Core Hybrid Event-B I: Single Hybrid Event-B Machines

Richard Banach[a,1], Michael Butler[b], Shengchao Qin[c], Nitika Verma[d], Huibiao Zhu[e,2]

[a]*School of Computer Science, University of Manchester, Oxford Road, Manchester, M13 9PL, U.K.*
[b]*School of Electronics and Computer Science, University of Southampton, Highfield, Southampton, SO17 1BJ, U.K.*
[c]*School of Computing, University of Teesside, Middlesbrough, Tees Valley, TS1 3BA, U.K.*
[d]*Department of Mathematics, Indian Institute of Technology, Hauz Khas, New Delhi, India.*
[e]*Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, 3663 Zhongshan Road North, Shanghai 200062, P.R. China.*

Abstract

Faced with the increasing need for correctly designed hybrid and cyber-physical systems today, the problem of including provision for continuously varying behaviour as well as the usual discrete changes of state is considered in the context of Event-B. An extension of Event-B called Hybrid Event-B is presented, that accommodates continuous behaviours (called pliant events) in between familiar discrete transitions (called mode events in this context). The continuous state change can be specified by a combination of indirect specification via ordinary differential equations, or direct specification via assignment of variables to values that depend on time, or indirect specification by demanding that behaviour obeys a time dependent predicate. The syntactic elements of the extension are discussed, and the semantics is described in terms of the properties of time dependent valuations of variables. Refinement is examined in detail, with reference to the notion of refinement inherited from discrete Event-B. A full suite of proof obligations is presented, covering all aspects of the new framework. A selection of examples and case studies is presented. A particular challenge —bearing in mind the desirability of conforming to existing intuitions about discrete Event-B, and the impact on tool support (as embodied in tools for discrete Event-B like Rodin)— is to design the whole framework so as to disturb as little as possible the existing structures for handling discrete Event-B.

1. Introduction

Today, we see an ever-increasing interaction between digital devices and the physical world. Once, it was enough to see this in terms of predominantly isolated systems, in which a single digital device interacted with a fixed suite of physical equipment, and to talk, therefore, of embedded systems. Nowadays though, this picture is proving more and more inadequate. It is more and more the case that families of such systems are coupled together using communication networks, and can thus influence each others'

Email addresses: banach@cs.man.ac.uk (Richard Banach), mjb@ecs.soton.ac.uk (Michael Butler), s.qin@tees.ac.uk (Shengchao Qin), nitika.iitd@gmail.com (Nitika Verma), hbzhu@sei.ecnu.edu.cn (Huibiao Zhu)

[1]A portion 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.

[2]Huibiao Zhu is supported by National Natural Science Foundation of China (No. 61361136002 and No. 61321064) and Shanghai Collaborative Innovation Center of Trustworthy Software for Internet of Things (ZF1213).

working. These days then, the talk is of *Cyber-Physical Systems* [45, 51, 52, 54, 49, 13, 18, 44, 1, 37], which is the name that has been adopted for these interacting families of embedded systems.

These new systems throw up novel challenges in terms of design technique, as it is increasingly difficult to ignore the continuous characteristics in their behaviours. Unfortunately, the usual kinds of approaches to the modeling, specification and development of conventional discrete systems, offer little help for developing the continuous aspects, simply because the usual semantic foundations of such approaches make almost no contact with what is needed for the continuous world.

That is not to say that discrete techniques have never impinged on the design of systems that are continuous as regards their physical characteristics — far from it. However, the usual way that purely discrete technologies interact with the continuous aspects is to tiptoe round them — predominantly because of the semantic inadequacy just mentioned. Often, the inconvenient continuous aspects are permitted to occur in only very simplified form, and then their consequences can typically be reduced to a small number of algebraic facts, which can be accommodated within the discrete world.

For very simple problems, this approach can almost be convincing, aside from the fact that the collection of algebraic facts that are accumulated, usually fail to come with the necessary invariants that bind them together — precisely because the required invariants emerge from the continuous world, which is being studiously ignored. Obviously this undermines the integrity of such a technique and weakens the dependability that it can deliver.

For more complex systems, the problem only gets worse. First, the design is approached from the purely continuous side (since it is too complicated to ignore the continuous aspects altogether). Conventional techniques from the continuous sphere are applied, until the design has reached a reasonable state. Then, some engineering heuristics are applied that turn a continuous design into a discrete one, after which, a kind of collective amnesia takes place. All thoughts of the continuous world are forgotten, and the discrete design that emerged from the earlier activity —which is regarded now as the top level spec— is treated as if it were the most obvious and natural way to abstractly specify the desired system.

Unfortunately, there is a major defect to this strategy. Specifications, by their nature, are intended to be as clear and perspicuous as the intrinsic nature of the problem will allow, so that they can be clearly related to domain level requirements, and properly understood by all problem domain stakeholders as easily as possible. An essential ingredient of this is simplicity of expression and of structure. The B Method [2, 3] —which is our concern in this paper— more than most, stresses the importance of starting out with a clear and simple view of the system-to-be, and of adding the complexity only gradually. However, that which is clear and perspicuous in the continuous world is not the same as that which is clear and perspicuous in the discrete world. The limiting processes that go into the construction of continuous world quantities, sweep away vast (in fact unbounded) quantities of the discrete level detail that goes into their bottom-up construction. This radically changes the nature of what is 'simple' in the two worlds.

In this paper we extend the formalism of Event-B so that it can deal with continuous behaviour as a first class citizen. This extends the reach of the B Method so that it is better able to capture the kind of developments needed to realise the cyber-physical systems spoken of earlier. As a byproduct, in enabling continuous behaviour to occur in native fashion at the most abstract levels of the development, the complex, unintuitive detail manufactured by discretization processes, takes its rightful place at the intermediate levels of a more broadly based development.

In cyber-physical systems design, the communication side of the communication / continuous interplay that has to be faced, can be handled by relatively conventional means. After all, communicating systems have been studied in computer science for many years, and Event-B is no exception in providing many examples of the modeling of communication (see e.g. [3]). This leaves the continuous side to be faced, and our extension of Event-B enables it to encompass hybrid behaviour in a first class way. This is the main objective of the present paper.

Our extension of Event-B is designed to cause as little disruption as possible to the existing structure of discrete Event-B. This point is important since considerable investment has already been made in tool support for Event-B, through projects like RODIN [39], DEPLOY [21] and ADVANCE [4], resulting in the current state of the Rodin tool [40]. This, we do not wish to spoil.

The remainder of the paper is as follows. In Section 2 we explore preexisting work in more detail and contrast some of its common features with what we do in this paper. In Section 3 we briefly review discrete Event-B. Section 4 is concerned with setting out the semantic foundations for incorporating continuous behaviours into Event-B in our approach. In Section 5 we define the core syntax of our Event-B extension, indicating how the issues discussed previously relate to it. In Section 7 we discuss the formal semantics of our framework, relying on standard results from the literature to handle routine matters. In Section 8 we discuss refinement in the extended Event-B framework. Section 9 collects together the proof obligations that keep all the issues discussed previously under control in a specific development. Section 10 describes a number of small case studies, starting with the bouncing ball, continuing with a simple discretization of continuous behaviour, and culminating with a simple study of the European Train Control System. Section 11 concludes.

2. Related Work and the Hybrid Event-B Approach

The framework for Hybrid Event-B that we will build below is similar in many respects to a number of formulations of hybrid systems in the literature. Hybrid systems themselves have been studied intensively for many years, and the literature is too large by now to cover everything in detail here. Some of the earliest work includes [35, 5, 6, 28, 33]. Shortly after these papers appeared, other works such as [34, 24, 25, 53] and [26, 43, 22, 8] were published. Slightly later formulations include [33, 14, 29, 30, 17, 7, 16, 23]. Of particular note is the survey [15], which covers a large number of these formulations, and especially, the tools that support them. A modern and unified theoretical overview of many of these established approaches is to be found in [46], and there is [38] which is closest to our approach. Moreover, a large body of work has appeared in the *International Conference on Hybrid Systems: Computation and Control* series of international meetings, and this, combined with the modern trends noted above, has joined with other relevant events, creating the major annual *CPS Week* meeting in recent years. We now comment on three characteristic that are frequently seen in this class of system.

The first characteristic of many extant systems for addressing hybrid behaviour, is that they are conceived with the strategy of verifying that a given hybrid system satisfies some desirable property — obviously this is a laudable aim in itself. Unfortunately, any language that is expressive enough to encompass a significant portion of hybrid behaviour is highly undecidable. As a consequence, the desire to make mechanisable inroads into the verification high level goal has led to many systems that curtail quite severely the expressivity of the language used to describe the candidate hybrid system, in order to lend some decidability to the problem. Even so, the needed decision procedures often have high complexity, adding yet more difficulties.

The second characteristic comes from this severe curtailment of expressivity inherent in the strategy just described, which chimes with a kind of bottom up approach. If one cannot express a problem in the most transparent way, its description will most likely reduce to a complex set of lower level subproblems (such as with discretization, discussed above). This only makes worse any challenge from high complexity decision procedures.

The third characteristic is a typical further consequence of this kind of strategy, namely that the connection between the formal description of the two sides of the framework can become weak. While the discrete side is invariably captured quite precisely, the side of the formalism that deals with the continuous side is either: precise but severely curtailed in expressivity; or is more encompassing regarding

the admitted continuous behaviour but significantly less precise regarding its foundations — in extreme cases delegating *all* aspects of continuous behaviour to, e.g., the semantics of a simulation tool.[3]

The extent to which any of these characteristics is present in any given formalism varies widely, of course. Our own approach for Hybrid Event-B attempts to bypass some of these difficulties by advocating a top down methodology. By starting with simple models, and designing the properties that they should satisfy along with them (rather than trying to discover those *post hoc*), and enriching both along the way to the final system, the aim is to keep the tractability of all aspects of design and verification much higher than if one was confronted with the final system outright — without any clues as to its underlying structure or design motivations.

A salient characteristic of the B-Method in general, of Event-B in particular, and of our hybrid extension of it, is the extent to which the top down approach is integral to the formalism. This approach has given Event-B considerable momentum worldwide [47], good reason to inspire our hybrid extension of it here. The top down approach also has some consequences regarding the issues mentioned above, which we comment on now.

Regarding verification, because we model at the highest level of abstraction possible, we avoid the pitfalls of an inherently bottom up approach, that would be forced by a low degree of expressivity. This has the advantage that we can attempt verification where it potentially has the least complexity; but it also has the disadvantage that we can easily write down models for which no verification strategy is known. We elaborate this point further shortly.

Regarding concerns about the formal description of the framework, our approach to the design of Hybrid Event-B is more readily distinguished from alternative approaches. First and foremost, we ground the semantics of the Hybrid Event-B framework-to-be in established facts from the world of textbook pure mathematics (facts concerning properties of suitable families of piecewise continuous real functions). This standpoint separates soundness-in-principle of the formalism (established by appeal to facts from mathematical analysis) from verifiability-in-practice (performed by executable algorithms running with acceptable complexity on specific classes of examples) — and leads to situations in which we know (semantically) certain generic facts on which we can rely, even though, in specific instances we cannot calculate their consequences. Still, this approach gives our formulation an equally consistent level of formal rigour for both the discrete and continuous parts of the theory, at least in principle.

In this paper we focus on the generic formal semantics. The preceding remarks imply that there is a non-trivial road to be navigated from the generic semantic world to the world of verifiable problem instances. We do not embark on that road in this paper, postponing those details to other publications.

Verifiability in practice is the primary concern of tools, and along with the theoretical development of this paper, there is an intention to enhance the Rodin tool [39] to incorporate the capability to verify suitable classes of practical examples. Typically, this capability will be somewhat open-ended, in line with the vast range of applied mathematics about which detailed consequences can be calculated, and the capability of the extended tool at any point will depend on the effort invested in tool enhancement up till then.[4]

What is needed for comprehensive verification goes beyond mere calculation of some continuous behaviour. Looking forward to the needs of the formal semantics, we require the calculation of the times of preemption of an episode of continuous behaviour by the next discrete transition, and the confirmation of invariants over a period of time; looking towards the needs of refinement, we additionally require confirmation of joint invariants over time. All this requires significant capability in symbolic calculation

[3] In fact, the behaviour of many commercial simulation tools intended for the modelling of physical systems is highly customer-driven, and makes no real contact with any foundational semantic concerns whatsoever [36].

[4] Thus, we envisage tool capability increasing over time. Despite this though, *every* version of such a tool will engage with some subset of the semantic world described in this paper, simplifying the conceptual challenge for practitioners.

for the tool, making the design of a suitable verification environment non-trivial, as stated.

Beyond these aspects, there are questions regarding the use of heuristic techniques, and of implementation. The reach of purely symbolic techniques will not cover all cases of interest, so more approximate techniques will need to be incorporated into the methodology. And when modelling has reached a sufficiently low level, code generation for appropriate parts of the system becomes relevant. Ideally, these aspects would be controlled by suitably incisive invariants, but it is to be noted that reasoning about approximate techniques is usually as difficult as the issues that cause their use in the first place, so this ideal may not be completely attainable.

Putting aside these questions of Hybrid Event-B internal strategy, the picture of system behaviour that it offers is quite similar to that offered by many of the systems mentioned at the beginning of this section. The majority of the works mentioned take an automata-theoretic view of hybrid systems, having named states for the discrete control. Within each of these, continuous behaviour evolves until the next preemption point, which is triggered by the truth of the guard condition of the next discrete state. We achieve a similar effect via the mode and pliant events of Hybrid Event-B, described below.

This relatively small degree of difference between formulations is in fact reassuring since, in Hybrid Event-B and in other approaches, among many things, we need to describe the physical world, and the physical world is as it is. Obviously, to be effective, any description of it must conform to the single existing reality. The combination of isolated discontinuous change of state, together with smoothly continuous behaviour has proved to be a useful framework in a number of formulations at the level of abstraction needed for applications.

3. Discrete Event-B

In this section we summarise discrete Event-B [3]. Event-B is characterised by proof obligations (POs) that define what consistency means for constructs, and for relationships between constructs. In keeping with a style we will follow throughout the paper, we do not quote the POs formally as we discuss various issues in the body of the paper, instead we accumulate all the POs, in Section 9, using a consistent notation, for better reference. The exception to this is when a PO of discrete Event-B needs to be modified in some way for the continuous extension. Then we quote the original form here.

3.1. Event-B Machines

Event-B consists of MACHINEs, supported by CONTEXTs. Contexts define the static data environment within which the dynamic behaviour of the machines takes place. Fig. 1 contains a context and a machine that depends on it. Contexts typically define sets and constants, the latter being any static mathematical objects needed by the machines that use them. Relationships between the objects introduced can be asserted using AXIOMS. Further properties that follow from those that are asserted may be declared in THEOREMS, which must be provable from the axioms. Furthermore, a context may extend another via an EXTENDS clause, making the entities defined there available.

An event has a STATUS field which indicates the role it plays in the development as a whole. An event may have *parameters*, declared by ANY. In general these include *inputs*, *local parameters* and *outputs*, indicated using notations $i?, l, o!$ respectively. While inputs and outputs are connected with the environment in the expected way, local parameters serve to resolve inherent nondeterminism in the event's actions. The WHERE clause gives the *guards*, which specify any constraints that the parameters have to satisfy, and any other conditions that have to hold before the event is enabled. If there are no parameters, then ANY ... WHERE is abbreviated to WHEN. The THEN clause gives the *actions* which specify the required updates to the values of the VARIABLES (i.e. specify the required change of state). Actions that update a set of variables *var* may take the most general form $var :| BApred(var, var')$, where $BApred(var, var')$ is a *before-after predicate* depending on the before-values *var* and the after-values

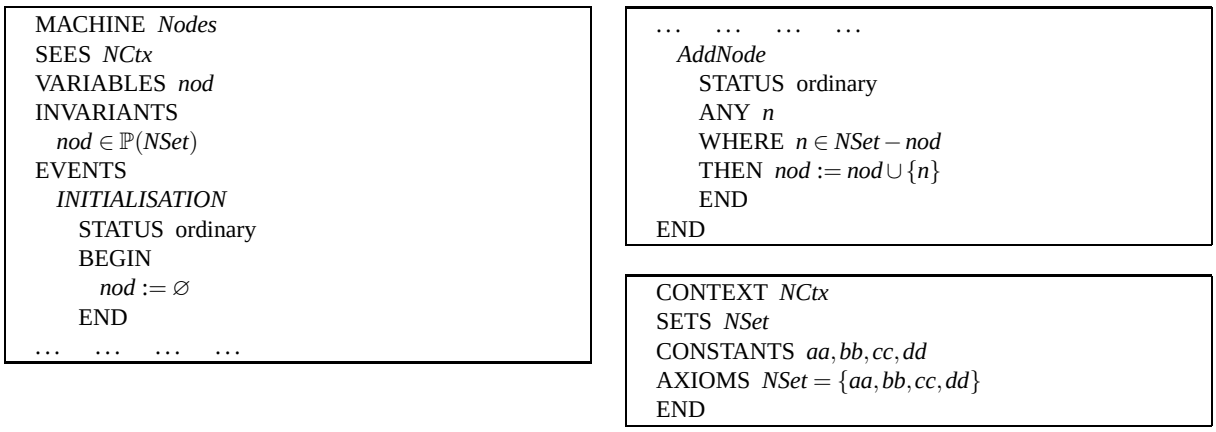

Figure 1: A simple Event-B machine, together with its context.

var', and specifying that *var* is to be updated to any after-values such that *BApred* is satisfied. There are simpler forms, e.g. $var := E(var)$, to handle straightforward assignment to the value of an expression. Among the events there is the *INITIALISATION* event, whose guard is posited to be true (indicated by the guardless BEGIN ... END syntax).

The behaviour of a machine must respect the INVARIANTS. This has a number of consequences. Firstly, the values established by the initialisation must satisfy the invariants. This is expressed formally in POs (11) and (12).

Secondly, each variable update must also preserve the invariants. Variable updates are implemented by event executions. If an event is to be executed, it must be enabled and be feasible. An event is enabled in the current state, if the event's guards are true in this state for an appropriate choice of values for the parameters. An event is said to be feasible iff, whenever in a putative before-state the invariants are true and the event's guards are also true, then there is an after-state for which the event's before-after-predicate becomes true (when evaluated with the mentioned before-state). This is expressed formally in PO (13). Furthermore, a feasible event is required to preserve the invariants. So if the invariants and the event's guards are true, and a chosen after-state makes the before-after-predicate true, then the after-state must also make the invariants true. This is expressed formally in PO (15).

For non-terminating systems, after every event, some event must become enabled. Since this is one point at which the conditions for discrete Event-B differ from those for our continuous extension, we quote the discrete Event-B PO here:

$$I(u) \Rightarrow (grd_{MoEv1}(u,l) \vee grd_{MoEv2}(u,l) \vee \ldots \vee grd_{MoEvN}(u,l)) \tag{1}$$

In (1), $MoEv1 \ldots MoEvN$ are the requisite events, with l as the parameter for each of them, and $I(u)$ is the invariant, where u is the state variable. For simplicity, we assumed that all parameter types were the same. It is possible to be more specific by separately quantifying each parameter occurrence.

3.2. Event-B Refinement

In Event-B, development progresses towards implementation via refinement. We give a small example of Event-B refinement in Fig. 2. It enhances the node set example above with a dynamically added set of node pairs, yielding a dynamically generated directed graph structure. The requirement of having directed edges between graph nodes is handled by adding a new variables, invariants and a new event *AddEdge*. Since *AddEdge* does not refine any existing event, its occurrences at runtime are considered to refine a 'notional abstract skip' event that is not present in the abstract model. Also, to prevent new

MACHINE *Nodes*

SEES *NCtx*
VARIABLES *nod*
INVARIANTS
  $nod \in \mathbb{P}(NSet)$

EVENTS
  INITIALISATION
    STATUS ordinary

    BEGIN
      $nod := \varnothing$
    END
  AddNode
    STATUS ordinary

    ANY n
    WHERE $n \in NSet - nod$
    THEN $nod := nod \cup \{n\}$
    END
END

MACHINE *Edges*
REFINES *Nodes*
SEES *NCtx*
VARIABLES nod, edg
INVARIANTS
  $nod \in \mathbb{P}(NSet)$
  $edg \in \mathbb{P}(NSet \times NSet)$
  $edg \subseteq nod \times nod$
EVENTS
  INITIALISATION
    STATUS ordinary
    REFINES *INITIALISATION*
    BEGIN
      $nod := \varnothing \ || \ edg := \varnothing$
    END
  AddNode
    STATUS ordinary
    REFINES *AddNode*
    ANY n
    WHERE $n \in NSet - nod$
    THEN $nod := nod \cup \{n\}$
    END
  AddEdge
    STATUS convergent
    ANY n, m
    WHERE $\{n, m\} \subseteq nod$
          $n \mapsto m \in NSet \times NSet - edg$
    THEN $edg := edg \cup \{n \mapsto m\}$
    END
VARIANT $\mathrm{card}(NSet \times NSet - edg)$
END

Figure 2: A refinement of the earlier Event-B machine.

events from taking permanent control at runtime, they must be 'convergent', i.e. they must decrease the $\mathbb{N}$-valued VARIANT, ensuring relative deadlock freedom.

