates differentiation with respect to time. (In fact the sematics of the $y, o! = E$ case is given in terms of the ODE $\mathcal{D}\, y, \mathcal{D}\, o! = \mathcal{D}\, E$, so that x, y and $o!$ satisfy the same regularity properties.) The COMPLY clause can be used to express any additional constraints that are required to hold during the pliant event via its before-during-and-after predicate *BDApred*. Typically, constraints on the permitted range of values for the pliant variables, and similar restrictions, can be placed here.

The COMPLY clause has another purpose. When specifying at an abstract level, we do not necessarily want to be concerned with all the details of the dynamics — it is often sufficient to require some global constraints to hold which express the needed safety properties of the system. The COMPLY clauses of the machine's pliant events can house such constraints directly, leaving it to lower level refinements to add the necessary details of the dynamics. The kind of pliant modalities that are the main focus of this paper are frequently to be found in the COMPLY clauses of pliant events.

The semantics of a Hybrid Event-B machine is as follows. It consists of a set of *system traces*, each of which is a collection of functions of time, expressing the value of each machine variable over the duration of a system run. (In the case of *HyEvBMch*, in a given system trace, there would be functions for clk, x, y, u, each defined over the duration of the run.)

Time is modelled as an interval $\mathcal{T}$ of the reals. A run starts at some initial moment of time, t_0 say, and lasts either for a finite time, or indefinitely. The duration of the run $\mathcal{T}$, breaks up into a succession of left-closed right-open subintervals: $\mathcal{T} = [t_0 \ldots t_1), [t_1 \ldots t_2), [t_2 \ldots t_3), \ldots$. The idea is that mode events (with their discontinuous updates) take place at the isolated times corresponding to the common endpoints of these subintervals t_i, and in between, the mode variables are constant and the pliant events stipulate continuous change in the pliant variables.

Although pliant variables change continuously (except perhaps at the t_i), continuity alone still allows for a wide range of mathematically pathological behaviours. To eliminate these, we make the following restrictions which apply individually to every subinterval $[t_i \ldots t_{i+1})$:

I Zeno: there is a constant δ_{Zeno}, such that for all i needed, $t_{i+1} - t_i \geq \delta_{\mathsf{Zeno}}$.

II Limits: for every variable x, and for every time $t \in \mathcal{T}$, the left limit $\lim_{\delta \to 0} x(t - \delta)$ written $\overrightarrow{x(t)}$ and right limit $\lim_{\delta \to 0} x(t + \delta)$, written $\overleftarrow{x(t)}$ (with $\delta > 0$) exist, and for every t, $x(t) = \overleftarrow{x(t)}$. [N. B. At the endpoint(s) of $\mathcal{T}$, any missing limit is defined to equal its counterpart.]

III Differentiability: The behaviour of every pliant variable x in the interval $[t_i \ldots t_{i+1})$ is given by the solution of a well posed initial value problem $\mathcal{D}\, xs = \phi(xs \ldots)$ (where xs is a relevant tuple of pliant variables and $\mathcal{D}$ is the time derivative). "Well posed" means that $\phi(xs \ldots)$ has Lipschitz constants which are uniformly bounded over $[t_i \ldots t_{i+1})$ bounding its variation with respect to xs, and that $\phi(xs \ldots)$ is measurable in t.

Regarding the above, the Zeno condition is certainly a sensible restriction to demand of any acceptable system, but in general, its truth or falsehood can depend on the system's full reachability relation, and is thus very frequently undecidable.

The stipulation on limits, with the left limit value at a time t_i being not necessarily the same as the right limit at t_i, makes for an easy interpretation of mode events that happen at t_i. For such mode events, the before-values are interpreted as the left limit values, and the after-values are interpreted as the right limit values.

The differentiability condition guarantees that from a specific starting point, t_i say, there is a maximal right open interval, specified by t_{MAX} say, such that a solution to the ODE system exists in $[t_i \ldots t_{\mathrm{MAX}})$. Within this interval, we seek the earliest time t_{i+1} at which a mode event becomes enabled, and this time becomes the preemption point beyond which the solution to the ODE system is abandoned, and the next solution is sought after the completion of the mode event.

In this manner, assuming that the *INITIALISATION* event has achieved a suitable intial assignment to variables, a system run is *well formed*, and thus belongs to the semantics of the machine, provided that at runtime:

- Every enabled mode event is feasible, i.e. has an after-state, and on its completion enables a pliant event (but does not enable any mode event). (1)
- Every enabled pliant event is feasible, i.e. has a time-indexed family of after-states, and EITHER: (2)

  (i) During the run of the pliant event a mode event becomes enabled. It preempts the pliant event, defining its end. ORELSE

  (ii) During the run of the pliant event it becomes infeasible: finite termination. ORELSE

  (iii) The pliant event continues indefinitely: nontermination.

Thus in a well formed run mode events alternate with pliant events.[2] The last event (if there is one) is a pliant event (whose duration may be finite or infinite).

[2] Many formalisms for hybrid systems permit a succession of mode events to execute before the next pliant event runs (to use our terminology). We avoid this for a number of reasons. Firstly, it spoils the simple picture that at each time, each variable has a unique value, and the runtime semantics of a variable is a straightforward function of time. Secondly, it avoids having to define the final value of a succession of mode events via a fixpoint calculation. Thirdly, and perhaps most importantly, it maintains the discrete Event-B picture in which events are (implicilty) seen as taking place at isolated points of real time, insofar as Event-B behaviours are seen as relating to the real world. We regard the overturning of such unstated assumptions as particularly dangerous in an engineering context — c.f. the Mars Lander incident, in which the U.S. and European teams interpreted measurements according to different units, without ever thinking to check which units were actually intended.

