Continuous Behaviour in Event-B: A Sketch

Richard Banach[1]*, Huibiao Zhu[2]**, Wen Su[2], and Xiaofeng Wu[2]

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

[2]Software Engineering Institute, East China Normal University,
3663 Zhongshan Road North, Shanghai 200062, P.R. China.
{hbzhu,wensu,xfwu}@sei.ecnu.edu.cn

Abstract. Including provision for continuously varying behaviour as well as discrete state change is considered for Event-B. An extension of Event-B is sketched that accommodates continuous events (called pliant events) in between familiar discrete events (called mode events).

1 Introduction

In this short paper, we briefly sketch an extension of Event-B that accommodates genuinely continuous behaviour (as well as discrete state changes). The motivation for this is to enable Event-B to engage better with problems exhibiting such behaviour in an essential way, as is increasingly needed in applications. A fully worked out presentation, including more extensive discussion of semantics, proof obligations for machine consistency and for refinement, and consideration of finegraining and coarsegraining issues, will appear elsewhere. We assume familiarity with Event-B.

2 Extending Event-B with Continuous Behaviour

To adequately capture behaviour over real time, we model time as an interval $\mathcal{T}$ of the real numbers $\mathbb{R}$, with a finite left endpoint to represent the time at which the initial state of the model is created, and with a right endpoint which is finite or infinite, depending on whether the dynamics is finite or infinite. Now, the values of all variables become functions of $\mathcal{T}$. By convention, $\mathcal{T}$ partitions into a sequence of left-closed right-open intervals, $\langle [t_0 \ldots t_1), [t_1 \ldots t_2), \ldots \rangle$, the coarsest partition such that all discontinuous changes take place at some boundary point t_i. We have two kinds of variable. **Mode variables** only change discontinuously between elements of a discrete type. These are

* The majority of the work reported in this paper was done while the first author was a visiting researcher at the Software Engineering Institute at East China Normal University. The support of ECNU is gratefully acknowledged.

** Huibiao Zhu is supported by National Basic Research Program of China (No. 2011CB302904), National High Technology Research and Development Program of China (No. 2011AA010101 and No. 2012AA011205), National Natural Science Foundation of China (No. 61061130541 and No. 61021004).

just like traditional B variables, and restricting to these recovers traditional Event-B. **Pliant variables** have types which include topologically dense sets, and which can evolve both continuously and via discrete changes. In a typical interval $[t_i \dots t_{i+1})$, the mode variables will be constant, but the pliant variables will change continuously. However, continuity alone still allows for a wide range of mathematically pathological behaviours, so we make the following restrictions:

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

With I-III in place, the behaviour of every pliant variable is piecewise smooth, with the smooth variation being described by a suitable differential equation (DE).

As well as two kinds of variable, we have two kinds of event. Mode events are like traditional Event-B events. They describe discontinuous changes, though they can involve both mode and pliant variables. Their syntax is identical to traditional Event-B events, except that before-values are to be interpreted as left limits (at the moment t_i that the event occurs), and after-values are to be interpreted as the corresponding right limits. For example, a straightforward generic mode event, decorated with this limit information, could be written in the usual notation as:

```
StdEv
    WHEN  $grd(\overrightarrow{u}, \overrightarrow{i})$
    ANY  $\overleftarrow{u}$
    WHERE  $BApred(\overrightarrow{u}, \overrightarrow{i}, \overleftarrow{u})$
    THEN  $u := \overleftarrow{u}$
    END
```

We also have pliant events. These involve changes to pliant variables alone, and they describe continuous change. While a mode event is a single before-/after-value pair, a pliant event is a family of before-/after-value pairs, parameterized by points in time falling within the relevant time interval $[t_i \dots t_{i+1})$. For every member of this family, the before-value is always the value at t_i, while the after-value is the value at t, for t in the open interval $(t_i \dots t_{i+1})$. Thus the change from before- to after-value does not take place instantaneously. Pliant events need new syntax, for which we give two variants:

```
PliEv
    STATUS  pliant
    WHEN  $grd(u(t_{\mathrm{L}(t)}))$
    WHERE  $BDApred(u(t), i(t), t)$
    SOLVE  $DE(u(t), i(t), t)$
    END
```

```
PliEv
    STATUS  pliant
    WHEN  $grd(u(t_{\mathrm{L}(t)}))$
    ANY  $u(t)$
    WHERE  $BDApred(u(t), i(t), t)$
    THEN  $u := u(t)$
    END
```

In the left hand syntax, we specify a differential equation to be solved. In the right hand syntax we just specify the continuous behaviour required directly, for those cases where this is known (since differentiating the behaviour, only to have to solve the resulting DE for it immediately afterwards is obviously wasteful).

Much of the structure of these two cases is similar, and we can discuss both of the cases together to begin with, starting from the top. After the header line we have the 'STATUS pliant' line. This introduces a new event status, the pliant status, signaling to any tool processing the syntax that a pliant event is being defined.