Ensuring the proper operation of this process is a collection of POs. These cover initialisation (20) and (21), feasibility and refinement of existing events (22)-(27), and 'refinement of skip' behaviour and convergence of 'new' events (28)-(29). Finally, a machine can also contain THEOREMS, which must be provable from the facts available to the machine.

4. Continuous Behaviours

In this section, we discuss, at an appropriately informal level, a number of issues that influence the way that our extension of discrete Event-B is designed.

4.0. Discrete Event-B behaviours. The states of an Event-B machine are given by valuations of the tuple of the machine's variables, i.e. functions from the tuple of variables that yield a tuple of values. Runs of Event-B machines are given as sequences of such valuations, each valuation being generated from its predecessor by some event. Of course, this does not correspond to the real world, where time is not discrete. So when runs of an Event-B machine are intended to reflect real world behaviour, each state is deemed to persist for an appropriate interval of time, and is then superseded by its successor. So the time dependence of the state is piecewise constant. In this paper, we extend this picture to also include continuously varying behaviour, taking into account several points as follows.

4.1. Time. We model time as an interval $\mathcal{T}$ of the reals $\mathbb{R}$, with a finite left endpoint and with a right endpoint which is either finite or infinite, depending on whether the dynamics is finite or infinite, and on whether the final transition (if there is one) lasts forever or not. The values of all variables become functions of $\mathcal{T}$. In our semantics, we will allow change of state to happen both continuously, and discontinuously. The discontinuous changes are restricted to isolated time points, so that $\mathcal{T}$ partitions into a sequence of intervals, $\mathcal{T} \equiv \langle [t_0 \ldots t_1), [t_1 \ldots t_2), \ldots \rangle$, each non-empty, left-closed, right-open. Times $t_0, t_1, t_2, t_3, \ldots$ specify the coarsest partition of $\mathcal{T}$ such that all mode transitions (specifying discontinuous change, see 4.6) take place at some boundary point t_i.[5] Note that the t_i are not given *a priori* but emerge via the runtime semantics. Additionally, below, 'piecewise continuous' always means continuous on non-empty, left-closed, right-open intervals.

4.2. Variables. Variables are partitioned into two subsets: **mode variables**, which are only permitted to change discontinuously, and **pliant variables**, whose types include topologically dense sets, and which are permitted to evolve both continuously and via discrete changes. Restricting to mode variables, we recover conventional Event-B. In practice, the pliant variables take values in 'nice' subsets of $\mathbb{R}$, i.e. subsets that can be specified by simple first order constraints over $\mathbb{R}$-valued variables. This is certainly needed if the formal semantics of Section 7 is to be made precise. Still, such constraints are quite sufficient to construct many quite exotic scenarios using the usual combinators.

4.3. Limits. We consider now how discontinuities are handled. 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 $\delta > 0$ in both cases) both exist, and for every t, $x(t) = \overleftarrow{x(t)}$. (At the endpoint(s) of $\mathcal{T}$, if is needed for any purpose, any missing limit is defined to equal its counterpart.) Thus all valuations are continuous from the right and have limits from the left. This space of functions is commonly known as Càdlàg,[6] and is much used in stochastic analysis (pointing to a subsequent smooth stochastic extension of our theory).

4.4. Differentiability. In an interval $[t_i \ldots t_{i+1})$, the behaviour of every pliant variable x is given, piecewise, 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' implies two conditions. Firstly: $\phi(xs, t)$ has a Lipschitz constant which is uniformly bounded over $[t_i \ldots t_{i+1})$. Specifically, there is a constant K such that for all $t \in [t_i \ldots t_{i+1})$, we have $|| \phi(xs_1, t) - \phi(xs_2, t) || \leq K || xs_1 - xs_2 ||$. Secondly: $\phi(xs, t)$ is measurable in t. (In the preceding, $|| \, . \, ||$ denotes the $\mathcal{L}^\infty$ norm of a real vector, i.e. the maximum absolute value of any of its components.) The conditions stated for the DE $\mathcal{D}\,xs = \phi$ imply that once initial values are specified, the solution xs exists and is unique in the Carathéodory sense, and is absolutely continuous over some maximal right-open interval. (See e.g. [48] for differential equations, and [50, 41, 32] for the biimplication between *absolute* continuity and differentiability almost everywhere (amounting to the Carathéodory interpretation of DEs).)

We included the word 'piecewise' here, because, for convenience and modelling fluency, pliant variables may also be directly assigned, e.g. $xs := E$. (See Section 5.) The expression E is constrained to yield piecewise absolutely continuous behaviour for xs during a left-closed right-open interval $[t_i \ldots t_{i+1})$. Thus, although a DE will yield absolutely continuous values during $[t_i \ldots t_{i+1})$, a direct assignment may have isolated discontinuities coming from the nature of E and not from machine $\mathcal{M}$'s mode events.

4.5. Zeno. We desire a constant δ_{Zeno}, such that for all i that are relevant, $t_{i+1} - t_i \geq \delta_{\mathsf{Zeno}}$. We say 'desire', since Zeno properties are extremely hard to establish statically, usually requiring a full knowledge

[5] Various approaches to hybrid system and timed automaton semantics take varying views on the closedness/openness of the intervals dividing up real time. All can be related to one another, modulo some low level technical details.

[6] From the French: continue à droite, limite à gauche.

of the dynamics. Moreover, in idealised modelling situations, Zeno behaviour may be tolerable, even if it is always unphysical in reality. Still, it would typically pose problems for mechanical calculation.[7]

4.6. Transitions. With the distinction between mode and pliant variables, there is a distinction between **mode transitions** and **pliant transitions**. Mode transitions are just conventional Event-B transitions, recording a discrete transition from before-values to after-values of some subset of (mode and pliant) variables, specified syntactically by an Event-B **mode event**.

Pliant transitions record piecewise continuous behaviour of some pliant variables during an interval $[t_i \ldots t_{i+1})$. Since any such interval is only determined at runtime, values t_i and t_{i+1} are unknown statically. So we introduce two generic constants, $\mathbb{t}_{\mathrm{L}}$ and $\mathbb{t}_{\mathrm{R}}$, to refer to the start and end of any such interval, both in the concrete syntax of the system definition, and in our discourse about its behaviour.

Pliant transitions are syntactically specified by **pliant events**. A pliant event can specify the initial conditions that have to hold for the pliant variables. It can also specify other guard conditions needed for the enabledness of the pliant transition (typically concerning mode variables). It also specifies the DE to be obeyed (subject to the conditions in 4.4).

As an alternative to writing a differential equation, if the required continuous behaviour is directly known, then it may be directly assigned to the pliant variable instead of writing a corresponding DE. Obviously this is very convenient, but to avert the pathologies inherent in mere continuity,[8] we insist that such continuous behaviours should also be piecewise absolutely continuous solutions to well posed initial value problems. One consequence of allowing direct assignments, is the possibility of discontinuities in the pliant variable behaviour being defined during $[\mathbb{t}_{\mathrm{L}} \ldots \mathbb{t}_{\mathrm{R}})$, as noted in 4.4.

Additionally, any further constraints that need to hold while the pliant transition runs can be specified within the pliant event. Parameters may be introduced in a pliant event. Their syntactic scope is the whole of the pliant event, and at runtime, they refer to functions of time over the interior of the relevant time interval, $(\mathbb{t}_{\mathrm{L}} \ldots \mathbb{t}_{\mathrm{R}})$). Inputs and local parameters should have the same properties as pliant variables. So they should be piecewise absolutely continuous solutions to well posed initial value problems.

4.7. Syntactic aspects of time. The semantic aspects of time must be connected with the syntax of events. Because of its special properties, i.e. as a read-only variable, the time variable must be declared as such. It is necessary to declare the initial value of $\mathcal{T}$, most conveniently done in the *INITIALISATION*. We also admit *clocks*. A clock, by definition, increases at the same rate as time during every pliant event (i.e. its time derivative is 1), so this property need not be mentioned in the syntax. Clocks can be updated in mode events. More exotic clocks can be implemented using normal pliant variables.

4.8. Interpretation of mode events. In discrete Event-B, an event describes how two successive valuations in a run are related. In Hybrid Event-B, if the mode transition is regarded as taking place at time t_q, then the before-values are normally interpreted as the left limits of the valuations at t_q, and the after-values are the right limits (which equal their values at t_q itself). Note that the parameters are regarded as being defined only at the time t_q itself, so do not possess limits.

The exception to 'normally' occurs when a pliant variable undergoes a discontinuity (at time t_q say) arising from a direct assignment (as in 4.4 and 4.6), and the after-value of the discontinuity enables the mode event (whether the before-value does so or not). Then, to aid fluency in modelling, particularly of edge-triggered phenomena, the discontinuity after-value plays the role of mode event before-value, the

[7] Our approach contrasts with many other approaches to the Zeno problem, which demand that any finite time interval contains only a finite number of transitions, or that the sequence of discrete transition times contains no accumulation points. But this still permits the sequence of times specified by $t_{i+1} - t_i = 1/i$, which, while satisfying the mentioned restrictions, nevertheless allows the t_i to get arbitrarily (and thus unphysically) close together.

[8] See standard texts on mathematical analysis, e.g. [42, 31, 27].

mode event executes at t_q, and variable values at t_q become as specified for after-values in the assignments of the mode event.

4.9. Interpretation of pliant events. As noted already, there are two ways of specifying pliant behaviour: via a DE, and directly. In both cases, the right hand side of the DE or assignment, may contain discontinuities. In the DE case, $\mathcal{D}\,xs = \phi$, the Carathéodory interpretation integrates over any discontinuity in ϕ, yielding behaviour that although absolutely continuous, is *nonsmooth*. See e.g. [19, 20].

In the direct assignment case, $xs := E$, any discontinuity in E remains visible in xs. Piecewise absolute continuity of E thus yields piecewise absolute continuity of xs. The interaction of such discontinuities with the enabling of mode events requires care, as already noted. If the discontinuity after-value enables the mode event, then the discontinuity after-value is superseded by the mode event after-value. (N. B. We deliberately disregard the case where the discontinuity before-value enables a mode event but the discontinuity after-value doesn't.)

The solution to a DE gives rise to its transition relation Q. For an interval such as $[\mathbb{t}_{\mathrm{L}} \ldots \mathbb{t}_{\mathrm{R}})$, for $t \in (\mathbb{t}_{\mathrm{L}} \ldots \mathbb{t}_{\mathrm{R}})$, $Q(\mathbb{t}_{\mathrm{L}}, t)$ is a t-indexed set of before-/after-value pairs, relating the valuation at time $\mathbb{t}_{\mathrm{L}}$ to the valuation at time t. This gives perhaps the closest correspondence to the before-after picture familiar from the discrete world. For direct assignments, the picture is exactly the same; any discontinuities encountered are not visible (as such) in the individual $Q(\mathbb{t}_{\mathrm{L}}, t)$ pairs of values.

Although beyond the scope of this paper, an additional benefit of the formalism described, arises in multi-machine systems. There, a mode transition in one machine may be sensed as a kink or discontinuity during pliant behaviour in another machine which does not experience a mode transition at the same time.

4.10. Mode and pliant event interleaving. In 4.0 we indicated that discrete Event-B transitions were isolated from each other in time, and that we want to preserve this picture in Hybrid Event-B. Consequently, pliant transitions and mode transitions must alternate. To ensure this, we stipulate that both kinds of events are feasible, and that at run time, each kind of transition enables the other kind. Therefore a Hybrid Event-B run ought to have the following properties, where we assume that the machine contains an *INITIALISATION* mode event to start a system run.

- Every enabled mode transition is feasible, i.e. has an after-state, and on its completion enables a pliant transition (but does not enable any mode transition).[9,10] (2)
- Every enabled pliant transition is feasible, i.e. has a time-indexed family of after-states, and EITHER: (3)
  - (i) During the run of the pliant transition a mode transition becomes enabled. Such a mode transition preempts the pliant transition, and defines the end of its family of after-states. ORELSE
  - (ii) During the run of the pliant transition it becomes infeasible, i.e. for some point in time, all the conditions stipulated cannot be satisfied simultaneously — finite termination. ORELSE
  - (iii) The pliant transition continues indefinitely — non-termination.

It is clear from (2), (3) that the time points t_i for a given run emerge at runtime. The construction of a given system trace thus proceeds piece by piece, determining the t_i as it goes. The set of successfully constructed system traces will constitute the semantics of the system. See Section 7 for details.

4.11. Preemption. In (3), and in earlier discussion, it is clear that as soon as a mode event becomes enabled, it preempts the current pliant event. This *eager* scheduling of mode events in Hybrid Event-B is the sharpest departure from discrete Event-B, since discrete Event-B schedules events *lazily*, as noted in 4.0. The difference is motivated by physical law, which is so relevant to the systems for which Hybrid

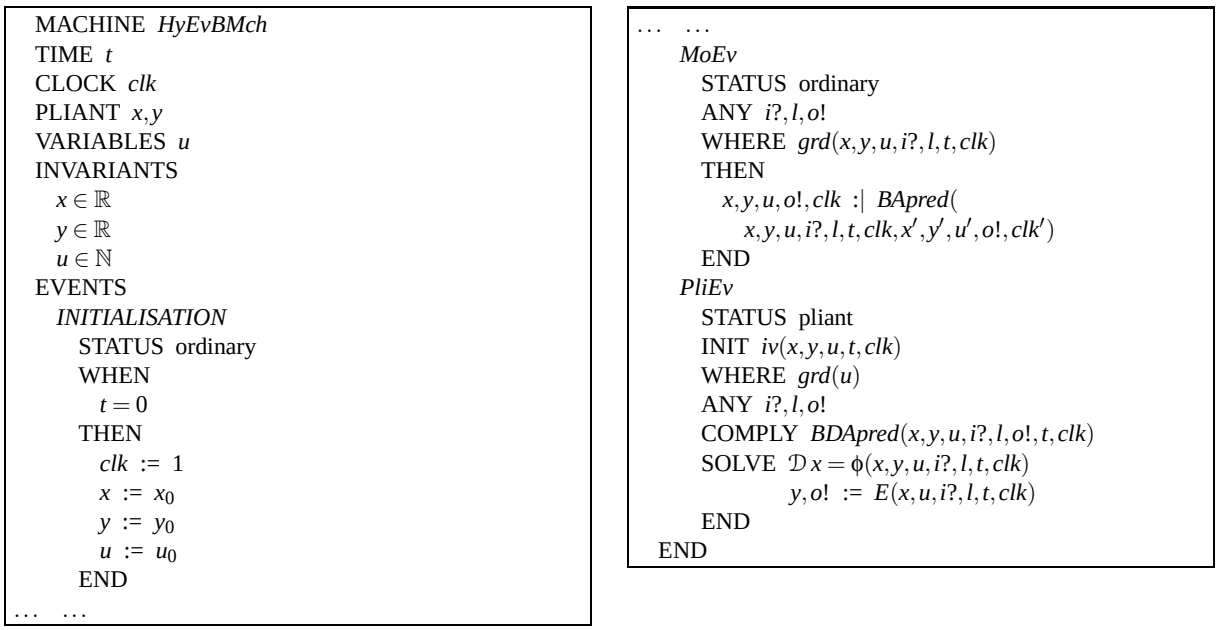

Figure 3: A schematic Hybrid Event-B machine.

Event-B is intended. Physical laws are all eager: e.g. a falling bouncing ball, when it hits a horizontal surface, does not have any choice about when to bounce up again; see Section 10.1 for more discussion.

5. Syntax of Core Hybrid Event-B Machines

Fig. 3 shows the elements of a Hybrid Event-B machine. After the machine name is the TIME declaration, which names the variable used to denote real time (if needed). This permits read-only access to time in the rest of the machine. Time is synchronised (via a WHEN clause) with the start of a run in the *INITIALISATION*. Next comes a CLOCK variable *clk*. This allows the restrictions discussed in Section 4.7 to be enforced. Then come the PLIANT and VARIABLES declarations. The former introduces the pliant variables, while the latter introduces the mode variables.

Next come the INVARIANTS. Where these declare typing information, the conventions used in discrete Event-B are extended to Hybrid Event-B in that the type of a pliant variable such as x or y in Fig. 3 is the set of values that it can take *at any given moment of time* (specifically $\mathbb{R}$ in the case of x,y).[11] Other invariants may be written as usual. The fact that time dependence is *not* part of the type of any variable, means that an occurrence of a variable in an invariant necessarily refers to its current value, which is at an arbitrary time during a system run. Consequently any invariant expression written in the INVARIANTS section has to be true *at all times* during a system run.

[9] We deliberately forbid successive mode transitions to occur, as is permitted in some alternative frameworks. This prevents the semantics of a 'mode event cascade' having to be defined via a fixed point calculation, and permits the characterisation of system traces as functions of time for each variable.

Regarding interfacing between continuous and discrete behaviour, it is helpful to have the discrete behaviours described in straightforward before-after terms. Such specifications can subsequently be refined to sequential code by conventional means, outside of, and below the level of abstraction of the present formalism.

[10] If a mode event has an input parameter, to facilitate simpler modelling, the semantics assumes that its value only becomes available at a time strictly later than the previous occurrence of a mode event, ensuring part of (2) automatically.

[11] In particular, in our formulation, the type of a pliant variable such as x is **not**, for example, $\mathbb{R}^+ \to \mathbb{R}$ (as it would be in some related formalisms), i.e. the time dependence is **not** mentioned explicitly in the type.

Then come the EVENTS, starting with the *INITIALISATION*. The STATUS of *INITIALISATION* is 'ordinary' — for simplicity, existing event status designations are taken over from discrete Event-B for mode events. The initial value of real time is synchronised to the initial state of the machine in the WHEN clause. Real time is read-only; it is never assigned. Other variables are assigned their initial values as usual, including assignment of initial values to clocks. If a nondeterministic initial assignment to some variables is needed, it can be achieved via the usual ANY ... WHERE ... THEN ... mechanism.

Then come the remaining mode events and pliant events. Mode events are as in discrete Event-B, aside from timing details, discussed in Sections 4.8 and 6.13. A mode event *MoEv* appears in Fig. 3.

Pliant events need new syntax. As mentioned in Section 4.6, pliant variables can be assigned values either via the solution of a DE, or directly by being assigned the value of a (time dependent) expression, or indeed by being assigned a value consistent with some (time dependent) predicate. We have a flexible syntax that accommodates all these possibilities.

A schematic pliant event is shown in *PliEv*. It starts with a new status declaration, 'STATUS: pliant', on which the remaining new syntax depends. Next come two guard clauses. The INIT guard specifies initial constraints that must hold concerning the pliant variables (and, if any, constraints that mix mode and pliant variables), and the WHERE guard specifies initial constraints that must hold concerning mode variables alone.[12]

The ANY clause introduces parameters $i?, l, o!$, satisfying the restrictions mentioned in Section 4.6. As with mode events, if there is no ANY clause, the WHERE clause can be renamed WHEN.

The COMPLY clause defines a before-during-after-predicate $\mathit{BDApred}(x, y, u, i?, l, o!, t, \mathit{clk})$. The *BDApred* predicate defines conditions that must hold for the duration of any pliant transition specified by *PliEv*. If *BDApred* is sharp enough, or *PliEv* is being specified in a sufficiently loose manner, then *BDApred* alone may be enough to specify the behaviour required of *PliEv*. As an expressiveness metaphor for the convenience of modellers, we allow pliant variables in COMPLY clauses to refer to time explicitly. Thus we permit occurrences of terms like '$\Xi(y(ex), \ldots)$', where Ξ is a predicate, y is a pliant variable and *ex* is an expression that evaluates to a time between $\mathbb{t}_{\mathrm{L}}$ and t.

Otherwise, the behaviour of the pliant variables during any pliant transition specified by *PliEv* may be further constrained by the SOLVE clause. This can contain DEs and direct assignments of pliant variables and outputs. The form of any DE in the SOLVE clause is required to be in general first order form, $\mathcal{D}x = \phi$, as discussed earlier, guaranteeing existence and uniqueness via standard machinery [48].

A direct assignment $y, o! := E$ is acceptable provided E is a piecewise absolutely continuous function of its piecewise absolutely continuous parameters. In that case, direct assignment is equivalent to solving $\mathcal{D}y, \mathcal{D}o! = \mathcal{D}E$, where the solution is reinitialised at points of discontinuity of E, and provided that it yields a consistent solution.

Although we are quite precise about the structure and meaning of SOLVE clauses, we are less prescriptive about the COMPLY clause (although, in practice, it will typically consist of straightforward algebraic constraints on the variables). To see why, consider a COMPLY clause like $x \in [0 \ldots 1]$. Unlike discrete Event-B, this specifies a time indexed family of assignments of values to $x(t)$ for all $t \in (\mathbb{t}_{\mathrm{L}} \ldots \mathbb{t}_{\mathrm{R}})$. Without any further restriction, this allows the function $x(t)$ to vary uncontrollably, despite the extreme simplicity of the constraint $x \in [0 \ldots 1]$. To address this, we stipulate that of all the functions of t that the bare *BDApred* in the COMPLY clause admits for the pliant variables, we consider only those that are piecewise absolutely continuous for $t \in [\mathbb{t}_{\mathrm{L}} \ldots \mathbb{t}_{\mathrm{R}})$. Thereby, we restrict the pliant variable behaviours mentioned in the *BDApred* to the same class of time functions that are specifiable using the earlier DE and direct assignment forms.