We note that this framework is quite close to the modern formulation of hybrid systems. (See eg. [20, 14] for representative formulations, or the large literature in the *Hybrid Systems: Computation and Control* series of international conferences, and the further literature cited therein.)

In reality, there are a number of semantic issues that we have glossed over in the framework just sketched. We refer to [10] for a more detailed presentation. As well as these, in line with the normal B-Method approach, there is a full suite of proof obligations that statically guarantees conformance with the semantics (see [10] again). We do not have space to quote them all, but we overview the ones that are most relevant to the remainder of the paper. First we quote pliant event feasibility:

$$\begin{aligned}
&I(u(\mathbb{t}_{\mathrm{L}})) \wedge iv_{PliEvA}(u(\mathbb{t}_{\mathrm{L}})) \wedge grd_{PliEvA}(u(\mathbb{t}_{\mathrm{L}}))\\
&\quad \Rightarrow (\exists\, \mathbb{t}_{\mathrm{R}} > \mathbb{t}_{\mathrm{L}} \bullet [\, (\mathbb{t}_{\mathrm{R}} - \mathbb{t}_{\mathrm{L}} > \delta_{\mathsf{Zeno}PliEvA}) \wedge \,]\\
&\qquad\qquad (\forall\, \mathbb{t}_{\mathrm{L}} < t < \mathbb{t}_{\mathrm{R}} \bullet (\exists\, u(t) \bullet BDApred_{PliEvA}(u(t), t) \wedge SOL_{PliEvA}(u(t), t))))
\end{aligned} \tag{3}$$

In (3), I is the machine invariant, iv and grd are guards, $PliEv$ is the pliant event in question, u refers to all the machine's variables as needed, SOL is the solution to the SOLVE clause of $PliEv$, $\mathbb{t}_{\mathrm{L}}$ and $\mathbb{t}_{\mathrm{R}}$ define the endopints of the interval in which the solution holds, and $\delta_{\mathsf{Zeno}PliEv}$ is the relevant Zeno constant — the term containing it can be omitted if it is too difficult to establish Zeno-freeness statically. Note that both *BDApred* and *SOL* have to hold throughout the interval. If either fails to do so, it signals the end of the feasible interval for *PliEv*. Next we quote pliant event invariant preservation:

$$\begin{aligned}
&I(u(\mathbb{t}_{\mathrm{L}})) \wedge iv_{PliEvA}(u(\mathbb{t}_{\mathrm{L}})) \wedge grd_{PliEvA}(u(\mathbb{t}_{\mathrm{L}})) \wedge\\
&(\exists\, \mathbb{t}_{\mathrm{R}} > \mathbb{t}_{\mathrm{L}} \bullet (\forall\, \mathbb{t}_{\mathrm{L}} < t < \mathbb{t}_{\mathrm{R}} \bullet BDApred_{PliEvA}(u(t), t) \wedge SOL_{PliEvA}(u(t), t)))\\
&\quad \Rightarrow (\forall\, \mathbb{t}_{\mathrm{L}} < t < \mathbb{t}_{\mathrm{R}} \bullet I(u(t)))
\end{aligned} \tag{4}$$

The last PO we quote is the correctness PO for pliant event refinement:

$$\begin{aligned}
&I(u(\mathbb{t}_{\mathrm{L}})) \wedge K(u(\mathbb{t}_{\mathrm{L}}), w(\mathbb{t}_{\mathrm{L}})) \wedge iv_{PliEvC}(w(\mathbb{t}_{\mathrm{L}})) \wedge grd_{PliEvC}(w(\mathbb{t}_{\mathrm{L}})) \Rightarrow\\
&\boldsymbol{(}\, \exists\, \mathbb{t}_{\mathrm{R}} > \mathbb{t}_{\mathrm{L}} \bullet (\forall\, \mathbb{t}_{\mathrm{L}} < t < \mathbb{t}_{\mathrm{R}} \bullet BDApred_{PliEvC}(w(t), t) \wedge SOL_{PliEvC}(w(t), t))\\
&\quad \Rightarrow (\forall\, \mathbb{t}_{\mathrm{L}} < t < \mathbb{t}_{\mathrm{R}} \bullet (\exists\, u(t) \bullet\\
&\qquad\qquad BDApred_{PliEvA}(u(t), t) \wedge SOL_{PliEvA}(u(t), t) \wedge K(u(t), w(t))))\boldsymbol{)}
\end{aligned} \tag{5}$$

In (5), *PliEvA* and *PliEvC* are the abstract and concrete pliant events. Furthermore, the form of (5) implies that time progresses in the same way in the abstract and concrete systems. This is a consequence of the single outer level quantification over time $\forall\, \mathbb{t}_{\mathrm{L}} < t < \mathbb{t}_{\mathrm{R}}$, indicated by the heavy parentheses. The uniform parameterisation over time implies that (5) demands that the joint invariant $K(u(t), w(t))$ holds throughout the two pliant events.

3 Pliant Modalities and Requirements in Hybrid Event-B

The framework described above, when elaborated in full detail, is certainly expressive enough to subsume the range of problems tackled in the hybrid and cyber-physical systems domain. However it does so by reducing all aspects of system behaviour to their descriptions in a language that is essentially first order logic with real and integer arithmetic, real functions and set theory, supplemented by differential equations. Any such language is, of course, rich enough to be highly undecidable. Despite this, many simple and intuitive properties of system behaviour nevertheless have descriptions in such a language that obscures their simplicity. This has two negative consequences. Firstly, it obscures readability and perspicuity for the user or designer, when straightforward ideas have to be written in a convoluted way. Secondly, it may make it relatively more difficult for a mechanised reasoning system to reach the most frequently intended consequences of these simple properties, when they are needed in verification.