In the remainder of the structure we see the notation $\mathrm{L}(t) = \mathit{max}\{i \mid t_i \leq t\}$, which has a counterpart $\mathrm{R}(t) = \mathit{min}\{i \mid t_i > t\}$. These map any time t to the left and right ends of the interval containing t during a run, and are used to refer generically to the initial and final time values of the interval during which the continuous behavour is specified.

The next line is the 'WHEN' line, and contains any required facts about the initial values of the relevant state variables when the pliant transition starts; it also contains any additional guard information. Unlike the guard of a mode event, it cannot depend on any input that the pliant event needs, since any such input will last throughout the interval $(t_{\mathrm{L}(t)} \ldots t_{\mathrm{R}(t)})$, and so its value at the time instant $t_{\mathrm{L}(t)}$ has measure zero; this is of insufficient weight to influence the start of a pliant event.

At this point, the two structures start to diverge. On the left, for the case governed by a differential equation, we have a 'WHERE' line, which contains a before-during-and-after predicate *BDApred*. Since this case is predominantly governed by the differential equation, there is often little or nothing for the *BDApred* to specify, so it will often be very simple, or can be omitted entirely. On the other hand, if there are additional constraints that need to hold during the pliant event, such as facts concerning specific values of time, deadlines, or anything else, such constraints can be placed here.

The next line is where the action is, since it includes the differential equation in the 'SOLVE' clause. The differential equation specifies what the values of the state variables are to be during the interval of interest, but it does so indirectly. In general, the DE depends on the current values of the state variables and on the inputs which are received through the course of the interval of interest. That completes the description of the left hand case.

On the right, we pick up at the 'ANY' line. This works a lot like the ANY clause of a mode event, but it (typically) names a family of after-values that is time dependent, defined over the open interval $(t_{\mathrm{L}(t)} \ldots t_{\mathrm{R}(t)})$. The named values are utilised in the before-during-and-after predicate *BDApred*. Unlike the previous case, where the *BDApred* predicate is typically simple or absent, this time, the *BDApred* predicate is the entity that actually does the hard work of specifying the after-values, so, unlike previously, it will be nontrivial. The after-values are actually assigned in the 'THEN' clause on the next line, just as for normal Event-B. As usual, if the expressions to be assigned are known explicitly, then the ANY and WHERE clauses can be omitted, and the required values can be assigned directly. All this is therefore just like normal Event-B, except that everything is parameterised by time. This completes the description of the other case.

A continuous Event-B machine, with mode and pliant events as described, is said to be **well formed** iff every mode transition enables a pliant transition (but no mode transition) on completion, every non-final pliant transition enables a mode transition during its execution (which then fires, preempting the pliant transition), and a final pliant transition either continues indefinitely (non-termination) or becomes undefined at some point (finite termination).

A run of such a machine starts with an initial mode transition which sets up the system initial system state, and then, pliant transitions alternate with mode transitions. The last transition (if there is one) is a pliant transition (whose duration may be finite or infinite).

Since time has a different character from other variables, if time is mentioned explicitly in a system model, then the name of the time variable has to be indicated to a tool such as Rodin. A convenient way of doing this is to have a 'TIME t' declaration. The value of time may be linked to the rest of the system model in the *INITIALISATION* event which may then be given a guard such as 'WHEN $t = 0$'.

An alternative approach to time utilises one or more *clocks*. The difference between a time variable and a clock variable is that a clock may run fast or slow with respect to (real) time, so its derivative must be specified during pliant events that use it. It can also be reset by mode events (specifically during initialisation). Clock variables can be declared as 'CLOCK clk'.

3 Discussion, POs

Above, we sketched the essentials of an extension of Event-B intended to cope with genuinely continuous behaviours, such as are increasingly needed in the hybrid and cyber-physical applications being developed today. That so few of these are developed using a refinement mindset is the main reason for considering Event-B here.

As with conventional Event-B, the semantics is expressed via proof obligations, which we cover briefly now. Events have to be feasible; mode events via the usual PO, pliant events via a PO that asserts a solution to the DE in some interval. Events have to preserve the invariants; mode events as usual, pliant events continuously over the course of the DE solution. Alternation between mode and pliant events is handled by POs that demand the relevant disjunctions of guards under appropriate conditions.

Properly defined refinement between machines is crucial of course. Explicit POs become much simpler if mode events must be refined by mode events (in the usual way), and if pliant events must be refined by pliant events (in a way that preserves the passage of time, and maintains the invariants during the course of the two pliant events). Relative deadlock freedom may be demanded of the mode events, and separately, of the pliant events.

One issue not present in conventional Event-B, is that relatively long lived pliant events may need to get broken up into short lived ones, in particular, when modeling the implementation of continuous behaviour by digital means utilising a high sampling frequency. To deal with this we can introduce suitable skips that momentarily interrupt the long lived pliant event. POs can be designed so that such skips do not alter the dynamics of the system, while neverthless breaking up a long transition into short steps that can later be refined in the usual way.