To aid modelling fluency, we define two further constructs permitted to occur as top level conjuncts

[12] There is no evident reason why initial constraints on mode and pliant variable might be separated, but it proves useful later.

in COMPLY clauses: skip and *INVARIANTS*. The former specifies constant behaviour, while the latter allows arbitrary piecewise absolutely continuous behaviour, provided the machine's invariants are respected. Both constructs can be used to specify behaviour for pliant variables not otherwise constrained in the event. To further simplify model description, when at least one of the COMPLY or SOLVE clauses contains non-trivial content, COMPLY *INVARIANTS* is understood to apply to any pliant variables whose behaviour is not specified in these clauses. So COMPLY *INVARIANTS* only needs to be written when both the COMPLY and SOLVE clauses have no (other) content. However, we insist that COMPLY skip must always be written when needed, since it defines specific behaviour.

In total then, the set of permitted behaviours for the pliant variables defined by a pliant event, consists of the intersection of those permitted by the COMPLY clause and those permitted by the SOLVE clause.

As already mentioned, in the absence of a SOLVE clause, the COMPLY clause can serve as an implicit specification of the required behaviour. This makes it very useful for specifying behaviours that have to obey global (though potentially time-dependent) constraints, without committing to any specific dynamics. We call such specifications *pliant envelopes*.[13]

Overall machine consistency requires that we check various properties of a Hybrid Event-B machine. Fortunately, a good portion of these are taken care of already in the purely discrete Event-B framework, and we have commented on them in Section 3. What remains are POs relevant exclusively to pliant events, and to the interaction between mode and pliant events.

Turning to the pliant event POs, pliant events firstly have to be feasible. This means that at a presumed starting time $\mathbb{t}_\mathrm{L}$, given that the invariants hold and the *iv* and *grd* clauses of the pliant event also hold, then for some duration of the pliant event defined by $\mathbb{t}_\mathrm{R} > \mathbb{t}_\mathrm{L}$, for all times $t \in (\mathbb{t}_\mathrm{L} \ldots \mathbb{t}_\mathrm{R})$, values for the variables exist, that satisfy the specification of the pliant event, i.e. that the COMPLY and SOLVE clauses are satisfied. The formal PO is (14).

Pliant events have to preserve the invariants. Thus, if at $\mathbb{t}_\mathrm{L}$ we have the invariants, and in the interval to $\mathbb{t}_\mathrm{R}$ a behaviour of the system satisfies the COMPLY and SOLVE clauses, then that behaviour must also satisfy the invariants throughout this interval. The formal PO is (16).

Note that a subtlety arises concerning the failure of invariants and *BDA* predicates. If an invariant ever fails during the construction of a system trace, then that trace is abandoned; failure of invariants is not permitted. However, if a *BDA* predicate fails during the construction of a system trace, it simply indicates that the pliant transition in question has become infeasible. Such infeasibility just indicates finite termination if no mode event became enabled during the course of the transition, c.f. (3).

Machine well-formedness is concerned with the expected alternation between mode and pliant transitions in a run. In going from a mode transition to a pliant transition, we demand that in any mode transition after-state, no mode event guard is true for any choice of parameter, but that some pliant event guard is true. The formal PO is (17). Conversely, in going from a pliant transition to a mode transition, we demand that no mode event is ever enabled *during* the transition, but that **either** the values of the variables at the endpoint $\mathbb{t}_\mathrm{R}$, *do* enable some mode event for some parameter, **or** the left limits at $\mathbb{t}_\mathrm{R}$ enable a mode event in case values at $\mathbb{t}_\mathrm{R}$ do not exist.[14]

We still have to be careful though. A final pliant transition runs forever or till it becomes infeasible. If we require such a final pliant transition in the system, for the relevant proof obligation to be effective (i.e. to not fail on final pliant transitions), we need to know statically which pliant events are supposed

[13] In [38] and in other works by Platzer, such specifications are called *differential invariants*. In the context of Event-B, where the word 'invariant' has strong connotations with literally time independent properties, we prefer an alternative terminology, to avoid potential misunderstanding.

[14] Observe that this definition handles the pliant/mode issue of Sections 4.8 and 4.9. If a pliant behaviour is continuous at $\mathbb{t}_\mathrm{R}$ then both options are equivalent. If there is a discontinuity at $\mathbb{t}_\mathrm{R}$, then presuming all discontinuities are right continuous (see Section 4.0), the correct value is used for the mode event guard. Otherwise, the left limit must be used.

to be final and which are not. For this purpose we introduce a new status tag for final pliant events, 'STATUS: pliant final'. This declares the tagged event as a final one and prevents the relevant check being demanded of it. See (18) for the formal PO.[15]

6. Further Technical Considerations

In this section we discuss some additional technical issues regarding Hybrid Event-B machines.

6.12. Mode event guard closure. Suppose expression $x < 3$ occurs in the guard of a mode event *MoEv*, where x is a pliant variable. Suppose x behaves as $x(t) = 4 - t$ during a pliant transition, where t starts at 0. Eventually, *MoEv* will become enabled, but since there is no 'earliest time *immediately* after $t = 1$', *MoEv* cannot execute at an identifiable time unless we replace $x < 3$ in the guard by $x \leq 3$, which becomes true exactly at $t = 1$ in our example. However, the negation of $x \leq 3$ is $x > 3$, which resurrects the problem. Our solution is to **allow** expressions like $x < 3$ in mode event guards, but to **interpret** them at runtime via the topological closure of the regions they define when constructing system traces. This interpretation ensures that mode transitions occur at specific times, but also allows mode events with non-overlapping guards (e.g. guards such as $x \leq 3$ together with $x > 3$, or more symmetrically, $x < 3$ together with $x > 3$) to be easily defined for more fluent modelling and reasoning purposes. In the semantics of Section 7, we restrict to pliant variables whose values are in (subsets of) $\mathbb{R}$. For such variables, we need merely to replace strict inequalities by nonstrict ones in determining guard closure.

We accept that adding such boundary values into mode event guards may give rise to pathological counterexamples in which the trajectory does not satisfy event definitions, or invariants, *as written*. However, we claim that these will have little impact in practice, since for the kind of engineering applications we envisage, the dynamics has to be locally stable in order to be useful. So, a small disturbance to trajectory data must have a relatively small effect on the trajectory, at least within some time range (the acceptable limits on such disturbances being highly application dependent). The chief thing is that reasoning about the system model allows the maintenance of the invariants to be proved, since these express what is important about the system. Provided any pathological behaviour permitted by the operational semantics arises from a disturbance set of measure zero, we can ignore it for practical purposes.

6.13. Event parameter availability. In early versions of discrete Event-B, any parameters needed by an event were simply assumed available, a natural view when parameters merely resolved nondeterminism. However, in more recent versions incorporating code generation, parameters can also be input parameters (decorated with ?), or output parameters (decorated with !); local parameters are written undecorated, as before. Considering that in discrete Event-B *all* connections with real time are neglected, the issue of *when* any parameter might become available does not really arise.

However, in Hybrid Event-B the issue needs more thought, because of the presence of real time. There are two design decisions to be made, one for mode transitions and the other for pliant transitions.

For mode transitions, we stipulate that input parameters become available at some time which is *strictly greater* than the time at which the most recent preceding mode transition occurred. At that moment, nondeterminism is resolved by choice of local parameters, and output parameters are calculated using the event's *BApred*. The strict inequality prevents runs contravening the condition in (2), that forbids a mode transition from immediately enabling another mode transition, and avoids the need to complicate mode event guards to achieve this effect. This mechanism also gives a convenient way of modelling stimuli from the environment that arise spontaneously (from the model's viewpoint).

[15] Restricting to statically knowable final pliant events theoretically constricts computational expressivity, but does so in way that can only be regarded as beneficial from an engineering standpoint.

For pliant events, we stipulate that all required parameters are available immediately that values exist (in the sense of existential quantification), that would enable the event, regardless of whether the event is then scheduled for execution. In practice, since pliant transitions occupy extended periods of time, their parameters will also need similar durations, so will most likely be held in permanent elements of any actual implementation. However Hybrid Event-B makes no assumptions about this and only assumes that parameters are available during transitions themselves.

6.14. Invariant checking. In modelling a system in which some physical attribute is to be confined to some region, the simplest approach is to define an invariant that confines the relevant variable to that region. Then enough events should be designed to ensure the invariant is maintained.

Often, mode events are involved in maintaining the invariant, having guards stating that the dynamics is at the boundary of the region, and with actions that cause a suitable change of course. This raises a technical niggle for the semantics.

In determining the trajectory of a pliant transition, the semantics first looks for the maximal interval within which the pliant event specifies a consistent dynamics. Only then is the next preemption point sought. In the situation we are discussing, the dynamics will therefore usually overshoot the desired region's boundary (breaking the invariant) before the discovery of the next preemption point. It is thus important that the invariant is not checked before the next preemption point has been found.

As modelling descends towards implementation, we would normally expect there to be some tolerance between the true region boundary and a mode event guard's view of it, to allow for quantization errors and similar effects.

6.15. *BDApred* $\mathbb{t}_{\mathrm{R}}$ **left-limits.** The considerations that made us impose a closure interpretation on mode event guards, and the remarks in 6.14, have implications also for the *BDA* predicates of pliant events. In the earlier description, a pliant event gave rise to a transition whose duration was a left-closed right-open interval $[\mathbb{t}_{\mathrm{L}} \ldots \mathbb{t}_{\mathrm{R}})$, its right endpoint being determined by the next preemption point, otherwise being determined by infeasibility beyond $\mathbb{t}_{\mathrm{R}}$. To maximise simplicity of modelling, we allow preemption to be defined by the truth of a mode event guard for variable values which: **either** arise in the interior of a piecewise absolutely continuous evolution, **or** arise as *finite* limits at $\mathbb{t}_{\mathrm{R}}$ in case the *BDApred* defining feasibility is not true beyond $\mathbb{t}_{\mathrm{R}}$.[16]

7. Formal Semantics

In this section we describe the formal semantics of Core Hybrid Event-B. In order to not waste space on repeating routine material, we rely extensively on existing work. We rely on [3] (especially Chapters 5, 9, 14) for the semantics of discrete Event-B; and on [48] (especially Chapter III §10) for differential equations in the sense of Carathéodory.

In this paper we define the semantics of a single Hybrid Event-B machine $\mathcal{M}$. For simplicity, the semantics performs several checks at runtime. In a practical system, most of this would be avoided by imposing syntactic tests, which would provably guarantee the runtime semantics (see Section 9.15).

We turn to the semantics itself. Firstly, we make precise a few points of terminology and convention.

- Time, referred to as t, takes values in the real left-closed right-open set $[t_0 \ldots +\infty)$, where t_0 is an initial value for time. For every other system variable *var*, there is a type U^{var}. If *var* is pliant, then U^{var} is $\mathbb{R}$. (4)

[16] Note that the latter case precludes the occurrence of a discontinuity at $\mathbb{t}_{\mathrm{R}}$.

- Time is a distinguished variable (read-only, never assigned by events, and synchronised with the machine during *INITIALISATION*). All state variables have interpretations which are functions of an interval of time starting at t_0; see (7). As well as directly referring to the time variable, time may be handled indirectly by using clock variables (declared as such), whose values may be reset by mode events. (5)
- The events of a machine $\mathcal{M}$ consist of mode events and pliant events. Given a valuation of all the state variables, inputs and local parameters, and time, a mode event is *enabled* iff the valuation lies in the topological closure of the set of tuples of values in which the WHERE clause of the event evaluates to true. Given a valuation of all the state variables, and time, a pliant event is *enabled* iff the INIT and WHERE clauses evaluate to true. (6)
- The semantics of $\mathcal{M}$ is a set of system traces $\mathcal{S}$. Each system trace $S \in \mathcal{S}$ is given by a time interval $\mathcal{T} = [t_0 \ldots t_{\text{FINAL}})$ (where t_{FINAL}, with $t_{\text{FINAL}} > t_0$, is finite or $+\infty$), and a set of time dependent variable interpretations $\zeta_{var} : \mathcal{T} \to U^{var}$, one for each state variable *var*. If $\mathcal{S}$ is empty we say that the semantics of $\mathcal{M}$ is VOID. (N. B. For reasons of simplicity, we omit inputs, local parameters and outputs from system traces. These are regarded as existing only for the duration of the transitions that they belong to; i.e., the single time value at which a mode transition occurs, or the topological interior of the interval during which a pliant transition takes place. With additional machinery, such parameters could be included in system traces.) (7)
- The set of traces $\mathcal{S}$ is constructed by the step by step process below, which describes how individual system traces are constructed incrementally.[17] Whenever a CHOOSE is encountered, the current trace-so-far is replicated as many times as there are different possible choices, a different choice is allocated to each copy, and the procedure is continued for each resulting trace-so-far. Whenever a TERMINATE is encountered, the current trace-so-far is complete and is added to the semantics $\mathcal{S}$, of $\mathcal{M}$. Whenever an ABORT is encountered, the current trace-so-far is abandoned (and eliminated from $\mathcal{S}$). If a VOID is encountered, the semantics is VOID. (8)

The construction of system traces is as follows.

[1] Let $\eta := 0$ (where η is a meta-level variable).

[2] Assuming the *INITIALISATION* is feasible, CHOOSE an initial assignment to all variables satisfying all the invariants of $\mathcal{M}$, thereby interpreting their values at time t_0. Otherwise, VOID.

[3] If any non-*INITIALISATION* mode event that does not have any inputs (but which may have local parameters or outputs), is enabled when the state variables have the values at t_η and enabling values exist for the local variables, then ABORT.

[4] With the state variables having the values at t_η, CHOOSE an enabled pliant event *PliEv* provided there is one, else ABORT.

[4.1] Considering all occurrences of differential equations and direct assignments in the SOLVE clause of *PliEv*, if any pliant variable *pli* appears in the left hand side of more than one occurrence then ABORT.

[5] If there does not exist a $t_{\text{MAX}} > t_\eta$ such that there is a simultaneous piecewise absolutely continuous solution of all the differential equations and direct assignments in the SOLVE clause of *PliEv* in the left-closed, right-open interval $[t_\eta \ldots t_{\text{MAX}})$, using state variable values at t_η as initial values, with these initial values required to satisfy the INIT and WHERE guards of *PliEv*, and with inputs and local parameters where needed, such that the *BDApred* (including any implicit *INVARIANTS* constraint) in the COMPLY clause of *PliEv* in the interval $(t_\eta \ldots t_{\text{MAX}})$ is satisfied, then ABORT.

[17] N. B. The process is not intended to be an executable sequential procedure. All traces-so-far are intended to be explored simultaneously and to completion, even if completion involves an infinite amount of time for a non-terminating system trace.

[6] Otherwise, CHOOSE a simultaneous solution as in **[5]**, let t_{MAX} be maximal such that the properties in **[5]** hold, and use the solution to assign the values of all pliant variables (and outputs) in the interval $[t_\eta \ldots t_{\mathrm{MAX}})$.

[6.1] For every mode variable, extend its value at t_η to a constant function in the interval $[t_\eta \ldots t_{\mathrm{MAX}})$.

[7] If no non-*INITIALISATION* mode event is enabled by the values of the state variables at any time t_{NEXT} in the open interval $(t_\eta \ldots t_{\mathrm{MAX}})$ (including left-limit at t_{MAX} itself), together with a choice of values for inputs and local parameters, then if the invariants of $\mathcal{M}$ are not satisfied in the open interval $(t_\eta \ldots t_{\mathrm{MAX}})$, then ABORT. Otherwise TERMINATE.

[8] CHOOSE $t_{\eta+1} > t_\eta$ such that **either** $t_{\eta+1}$ is the earliest time at which a non-*INITIALISATION* mode event without inputs (but potentially having suitably chosen local parameters) is enabled according to the criteria in **[7]**, **or** a non-*INITIALISATION* mode event having inputs is enabled (with a suitable choice of inputs and local parameters) according to the criteria in **[7]** at $t_{\eta+1}$ and there is no non-*INITIALISATION* mode event without inputs that is enabled according to the criteria in **[7]** at any time between t_η and $t_{\eta+1}$.

[9] If the invariants are not satisfied in the open interval $(t_\eta \ldots t_{\eta+1})$, then ABORT.

[10] Let $\eta := \eta + 1$.

[10.1] Let *MoEvs* be the set of non-*INITIALISATION* mode events that are enabled when all state variables *var* are interpreted as their values $var(t_\eta)$ at t_η (or their left-limit values $\overrightarrow{var(t_\eta)}$ at t_η if $t_\eta = t_{\mathrm{MAX}}$), and suitable values are chosen for inputs and local parameters where needed.

[10.2] CHOOSE an enabled event from *MoEvs*, and an assignment to all state variables and outputs according to its *BApred*, such that all the invariants of $\mathcal{M}$ are satisfied, thereby (re)interpreting those variable values at time t_η. Otherwise ABORT.

[10.3] For any other state variable *var* without a value at t_η, interpret its value at t_η as its left-limit at t_η, i.e. as $\overrightarrow{var(t_\eta)}$, provided this is finite. Otherwise ABORT.

[10.4] Discard the interpretation of all state variables in the open interval $(t_\eta \ldots t_{\mathrm{MAX}})$, where t_{MAX} is the value determined in **[6]**. (If $t_\eta = t_{\mathrm{MAX}}$ then the interval is empty.)

[11] Goto **[3]**.

Regarding the soundness of the above construction, since we can take some basic things like mode event update semantics and the semantics of the existence of solutions to differential equations for granted, the key remaining issue is whether the handover from pliant to mode transitions, and from mode to pliant transitions, is well defined.

We observe that the handover from pliant to mode transitions is trouble-free as follows. Consider first, mode events without inputs. Since the set of values at which the WHERE guard of any such mode event is interpreted is closed (by (6)), then this set, with dependence on local parameters existentially quantified away, is also closed. Then, since the system trajectory is a piecewise continuous function during any interval in which a pliant rule is active, if the system trajectory meets the quantified closure at all during such an interval, it first meets it at some specific time point. (This happens regardless of whether the time point occurs in the interior of the interval or at its end, and takes into account our earlier discussion of discontinuities.) In both of these cases the time $t_{\eta+1}$ will be strictly greater than t_η, since the test in **[3]** has earlier been passed, by assumption. Since there are only finitely many rules, the minimum of such time points across all of the rules to which these considerations apply, is a unique well defined time point $t_{\eta+1} > t_\eta$ at which the pliant transition is to be preempted — if it is to be preempted by a mode event without inputs.

Secondly, consider mode events with inputs. Point **[8]** stipulates that $t_{\eta+1}$ is to be chosen so that $t_{\eta+1} > t_\eta$ is satisfied, in line with remarks in Section 6.13. Thus, even though a mode event with inputs

can have its WHERE guard satisfied by state variable values (plus inputs and local parameters) at time t_η (since such a situation is excluded from causing an ABORT in **[3]**), in **[8]** t_η is never selected as preemption point. Apart from this, mode events with inputs can cause the selection of preemption point at any time at which their WHERE guard is satisfiable, provided this is not later than a preemption point that could be selected according to the first case. With a preemption point selected, a consistent set of mode updates can be derived, by **[7]**, **[10.1]**, **[10.2]**.

Note the careful wording in **[10.2]**. **If** a machine has a mode event without inputs, *MoEvX* say, enabled at t_η, **then** the machine has to execute *some* mode event at t_η (to comply with the remarks in Section 4.11), but the event does not have to be *MoEvX*. The same does not apply to mode events with inputs (that would be enabled at t_η if inputs were supplied). The semantics has the option of simply not supplying the required inputs at t_η.

We argue that the handover from mode to pliant transitions is also consistent. Upon completion of a mode transition, some pliant events will (typically) be enabled, **[4]**, required to be unambiguous and consistent by **[4.1]**. One can then be selected to run **[5]**, **[6]**, in an ensuing nonempty interval.

With suitable attention to routine details, the above remarks can be turned into a formal proof of the consistency of the definition of system traces. The alternation between mode and pliant transitions is a structural feature that can be policed by proof obligations that enforce a static version of these constraints. These new POs, specific to Hybrid Event-B, are given in (17) and (18).

We observe that for pliant transitions, the invariants are checked only after their endpoint has been determined, in line with remarks in Section 6.14. Only the open interval $(t_\eta \ldots t_{\eta+1})$ needs to be checked since variable values at t_η are confirmed to satisfy the invariants during the preceding mode event.

The above semantics, although for a single machine, is still an *open* semantics in that outputs are delivered to the environment, and inputs are accepted from the environment provided they are piecewise absolutely continuous. Such inputs might be produced by some other Hybrid Event-B machine outside the discourse, and, specifically, might themselves have isolated discontinuities. However, our interpretation of direct assignment and use of the Carathéodory interpretation of differential equations ensures that a well defined meaning is available.