The aim of this paper is to introduce a range of syntactic primitives for properties of real functions that occur commonly in hybrid applications, along with their definitions. Not only can this make system models more readable, but it can also be exploited by reasoning systems in order to optimise verification.

The primitives we introduce are in many ways really rather simple: constancy, boundedness, monotonicity, and so on. While easy to grasp, their defnitions in terms of base logical primitives are always more complicated than one feels they ought to be. As well as that though, and in stark contrast to the situation for purely discrete applications, there are not-so-obvious connotations with requirements that are worth exploring.

In a purely discrete application, when we write an oversimplified abstract model (as we are strongly encouraged to do at the outset of system development in Event-B), we are never in any doubt that what we define (as long as it is not patently ridiculous), is ultimately implementable. The basic reason for this is that the discrete data types we use in such early models are clearly implementable on the discrete hardware that we ultimately target them to. The case with continuously varying quantities is subtly different. Usually, we build continuous models to reflect the capabilities of real physical phenomena or real physical equipment. In such cases, the modelling flexibility that we have is severely curtailed, because the physical behaviour that we want to model is normally confined within quite tightly constrained parameters. Postulating a behaviour, no matter how desirable, is pointless unless we are confident about ultimate implementabiliy. In practice this usually means that we have to start modelling at a level considerably closer to the physical level than we might like. Only when we know that we are working within realistic parameters, can we permit ourselves to abstract from the details, to create a simplified model that deals just with the coarsest aspects of the dynamics.

Having pursued a strategy as just described, it would be reasonable at this point to question the purpose of the simplified model, given that a more precise description is already available. The response to that would be, that there could easily be properties of the system that are much more convenient to deal with when cast in in terms of the simplified model than they would be if cast in in terms of the more precise model. *In extremis*, some properties might only be tractable in the simplified model.

We infer that consideration of the continuous sphere can bring with it an inversion of the usual Event-B relationship between requirements and refinement. Usually in Event-B, once having incorporated a requirement into a system model, further models of the development are refinements of it. In the continuous case though, we have argued that a subsequent model might be an abstraction instead.

Technically, a pliant modality is a property of a pliant variable that is given a specific name. Such modalities may occur in two places in a Hybrid Event-B machine. One possibility is in the INVARIANTs of the machine. The properties defined by such modalities must hold throughout any run of the machine. The other possibility is in the COMPLY clause of a pliant event. In this case, the relevant property is only required to hold during the execution of the pliant event, and indeed, if at some point during the execution of the pliant event, the modality contradicts other properties in the event's definition, it merely serves to define the limit of feasibility of the pliant event (i.e. PO (3) fails), signalling termination. We now discuss some modalities in more detail.

4 The CONTINUOUS Modality and its Relatives

Our first, and simplest, modality is the continuous modality, which declares that a pliant variable's behaviour is (globally) continuous throughout the duration of any run of the system. Such a restriction is appropriate for a model of a physical variable which is not expected to undergo impulses during the dynamics, and would be written amongst the INVARIANTS. Since the semantics of Hybrid Event-B ensures that the behaviour of any pliant variable is absolutely continuous during any pliant event, the continuous modality just prevents the variable's value from being discontinuously reassigned by a mode event, a condition that is particularly easy to enforce statically. We write the continuous modality thus:

$$\mathsf{CONTINUOUS}(f) \;\equiv\; \ldots \tag{6}$$

where the ellipsis stands for one of a number of equivalent definitions of absolute continuity. (See e.g., [21] for details.) Related to the CONTINUOUS modality, and a little harder to enforce, are modalities that assert the derivative, or *n*-th derivative of f are globally continuous:

$$\mathsf{DIFFERENTIABLE}(f) \;\equiv\; \mathsf{CONTINUOUS}(\mathcal{D} f) \tag{7}$$

$$n\text{-}\mathsf{DIFFERENTIABLE}(f) \;\equiv\; \mathsf{CONTINUOUS}(\mathcal{D}^n f) \tag{8}$$

5 The CONST Modality

Our next modality is the constant modality, which declares that a pliant variable remains constant. At this point, the reader may well question the need for such a modality. Surely, if a variable is to be constant, aside from the possibility of declaring a constant instead, we could declare a mode variable, and declare it as CONTINUOUS. And there are other, similar possibilities. Pursuing such reasoning, the case we cannot cover by existing means is when we need to introduce a quantified variable, which we require to remain constant within the scope of the COMPLY clause of a pliant event, but whose value is not determinable statically. The definition of the constant modality for a variable f is thus:

$$\mathsf{CONST}(f) \;\equiv\; (\forall t \bullet \mathbb{t}_{\mathrm{L}} \le t < \mathbb{t}_{\mathrm{R}} \Rightarrow f(\mathbb{t}_{\mathrm{L}}) = f(t)) \tag{9}$$

The form of the CONST modality that we show in (9) is the one that is appropriate for a COMPLY clause. This is characterised by the presence of $\mathbb{t}_{\mathrm{L}}$ and $\mathbb{t}_{\mathrm{R}}$ in (9). These identifiers are free in (9) and bind to the intial and final times repectively of the duration of the pliant event at runtime. (The former fact follows from the assumption of the pliant event's guards as properties of $\mathbb{t}_{\mathrm{L}}$ in POs featuring $\mathbb{t}_{\mathrm{L}}$, and the latter from the assumption that $\mathbb{t}_{\mathrm{R}} > \mathbb{t}_{\mathrm{L}}$ in these POs (which is usually sufficient) — as is done in (3)-(5).) The form of (9) appropriate for an INVARIANT follows by removing the references to $\mathbb{t}_{\mathrm{R}}$ and replacing references to $\mathbb{t}_{\mathrm{L}}$ by references to the initial time t_0, leaving:

$$\mathsf{CONST}(f) \;\equiv\; (\forall\, t \bullet t_0 \leq t \Rightarrow f(t_0) = f(t)) \tag{10}$$

The CONST modality has an associated form (in which E is an expression) for cases when we can determine the value that f is to have during the relevant interval:

$$\mathsf{CONST}(f, E) \;\equiv\; (\forall\, t \bullet \mathbb{t}_{\mathrm{L}} \leq t < \mathbb{t}_{\mathrm{R}} \Rightarrow f(\mathbb{t}_{\mathrm{L}}) = f(t) = E(\mathbb{t}_{\mathrm{L}})) \tag{11}$$

In future, we will just show the COMPLY form of any modality, as the INVARIANT form follows readily, by simply removing references to $\mathbb{t}_{\mathrm{R}}$, and replacing values at $\mathbb{t}_{\mathrm{L}}$ with intial values where necessary.