Definition 7.1. *A Hybrid Event-B machine $\mathcal{M}$ is said to be non-void iff its semantics is not* VOID*, i.e. its set of system traces $\mathcal{S} \neq \varnothing$. It is said to be correct iff it is non-void, and also, during the construction of its semantics, no* ABORT *is ever encountered.*

8. Refinement

It is desirable that as far as possible Hybrid Event-B refinement should add to, rather than modify, the existing notion of refinement in discrete Event-B. Seeking to fulfil this aim restricts the design of Hybrid Event-B refinement quite severely. This has the benefit of limiting the complexity of the POs that capture the new notion, making it more practicable and useful.

We base our design on two principal assumptions. Firstly, we assume that in discrete Event-B, the events take place at (real world) times appropriate to the application context.[18] Secondly, we assume that in refining an abstract model *A* to a concrete model *C*, the application context remains the same, and the timings of those *C* events that are refinements of *A* events remain unaltered. Therefore, if the refinement to *C* introduced new events, the timings of occurrences of these will interleave the timings of occurrences of the events inherited from *A*.

[18] This is indeed an assumption. In discrete transition systems, the occurrence of an event, *instantly* enables any successor. That this successor does not run immediately is an interpretation that is imposed from outside the formal framework.

In Hybrid Event-B refinement we assume that time flows at the same rate in both the abstract and concrete systems. Consequently, the times at which abstract states and concrete states should be compared, in relations like the joint invariant, should be the same. Thus, relations like the joint invariant, will be required to hold at all individual times. On this basis, the coincidence of the times at which abstract and corresponding concrete mode events are deemed to occur becomes *derivable* in Hybrid Event-B.

Thus, suppose a mode event *MoEvA* becomes enabled in *A*. Then, by relative deadlock freedom for mode events, some concrete mode event *MoEvC* becomes enabled in *C*. Since the times at which the abstract and concrete states being compared in the relative deadlock freedom PO are the same, the times at which *MoEvC* and *MoEvA* become enabled are the same. Conversely, suppose a mode event *MoEvC* becomes enabled in *C*. Then *MoEvC* is either an 'old' event or a 'new' event. If it is an old event, then using guard strengthening for mode events, some abstract event *MoEvA* simultaneously becomes enabled in *A*. If it is a new event, a 'notional skip' is enabled. However, the concept of 'notional skip' acquires, in Hybrid Event-B, additional connotations, not present in discrete Event-B.

In discrete Event-B, it makes no difference whether we view a 'notional skip' as actually running or not. The point is that when an event executes (in general, changing the machine state), a choice point is generated for the scheduler to select the next enabled event to run. However, if the event that ran was a skip, the choices available remain the same as before, since the state has not changed. So running or not running a skip event has no influence on the scheduler.

In Hybrid Event-B though, in between the mode transitions, pliant transitions run. Now, it makes a difference whether we view a notional skip as actually executing or not. If it executes, then fresh choices may become available to the scheduler, since the pliant transition preceding the skip will have changed the state. This would be an unwelcome complication. Therefore, we determine that **in Hybrid Event-B, notional** skips **do not introduce scheduling choice points**.

We illustrate the above in a schematic example. Fig.4 shows a fragment of the refinement of an abstract run. Time goes left to right. The narrowly spaced vertical bars represent mode events, taking place instantaneously. The horizontal lines represent the pliant events that interleave them, having non-zero durations. At the abstract level we have the events $MoEvA_1$, $PLiEvA_1$, $MoEvA_2$, $PliEvA_2$, $MoEvA_3$. The mode events are refined by concrete mode events $MoEvC_1$, $MoEvC_2$, $MoEvC_3$. Between $MoEvC_1$ and $MoEvC_2$ there is pliant event $PLiEvC_1$ which refines $PLiEvA_1$. By the argument above, $MoEvA_1$ and $MoEvC_1$ are simultaneous, as are $MoEvA_2$ and $MoEvC_2$, and noting that mode transitions both enable and preempt pliant transitions, we conclude that the durations of $PLiEvC_1$ and $PLiEvA_1$ are the same.

In between $MoEvC_2$ and $MoEvC_3$, there are some 'new' concrete mode events, $MoEvC_{2,1}$ and $MoEvC_{2,2}$, and interleaving these, are shorter pliant events $PliEvC_{2,1}$, $PliEvC_{2,2}$ and $PliEvC_{2,3}$. The sequence $PliEvC_{2,1}$, $MoEvC_{2,1}$, $PliEvC_{2,2}$, $MoEvC_{2,2}$, $PliEvC_{2,3}$ refines $PliEvA_2$ — if we take due account of the 'notional skips' that are needed to abstract $MoEvC_{2,1}$ and $MoEvC_{2,2}$, indicated by the heavier strokes through the $PliEvA_2$ timeline. Overall, the duration of the sequence $PliEvC_{2,1}$, $MoEvC_{2,1}$, $PliEvC_{2,2}$, $MoEvC_{2,2}$, $PliEvC_{2,3}$, equals that of $PliEvA_2$ because $MoEvA_2$ and $MoEvA_3$ fix the endpoints via their refinements $MoEvC_2$ and $MoEvC_3$. In general, the time period during which an abstract pliant transition runs must consist of one or more concrete pliant event durations, as Fig. 4 shows.

Hybrid Event-B needs proof obligations to guarantee the behaviour just described, while disturbing discrete Event-B as little as possible. It turns out that we can deal with mode events essentially as in discrete Event-B, for which the POs are standard. The only remaining point concerns variants and convergence, to which we return below.

Regarding pliant transitions, an abstract pliant transition starts at the same moment as a refining concrete pliant transition. This requires pliant guard strengthening, which works like mode guard strengthening. Thus, if the abstract and concrete invariants hold, and the concrete pliant INIT and WHERE guards hold, then so too must the abstract pliant INIT and WHERE guards. The formal PO is (31).

After guard strengthening comes invariant preservation. Since we demand that invariants are true at

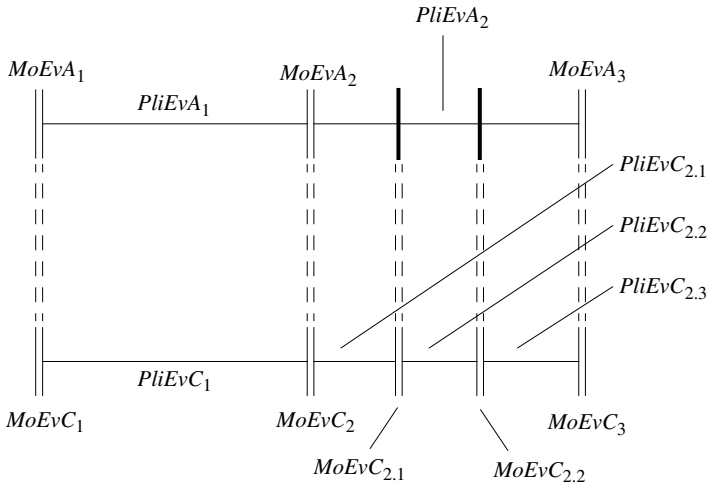

Figure 4: Typical phenomena observed during the refinement of some abstract transitions. The progress of time is correlated in the abstract and concrete systems, implying that the endpoints of abstract and concrete coincide.

all times, if the invariants and concrete guards are all true initially, then for the common duration of both pliant events, the concrete *BDApred* and the predicate SOL_{PliEvC} that defines the concrete solution[19] imply the existence of abstract states and parameters that cause the abstract *BDApred* and solution predicate SOL_{PliEvA} to hold. See (32) for the formal details. This covers cases in which the concrete pliant event refines an abstract one.

The remaining case is when a concrete pliant transition is an instance of a 'new' concrete pliant event, and occurs after a 'new' concrete mode event (the latter refining a 'notional abstract skip'), for example $PliEvC_{2.2}$ in Fig. 4. The point here is that the new mode transition (and its following pliant transition) run while some abstract pliant transition is also running and continually changing the abstract state, a situation absent from discrete Event-B due to piecewise constant behaviour.

The new concrete mode event is unproblematic. Its guard strengthens the true guard of an abstract notional skip, and the discrete Event-B invariant preservation PO for new mode events works as required, since all the invariants are true by assumption in its before-state, hence easy to re-verify in the after-state.

We turn to the new concrete pliant events. These are trickier due to the continuously changing abstract state in a period preceding the new concrete pliant transition. This aspect makes a comparison between the new concrete pliant event's guards (at the moment it starts) and the guards of the abstract event it refines (which started earlier), much more questionable.

It was for this reason we split pliant events' guards into two: the INIT guard, involving pliant variables and combinations of pliant and mode variables, and the WHERE guard, permitted to involve mode variables alone. The mode variables in the WHERE guard of the abstract pliant event being refined by a new concrete pliant event, have piecewise constant trajectories which do not change throughout any transition defined by the abstract pliant event, no matter how many new concrete pliant events contribute to the refinement. Therefore, it is reasonable to construct a guard strengthening PO for new concrete pliant events that refers just to the WHERE guard variables. Syntactically, we indicate the alternative guard strengthening tactic via a new event status 'pliant convergent'.

Invariant preservation is the same for old concrete pliant events and for new ones. In both cases, the concrete event has to name the abstract event it refines, since both the abstract and concrete behaviours

[19] SOL_{PliEvC} is the formal name of the transition relation Q discussed in Section 4.9.

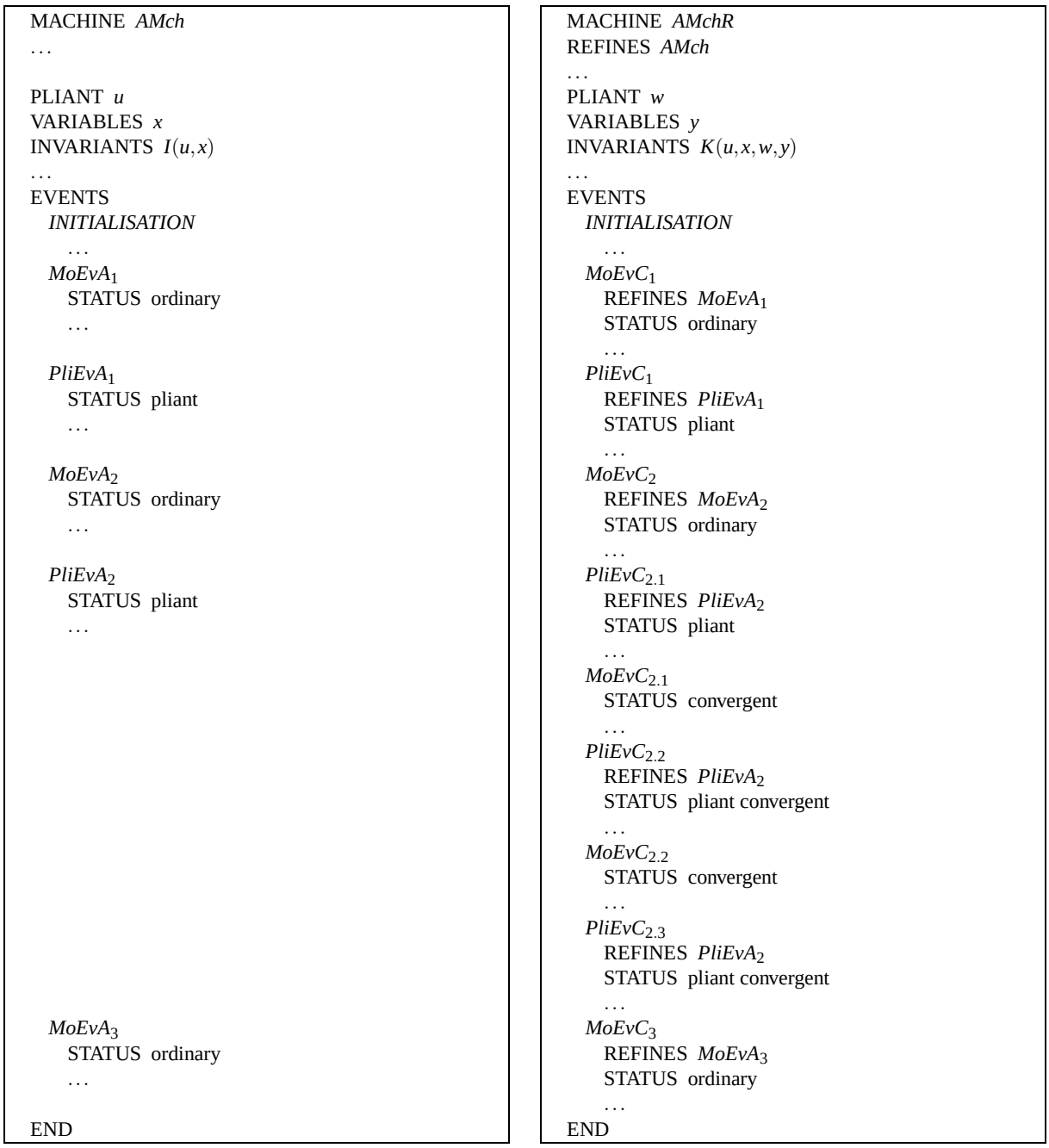

Figure 5: Syntax for expressing a machine and its refinement, a fragment of whose behaviour is shown in Fig. 4.

are non-trivial. Moreover the abstract guard, which causes the problems just addressed, does not figure in the PO, the formal expression for which is (32).

Next is relative deadlock freedom. If, in a given abstract state, some abstract event is enabled, then viewed through the abstract and joint invariants, a corresponding concrete state should enable some concrete event. The requirements are the same for mode and pliant events, expressed in the POs (35) and (36), two individual POs to maintain the separation between mode and pliant aspects.

The final topic in this section is convergence and variants. Suppose that discrete convergence holds for new mode events via a variant V defined on a well-founded set. This gives us relative non-Zenoness;

if the abstract system is Zeno-free, then the concrete system cannot have a Zeno point at any finite time.

Now suppose all concrete pliant events last for at least $\delta_{\mathsf{Zeno},C}$. Suppose a concrete run contains an unbounded sequence of new pliant transitions, refining a single abstract pliant transition. Two facts follow. Firstly, the unbounded sequence must occur at the end of the run. Secondly, the occurrences of the new pliant transitions must be interleaved with occurrences of new mode transitions only (since if not, an old concrete mode transition would refine an old abstract mode transition, which would preempt the single abstract pliant transition, a contradiction).

Likewise, suppose a concrete run contains an unbounded sequence of new mode transitions, part of the refinement of a single abstract pliant transition. The new mode transitions must be interleaved with new pliant transitions only (since if not, an old pliant transition would refine an old abstract pliant transition, implying the original single abstract pliant transition was preempted, a contradiction).

The above shows two things. The first is that tackling Zeno properties is most profitably done at the most abstract level possible, since lower level models may then inherit relative Zeno-freedom. The second is that with non-Zenoness in both models, concrete divergence takes unbounded time, and implies an unbounded abstract pliant transition at the end of the run. This is in line with conventional views of divergence through refinement.

Thus, convergence in the mode event and pliant event regimes of Hybrid Event-B are closely connected. In practice, it is still often easiest to address convergence in the discrete regime, since it avoids potential problems around asymptotic approach to convergence in the pliant regime.

In Fig. 5, we give the relevant syntactic details that connect the syntactic descriptions of the various events in Fig. 4 that we discussed above. These are sufficient to enable a tool to generate the required POs in the correct form.

Definition 8.1. *A Hybrid Event-B machine $\mathcal{MR}$ correctly refines a Hybrid Event-B machine $\mathcal{M}$ iff for every system trace SR of $\mathcal{MR}$ there is a system trace S of $\mathcal{M}$ such that:*

(i) If SR occupies the time interval $[t_0 \ldots t_{\mathrm{FIN}R})$, then S occupies a time interval $[t_0 \ldots t_{\mathrm{FIN}})$, where $t_{\mathrm{FIN}R} \leq t_{\mathrm{FIN}}$.

(ii) For each t in $[t_0 \ldots t_{\mathrm{FIN}R})$, all the invariants hold.

(iii) At each occurrence of a mode event in S there is an occurrence of a mode event in SR.

9. Proof Obligations

In this section, we gather together the proof obligations discussed above. Of course, the main purpose of the POs is to give a static guarantee of correctness, and we turn to this aspect at the end of the section.

For clarity below, when dealing with mode events, viewed as taking place instantaneously, we write just the variable names involved, e.g. u. When dealing with pliant events, viewed as defining time-indexed families of before-after pairs of states, we indicate time dependence explicitly. We write e.g. $u(t)$, while not excluding other forms of time dependence e.g. $u(t-1)$, (provided their use yields piecewise absolutely continuous behaviours). First we summarise the new status tags introduced earlier.

9.1. New STATUS Tags

For ease of reference, we summarise the various additional status tags introduced through the course of the paper to indicate various attributes of pliant events.

Tag	Remarks	(9)
pliant	an ‘ordinary’ pliant event	
pliant convergent	a ‘new’ pliant event of a refinement	
pliant final	a final pliant event that does not need to enable any mode event	

9.2. Contexts

Contexts define the static mathematical apparatus with which machines are specified. Contexts can be *extended* as in discrete Event-B, which implies that any axioms assumed in an abstract context, must be proved to still hold in the instantiation provided by the extension. Thus if $Stat_A$ contains the static definitions of a context Con_A, containing axioms $Axioms_A$, and $Stat_E$ contains the static definitions of a context Con_E, which extends $Stat_A$, containing axioms $Axioms_E$, the following PO must hold.

$$Stat_A \wedge Stat_E \wedge Axioms_E \Rightarrow Axioms_A \tag{10}$$

9.3. Machine Initialisation POs

For a machine A with variables u, initialisation event $Init_A$ and invariant I to be well defined, the initialisation PO has to first of all be feasible:

$$\exists u' \bullet Init_A(u') \tag{11}$$

so at least one initial state exists. Also, any initial state has to establish the invariants:

$$Init_A(u') \Rightarrow I(u') \tag{12}$$

Primes are used in (11) and (12), since initialisation is regarded as a kind of event in Event-B.

9.4. Machine Consistency POs

Machine consistency begins with feasibility POs for both mode and pliant events. For a mode event $MoEvA$, with state variables u, parameters $i?, l, o!$ and guard grd_{MoEvA}, given invariants I and with before-after-predicate $BApred_{MoEvA}(u, i?, l, o!, u')$, the PO reads:

$$I(u) \wedge grd_{MoEvA}(u, i?, l) \Rightarrow (\exists u', o! \bullet BApred_{MoEvA}(u, i?, l, o!, u')) \tag{13}$$

Note that in (13) we do not use the topological closure of the state space region defined by grd_{MoEvA}, in line with our remarks in Section 6.12. The topological closure is relevant to the runtime semantics of a Hybrid Event-B machine, but should be ignored for static verification.

For a pliant event $PliEvA$, with state u, parameters $i?, l, o!$, INIT guard iv_{PliEvA} and WHERE guard grd_{PliEvA}, given invariants I, and with before-during-after-predicate $BDApred_{PliEvA}$, feasibility asserts that there is an open interval given by some $\mathbb{t}_R > \mathbb{t}_L$ within which the pliant event specifies a behaviour of the machine. This means that there is a solution predicate SOL_{PliEvA} which, either solves the differential equation of, or expresses the direct assignment in, the SOLVE clause of $PliEvA$, and that in the interval $(\mathbb{t}_L \ldots \mathbb{t}_R)$, both SOL_{PliEvA} and $BDApred_{PliEvA}$ are jointly satisfied:

$$\begin{aligned} &I(u(\mathbb{t}_L)) \wedge iv_{PliEvA}(u(\mathbb{t}_L)) \wedge grd_{PliEvA}(u(\mathbb{t}_L)) \\ &\quad \Rightarrow (\exists \mathbb{t}_R > \mathbb{t}_L \bullet [\,(\mathbb{t}_R - \mathbb{t}_L \geq \delta_{\mathsf{Zeno}PliEvA}) \wedge]\,(\forall \mathbb{t}_L \leq t < \mathbb{t}_R \bullet (\exists u(t), i?(t), l(t), o!(t) \bullet \\ &\qquad BDApred_{PliEvA}(u(t), i?(t), l(t), o!(t), t) \wedge SOL_{PliEvA}(u(t), i?(t), l(t), o!(t), t)))) \end{aligned} \tag{14}$$

In (14) the portion in bold square brackets expresses the Zeno property of $PliEvA$, presuming that $\delta_{\mathsf{Zeno}PliEvA}$ is a suitable constant. The square brackets indicate that it may be regarded as optional, since Zeno properties are often so hard to prove statically.

Machine consistency continues with invariant preservation. For mode events, with the conventions used in (13) we have:

$$I(u) \wedge grd_{MoEvA}(u, i?, l) \wedge BApred_{MoEvA}(u, i?, l, o!, u') \Rightarrow I(u') \tag{15}$$

Machine consistency also includes invariant preservation for pliant events:

$$\begin{aligned} &I(u(\mathbb{t}_L)) \wedge iv_{PliEvA}(u(\mathbb{t}_L)) \wedge grd_{PliEvA}(u(\mathbb{t}_L)) \wedge (\exists \mathbb{t}_R > \mathbb{t}_L \bullet \mathrm{TRM}(\mathbb{t}_R) \wedge (\forall \mathbb{t}_L \leq t < \mathbb{t}_R, u(t), \\ &i?(t), l(t), o!(t) \bullet BDApred_{PliEvA}(u(t), i?(t), l(t), o!(t), t) \wedge SOL_{PliEvA}(u(t), i?(t), l(t), o!(t), t))) \\ &\quad \Rightarrow (\forall \mathbb{t}_L \leq t < \mathbb{t}_R \bullet I(u(t))) \end{aligned} \tag{16}$$

In (16), for a **nonfinal** pliant event, $\text{TRM}(\mathbb{t}_R)$ signifies that $\mathbb{t}_R$ is (at least as big as) the preemption time of a pliant transition specified by the event and started at $\mathbb{t}_L$ (i.e. $\mathbb{t}_R$ records the termination time of the transition). The minimum value of $\mathbb{t}_R$ is obtainable via the calculation needed for well-formedness in the PO (18). For a **final** pliant event, $\text{TRM}(\mathbb{t}_R)$ signifies that (16) must be true for unboundedly large $\mathbb{t}_R$.

9.5. Machine Well Formedness POs

Well formedness statically checks that mode and pliant steps alternate during a system run. If u is an after-state of a transition of mode event *MoEvA*, then it: disables mode events that **do not have inputs**[20] (by ensuring that the disjunction of those mode events' guards evaluates to false), and enables some pliant event (by ensuring that the disjunction of pliant event initial values and guards evaluates to true).

$$\begin{aligned}
&\exists u_0, i_0?, l_0, o_0! \bullet I(u_0) \wedge grd_{MoEvA}(u_0, i_0?, l_0) \wedge BApred_{MoEvA}(u_0, i_0?, l_0, o_0!, u) \wedge I(u) \\
&\quad \Rightarrow \neg [\ \exists l \bullet grd_{MoEvA1}(u,l) \vee grd_{MoEvA2}(u,l) \ldots grd_{MoEvAN}(u,l)\ ] \wedge \\
&\qquad [\ (iv_{PliEvA1}(u) \wedge grd_{PliEvA1}(u)) \vee (iv_{PliEvA2}(u) \wedge grd_{PliEvA2}(u)) \vee \ldots \vee \\
&\qquad\ (iv_{PliEvAM}(u) \wedge grd_{PliEvAM}(u))\ ]
\end{aligned} \tag{17}$$

In (17), we have simplified matters by assuming that all mode event local parameters have the same type.

Dually, if *PliEvA* is a **nonfinal** pliant event, then the end of the state trajectory in any of its pliant transitions enables some mode event. Since pliant transitions do not, typically, become infeasible when preempted, (18) does not demand that pliant events are disabled. We again simplify (18) a little by assuming that all the mode event inputs and local parameters respectively have the same types.

$$\begin{aligned}
&I(u(\mathbb{t}_L)) \wedge iv_{PliEvA}(u(\mathbb{t}_L)) \wedge grd_{PliEvA}(u(\mathbb{t}_L)) \wedge (\exists\, \mathbb{t}_R > \mathbb{t}_L \bullet (\forall\, \mathbb{t}_L \leq t < \mathbb{t}_R, u(t), i?(t), l(t), o!(t) \bullet \\
&BDApred_{PliEvA}(u(t), i?(t), l(t), o!(t), t) \wedge SOL_{PliEvA}(u(t), i?(t), l(t), o!(t), t) \wedge \text{MAXIMAL}(\mathbb{t}_R) \wedge \\
&\neg [\ \exists i?, l \bullet grd_{MoEvA1}(u(t), i?, l) \vee grd_{MoEvA2}(u(t), i?, l) \vee \ldots \vee grd_{MoEvAN}(u(t), i?, l)\ ]\ )) \\
&\quad \Rightarrow \text{WELLDEF}(\mathbb{t}_R) \wedge [\ \exists i?, l \bullet grd_{MoEvA1}(\overbrace{u(\mathbb{t}_R)}, i?, l) \vee grd_{MoEvA2}(\overbrace{u(\mathbb{t}_R)}, i?, l) \vee \ldots \vee \\
&\qquad\qquad grd_{MoEvAN}(\overbrace{u(\mathbb{t}_R)}, i?, l)\ ]
\end{aligned} \tag{18}$$

In (18), the term $\text{MAXIMAL}(\mathbb{t}_R)$ abbreviates the statement that there is no greater value of $\mathbb{t}_R$ such that the properties stated in the assumptions hold. Likewise, the term $\text{WELLDEF}(\mathbb{t}_R)$ insists that all variables have well defined values at $\mathbb{t}_R$, whether through, continuity, discontinuity or left-limit at $\mathbb{t}_R$. The PO (18) covers two cases. In both cases the assumptions state that there is no time strictly less than $\mathbb{t}_R$ such that the pliant solution exists and a mode event is enabled. Regarding the conclusions, in the first case, the solution exists at (and necessarily beyond) $\mathbb{t}_R$, and is either continuous there, or suffers a discontinuity precisely at $\mathbb{t}_R$ — in which case the overarrows in the terms $\overrightarrow{u(\mathbb{t}_R)}$ are disregarded (indicated by the bold parentheses surrounding the overarrows), and the actual value $u(\mathbb{t}_R)$ is used to enable some mode event. In the second case the solution becomes infeasible at $\mathbb{t}_R$, and the left limit is needed. As noted above, the calculation needed for $\mathbb{t}_R$ in (18) yields the duration of any pliant transition.

9.6. The Zeno Property

The discussion in Section 4 noted the desirability of non-Zenoness. In fact we already addressed this in PO (14), since proving it with the Zeno terms for all pliant events gives global non-Zenoness, as the number of pliant events is finite.

[20] The semantics ensures mode event inputs are not available at the same time as previously scheduled mode transitions.

9.7. Measurability and the Lipschitz PO

Two conditions discussed in Section 4 were the Lipschitz and measurability criteria for differential equations. Regarding measurability in time of the right hand side of DEs, we can rest easy. Non-measurable functions require considerable mathematical ingenuity to construct, and do not figure in engineering applications.

The Lipschitz criterion is of more relevance. Standard references, e.g. [48], delight in showing the pathologies that arise regarding existence and uniqueness of solutions to DEs when some aspect of the Lipschitz condition fails. The easiest way to guarantee it is to demand a uniform Lipschitz bound on the right hand side of every DE that we have to deal with. Thus, let $\mathcal{D}\,xs = \phi(xs,t)$ be a DE specifying the behaviour of some pliant event. Then the uniform bound condition reduces to:

$$\exists K \bullet \forall t \bullet ||\,\phi(xs_1,t) - \phi(xs_2,t)\,|| \leq K\,||xs_1 - xs_2\,|| \tag{19}$$

where $||\,.\,||$ denotes the $\mathcal{L}^\infty$ norm of a real vector, i.e. the maximum absolute value of any of its components. Normally, the truth of such a property will follow from generic properties of the class of DEs being used, so will not normally need to be verified explicitly.

9.8. Absolute Continuity in the Direct Assignment Case

Besides differential equations, a pliant event may be specified via a direct assignment, for example $xs := E(xs,t)$. As we stated in Section 5, we demand directly that E is piecewise absolutely continuous, so the property we need for xs is immediate.

9.9. Absolute Continuity in the Implicit Case

A pliant event may also be specified more indirectly, via the *BDApred* alone (rather than just using the *BDApred* as an additional constraint). Aside from the need for all behaviours to be absolutely continuous, we do not place further restrictions on what is permitted to be specified by this means. While, theoretically, this opens the door to defining a wide range of truly exotic behaviours, in practice these are of no interest for engineering applications, since the content of *BDApred* will normally exclude excessively wild behaviour.

One consequence of permitting 'pure *BDApred* specification' is that various POs relating to pliant events are affected. However, this is rather trivial. Since *any* piecewise absolutely continuous behaviour *SOL* satisfying the *BDApred* is allowed, the combination *BDApred* $\wedge$ *SOL* (this being the only context in which *SOL* appears in any PO) reduces to just *BDApred* in the PO.

9.10. Refinement POs

Suppose that as well as machine A as above, we have another machine C, with state variable w, and joint invariant $K(u,w)$, which is a refinement of A. This means that the concrete (joint) invariant is a relation over both u and w, aligning with the B-Method view that a refinement is an *enhancement* of its abstract counterpart rather than a replacement for it. The next sections cover the relevant POs.

9.11. Refinement Initialisation POs

Concrete initialisation feasibility is:

$$\exists w' \bullet Init_C(w') \tag{20}$$

while correct initialisation of C is relative to A:

$$Init_C(w') \Rightarrow (\exists u' \bullet Init_A(u') \wedge K(u',w')) \tag{21}$$

9.12. Refinement Mode Event Consistency POs

Next are the concrete event POs. Let the concrete mode event that refines an abstract mode event *MoEvA* is called *MoEvC*. Let *MoEvC* have state w, input, local and output parameters $j?, k, p!$, guard $grd_{MoEvC}(w, j?, k)$, and before-after predicate $BApred_{MoEvC}(w, j?, k, p!, w')$. Then, given the concrete invariant $K(u, w)$, event feasibility is:

$$\exists u \bullet K(u,w) \wedge grd_{MoEvC}(w,j?,k) \Rightarrow (\exists w', p! \bullet BApred_{MoEvC}(w,j?,k,p!,w')) \tag{22}$$

Two POs must hold if *MoEvC* refines *MoEvA*. The first, guard strengthening, states that when the invariants hold, the concrete guard implies the abstract one:

$$\begin{aligned} &I(u) \wedge K(u,w) \wedge grd_{MoEvC}(w,j?,k) \\ &\quad \Rightarrow (\exists i?, l \bullet grd_{MoEvA}(u,i?,l)) \end{aligned} \tag{23}$$

The second, invariant preservation, also referred to as the correctness PO, reads:

$$\begin{aligned} &I(u) \wedge K(u,w) \wedge grd_{MoEvC}(w,j?,k) \wedge BApred_{MoEvC}(w,j?,k,p!,w') \\ &\quad \Rightarrow (\exists i?, l, o!, u' \bullet BApred_{MoEvA}(u,i?,l,o!,u') \wedge K(u',w')) \end{aligned} \tag{24}$$

While the guard strengthening and correctness POs, (23) and (24) express what needs to be true for *MoEvC* to refine *MoEvA*, they do not indicate how particular abstract $i?, l, o!, u'$ are to be found for given concrete $j?, k, p!, w'$. This is remedied by providing a witness relation $W(u, i?, l, o!, u', w, j?, k, p!, w')$ that can be used to indicate appropriate values. The witness itself has to be feasible:

$$\begin{aligned} &I(u) \wedge K(u,w) \wedge grd_{MoEvC}(w,j?,k) \wedge BApred_{MoEvC}(w,j?,k,p!,w') \\ &\quad \Rightarrow (\exists i?, l, o!, u' \bullet W(u,i?,l,o!,u',w,j?,k,p!,w')) \end{aligned} \tag{25}$$

Given a feasible witness which is appropriate for the problem, the guard strengthening PO changes to:

$$\begin{aligned} &I(u) \wedge K(u,w) \wedge grd_{MoEvC}(w,j?,k) \wedge W(u,i?,l,o!,u',w,j?,k,p!,w') \\ &\quad \Rightarrow grd_{MoEvA}(u,i?,l) \end{aligned} \tag{26}$$

while the correctness PO changes to:

$$\begin{aligned} &I(u) \wedge K(u,w) \wedge grd_{MoEvC}(w,j?,k) \wedge BApred_{MoEvC}(w,j?,k,p!,w') \wedge \\ &W(u,i?,l,o!,u',w,j?,k,p!,w') \\ &\quad \Rightarrow BApred_{MoEvA}(u,i?,l,o!,u') \wedge K(u',w') \end{aligned} \tag{27}$$

where in (26) and (27), there are no more existential quantifiers to find values for.

If machine *C* has 'new' events that refine notional abstract skips, then the preceding simplifies. The abstract state does not change, so there is no abstract input either. This obviates the need for existential quantification, or witnesses. The result is:

$$I(u) \wedge K(u,w) \wedge grd_{NewEvC}(w,j?,k) \wedge BApred_{NewEvC}(w,j?,k,p!,w') \Rightarrow K(u,w') \tag{28}$$

New events are normally prevented from 'taking control of the run forever', which is achieved by demanding that each execution of a new event decreases a variant V. We can retain this criterion in Hybrid Event-B, and the PO reads:

$$BApred_{NewEvC}(w,j?,k,p!,w') \Rightarrow V(w') < V(w) \tag{29}$$

A possibility in Hybrid Event-B is the fact that it might be harder to restrict the type of the variant to an '*obviously* well founded' set. But in engineering applications this can usually be overcome with a little ingenuity.

9.13. Refinement Pliant Event Consistency POs

Turning to pliant events, we demand that abstract pliant events are refined by concrete pliant events. We start with relative event feasibility, which again features an optional Zeno term, and is again like the abstract case, aside from the existentially quantified abstract state:

$$\begin{array}{l}(\exists u(\mathbb{t}_{\mathrm{L}}) \bullet I(u(\mathbb{t}_{\mathrm{L}})) \wedge K(u(\mathbb{t}_{\mathrm{L}}), w(\mathbb{t}_{\mathrm{L}})) \wedge iv_{PliEvC}(w(\mathbb{t}_{\mathrm{L}})) \wedge grd_{PliEvC}(w(\mathbb{t}_{\mathrm{L}})) \\ \quad \Rightarrow (\exists \mathbb{t}_{\mathrm{R}} > \mathbb{t}_{\mathrm{L}} \bullet [\,(\mathbb{t}_{\mathrm{R}} - \mathbb{t}_{\mathrm{L}} \geq \delta_{\mathsf{Zeno}PliEvC}) \wedge]\,(\forall \mathbb{t}_{\mathrm{L}} < t < \mathbb{t}_{\mathrm{R}} \bullet (\exists w(t), j?(t), k(t), p!(t) \bullet \\ \qquad BDApred_{PliEvC}(w(t), j?(t), k(t), p!(t), t) \wedge SOL_{PliEvC}(w(t), j?(t), k(t), p!(t), t)))))\end{array} \tag{30}$$

Next is the analogue of guard strengthening. This comes in two forms, differing in whether the term $iv_{PliEv_A}(u(\mathbb{t}_{\mathrm{L}}))$ is included or not (indicated by enclosing it in heavy square brackets):

$$\begin{array}{l}I(u(\mathbb{t}_{\mathrm{L}})) \wedge K(u(\mathbb{t}_{\mathrm{L}}), w(\mathbb{t}_{\mathrm{L}})) \wedge iv_{PliEvC}(w(\mathbb{t}_{\mathrm{L}})) \wedge grd_{PliEvC}(w(\mathbb{t}_{\mathrm{L}})) \\ \quad \Rightarrow [\, iv_{PliEvA}(u(\mathbb{t}_{\mathrm{L}})) \wedge]\, grd_{PliEvA}(u(\mathbb{t}_{\mathrm{L}}))\end{array} \tag{31}$$

The conditions for ignoring $iv_{PliEv_A}(u(\mathbb{t}_{\mathrm{L}}))$ come from refinement, as discussed in Section 8.

The correctness PO becomes:

$$\begin{array}{l}I(u(\mathbb{t}_{\mathrm{L}})) \wedge K(u(\mathbb{t}_{\mathrm{L}}), w(\mathbb{t}_{\mathrm{L}})) \wedge iv_{PliEvC}(w(\mathbb{t}_{\mathrm{L}})) \wedge grd_{PliEvC}(w(\mathbb{t}_{\mathrm{L}})) \Rightarrow \\ \quad \big(\exists \mathbb{t}_{\mathrm{R}} > \mathbb{t}_{\mathrm{L}} \bullet \mathrm{TRM}(\mathbb{t}_{\mathrm{R}}) \wedge (\forall \mathbb{t}_{\mathrm{L}} < t < \mathbb{t}_{\mathrm{R}}, w(t), j?(t), k(t), p!(t) \bullet \\ \quad BDApred_{PliEvC}(w(t), j?(t), k(t), p!(t), t) \wedge SOL_{PliEvC}(w(t), j?(t), k(t), p!(t), t)) \\ \qquad \Rightarrow (\forall \mathbb{t}_{\mathrm{L}} < t < \mathbb{t}_{\mathrm{R}} \bullet (\exists u(t), i?(t), l(t), o!(t) \bullet \\ \qquad\quad BDApred_{PliEvA}(u(t), i?(t), l(t), o!(t), t) \wedge SOL_{PliEvA}(u(t), i?(t), l(t), o!(t), t) \wedge \\ \qquad\quad K(u(t), w(t)))))\big)\end{array} \tag{32}$$

The form of (32) implies a number of things. The main one is that time progresses at the same rate in the abstract and concrete systems. This is a consequence of citing the same time value in both occurrences of time in K in the conclusion of the inner (universally quantified), implication; and also, of using the same $\mathbb{t}_{\mathrm{R}}$ value in both the assumptions and conclusions of this implication (as enforced by the scope of the existential quantification over $\mathbb{t}_{\mathrm{R}}$). The termination term $\mathrm{TRM}(\mathbb{t}_{\mathrm{R}})$ refers to preemption (or nontermination) of a concrete transition started at $\mathbb{t}_{\mathrm{L}}$. So (32) assures us that a simulating pair of pliant transitions lasts as long at the abstract level as at the concrete level.

The PO (32) suffers from the same problem as (24), namely that there is no indication of how to find suitable $u(t), i?(t), l(t), o!(t)$ for any given $w(t), j?(t), k(t), p!(t)$, a situation made worse by the fact that these quantities now depend on time.

The remedy is the same as before. We introduce $W(u(t), i?(t), l(t), o!(t), w(t), j?(t), k(t), p!(t))$, a pliant witness relation, to point the way. Note that guard strengthening no longer requires a witness, since it does not involve any of the parameters in the pliant case.

The witness relation $W(u(t), i?(t), l(t), o!(t), w(t), j?(t), k(t), p!(t))$ has to be as feasible as the concrete transition needs to last:

$$\begin{array}{l}I(u(\mathbb{t}_{\mathrm{L}})) \wedge K(u(\mathbb{t}_{\mathrm{L}}), w(\mathbb{t}_{\mathrm{L}})) \wedge iv_{PliEvC}(w(\mathbb{t}_{\mathrm{L}})) \wedge grd_{PliEvC}(w(\mathbb{t}_{\mathrm{L}})) \Rightarrow \\ \quad \big(\exists \mathbb{t}_{\mathrm{R}} > \mathbb{t}_{\mathrm{L}} \bullet \mathrm{TRM}(\mathbb{t}_{\mathrm{R}}) \wedge (\forall \mathbb{t}_{\mathrm{L}} < t < \mathbb{t}_{\mathrm{R}}, w(t), j?(t), k(t), p!(t) \bullet \\ \quad BDApred_{PliEvC}(w(t), j?(t), k(t), p!(t), t) \wedge SOL_{PliEvC}(w(t), j?(t), k(t), p!(t), t)) \\ \qquad \Rightarrow (\forall \mathbb{t}_{\mathrm{L}} < t < \mathbb{t}_{\mathrm{R}} \bullet (\exists u(t), i?(t), l(t), o!(t) \bullet W(u(t), i?(t), l(t), o!(t), w(t), j?(t), k(t), p!(t)))))\big)\end{array} \tag{33}$$

With the help of the witness, the PO (32) becomes:

$$
\begin{array}{l}
I(u(\mathbb{t}_{\mathrm{L}})) \wedge K(u(\mathbb{t}_{\mathrm{L}}), w(\mathbb{t}_{\mathrm{L}})) \wedge iv_{PliEvC}(w(\mathbb{t}_{\mathrm{L}})) \wedge grd_{PliEvC}(w(\mathbb{t}_{\mathrm{L}})) \Rightarrow \\
\quad \big(\exists\, \mathbb{t}_{\mathrm{R}} > \mathbb{t}_{\mathrm{L}} \bullet \mathrm{TRM}(\mathbb{t}_{\mathrm{R}}) \wedge (\forall\, \mathbb{t}_{\mathrm{L}} < t < \mathbb{t}_{\mathrm{R}}, w(t), j?(t), k(t), p!(t) \bullet \\
\quad BDApred_{PliEvC}(w(t), j?(t), k(t), p!(t), t) \wedge SOL_{PliEvC}(w(t), j?(t), k(t), p!(t), t) \wedge \\
\quad W(u(t), i?(t), l(t), o!(t), w(t), j?(t), k(t), p!(t))) \\
\qquad \Rightarrow (\forall\, \mathbb{t}_{\mathrm{L}} < t < \mathbb{t}_{\mathrm{R}} \bullet \\
\qquad\quad BDApred_{PliEvA}(u(t), i?(t), l(t), o!(t), t) \wedge SOL_{PliEvA}(u(t), i?(t), l(t), o!(t), t) \wedge \\
\qquad\quad K(u(t), w(t))\big)
\end{array}
\tag{34}
$$

9.14. Refinement Relative Deadlock Freedom POs

Acting in tandem with feasibility, relative deadlock freedom guarantees that, despite guards being individually *strengthened* during refinement (see (24)), all together (i.e. taking new events into account) the concrete system is enabled 'at least as much' as the abstract one.