6 The PLENV Modality and its Relatives

The PLENV modality is at the opposite end of the expressivity spectrum to CONST. It constrains its argument f to remain within a PL*iant* ENV*elope* specified by a lower bound function and/or an upper bound function. It is easiest to build up from the simplest cases, so we start with dynamic lower and dynamic upper bounds alone:

$$\mathsf{PLENVL}(f, lb) \;\equiv\; (\forall\, t \bullet \mathbb{t}_{\mathrm{L}} \leq t < \mathbb{t}_{\mathrm{R}} \Rightarrow lb(t) \leq f(t)) \tag{12}$$

$$\mathsf{PLENVU}(f, ub) \;\equiv\; (\forall\, t \bullet \mathbb{t}_{\mathrm{L}} \leq t < \mathbb{t}_{\mathrm{R}} \Rightarrow f(t) \leq ub(t)) \tag{13}$$

The general dynamically bounded PLENV modality now follows:

$$\mathsf{PLENV}(f, lb, ub) \;\equiv\; \mathsf{PLENVL}(f, lb) \wedge \mathsf{PLENVU}(f, ub) \tag{14}$$

From these forms, several useful forms follow by restricting the various bounds involved to constants. We can easily define these by combining our existing definitions with the CONST modality, which is actually the most transparent way to do it.

The first two modalities just restrict PLENVL and PLENVU, giving conventional notions of lower and upper bounds:

$$\mathsf{LBND}(f, E) \;\equiv\; (\exists\, lb \bullet \mathsf{PLENVL}(f, lb) \wedge \mathsf{CONST}(lb, E)) \tag{15}$$

$$\mathsf{UBND}(f, E) \;\equiv\; (\exists\, ub \bullet \mathsf{PLENVU}(f, ub) \wedge \mathsf{CONST}(ub, E)) \tag{16}$$

Associated with these is the BND modality, combining the two of them:

$$\mathsf{BND}(f, E_L, E_U) \;\equiv\; \mathsf{LBND}(f, E_L) \wedge \mathsf{UBND}(f, E_U) \tag{17}$$

Finally we have versions of PLENV in which one bound but not the other is held constant:

$$\mathsf{PLENVLC}(f, E_L, ub) \;\equiv\; \mathsf{LBND}(f, E_L) \wedge \mathsf{PLENVU}(f, ub) \tag{18}$$

$$\mathsf{PLENVUC}(f, lb, E_U) \;\equiv\; \mathsf{PLENVL}(f, lb) \wedge \mathsf{UBND}(f, E) \tag{19}$$

The above pliant modalities give us a selection of primitives that can capture high level aspects of pliant variable behaviour in a concise and perspicuous way. Of course the aspects thus captured are not the only ones that can be exhibited by pliant variables, but they are typical of the relatively simple properties favoured in many engineering designs. We can thus expect them to have wide applicability, especially when they are combined with the possibility of making f a real-valued function of several variables (each depending on t). Moreover, we have to be sure that particular constants or bounds used are actually achievable, so design of these high level properties usually goes hand in hand with the lower level design that refines/implements it.

7 Monotonic Modalities

In this section we consider modalities connected with monotonicity. For functions of time delivering a single real value, the following are the evident possibilities.

$$\mathsf{MONINC}(f) \;\equiv\; (\forall\, t_1, t_2 \bullet \mathbb{t}_{\mathrm{L}} \le t_1 \le t_2 < \mathbb{t}_{\mathrm{R}} \Rightarrow f(t_1) \le f(t_2)) \tag{20}$$
$$\mathsf{MONDEC}(f) \;\equiv\; (\forall\, t_1, t_2 \bullet \mathbb{t}_{\mathrm{L}} \le t_1 \le t_2 < \mathbb{t}_{\mathrm{R}} \Rightarrow f(t_1) \ge f(t_2)) \tag{21}$$

Compared with the modalities of Section 6, the above modalities can be used more freely at an abstract level, since they do not assert specific numerical measures for the rate of increase/decrease, making it relatively easier to postpone the determination of these rates during the development.

8 Convex and Concave Modalities

In the last section, the key properties depended on comparing the function under consideration at two points. Convexity and concavity depend on a comparison at three points, and are essentially concerned with capturing liberal notions of a function increasing or decreasing in an “accelerating” manner (bending upwards — convexity), or in a “decelerating” manner (bending downwards — concavity):

$$\begin{aligned}\mathsf{CVEX}(f) \;\equiv\; (\forall\, t_1, t_2, \lambda \bullet \mathbb{t}_{\mathrm{L}} \le t_1 \le t_2 < \mathbb{t}_{\mathrm{R}} \wedge 0 < \lambda < 1 \Rightarrow \\ f(\lambda\, t_1 + (1-\lambda)t_2) \le \lambda f(t_1) + (1-\lambda) f(t_2))\end{aligned} \tag{22}$$
$$\begin{aligned}\mathsf{CCAVE}(f) \;\equiv\; (\forall\, t_1, t_2, \lambda \bullet \mathbb{t}_{\mathrm{L}} \le t_1 \le t_2 < \mathbb{t}_{\mathrm{R}} \wedge 0 < \lambda < 1 \Rightarrow \\ f(\lambda\, t_1 + (1-\lambda)t_2) \ge \lambda f(t_1) + (1-\lambda) f(t_2))\end{aligned} \tag{23}$$

These modalities evidently have similar levels of flexibility for abstract use as we saw in the previous section. Equally evidently, we can imagine more and more complex properties built along similar lines.