For mode events, utilising the witness relation $W(u, i?, l, o!, u', w, j?, k, p!, w')$ given earlier, and assuming at both levels that all events have the same parameter types, the PO reads:

$$
\begin{array}{l}
I(u) \wedge K(u, w) \wedge (\exists\, o!, p!, u', w' \bullet W(u, i?, l, o!, u', w, j?, k, p!, w')) \wedge \\
[\, grd_{MoEvA1}(u, i?, l) \vee grd_{MoEvA2}(u, i?, l) \vee \ldots \vee grd_{MoEvAN}(u, i?, l) \,] \\
\quad \Rightarrow grd_{MoEvC1}(w, j?, k) \vee grd_{MoEvC2}(w, j?, k) \vee \ldots \vee grd_{MoEvCM}(w, j?, k)
\end{array}
\tag{35}
$$

We also demand relative deadlock freedom in the continuous sphere. Note that we don't need a witness here, since pliant events do not have parameters that can be sensed at the initial instant of a pliant transition.

$$
\begin{array}{l}
I(u) \wedge K(u(\mathbb{t}_{\mathrm{L}}), w(\mathbb{t}_{\mathrm{L}})) \wedge [\, (iv_{PliEvA1}(u(\mathbb{t}_{\mathrm{L}})) \wedge grd_{PliEvA1}(u(\mathbb{t}_{\mathrm{L}}))) \vee \\
(iv_{PliEvA2}(u(\mathbb{t}_{\mathrm{L}})) \wedge grd_{PliEvA2}(u(\mathbb{t}_{\mathrm{L}})) \vee \ldots \vee (iv_{PliEvAM}(u(\mathbb{t}_{\mathrm{L}})) \wedge grd_{PliEvAM}(u(\mathbb{t}_{\mathrm{L}})) \,] \\
\quad \Rightarrow [\, (iv_{PliEvC1}(w(\mathbb{t}_{\mathrm{L}})) \wedge grd_{PliEvC1}(w(\mathbb{t}_{\mathrm{L}}))) \vee (iv_{PliEvC2}(w(\mathbb{t}_{\mathrm{L}})) \wedge grd_{PliEvC2}(w(\mathbb{t}_{\mathrm{L}})) \vee \ldots \vee \\
\qquad (iv_{PliEvCN}(w(\mathbb{t}_{\mathrm{L}})) \wedge grd_{PliEvCN}(w(\mathbb{t}_{\mathrm{L}})) \,]
\end{array}
\tag{36}
$$

9.15. Correctness

The objective of having static POs is to enable us to conclude, statically, that runtime errors do not occur. In this section we examine some correctness properties that follow from the POs above.

Theorem 9.1. *Let $\mathcal{M}$ be a Hybrid Event-B machine. Suppose that no event (whether mode or pliant) has an inconsistent specification for the update of any variable. Suppose that the POs listed earlier in this section hold. Then the Hybrid Event-B machine $\mathcal{M}$ is correct according to Definition 7.1.*

Proof: It will be sufficient to go through the steps of the formal semantics in Section 7, and to confirm that the static properties assumed are sufficient to ensure that the ABORT or VOID cases are never encountered.

Regarding step **[2]**, we assume that initialisation assigns values to all variables, consistent with the invariants.

Next, the mode-to-pliant machine well-formedness PO (17) guarantees that no mode event without inputs is enabled, passing step **[3]**; it also guarantees that there is an enabled pliant event governing the subsequent behaviour, passing step **[4]**. The check in **[4.1]** is passed, by assumption.

Pliant event feasibility, (14), ensures that in step **[5]**, some nonempty interval $(t_0 \ldots t_{\mathrm{MAX}})$ can be found, leading to a choice of explicit solution for some maximal t_{MAX} in **[6]**. Step **[6.1]** is unproblematic.

If no mode event becomes enabled during (or at the end of) the interval $(t_0 \ldots t_{\mathrm{MAX}})$ then, the invariant preservation PO (16) guarantees successful termination at t_{MAX}, by **[7]**.

Otherwise, the next cycle of execution starts, and step **[8]** determines the next preemption point $t_{\eta+1}$. PO (18) guarantees that however this preemption point is determined, whether by continuous or discontinuous behaviour assigning variable values, or by left-limit values at the end of a region of feasibility, all variables are well defined and enable a non-*INITIALISATION* mode event. Step **[10.1]** determines the set of enabled non-*INITIALISATION* mode events at t_η, and step **[10.2]** chooses a mode transition selected from them. Step **[10.3]** completes the (re)definition of variable values at t_η. Because of PO (18), none of these steps can ABORT. Finally, step **[10.4]** cleans up the time interval $(t_\eta \ldots t_{\mathrm{MAX}})$. The proof then continues as from the third paragraph above, though it deals with a generic t_η instead of t_0. We are done. □

Note that the above proof, while asserting correctness as in Definition 7.1, does not assure the absence of Zeno phenomena, *unless* we are able to include the $\delta_{\mathsf{Zeno}PliEvA}$ terms in the POs that contain them. Note also that mode event guard closure was never mentioned in either the POs or the proof. Although it is useful for runtime semantics, it may give rise to phenomena beyond the reach of static verification.

Theorem 9.2. *Let $\mathcal{M}$ and $\mathcal{MR}$ be Hybrid Event-B machines. Suppose the conditions of Theorem 9.1 are satisfied for both machines. Suppose that the refinement POs hold for $\mathcal{M}$ and $\mathcal{MR}$. Then $\mathcal{MR}$ refines $\mathcal{M}$ in the sense of Definition 8.1.*

Proof: The proof proceeds by induction. Let *SR* be a system trace of $\mathcal{MR}$, given by a collection of time dependent valuations for all the variables of $\mathcal{MR}$ over an interval $[t_0 \ldots t_{\mathrm{FIN}R})$. We show that we can simulate *SR* by a system trace *S* of $\mathcal{M}$, such that all the invariants of both machines hold, and at each occurrence of a mode event in *S*, there is an occurrence of a mode event in *SR*.

System trace *SR* starts with an initial state satisfying $\mathcal{MR}$'s invariants, and the initialisation refinement POs ensure a corresponding $\mathcal{M}$ initial state satisfying $\mathcal{M}$'s invariants. Thereafter, pliant transitions and mode transitions alternate in *SR*. POs (30)-(32) ensure that the first pliant transition of *SR* can be correctly simulated until it is preempted by the next mode transition of *SR*. (That the abstract system trace *S* cannot be preempted sooner than the next mode transition of *SR* follows by the mode event relative deadlock freedom PO (35), which would enable an $\mathcal{MR}$ mode event, forcing an earlier *SR* preemption.)

Then POs (22)-(28) ensure that the mode transition is correctly simulated, whether by an 'old' abstract transition or by a 'notional skip', both of which preserve the invariants. The subsequent pliant transition of *SR* may be for an 'old' or a 'new' event. In both cases, given that this *SR* transition is feasible by assumption, the refinement correctness PO for pliant events (32) guarantees that the simulating abstract pliant transition is feasible and executes, preserving the invariants. (In particular, in the case of a 'new' event simulated by a 'notional skip', it prevents the previous abstract transition from becoming infeasible precisely at the moment of preemption.)

The inductive process continues to cover all of the interval $[t_0 \ldots t_{\mathrm{FIN}R})$, giving a simulating abstract system trace *S* lasting at least as long as *SR*. It is also clear that for each mode transition in *S* (disregarding the notional skip' transitions) there is a mode transition in *SR* which gave rise to the *S* transition through simulation. We are done. □

We point out that although the above account discussed machines in terms of their state variables alone, similar considerations apply when events feature parameters. (This typically necessitates suitable existential claims in the hypotheses regarding inputs etc.).

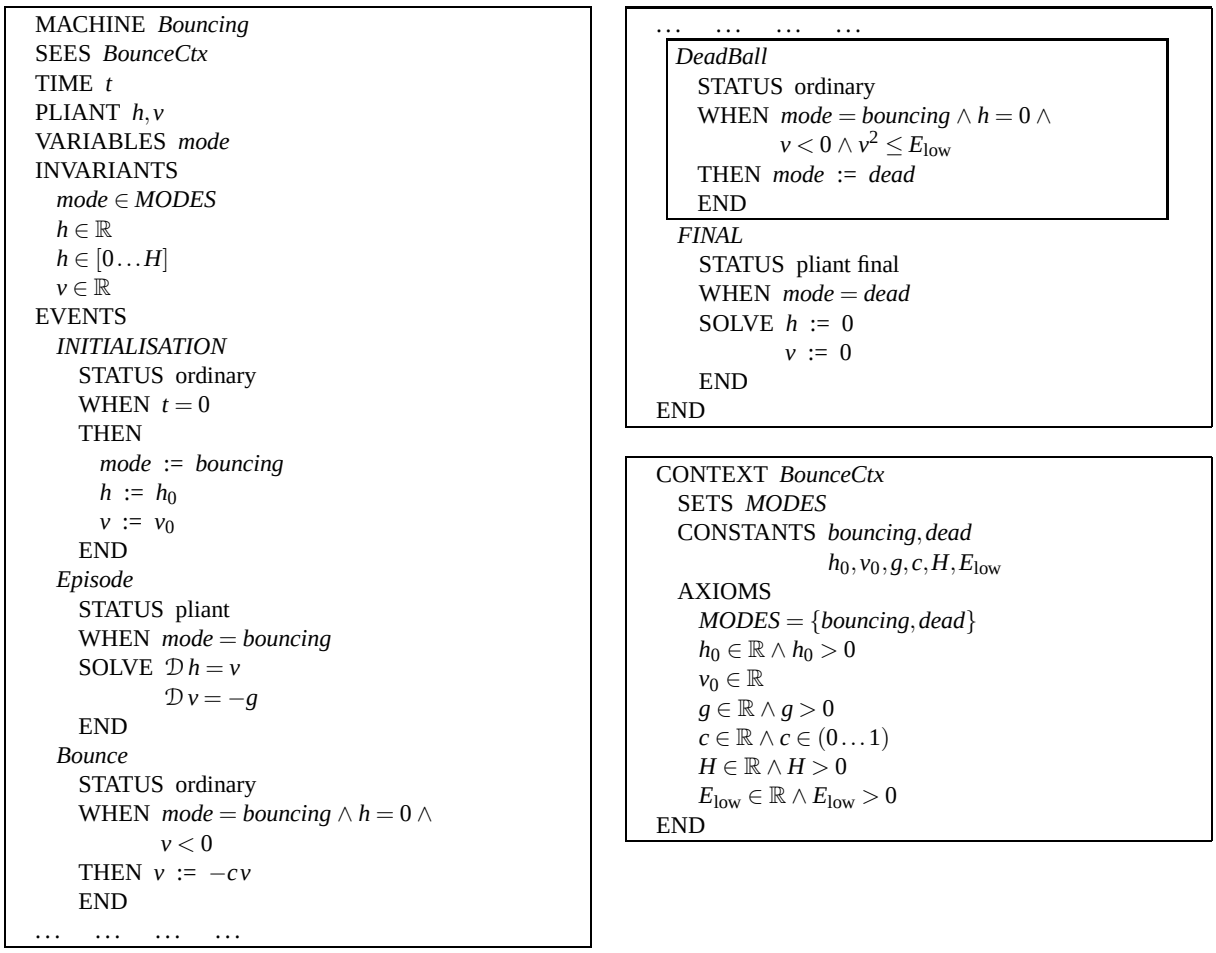

Figure 6: A Hybrid Event-B machine for the bouncing ball.

10. Case Studies

In this section we look at a number of relatively small case studies that illustrate the framework we have described previously. Somewhat larger case studies can be found in [12, 11, 9, 10].

10.1. The Bouncing Ball

We treat a favourite example, the bouncing ball — a nice account can be found in [38]. A pointlike ball of unit mass is subject to gravity g, and bounces vertically over some point on a horizontal surface, starting at time $t = 0$. The ball's height above the surface is $h(t)$, initially set to $h_0 > 0$ at $t = 0$, and its vertical velocity is $v(t)$ (positive values indicating upward movement), initially v_0 at $t = 0$. Whenever the ball hits the surface, the speed diminishes by a factor $c < 1$, and the kinetic energy by a factor c^2. When the ball's energy is low enough, the bounce may simply absorb all the energy, leaving the ball stationary on the horizontal surface.

To understand this ball's behaviour, let us consider a single full bouncing episode, with the ball leaving the surface with velocity $\tilde{v}$. Such an episode reaches a height $\tilde{h}$ given by $g\tilde{h} = \frac{1}{2}\tilde{v}^2$, since this expresses the conversion of pure kinetic energy at the surface to pure potential energy at the highest point. Since the energy is diminished after the ball returns to the surface, the maximum height reached during any individual full episode is an upper bound for any remaining dynamics of the ball. Therefore, if we wish to impose an invariant such as $h(t) \leq H$ (where H is a constant), it is sufficient to check whether the property is maintained through the first (partial) episode, and through the next (full) episode.

At time $t = 0$ the energy is $g h_0 + \frac{1}{2} v_0^2$. This becomes pure kinetic energy when the ball reaches the ground, at which point it has a velocity $-v_{\max}$ given by:

$$v_{\max} = \sqrt{2\left(g h_0 + \frac{1}{2} v_0^2\right)} \tag{37}$$

If the ball happened to be moving upwards at $t = 0$, then it would reach a height $h_{\max}$ given by $g h_{\max} = \frac{1}{2} v_{\max}^2$, and this would be the maximum height it would ever reach. If the ball was moving downwards at $t = 0$, then it would lose speed by the factor c upon bouncing, and, rebounding at a velocity $c v_{\max}$, would subsequently reach a maximum height $h_{\widetilde{\max}}$ given by $g h_{\widetilde{\max}} = \frac{1}{2}(c v_{\max})^2$. These facts provide the basis for a case analysis that determines whether an invariant like $h(t) \leq H$ is respected or not, depending on the initial values. (Of course the above account depended on our knowing about energy and its conservation, allowing us to shortcircuit a more laborious solution of the system as might be performed by an unsophisticated mechanised reasoner, which would simply integrate the equations episode by episode, arriving eventually at the same conclusions.)

A Hybrid Event-B model for the system appears in Fig. 6. The context *BounceCtx* collects all the easy-to-forget facts concerning the constants that play a role in the system, without which the observations made above would not be provable. The *INITIALISATION* synchronises real time to 0, and assigns the other variables their initial values. The *Episode* pliant event describes a bouncing episode. It has no constraints on the initial values of variables except that it checks that the *mode* is *bouncing*. Mode event *Bounce* discontinuously flips the velocity of the ball when it hits the horizontal surface, and when the energy of the ball is small enough ($v^2 \leq E_{\text{low}}$), instead of bouncing, the ball has the option of resting on the horizontal surface and enabling the *FINAL* pliant event that brings the dynamics to an end.

Without the mode event *DeadBall*, the system would exhibit Zeno behaviour — the system's energy is conserved except at bounces, and since each bounce depletes the energy by a multiplicative factor c^2, an infinite number of these would be needed to consume all the energy. Since the duration of a bouncing episode is proportional to the 'lift-off energy', successive episode durations would be similarly reduced, leading to a Zeno point at a finite point in time. Note that this illustrates well the fact that Zeno behaviour is generally intimately connected with reachability.

With *DeadBall*, Zeno behaviour is not excluded — it could be though, by strengthening the guard of *Bounce* to exclude bouncing at low energy.

The bouncing ball also illustrates the utility of allowing mode event guards to define non-closed regions of the state space, even though such mode event guards are potentially reinterpreted as their closure at runtime. In the event *Bounce*, the guard, $mode = bouncing \wedge h = 0 \wedge v < 0$ specifies a non-closed region, its closure being $mode = bouncing \wedge h = 0 \wedge v \leq 0$. Statically, the after-state established by *Bounce* in the case that $v = 0$ is the same as the before-state, so re-establishes the guard of *Bounce*, and causes a failure of the PO (17). Dynamically though, we know that $v = 0$ cannot be reached after any finite number of events if $v_0 \neq 0$, so insisting on statically closed guards would lead to inconvenient modelling metaphors.

10.2. A Simple Refinement-Based Discretization Example

In this example, we examine a simple case of discretization. In the left part of Fig. 7, there is a simple Hybrid Event-B machine *ExUp*. It has a single mode variable *md* and a single pliant variable *x*. As well as time *t*, we have a clock variable *clk*, included to show the syntax. The mode variable *md* has two possible values, *stat* and *dyn*. Time is defined as the non-negative reals, and *x* has values in the closed interval $[0 \ldots 10]$.

Machine *ExUp* has four events: *INITIALISATION*, *IncPli*, *Stop*, *FINAL*. Upon initialisation, which is synchronised with time 0, the clock is set to 1, the mode *md* becomes *dyn*, and *x* is set to 0. Upon

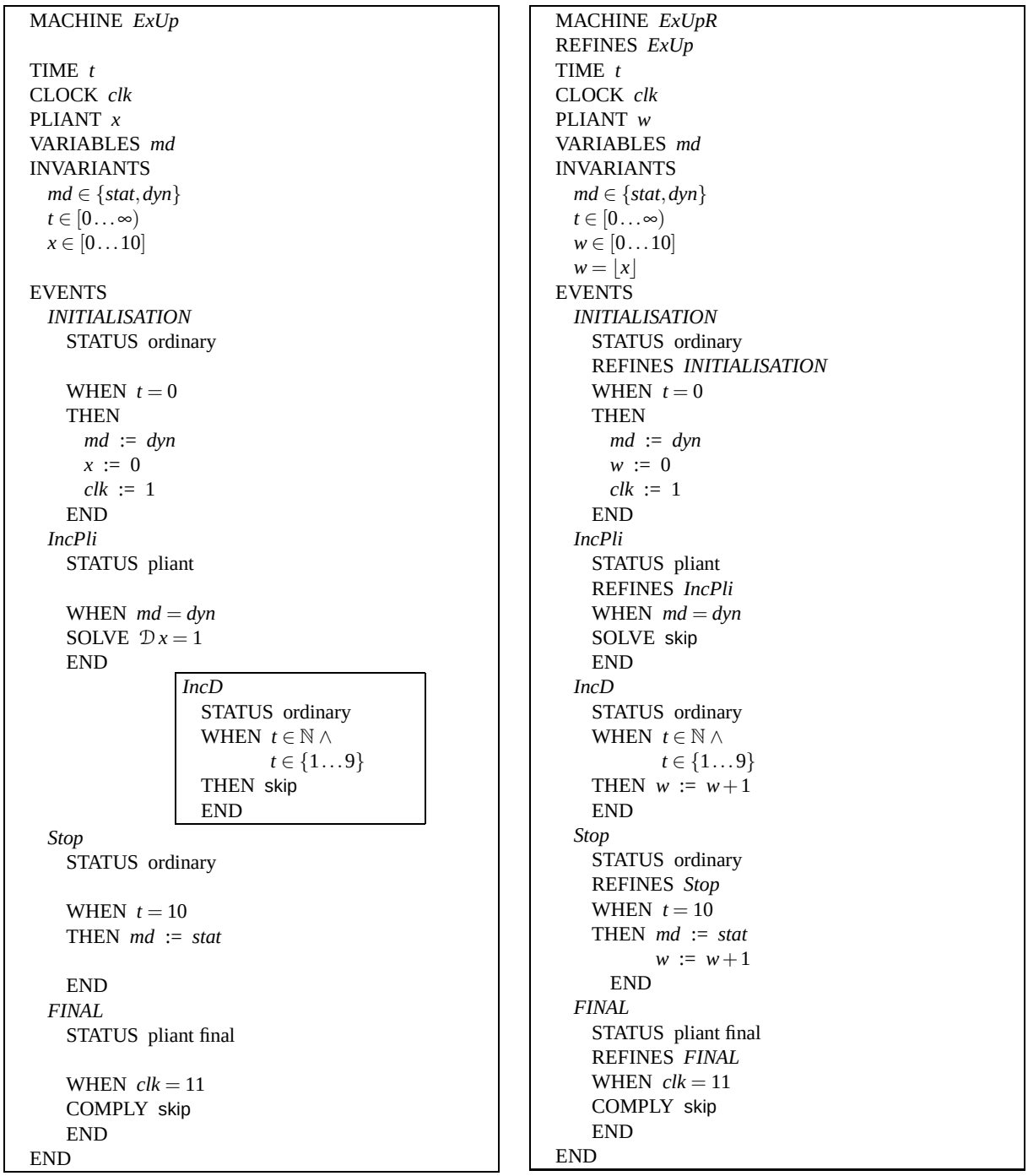

Figure 7: A simple example of discretization via refinement.

md becoming *dyn*, the pliant event *IncPli* becomes enabled, which causes *x* to increase at a steady rate since its derivative is set to 1. The clock *clk* also increases at this rate, by definition. The behaviour of *IncPli* continues for 10 time units, whereupon the mode event *Stop* changes the mode to *stat* disabling *IncPli*. By this time, the clock has reached 11, which enables the pliant event *FINAL*, which takes over, maintaining the value of *x* unchanged for the rest of time.

Shown in a box, indented, is a 'notional skip', *IncD*, that will be refined to a real mode event in machine *ExUpR*. It is included to illustrate that, unlike for discrete Event-B, the notional skip has to

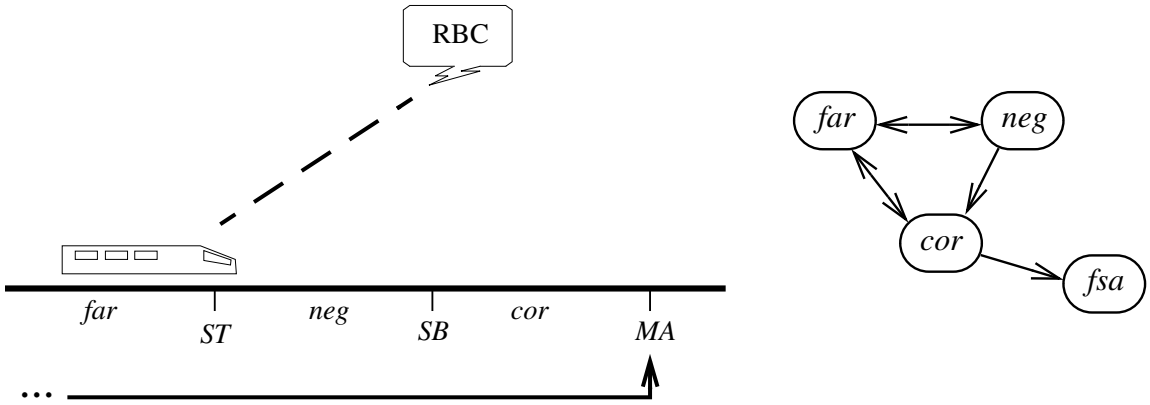

Figure 8: The European Train Control System. A movement authority, defined by its endpoint *MA*, start braking position *SB*, and start talking position *ST*; dividing the track into a *far* region, a re-*neg*otiate region, and a *cor*rection region (together with the transition diagram for the corresponding modes).

be envisaged as happening at some specific time(s), because the real events that refine them, *do* have to happen at specific times.

Machine *ExUp* is refined to *ExUpR*. The main feature of this refinement is the introduction of pliant variable w, and joint invariant $w = \lfloor x \rfloor$. In *ExUpR*, event *IncD* is now a real event, and machine *ExUpR* evidently has shorter *IncPli* pliant events (of duration one time unit instead of ten), since *IncD* preempts the refined *IncPli* frequently.

Event *IncD* refines the notional skip. Note that despite the discontinuity that the concrete *IncD* specifies, it does nevertheless refine skip. To see this better, consider a small interval surrounding $t = 5$. The behaviour of x is continuous through $t = 5$, consistent with a skip taking place at any chosen moment, including $t = 5$. On the other hand, the behaviour of w jumps from 4 to 5 at $t = 5$. Just before $t = 5$, we have $x < 5$, so $\lfloor x \rfloor = w = 4$, a fact that persists to the left limit: $\overrightarrow{\lfloor x \rfloor(5)} = \overrightarrow{w(5)} = 4$. But as soon as $t = 5$, then $x = 5$ holds, so $\lfloor x \rfloor(5) = w(5) = 5$. These two facts confirm that the behaviour of w refines skip at $t = 5$.

Observe that this example illustrates a particularly benign instance of discretization. The previously smooth (but non-trivial) behaviour of *IncPli* and trivial behaviour of (the notional) *IncD*, is replaced by a trivial behaviour of *IncPli* and non-trivial behaviour of *IncD*. This is a typical 'zero order hold', in which boundary values of pliant transitions corresponding to isolated observations and actuations, define constant behaviour in the next interval.

10.3. The European Train Control System

In our last example we present a simple treatment of the European Train Control System (ETCS), broadly based on the models in [38]. For ease of comparison, we use the same notations as [38] for variables where possible (even though this strays beyond the usual lexical conventions of Event-B).

Unlike older train control systems which confined trains to a succession of statically defined rail track sections, with consequent latencies when crossing section boundaries, the rail track is organised into dynamically controlled **movement authorities**. The key invariants are that **distinct movement authorities are always disjoint**, that **each movement authority contains (at most) one train**, and that **each train is in some movement authority**. If these are always maintained, then trains cannot collide.

Fig. 8 shows a movement authority. The movement authority is split into successive regions *far*, *neg* and *cor*, the last of which terminates the movement authority at limit *MA*. Within *far* the train can travel freely. When point *ST* (start talking) is reached, which is the boundary between *far* and *neg*, the train

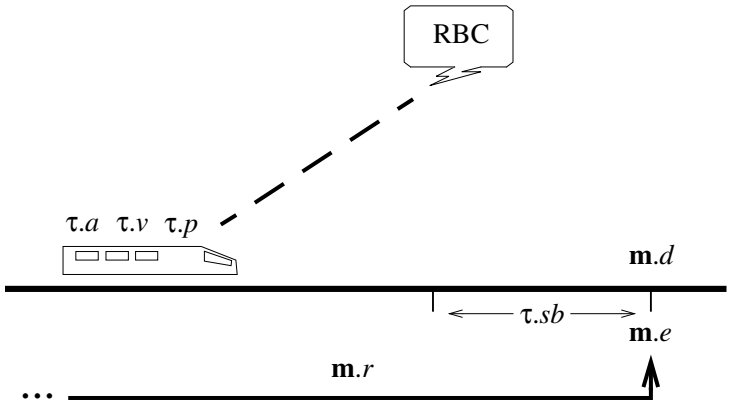

Figure 9: The European Train Control System. A generalised movement authority, defined by its recommended speed limit $\mathbf{m}.r$, end position $\mathbf{m}.e$ and demanded speed limit (at end) $\mathbf{m}.d$. This is used to control the train parameters: train acceleration $\tau.a$, train speed $\tau.v$, and train position $\tau.p$. The essential safety invariant is $\tau.p \geq \mathbf{m}.e \Rightarrow \tau.v \leq \mathbf{m}.d$.

enters the negotiation region, and starts to negotiate with the Radio Block Controller (RBC) about an extension to the movement authority. If this is successful, then the movement authority is extended and the train once more finds itself in a new *far* region. If the negotiation is unsuccessful for some reason (e.g. unreliable communication, or an emergency situation beyond *MA*), and the train crosses the point *SB* (start braking), it finds itself in the *cor* (correction) region, at which point it goes into emergency braking mode. The design is such that emergency braking must bring the train to a standstill before *MA*. Once the train has stopped, manual intervention is needed to restart the system.

Following [38], we actually model a *generalised movement authority*, shown in Fig. 9. This formulation checks whether the emergency braking distance in the *cor* region (modelled by train variable $\tau.sb$) is adequate, by reconciling it with the other dynamical variables of the train motion.

The heart of the model consists of train variables and movement authority variables, supported by suitable constants and other variables. The train variables are $\tau.p$, $\tau.v$ and $\tau.a$ which represent the current position, velocity and acceleration of the train, respectively, together with the train emergency braking distance $\tau.sb$ (which corresponds to $MA - SB$ earlier). The movement authority variables are $\mathbf{m}.r$, $\mathbf{m}.e$ and $\mathbf{m}.d$. These represent respectively the *recommended* speed (in what would correspond to the *far* and *neg* regions of the earlier model), the movement authority *endpoint* (corresponding to *MA* earlier), and the *demanded* speed at the endpoint (corresponding to the maximum permissible speed when the endpoint is reached).

The object of the exercise is to ensure that $\mathbf{m}.sb$ is of sufficient length, that should it happen that the train passes the *SB* point, maximum deceleration is capable of reducing the speed to no more than $\mathbf{m}.d$ by the time $\mathbf{m}.e$ is reached, i.e. to maintain the invariant $\tau.p \geq \mathbf{m}.e \Rightarrow \tau.v \leq \mathbf{m}.d$.

We now describe a Hybrid Event-B machine to capture this situation. The static data is in the CONTEXT *ECTS_Ctx* in Fig. 10. It contains the *normal* and *emergency* mode constants, and the *emrg* and *newMA* message values. It also contains the maximum train deceleration b and maximum train acceleration A, and also ε, which is the polling interval.[21] In addition, it contains two static functions, bd and od, which we will need later.

The *ETCS_Mch* machine itself is in Fig. 11. Aside from variables already mentioned, there is a clock $\tau.clk$ to implement the polling. Note that only $\tau.p$ and $\tau.v$ are declared pliant since they change

[21] We follow [38] in having a top level model which is already a polling model. An alternative approach, which will be pursued elsewhere, starts with a 'more continuous' abstract top level model, and introduces polling further down the development.

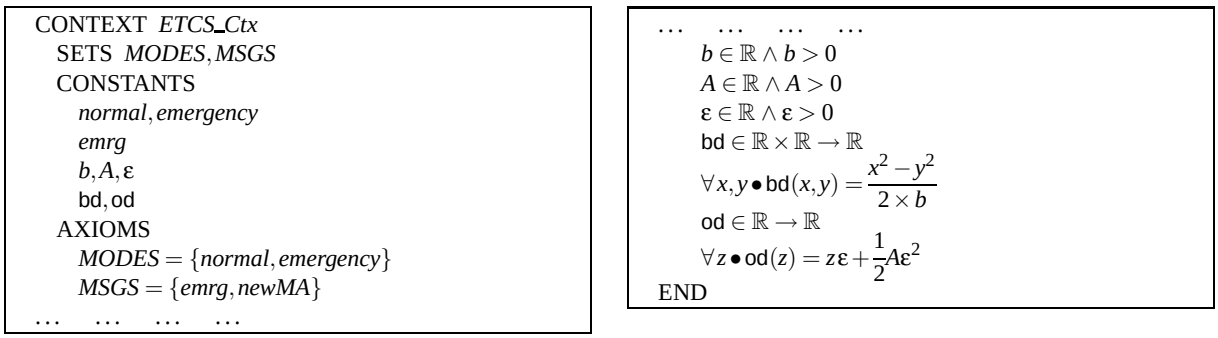

Figure 10: Static data for the European Train Control System.

continuously. Other variables are piecewise constant (albeit having values in $\mathbb{R}$), so are mode variables. An important feature of Fig. 11 is *inv*9, which expresses the key safety property, $\tau.p \geq \mathbf{m}.e \Rightarrow \tau.v \leq \mathbf{m}.d$.

We now consider the behaviour of the system. The radio block controller has the exclusive mode event *EMERGENCY*, to declare that emergency braking is required, and participates in the mode event *MOVEMENT_AUTHORITY*, whereby new data are assigned to the movement authority, and the train simultaneously reacts by updating its emergency braking point $\tau.sb$. Both mode events have input parameters, so, according to the semantics, the needed values become available at undetermined times that do not clash with any other mode event occurrences. Note that *EMERGENCY* can only occur once. Having happened, an emergency brings the system to rest, completing the dynamics.

Turning to the *MOVEMENT_AUTHORITY* event, we see that when prompted by the receipt of the input parameter *newMA* from the environment it reassigns the movement authority variables, $\mathbf{m}.r$, $\mathbf{m}.e$, $\mathbf{m}.d$, according to nondeterministically chosen values r, e, d, subject to some restrictions as follows. Firstly, the event can only take place in *normal* mode. Secondly, the values assigned must all be positive, consistent with the restriction that, when under automatic control, the train can only move forwards. Thirdly, the new values for $\mathbf{m}.r$ and $\mathbf{m}.d$ must satisfy $r \geq d$, i.e. the recommended (i.e. cruising) speed is greater than the demanded (i.e. limiting) speed, which is also expressed in *inv*8. This is a natural property to expect, and although not essential, it simplifies some case analysis below. Fourthly, there are two further dynamical restrictions on the new movement authority values.

To understand the first, there is a requirement that any update to a movement authority must be no more demanding than its predecessor, in case the train is already braking as hard as it can in order to remain within the current movement authority. Consequently, if the new demanded speed d is greater than the current one $\mathbf{m}.d$, then since the train is (by assumption) guaranteed to be capable of remaining within the current movement authority (i.e. to not go past $\mathbf{m}.e$), we need only ensure that the new endpoint e is no earlier than the current one, $(d \geq \mathbf{m}.d \Rightarrow e \geq \mathbf{m}.e)$.

To understand the second, consider the following. When ideal one-dimensional motion is governed by acceleration that is piecewise constant over time, then velocity is piecewise linear over the same time periods within which the acceleration is constant, *each piece with respect to an origin of time appropriate to ensuring continuity (though not differentiability) of the velocity as a whole*. Furthermore, in this situation, position is piecewise quadratic, again over the same time periods within which the acceleration is constant, and such that each piece is quadratic with respect to the same origin of time that applied to the velocity, and with an initial value that ensures continuity (though not differentiability beyond first order) of the position as a whole. Thus, during a period of constant ac- or de- celeration a, the velocity behaves like $v = at$ and the position like $d = d_0 + \frac{1}{2}at^2$, with respect to an appropriate origin for time t, and initial position d_0. Eliminating t, we find $d = d_0 + v^2/2a$, so that over some period of constant celeration where the velocity does not cross 0, we have:

MACHINE *ETCS_Mch*
SEES *ETCS_Ctx*
CLOCK $\tau.clk$
PLIANT
  $\tau.p, \tau.v$
VARIABLES
  $\tau.a, \tau.sb$
  mode
  $\mathbf{m}.r, \mathbf{m}.e, \mathbf{m}.d$
INVARIANTS
  $inv0 : \tau.p \in \mathbb{R} \land \tau.p \geq 0$
  $inv1 : \tau.v \in \mathbb{R} \land \tau.v \geq 0$
  $inv2 : \tau.a \in \mathbb{R} \land \tau.a \in [-b \ldots A]$
  $inv3 : \tau.sb \in \mathbb{R} \land \tau.sb \geq 0$
  $inv4 : mode \in MODES$
  $inv5 : \mathbf{m}.r \in \mathbb{R} \land \mathbf{m}.r \geq 0$
  $inv6 : \mathbf{m}.e \in \mathbb{R} \land \mathbf{m}.e \geq 0$
  $inv7 : \mathbf{m}.d \in \mathbb{R} \land \mathbf{m}.d \geq 0$
  $inv8 : \mathbf{m}.r \geq \mathbf{m}.d$
  $inv9 : \tau.p \geq \mathbf{m}.e \Rightarrow \tau.v \leq \mathbf{m}.d$
EVENTS
  INITIALISATION
    STATUS ordinary
    BEGIN
      $\tau.clk := 0$
      $\tau.p, \tau.v := 0, 0$
      $\tau.a, \tau.sb := 0, 0$
      $mode := normal$
      $\mathbf{m}.r, \mathbf{m}.e, \mathbf{m}.d := 0, 0, 0$
    END
  MOVEMENT_AUTHORITY
    STATUS ordinary
    ANY $msg?, r, e, d$
    WHERE
      $mode = normal \land$
      $msg? = newMA \land$
      $r \in \mathbb{R} \land r > 0 \land$
      $e \in \mathbb{R} \land e > 0 \land$
      $d \in \mathbb{R} \land d > 0 \land$
      $r \geq d \land$
      $(d \geq \mathbf{m}.d \Rightarrow e \geq \mathbf{m}.e) \land$
      $(d \leq \mathbf{m}.d \Rightarrow e \geq \mathbf{m}.e + (\mathbf{m}.d^2 - d^2)/2b)$
    THEN
      $\mathbf{m}.r, \mathbf{m}.e, \mathbf{m}.d := r, e, d$
      $\tau.sb := \mathsf{bd}(r + A\varepsilon, d) + \mathsf{od}(r)$
    END
  EMERGENCY
    STATUS ordinary
    ANY $msg?$
    WHERE
      $msg? \in MSGS \land msg? = emrg \land$
      $mode = normal$
    THEN $mode := emergency$
    END
... ...

... ...
  DRIVE
    STATUS pliant
    WHEN $\tau.clk = 0$
    COMPLY $\tau.v \geq 0 \land$
           $\tau.clk < \varepsilon$
    SOLVE $\mathcal{D}\,\tau.v = \tau.a$
          $\mathcal{D}\,\tau.p = \tau.v$
    END
  SPEED_OK
    STATUS ordinary
    ANY a
    WHEN
      $\tau.clk = \varepsilon \land mode = normal \land$
      $\mathbf{m}.e - \tau.p \geq \tau.sb \land \tau.v \leq \mathbf{m}.r \land$
      $a \in [-b \ldots A]$
    THEN $\tau.a := a$
         $\tau.clk := 0$
    END
  SPEED_HIGH
    STATUS ordinary
    WHERE
      $\tau.clk = \varepsilon \land mode = normal \land$
      $\mathbf{m}.e - \tau.p \geq \tau.sb \land \tau.v \geq \mathbf{m}.r$
    THEN $\tau.a := -b$
         $\tau.clk := 0$
    END
  AUTOMATIC_TRAIN_PROTECTION
    STATUS ordinary
    WHEN
      $\tau.clk = \varepsilon \land (mode = emergency \lor$
            $\mathbf{m}.e - \tau.p \leq \tau.sb)$
    THEN $\tau.a := -b$
         $\tau.clk := 0$
    END
  FULL_STOP
    STATUS ordinary
    WHEN $\tau.v = 0 \land mode = emergency$
    THEN $\tau.a := 0$
         $\tau.clk := 0$
    END
  FINAL_TRAIN
    STATUS pliant final
    WHEN
      $\tau.clk = 0 \land mode = emergency \land$
      $\tau.v = 0 \land \tau.a = 0$
    COMPLY skip
    END
END

Figure 11: A Hybrid Event-B machine for the European Train Control System.

$$\text{relative displacement} = \frac{\text{difference in squared velocity}}{2 \times \text{celeration}} \tag{38}$$

where both the numerator and denominator of (38) are positive.

Returning to the last *MOVEMENT_AUTHORITY* guard, if the new demanded speed d is less than the current one $\mathbf{m}.d$, then for the new endpoint e, we must allow an extra distance at least enough to permit maximum braking to successfully bring the train down to velocity d in the worst case. The worst case is given by assuming that the train started braking as hard as possible as late as possible to still remain within the current movement authority. In that case, when the train arrives at the current endpoint $\mathbf{m}.e$, it will be travelling at velocity $\mathbf{m}.d$, by definition. Therefore, to be going at d by the time e is reached, we must add at least $(\mathbf{m}.d^2 - d^2)/2b$ extra displacement onto $\mathbf{m}.e$ to remain feasible, where b is the maximum braking deceleration. Hence $(d \leq \mathbf{m}.d \Rightarrow e \geq \mathbf{m}.e + (\mathbf{m}.d^2 - d^2)/2b)$. We discuss the update to $\tau.sb$ in *MOVEMENT_AUTHORITY* later.

The remaining events refer purely to the train. The only non-final pliant event is *DRIVE*, which is scheduled whenever the clock is reset to 0, and lasts for a period $\tau.clk < \varepsilon$. At the left limit of the endpoint of this period, various mode events can become enabled (via a guard $\tau.clk = \varepsilon$), so by the semantics in Section 7, such events can continue the system trace. The *DRIVE* event itself merely stipulates that the train follows the laws of Newtonian mechanics during any pliant transition specified by this event.

The event *SPEED_OK* stipulates that in *normal* mode, at the end of a polling interval, if the train's current speed does not exceed the recommended maximum and the train has not reached the emergency braking zone, the acceleration for the next polling interval can be set arbitrarily between its static minimum and maximum values. The clock is reset and *DRIVE* is re-enabled for the next polling interval.

The next event is *SPEED_HIGH*. If, in *normal* mode, at the end of a polling interval, the train's current speed exceeds the recommended maximum and the train has not reached the emergency braking zone, the acceleration for the next polling interval is set, for simplicity, to its static minimum. The clock is reset and *DRIVE* is re-enabled for the next polling interval.[22]

If, by the end of a polling interval, the mode has been set to *emergency* or the emergency braking zone has been entered, then in the next event, *AUTOMATIC_TRAIN_PROTECTION*, the acceleration is set to maximum braking and the clock is reset. The actions of this event are identical to those of *SPEED_HIGH* in this very simple model (essentially for the reasons explained in footnote 22).

The last mode event, *FULL_STOP*, is triggered in *emergency* mode when the velocity reaches 0, at which point the acceleration is set to 0 too, and the train's motion stops, enabling the final pliant event *FINAL_TRAIN*, which keeps the train at rest indefinitely henceforth.

We return to the *MOVEMENT_AUTHORITY* event. The job of the train's portion of the event is to update its start braking variable $\tau.sb$, so that it remains consistent with the requirement of being able to decelerate to the new demanded speed d by the time the new endpoint of the movement authority e is reached.