9 A Simple Case Study

In this section we give a simple case study, based on examples in [9, 7, 8, 17, 18, 4], and simplified for a briefer description. The case study is based on a nuclear power station, where the reactor naturally tends to heat up despite heat being extracted for steam generation. To keep the temperature θ under control, a control rod, which absorbs neutrons, is inserted into the reactor, and the resulting lower neutron flux slows down the nuclear reaction, lowering the temperature. To prevent the reaction from stopping completely, the control rod is withdrawn after a period, and must cool down before it can be used again. In the works cited, more than one rod is used, leading to an interesting interplay between heating and cooling periods, and the recovery time of the rods.

For our purposes we simplify matters by having only one control rod, by assuming that the heating and cooling periods are equal, both lasting 3 time units, and that the 3 time units of the reactor heating period are sufficient for the control rod (which was extracted at the start of the reactor heating period) to itself have cooled down enough for rod reuse at the start of the next reactor cooling period.

In Fig. 2 we show a number of Hybrid Event-B machines that address this scenario. We postpone discussion of the *NucSkip* machine for now, and start with the *NucMon* machine. The *NucMon* machine models the behaviour just described at an abstract level. At initialisation time, the temperature θ is assumed to be θ_0 and a heating period is about to start (i.e. the rod is out). The *INITIALISATION* enables the *WaitRodOut* pliant event which lasts for the duration of the heating period. This specifies that the temperature is to increase in a monotonic fashion, using the MONINC modality, and furthermore states that the initial and final temperature values over the heating period differ by $\varDelta$.[3]

The significance of the latter is that, on the basis of monotonicity, we can deduce that the temperature throughout the heating period remains less than the final value attained during the period, $\theta(\mathbb{t}_{\mathrm{L}}) + \varDelta$. This in turn is sufficient to guarantee that the invariant $\theta \leq \varTheta_{\mathrm{TOL}}$ is satisfied throughout the heating period provided that $\theta_0 + \varDelta \leq \varTheta_{\mathrm{TOL}}$ holds, where $\varTheta_{\mathrm{TOL}}$ is the maximum tolerable reactor temperature that still assures safety. Since we have not included this last condition in the model, it would emerge as a missing hypothesis in any attempt to prove the pliant event invariant preservation PO (4) for *WaitRodOut*.

Similar arguments apply for the remainder of the behaviour of the *NucMon* machine. Thus, after 3 time units, mode event *RodIn* is enabled and preempts pliant event *WaitRodOut*. *RodIn* changes the *rod* variable from *out* to *in*, which enables *WaitRodIn*. In line with our simple modelling, the *WaitRodIn* pliant event exactly reverses the effect of *WaitRodOut*. The behaviour of *WaitRodIn* is also specified using a modality. This time it is the MONDEC modality, and it is again stated that the initial and final tempertaure values over the heating period differ by $\varDelta$. This time monotonicity guarantees

[3] In the *WaitRodOut* event, the final temperature value is referred to as $\theta(\mathbb{t}_{\mathrm{R}})$. This is slightly incorrect technically, since the actual value of θ at the time point referred to by $\mathbb{t}_{\mathrm{R}}$ falls outside the left closed right open interval $[\mathbb{t}_{\mathrm{L}} \ldots \mathbb{t}_{\mathrm{R}})$ which is the actual period during which any execution of *WaitRodOut* is active. A more technically correct reference to the final temperature reached would be $\mathsf{LLIM}(\theta(\mathbb{t}_{\mathrm{R}}))$, where LLIM is an additional modality that refers to the left limit of the expression enclosed. Instead of using such machinery, we have implicitly used a convention stipulating that any reference to a value at $\mathbb{t}_{\mathrm{R}}$ is in fact a reference to the relevant left limit value. This convention is to be understood as applying throughout Fig. 2. (Actually, since the temperature θ is always continuous in the behaviour specified in the *NucMon* machine, there is no difference between the actual $\mathbb{t}_{\mathrm{R}}$ value, and the corresponding left limit value at $\mathbb{t}_{\mathrm{R}}$, since no mode event of *NucMon* alters θ discontinuously.)

```
MACHINE $NucSkip$
CLOCK $clk$
PLIANT $\theta$
VARIABLES $rod$
INVARIANTS
  $\theta \in \mathbb{R} \wedge \theta \leq \Theta_{\mathrm{TOL}}$
  $rod \in \{in, out\}$
EVENTS
  $INITIALISATION$
    STATUS ordinary
    BEGIN
      $clk$ := 0
      $\theta$ := $\theta_0$
      $rod$ := $out$
    END
  $Wait$
    STATUS pliant
    INIT $clk = 0$
    COMPLY skip
    END
  $RodIn$
    STATUS ordinary
    WHEN $clk = 3 \wedge$
            $rod = out$
    THEN $rod$ := $in$
         $\theta$ := $\theta + \Delta$
         $clk$ := 0
    END
  $RodOut$
    STATUS ordinary
    WHEN $clk = 3 \wedge$
            $rod = in$
    THEN $rod$ := $out$
         $\theta$ := $\theta - \Delta$
         $clk$ := 0
    END
END
```

```
MACHINE $NucMon$

CLOCK $clk$
PLIANT $\theta$
VARIABLES $rod$
INVARIANTS
  $\theta \in \mathbb{R} \wedge \theta \leq \Theta_{\mathrm{TOL}}$
  $rod \in \{in, out\}$
EVENTS
  $INITIALISATION$
    STATUS ordinary
    BEGIN
      $clk$ := 0
      $\theta$ := $\theta_0$
      $rod$ := $out$
    END
  $WaitRodOut$
    STATUS pliant
    INIT $clk = 0$
    WHERE $rod = out$
    COMPLY
      MONINC$(\theta) \wedge$
      $\theta(\mathbb{t}_{\mathrm{R}}) = \theta(\mathbb{t}_{\mathrm{L}}) + \Delta$
    END
  $RodIn$
    STATUS ordinary
    WHEN $clk = 3 \wedge$
            $rod = out$
    THEN $rod$ := $in$
         $clk$ := 0
    END
  $WaitRodIn$
    STATUS pliant
    INIT $clk = 0$
    WHERE $rod = in$
    COMPLY
      MONDEC$(\theta) \wedge$
      $\theta(\mathbb{t}_{\mathrm{R}}) = \theta(\mathbb{t}_{\mathrm{L}}) - \Delta$
    END
  $RodOut$
    STATUS ordinary
    WHEN $clk = 3 \wedge$
            $rod = in$
    THEN $rod$ := $out$
         $clk$ := 0
    END
END
```

```
MACHINE $NucLinear$
REFINES $NucMon$
CLOCK $clk$
PLIANT $\theta$
VARIABLES $rod$
INVARIANTS
  $\theta \in \mathbb{R} \wedge \theta \leq \Theta_{\mathrm{TOL}}$
  $rod \in \{in, out\}$
EVENTS
  $INITIALISATION$
    STATUS ordinary
    BEGIN
      $clk$ := 0
      $\theta$ := $\theta_0$
      $rod$ := $out$
    END
  $WaitRodOut$
    STATUS pliant
    INIT $clk = 0$
    WHERE $rod = out$
    SOLVE
      $\mathcal{D}\,\theta = \Delta/3$
    END

  $RodIn$
    STATUS ordinary
    WHEN $clk = 3 \wedge$
            $rod = out$
    THEN $rod$ := $in$
         $clk$ := 0
    END
  $WaitRodIn$
    STATUS pliant
    INIT $clk = 0$
    WHERE $rod = in$
    SOLVE
      $\mathcal{D}\,\theta = -\Delta/3$
    END

  $RodOut$
    STATUS ordinary
    WHEN $clk = 3 \wedge$
            $rod = in$
    THEN $rod$ := $out$
         $clk$ := 0
    END
END
```

Fig. 2. Hybrid Event-B machines for the nuclear reactor case study.

that the temperature throughout the cooling period remains less than the initial value at the beginning of the period, $\theta(\mathbb{t}_{\mathrm{L}})$.

After 3 time units, mode event *RodOut* is enabled and preempts *WaitRodOut*. Because of our particularly simple modelling, *RodOut* returns the machine to exactly the state at intialisation, and the whole cycle of behaviour repeats indefinitely.

We now turn to machine *NucLinear*. This machine refines *NucMon*. Mostly it is identical to *NucMon*. To save space, we have not included the REFINES clauses for the individual events — it is to be assumed that each event of *NucLinear* refines the similarly named event in *NucMon*. The only differences between the two machines are in the *WaitRodOut* and *WaitRodIn* pliant events. While these events are specified loosely in the *NucMon* machine via modalities that admit an uncountable infinity of realisations, in the *NucLinear* machine, their behaviours are defined deterministically, via the differential equations $\mathcal{D}\,\theta = \pm\mathit{\Delta}/3$. Since the solutions of these, $\theta(t-\mathbb{t}_{\mathrm{L}}) = \theta(\mathbb{t}_{\mathrm{L}}) \pm (\mathit{\Delta}/3)(t-\mathbb{t}_{\mathrm{L}})$ are obviously monotonic, and also satisfy the properties needed at $\mathbb{t}_{\mathrm{R}} = \mathbb{t}_{\mathrm{L}} + 3$, they satisfy the specifications of *WaitRodOut* and *WaitRodIn* in *NucMon*, and hence we will be able to discharge the correctness PO for pliant event refinement (5) for these events, which is the only non-identity part of the refinement.

Reexamining the preceding from a requirements perspective, it is reasonable to presume that the crucial elements of the two machines *NucMon* and *NucLinear* were designed at least partly in tandem. The abstract specifications of pliant events *WaitRodOut* and *WaitRodIn* in *NucMon* must have been designed with at least a good degree of confidence that the rate of increase/decrease of temperature that they demanded was feasible, i.e. that realisations via the detailed behaviours of *WaitRodOut* and *WaitRodIn* in the *NucLinear* machine were achievable using the physical apparatus available.

Now we turn to the *NucSkip* machine. Its aim is to capture as much as possible of the system behaviour within mode events. We see this in the fact that the counterparts of the two pliant events *WaitRodOut* and *WaitRodIn* in the other two machines are required merely to skip in this one (over an extended time period), something which warrants the two events being combined into a single *Wait* event (whose somewhat superfluous nature is highlighted by the box surrounding it). Along with *Wait* just skipping, the two mode events *RodIn* and *RodOut* take on the additional job of recording the movements in temperature via increments of $\pm\mathit{\Delta}$.