Before resolving the implications of this we observe that if the train is travelling at velocity $\tau.v$, then by (38), to reduce speed to $\mathbf{m}.d$ (assuming that the train is braking at rate b and that $\tau.v \geq \mathbf{m}.d$) requires a braking distance:

[22] N.B. In [38], for the corresponding situation, braking is set arbitrarily between $-b$ and 0 (i.e. it permits no braking at all *in extremis*), but the ensuing safety discussion of the system is always phrased in terms of the train *choosing* maximum braking when appropriate. This is in line with the control engineering concern of *controllability*, i.e. the ability to choose a suitable behaviour for the system under particular circumstances by suitably assigning the controlled variables. This approach amounts to an *angelic choice* of course. In the B-Method, system behaviour is always analysed with respect to *demonic choice*, so we have made the behaviour here more deterministic in order to more easily address the safety requirements.

$$\mathsf{bd}(\tau.v, \mathbf{m}.d) = \frac{\tau.v^2 - \mathbf{m}.d^2}{2 \times b} \tag{39}$$

This means that at all times it must hold that,

$$\mathsf{bd}(\tau.v, \mathbf{m}.d) \leq \tau.sb \tag{40}$$

i.e. (40) must be an invariant.

To go from this to a safety property and to a safe assignment of $\tau.sb$ in *MOVEMENT_AUTHORITY*, we must relate (40) to the data of a movement authority, to the timing of events in the train, and to how movement authority data changes during the *MOVEMENT_AUTHORITY* event.

If the train is travelling at velocity $\tau.v$ and $\tau.v \leq \mathbf{m}.r$, then the maximum speed attainable (within an unchanging movement authority) is $\mathbf{m}.r + A\varepsilon$. This is because the only event that can make the acceleration positive is *SPEED_OK*, and this event is still enabled when $\tau.v = \mathbf{m}.r$. At that point *SPEED_OK* might choose to set $\tau.a$ to as much as A, which could increase the speed to as much as $\mathbf{m}.r + A\varepsilon$ over the next polling interval. After that, *SPEED_OK* will be disabled and the only other mode events all make $\tau.a$ nonpositive; so speed $\mathbf{m}.r + A\varepsilon$ cannot be exceeded.

In going from $\mathbf{m}.r$ to $\mathbf{m}.r + A\varepsilon$ the train travels an overshoot distance which is at most:

$$\mathsf{od}(\mathbf{m}.r) = \mathbf{m}.r\,\varepsilon + \frac{1}{2}A\varepsilon^2 \tag{41}$$

Therefore, if $\tau.v \leq \mathbf{m}.r$ holds at some point and the movement authority does not change, then

$$\mathsf{bd}(\mathbf{m}.r + A\varepsilon, \mathbf{m}.d) + \mathsf{od}(\mathbf{m}.r) \leq \tau.sb \tag{42}$$

represents a safe static weakening of (40) for the remainder of the movement authority. (Note that we have used *inv*8 here.)

Alternatively, if the train is travelling at velocity $\tau.v$ and $\tau.v \geq \mathbf{m}.r$, then on the next polling occurrence, the train will be compelled to reduce speed to $\mathbf{m}.r$. During this speed reduction the train will travel a distance, at most:

$$\mathsf{bd}(\tau.v + A\varepsilon, \mathbf{m}.r) + \mathsf{od}(\tau.v) \tag{43}$$

and, if $\mathbf{m}.e$ is close enough and deceleration is to continue down to $\mathbf{m}.d$, it will require a further distance of $\mathsf{bd}(\mathbf{m}.r, \mathbf{m}.d)$ to reach demanded speed, making a total of $\mathsf{bd}(\tau.v + A\varepsilon, \mathbf{m}.d) + \mathsf{od}(\tau.v)$.

The above facilitates a case analysis for determining a safe value of $\tau.sb$ when the movement authority is updated to a new tuple of values r, e, d.

If $\tau.v \leq r$, then we can use the first case above to set $\tau.sb$ to $\mathsf{bd}(r + A\varepsilon, d) + \mathsf{od}(r)$. If $\tau.v \geq r$, then we can rely on *SPEED_HIGH* or *AUTOMATIC_TRAIN_PROTECTION* to 'immediately'[23] start braking to reduce the speed to r. After that, an assignment of $\tau.sb$ to $\mathsf{bd}(r + A\varepsilon, d) + \mathsf{od}(r)$ will take care of deceleration to demanded speed when needed. Thus the value to be assigned to $\tau.sb$ is the same in both cases, although the justification is different in the two branches. This completes our discussion of *MOVEMENT_AUTHORITY* and of the ECTS case study.

[23] 'Immediately' means within an overshoot tolerance of $\mathsf{od}(\tau.v)$ which will have been allowed for in a preceding movement authority.

10.4. Some Behaviours of the ETCS Hybrid Event-B Machine.

Superficially all seems well. However, when we look at things in more detail, potentially undesirable system behaviours become apparent.

Consider the following system behaviour **SB1**. The system is initialised. During the first polling interval nothing changes (except the clock). At the next mode transition, *SPEED_OK* is enabled, chooses $\tau.a = A$ and is scheduled; the train starts to accelerate. At the end of the next polling interval the invariants are checked and since the speed is now $A\varepsilon$, invariant *inv*9 fails. Therefore **SB1** ABORTs. We conclude that the ETCS machine cannot be correct according to the criteria in Definition 7.1.

Now consider system behaviour **SB2**. The first polling interval is as in **SB1**. At the next mode transition, *SPEED_OK* chooses $\tau.a = 0$; the train remains stationary. Subsequent mode and pliant transitions are replicas of these two. The completely stationary behaviour carries on indefinitely. Since no ABORT is encountered, we conclude that the ETCS machine is at least non-void according to the criteria in Definition 7.1.

The reason for the failure of **SB1** is not hard to find. The initialisation of $\tau.sb$ did not take into account the more delicate reasoning that revealed the need for od in calculating $\tau.sb$.

Now consider system behaviour **SB3[**$\tau.sb/\mathsf{od}(0)$**]**, in which we change the initialisation so that $\tau.sb$ is initialised to $\mathsf{od}(0)$. Now, after the first polling interval (during which, the only thing that changes is the clock), only *AUTOMATIC_TRAIN_PROTECTION* is enabled and $\tau.a$ is set to $-b$. In the next polling interval *DRIVE* is infeasible since, with an initial velocity of 0 and negative $\tau.a$, it becomes impossible to COMPLY with $\tau.v \geq 0$ for any finite time. So **SB3[**$\tau.sb/\mathsf{od}(0)$**]** also ABORTs.

Consider next system behaviour **SB4**, in which, exactly at the end of the first polling interval (i.e. the first occurrence of $\tau.clk = \varepsilon$), a *MOVEMENT_AUTHORITY* event occurs which sets the movement authority data to 'sensible values' that permit the train to move forward while maintaining the invariants. Suppose the train reaches the emergency braking zone, i.e. *AUTOMATIC_TRAIN_PROTECTION* becomes enabled. The train decelerates, and suppose its velocity reaches 0 when the clock reads $\tau.clk = \varepsilon/2$, making the *DRIVE* event no longer feasible. Suppose no mode event occurs at this time. Then we have successful finite termination.

Now consider system behaviour **SB5**. This is just like **SB4**, but when the train has stopped midway through a polling interval at $\tau.clk = \varepsilon/2$, a *MOVEMENT_AUTHORITY* event occurs precisely at that moment (because the environment produced suitable r, e, d values just then) that sets the movement authority data to some new sensible values that (in their own terms) permit sensible progress of the train. After the *MOVEMENT_AUTHORITY* event occurrence, the *DRIVE* pliant event is disabled (because $\tau.clk \neq \varepsilon$). Since there is no other enabled pliant event after the *MOVEMENT_AUTHORITY* event, the semantics causes an ABORT.

Finally, consider system behaviour **SB6**. This is like **SB5**, except that the original movement authority data are such that the train comes to a standstill at a polling interval boundary, i.e. $\tau.clk = \varepsilon$. A *MOVEMENT_AUTHORITY* event occurs precisely then, reassigning the movement authority data to new sensible values. This time the train can continue moving according to the new data and there is no ABORT.

The above scenarios, consequences of a fairly uncritical transliteration of the ECTS case study in [38], serve to show a number of things. Firstly, they illuminate some of the darker corners of the Hybrid Event-B semantics of Section 7. This, although giving a defined behaviour for all Hybrid Event-B projects is, in practice, such that we would want to exclude the more undesirable of the possibilities via suitably stringent static checks. Secondly, the uncritical transliteration discarded a number of the properties inherent in the original $d\mathcal{L}$ programs in [38]. For example, in the original treatment of [38], *MOVEMENT_AUTHORITY* was only scheduled at polling interval boundaries, and also, *AUTOMATIC_TRAIN_PROTECTION*, if enabled, always overrode the *SPEED_OK* and *SPEED_HIGH* provisions due to being sequentially composed after them — such issues are easy to fix via more careful programming

and this would obviously be taken care of in a more serious attempt at ECTS via Hybrid Event-B. Thirdly, we also saw the consequences of the purely demonic policy of the B-Method approach, versus the option of using angelic choice as utilised in controllability arguments. This forced us to change the behaviour of *SPEED_HIGH*, in order to get any guarantee that when the train needed to, then (aside from emergencies), it could actually be relied on to slow down.

11. Conclusions

In this paper we recalled conventional Event-B before embarking on a design of an extension that would cope with the demands of the continuous behaviours exhibited by today's hybrid and cyber-physical systems. We examined in some detail the often unstated assumptions behind the relationship between discrete event based systems (such as discrete Event-B) and the real world, in order that the extension that we eventually presented disturbed existing Event-B conventions and assumptions as little as possible.[24] As well as seeking to minimise the human risk that accompanies inadvertent change to unspoken assumptions, seeking to stay as consistent as possible with the existing framework for discrete Event-B enables us to undermine as little as possible the existing features of Event-B as implemented in the Rodin tool, in which so much effort has been invested to date.

We then examined how these conventions and assumptions could be extended to encompass the needs of Hybrid Event-B. The exercise focused on the semantic domain, to determine the universe of mathematical objects in which the extended language would take its values. Given the nature of typical engineering applications, in which discrete discontinuities in signals commonly occur as systems move from mode to mode, the chosen universe was the world of piecewise absolutely continuous functions of time, which allowed characterisation in various ways, DEs, assignments, and predicates with models in (sets of) such functions. We also examined the implications of imposing a Zeno condition.

After that we presented Hybrid Event-B itself, giving the syntax and semantics for a Hybrid Event-B machine. We then moved on to consider refinement. In seeking to disturb existing Event-B as little as possible, we kept continuous behaviour apart from the existing discrete event framework as far as possible, and this goal proved achievable.

In Section 9 we gathered together the proof obligations that would give substance to the semantics of this framework in the Event-B style, and we gave two simple correctness results. In the last section we gave a collection of examples of Hybrid Event-B modelling. After considering the bouncing ball and a simple discretization problem, we ended with a simple version of the European Train Control System. This case study, deliberately patterned rather loosely after the models in [38], gave us an opportunity to discuss how some of the darker corners of the semantics of Hybrid Event-B could be exercised by imprudently designed Hybrid Event-B specifications. Future work will extend the present account to multiple Hybrid Event-B machines, and further, to include stochastic behaviour as first class citizen.

References

[1] Report: Cyber-Physical Systems, 2008. http://iccps2012.cse.wustl.edu/_doc/CPS_Summit_Report.pdf.

[2] J.R. Abrial, The B-Book: Assigning Programs to Meanings, Cambridge University Press, 1996.

[24] One is tempted to think of the Mars Lander story, in which the different measurement units used by the co-operating teams led to the disaster. Unquestioningly assumed to be the same, but different in reality (the USA using Imperial units, and the EU using metric units), the mismatch, which led to the crash of the vehicle, vividly illustrates the potentially destructive power of innocent change to unspoken assumptions.

[3] J.R. Abrial, Modeling in Event-B: System and Software Engineering, Cambridge University Press, 2010.

[4] ADVANCE. European Project ADVANCE. IST-287563 http://http://www.advance-ict.eu/.

[5] R. Alur, C. Courcoubetis, T. Henzinger, P.H. Ho, Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems, in: Proc. Workshop on Theory of Hybrid Systems, volume 736 of *LNCS*, Springer, 1993, pp. 209–229.

[6] R. Alur, D. Dill, A Theory of Timed Automata, Theor. Comp. Sci. 126 (1994) 183–235.

[7] G. Audemard, M. Bozzano, A. Cimatti, R. Sebastiani, Verifying Industrial Hybrid Systems with MathSAT, in: Proc. BMC-04, volume 119, ENTCS, 2005, pp. 17–32.

[8] R.J. Back, L. Petre, I. Porres, Continuous Action Systems as a Model for Hybrid Systems, Nordic J. Comp. 8 (2001) 2–21. Extended version of FTRTFT-00, LNCS 1926, 202-213.

[9] R. Banach, Pliant Modalities in Hybrid Event-B, in: Liu, Woodcock, Zhu (Eds.), Proc. Jifeng He Festschrift 2013, volume 8051 of *LNCS*, Springer, 2013, pp. 37–53.

[10] R. Banach, The Landing Gear Case Study in Hybrid Event-B, in: Boniol, Weils, Ait Ameur, Schewe (Eds.), Proc. ABZ 2014: Landing Gear Case Study, volume 433 of *CCIS*, Springer, 2014, pp. 126–141.

[11] R. Banach, M. Butler, A Hybrid Event-B Study of Lane Centering, in: Aiguier, Boulanger, Krob, Marchal (Eds.), Proc. CSDM-13, Springer, 2013, pp. 97–111.

[12] R. Banach, M. Butler, Cruise Control in Hybrid Event-B, in: Liu, Woodcock, Zhu (Eds.), Proc. ICTAC-13, volume 8049 of *LNCS*, Springer, 2013, pp. 76–93.

[13] L. Barolli, M. Takizawa, F. Hussain, Special Issue on Emerging Trends in Cyber-Physical Systems, J. Amb. Intel. Hum. Comp. 2 (2011) 249–250.

[14] Bender, K. and Broy, M. and Péter, I. and Pretschner, A. and Stauner, T., Model Based Development of Hybrid Systems: Specification, Simulation, Test Case Generation, in: Modelling, Analysis, and Design of Hybrid Systems, volume 279, Springer, LNCIS, 2002, pp. 37–51.

[15] L. Carloni, R. Passerone, A. Pinto, A. Sangiovanni-Vincentelli, Languages and Tools for Hybrid Systems Design, Foundations and Trends in Electronic Design Automation 1 (2006) 1–193.

[16] A. Cimatti, M. Roveri, Requirements Validation for Hybrid Systems, in: Proc. CAV-09, volume 5643, Springer, LNCS, 2009, pp. 188–203.

[17] E. Clarke, A. Fehnker, Z. Han, B. Krogh, O. Stursberg, M. Theobald, Verification of Hybrid Systems Based on Counterexample-Guided Abstraction Refinement, in: Proc. TACAS-03, volume 2619, Springer, LNCS, 2003, pp. 192–207.

[18] E. Clarke, P. Zuliani, Statistical Model Checking for Cyber-Physical Systems, in: Bultan, Hsiung (Eds.), Proc. ATVA-11, volume 6996 of *LNCS*, Springer, 2011, pp. 1–12.

[19] F. Clarke, Optimization and Nonsmooth Analysis, Society for Industrial Mathematics, 1987.

[20] F. Clarke, Y. Ledyaev, R. Stern, P. Wolenski, Nonsmooth Analysis and Control Theory, Springer, 1997.

[21] DEPLOY. European Project DEPLOY. IST-511599 http://www.deploy-project.eu/.

[22] A. Deshpande, A. Göllü, P. Varaiya, SHIFT: A Formalism and a Programming Language for Dynamic Networks of Hybrid Automata, in: Proc. Hybrid Systems IV, volume 1273, Springer, LNCS, 1997, pp. 113–133.

[23] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, O. Maler, SpaceEx: Scalable Verification of Hybrid Systems, in: Proc. CAV-11, volume 6806, Springer, LNCS, 2011, pp. 379–395.

[24] V. Friesen, A. Nordwig, M. Weber, Object-Oriented Specification of Hybrid Systems using UML, h and ZimOO, in: Proc. ZUM-98, volume 1493, Springer, LNCS, 1998, pp. 328–346.

[25] V. Friesen, A. Nordwig, M. Weber, Toward an Object-Oriented Design Methodology for Hybrid Systems, Object-Oriented Technology and Computing Systems Re-Engineering (1999) 1.

[26] Grosu, R. and Stauner, T. and Broy, M., A Modular Visual Model for Hybrid Systems, in: Proc. FTRTFT-98, volume 1486, Springer, LNCS, 1998, pp. 75–91.

[27] R. Haggarty, Fundamentals of Mathematical Analysis, Prentice Hall, 1993.

[28] J. He, From CSP to Hybrid Systems, in: Roscoe (Ed.), A Classical Mind, Essays in Honour of C.A.R. Hoare, Prentice-Hall, 1994, pp. 171–189.

[29] T. Henzinger, The Theory of Hybrid Automata, in: Proc. IEEE LICS-96, IEEE, 1996, pp. 278–292. Also http://mtc.epfl.ch/~tah/ Publications/the_theory_of_hybrid_automata.pdf.

[30] Y. Kesten, Z. Manna, A. Pnueli, Verification of Clocked and Hybrid Systems, Acta Informatica 36 (2000) 837–912.

[31] S. Lang, Real and Functional Analysis, Springer, 1993.

[32] R. Leadbetter, S. Cambanis, V. Pipiras, A Basic Course in Measure and Probability, Cambridge University Press, 2014.

[33] N. Lynch, R. Segala, F. Vaandrager, Hybrid I/O Automata, Information and Computation 185 (2003) 105–157. MIT Technical Report MIT-LCS-TR-827d.

[34] N. Lynch, R. Segala, F. Vaandrager, H. Weinberg, Hybrid I/O Automata, Springer, 1996.

[35] Z. Manna, A. Pnueli, Verifying Hybrid Systems, in: Hybrid Systems, volume 736, Springer, LNCS, 1993, pp. 4–35.

[36] P. Mosterman. Private Communication.

[37] National Science and Technology Council, Trustworthy Cyberspace: Strategic plan for the Federal Cybersecurity Research and Development Program, 2011.
http://www.whitehouse.gov/sites/default/files/microsites/ostp/fed_cybersecurity_rd_strategic_plan_2011.pdf.

[38] A. Platzer, Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics, Springer, 2010.

[39] RODIN. European Project RODIN (Rigorous Open Development for Complex Systems) IST-511599 http://rodin.cs.ncl.ac.uk/.

[40] RODIN Tool. http://www.event-b.org/ http://www.rodintools.org/ http://sourceforge.net/projects/rodin-b-sharp/.

[41] H. Royden, P. Fitzpatrick, Real Analysis, Pearson, 2010.

[42] W. Rudin, Principles of Mathematical Analysis, McGraw-Hill, 1976.

[43] T. Stauner, B. Rumpe, P. Scholz, Hybrid System Model, 1999. TUM-I9903.

[44] M. Stehr, M. Kim, C. Talcott, Toward Distributed Declarative Control of Networked Cyber-Physical Systems, in: Yu, Liscano, Chen, Zhang, Zhou (Eds.), Proc. UIC-10, volume 6406 of *LNCS*, Springer, 2010, pp. 397–413.

[45] J. Sztipanovits, 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.

[46] P. Tabuada, Verification and Control of Hybrid Systems: A Symbolic Approach, Springer, 2009.

[47] L. Voisin, J.R. Abrial, The Rodin Platform has Turned Ten, in: Ait Ameur, Schewe (Ed.), Proc. ABZ-14, volume 8477, Springer, LNCS, 2014, pp. 1–8.

[48] W. Walter, Ordinary Differential Equations, Springer, 1998.

[49] J. White, S. Clarke, C. Groba, B. Dougherty, C. Thompson, D. Schmidt, R&D Challenges and Solutions for Mobile Cyber-Physical Applications and Supporting Internet Services, J. Internet Serv. Appl. 1 (2010) 45–56.

[50] Wikipedia, Absolute continuity.

[51] J. Willems, Open Dynamical Systems: Their Aims and their Origins. Ruberti Lecture, Rome, 2007. http://homes.esat.kuleuven.be/~jwillems/Lectures/2007/Rubertilecture.pdf.

[52] L. Zhang, J. He, A Formal Framework for Aspect-Oriented Specification of Cyber Physical Systems, in: Lee, Howard, Slezak (Eds.), Proc. ICHIT-11, volume 206 of *CCIS*, Springer, 2011, pp. 391–398.

[53] C. Zhou, J. Wang, A. Ravn, A Formal Description of Hybrid Systems, in: Alur, Sontag, Henzinger (Eds.), Proc. HS-95, volume 1066, Springer, LNCS, 1995, pp. 511–530.

[54] L. Zhühlke, L. Ollinger, Agile Automation Systems Based on Cyber-Physical Systems and Service Oriented Architectures, in: Lee (Ed.), Proc. ICAR-12, volume 122 of *LNEE*, Springer, 2012, pp. 567–574.