In fact, with a suitably designed (and nontrivial) retrieve relation, the *NucSkip* and *NucMon* machines are interrefinable. To see this, we would have to rename the variables in the two machines in order to properly define the retrieve relation. While we do not pursue this in full detail, we can point out the essentials as follows. The retrieve relation $R(\theta_{NucSkip}, \theta_{NucMon})$ (which is oriented so that *NucSkip* is the abstract model and *NucMon* is the concrete model) has to take θ_{NucMon} and relate it to either θ_0 or to $\theta_0 + \mathit{\Delta}$ according to the value of *rod*, thus:

$$\begin{aligned} &K(\theta_{NucSkip}, \theta_{NucMon}) \;\equiv \\ &\qquad (rod = out \wedge \theta_0 \leq \theta_{NucMon} < \theta_0 + \mathit{\Delta} \wedge \theta_{NucSkip} = \theta_0) \vee \\ &\qquad (rod = in \wedge \theta_0 + \mathit{\Delta} \geq \theta_{NucMon} > \theta_0 \wedge \theta_{NucSkip} = \theta_0 + \mathit{\Delta}) \end{aligned} \tag{24}$$

The technically most interesting points regarding the refinement concern how the joint invariant $K(\theta_{NucSkip}, \theta_{NucMon})$ is preserved across the discontinuities at the mode events.

However, it is not hard to see that it all works out as required. For example, as an occurrence of *RodIn* approaches, θ_{NucMon} is slightly less than $\theta_0 + \varDelta$, while $\theta_{NucSkip}$ is still θ_0, satisfying $K(\theta_{NucSkip}, \theta_{NucMon})$. Then, as soon as *RodIn* completes, θ_{NucMon} immediately starts to decrease from $\theta_0+\varDelta$, while $\theta_{NucSkip}$ is now $\theta_0+\varDelta$, again satisfying $K(\theta_{NucSkip}, \theta_{NucMon})$ because of the altered value of *rod*. Event *RodOut* is similar, and, filling in the remaining details, the claimed interrefinability of *NucSkip* and *NucMon* can be established.

The interrefinability allows us to regard *NucSkip* as either an abstraction or a refinement of *NucMon*. The mathematics of the refinement is in fact typical of digital implementation techniques, whereby continuous behaviours are implemented by discretized means, under benign enough conditions. However, in this case, the relatively long duration of the pliant events, makes viewing *NucSkip* as an *implementation* rather unconvincing.

The view that *NucSkip* is an abstraction is more productive. Besides allowing for the passage of time, the pliant events of *NucSkip* do nothing. Nevertheless, despite this relative triviality, whether or not the crucial invariant $\theta \leq \varTheta_{\mathrm{TOL}}$ is preserved can still be determined from such a model. Note that the determination of this requires discrete computation exclusively, in contrast to making the same determination more directly from the *NucMon* or *NucLinear* machines, a potentially important simplification in the context of mechanical reasoning.

The *NucSkip* machine is a whisker away from a conventional Event-B machine. In fact, noting that the passage of time has no significance in this model, we could dispense with the pliant events completely, and be left with a genuine Event-B machine. In [17, 18, 4] the authors pursue a very similar approach in comparable examples. There, the abstract Event-B models just alluded to, are refined to more concrete Event-B models which handle the continuous behavours by packaging up pieces of continuous behavour in lambda abstractions, and assembling the overall behaviour as a succession of modifications that take place at the discrete transitions that correspond to our mode events. (This is patterned after the manner in which hybrid systems are modelled in the action systems framework [9, 7, 8].)

On the one hand, there is little difference between the approach we have developed here and the works of these other authors, if the aim is to explore the system's reachability relation and invariant preservation properties through the extreme values attained during the system's runs, these being computed using the discrete versions of the models. On the other hand, one consequence of using a (pure) discrete Event-B model for the management of these extreme values, is that the joint invariant defining its refinement to a more detailed model (taking real time and continuous behaviour into account) is typically restricted to holding only at the moments that the discrete events take place, i.e. pointwise at a few isolated moments of time, despite the fact that the continuous behaviour is active all the time. This observation severely weakens the connection between the models used, and the actual real world requirements that ought to be captured in the invariants.

To caricature the above in the context of our case study, it is no good (from the viewpoint of the real world requirements) if we are sure that the reactor temperature is safe at 3pm and at 4pm (because these are the times at which the discrete Event-B model's events take place and so the invariant can be asserted) and yet the reactor core is able to suffer a meltdown at 3.30pm (because no event took place then, and so the invariant could not be asserted at this time).

In our approach, the modalities we have introduced earlier enable us to conveniently package up useful properties of the continuous behaviors of interest, in a way that allows us to reduce the maintenance of the invariants that we are concerned with to discrete computations very similar to those of the previous approaches, *while nevertheless retaining fully the ability to rigorously assert the invariants needed at all times relevant to the requirements, and not just at those times when discrete events take place*. In this sense we regard our approach as an improvement on the pure discrete Event-B approach, beyond simply offering a more direct treatment of hybrid behaviour.

The above discussion also highlights the fact that mere (inter)refinability alone can be a very weak relationship between system models, unless the refinement relationship is appropriately validated against the application requirements. This is particularly important when purely discrete models are being related to models which treat real time as an indispensable element. Therefore it is crucial that the content of any such refinement relationship is critically evaluated. Our case study illustrates this particularly well since the to-all-intents-and-purposes discrete model *NucSkip* bears almost no visible relationship to the more 'realistic' *NucLinear* model that purports to reflect the behaviour of the physical system in a recognisable way. After all, the physical temperature *does not* undergo discontinuous changes when the control rod is inserted or withdrawn; and the physical temperature *does not* remain invariant in the time intervals between insertions/withdrawals. So if we are to use such a discrete model, it is vital that we can relate its behaviour to more realistic models via strong and convincing invariants.

10 Conclusions

In the previous sections we briefly reviewed Hybrid Event-B, and then discussed the motivations for introducing pliant modalities into the user level language. Basically, these included readability for application designers, so that intuitively straightforward properties would not be masked by convoluted low level descriptions. Furthermore, the possibility of simplifying the challenge for mechanised reasoning systems when dealing with large applications, by raising the level of reasoning abstraction via these modalities, provided an additional motivating factor. The motivating discussion went on to consider the interesting issue of where and how in a refinement hierarchy, the requirements concerning continuously varying entities were to be addressed, given that these are invariably strongly linked to real world properties of physical apparatus.

We then introduced a selection of simple modalities for functions of time returning a single real value, illustrating one of them in a case study based on rising and falling temperature in a nuclear reactor. The intention was not to give an exhaustive list of all modalities that might ever possibly be useful, but to give a representative selection.

In general, we would expect that an automated environment for supporting Hybrid Event-B application development would have the capability of incorporating user defined modalities introduced to support specific application.

The connection with automated environments and mechanised reasoning systems deserves further consideration. One aspect that we have not described in detail in this relatively brief paper, is that along with its definition, each modality needs a selection of properties that a verification environment for some application can readily make use of. In other words each modality needs to be supplied as a *theory* containing not only its definition, but also relevant properties.

Such properties need to be true of course. For very simple modalities such as those discussed in this paper, the properties of interest have been well known in the mathematics literature for around two centuries (see for example [16, 13, 12]). So full scale mechanical verification for these is perhaps superfluous. On the other hand, as the modalities introduced by users become more sophisticated and less conventional, the need for mechanical corroboration of their claimed properties becomes more pressing.

The mechanical corroboration issue raises an interesting challenge not present in the verification of purely discrete systems. This is that, whereas the *proof* of the properties of typical pliant modalities requires low level reasoning typical of arguments in mathematical analysis, usually needing a considerable number of interleavings of quantifications over higher order structures to be considered, the *use* of such properties in an application context is overwhelmingly symbolic and algebraic, needing perceptive strategies for equational substitution of equals for equals, and similar techniques. These are conflicting requirements for a verification system. The best proposal would therefore be to use different tools to address these different requirements: a system specifically geared to higher order reasoning and multiple nested quantifications at the low level, and a system more geared to decomposition and algebraic reasoning at the applications level. Such considerations pose an interesting challenge for any tool (eg. the Rodin Tool, [15]) that contemplates an extension to deal with the capabilities of Hybrid Event-B.

References

1. Report: Cyber-Physical Systems (2008), http://iccps2012.cse.wustl.edu/_doc/CPS_Summit_Report.pdf
2. Abrial, J.R.: The B-Book: Assigning Programs to Meanings. Cambridge University Press (1996)
3. Abrial, J.R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press (2010)
4. Abrial, J.R., Su, W., Zhu, H.: Formalizing Hybrid Systems with Event-B. In: Proc. ABZ-12. Springer, LNCS (2012)
5. Back, R., Sere, K.: Stepwise Refinement of Action Systems. In: Proc. MPC-89. vol. 376, pp. 115–138. Springer, LNCS (1989)
6. Back, R., von Wright, J.: Trace Refinement of Action Systems. In: Proc. CONCUR-94. vol. 836, pp. 367–384. Springer, LNCS (1994)
7. Back, R.J., Cerschi Seceleanu, C.: Modeling and Verifying a Temperature Control System using Continuous Action Systems. In: Proc. FMICS-00 (2000)
8. Back, R.J., Cerschi Seceleanu, C., Westerholm, J.: Symbolic Simulation of Hybrid Systems. In: Strooper, P., Muenchaisri, P. (eds.) Proc. APSEC-02. pp. 147–155. IEEE Computer Society Press (2002)
9. Back, R.J., Petre, L., Porres, I.: Continuous Action Systems as a Model for Hybrid Systems. Nordic J. Comp. 8, 2–21 (2001)

10. Banach, R., Butler, M., Qin, S., Verma, N., Zhu, H.: Core Hybrid Event-B: Adding Continuous Behaviour to Event-B (2012), submitted.
11. Barolli, L., Takizawa, M., Hussain, F.: Special Issue on Emerging Trends in Cyber-Physical Systems. J. Amb. Intel. Hum. Comp. 2, 249–250 (2011)
12. Haggarty, R.: Fundamentals of Mathematical Analysis. Prentice Hall (1993)
13. Lang, S.: Real and Functional Analysis. Springer (1993)
14. Platzer, A.: Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics. Springer (2010)
15. Rodin: European Project Rodin (Rigorous Open Development for Complex Systems) IST-511599 http://rodin.cs.ncl.ac.uk/
16. Rudin, W.: Principles of Mathematical Analysis. McGraw-Hill (1976)
17. Su, W., Abrial, J.R., Zhu, H., Huang, R.: From Requirements to Development: Methodology and Example. In: Proc. ICFEM-11. vol. 6991, pp. 437–455. Springer, LNCS (2011)
18. Su, W., Abrial, J.R., Zhu, H., Huang, R.: Complementary Methodologies for Developing Hybrid Systems with Event-B. In: Proc. ICFEM-12. vol. 7504, pp. 230–248. Springer, LNCS (2012)
19. Sztipanovits, J.: Model Integration and Cyber Physical Systems: A Semantics Perspective. In: Butler, Schulte (eds.) Proc. FM-11. Springer, LNCS 6664, p.1, http://sites.lero.ie/download.aspx?f=Sztipanovits-Keynote.pdf (2011), Invited talk, FM 2011, Limerick, Ireland
20. Tabuada, P.: Verification and Control of Hybrid Systems: A Symbolic Approach. Springer (2009)
21. Wikipedia: Absolute continuity.
22. Willems, J.: Open Dynamical Systems: Their Aims and their Origins. Ruberti Lecture, Rome (2007), http://homes.esat.kuleuven.be/~jwillems/Lectures/2007/Rubertilecture.pdf