Moded and Continuous Abstract State Machines

Richard Banach*[1] and Huibiao Zhu**[2]

[1]Department of Computer Science, University of Manchester, Manchester, M13 9PL, U.K.
[2]Shanghai Key Laboratory of Trustworthy Computing, MOE International Joint Laboratory of Trustworthy Software, International Research Center of Trustworthy Software, East China Normal University, Shanghai, China.
richard.banach@manchester.ac.uk , hbzhu@sei.ecnu.edu.cn

Abstract. In view of the increasing importance of cyber-physical systems, and of their correct design, the Abstract State Machine (ASM) framework is extended to include continuously varying quantities as well as the conventional discretely changing ones. This opens the door to the more faithful modelling of many scenarios where digital systems have to interact with the continuously varying physical world. Transitions in the extended framework are thus either *moded* (catering for discontinuously changing quantities), or *pliant* (catering for smoothly changing quantities). An operational semantics is provided, first for monolithic systems, and this is then extended to give a semantics for systems consisting of several distinct subsystems. This allows each subsystem to undergo its own subsystem-specific mode and pliant transitions. Refinement is elaborated in the extended context for both monolithic and composed systems. The formalism is illustrated using an example of a bouncing tennis ball.

1 Introduction

Conventional model based formal refinement technologies (e.g. [17, 54, 2, 3, 13]) are based on purely discrete concepts. These are typically ill suited to modelling applications which are best expressed using continuous mathematics. So there is a mismatch between the continuous modelling needed at the abstract level, and the discrete techniques used close to code in hybrid and cyber-physical (CPS) systems [48, 37, 38, 22, 1, 36, 47, 15, 55, 8, 53, 29, 18].

Hybrid and CPS systems display considerable complexity in their behaviour, which poses challenges for verification techniques. One well respected way of confronting verification complexity is the top-down design and development approach. Supported by suitable formal notions, it allows complex behaviour to be

* A large 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.

** Huibiao Zhu is supported by National Key Research and Development Program of China (Grant No. 2018YFB2101300), National Natural Science Foundation of China (Grant No. 61872145), and Shanghai Collaborative Innovation Center of Trustworthy Software for Internet of Things (Grant No.ZF1213).

approached in stages, with properties that have been established previously persisting in suitable form as more design detail is added. This makes it eminently suited to confront the challenges posed by complex systems development.

The Abstract State Machine (ASM) approach [13, 11] is an established methodology for top-down design and development. It differs from many other formal approaches by having a very liberal type system, based on universal algebra rather than a fixed collection of low level built-in types. Following on from this, the model of state update in ASM is based on the idea of modifying dynamic functions, a generalisation of the idea of state variables (although, most often, this full generality is not needed).

In this paper we present an extension of the ASM formalism that enables us to treat continuously changing quantities fluently, especially at the abstract level. This is essential if we are to model hybrid and cyber-physical systems effectively. We also develop the needed extension of ASM refinement. The ASM extension is based on restricting the continuous behaviours that are permitted to those which can be described, piecewise, by solutions to well posed initial value problems, this being sufficient for most engineering purposes. These fundamental ideas are applied both to monolithic systems, and to systems consisting of several cooperating subsystems.

The rest of this paper is structured as follows. In Section 2 we review the essentials of ASM, and then describe the continuous extension. We base this on a discussion of the desired semantic domain first, and then construct the syntax and the desired semantics to map cleanly to it. Section 3 covers a simple example concerning a tennis ball bouncing back and forth over a tennis net. Section 4 discusses the formal operational semantics of the given description. The formal semantics lends itself to defining the semantics of composed (or decomposed) systems, which we also discuss. With the detailed semantics covered, in Section 5 we return to the tennis ball example to explore some of its more subtle aspects. Then in Section 6 we develop the refinement machinery relevant to Continuous ASM. This is given as a minimal generalisation of the discrete formulation, and is followed by a discussion of compositionality issues. Section 7 returns to the tennis ball and discusses a simple refinement scenario for the example. Section 8 discusses related work, while Section 9 concludes.

2 ASM, Discrete and Continuous

In this section we review the essentials of ASM [13, 11], and extend the formalism to cope with continuously varying quantities. The advantage of considering such a clean extension of a discrete formalism is that it opens the door to adapting existing tools for the discrete formalism, rather than having to start from scratch.

2.1 Discrete Basic ASM Models

A definitive description of conventional, discrete ASM, is given in [13]. Here, we give an overview sufficient to prepare the ground for the continuous extension.

As noted above, ASM is founded on concepts of universal algebra [25, 50]. This starts by defining *signatures*, from which we can generate *algebras* and then look for *models* that satisfy the constraints imposed by the algebras. These static structures constitute a universe within which ASM dynamics runs its course.

Focusing on the concept of basic ASM, the key update notion is carried by *dynamic functions*, functions that get (partly) redefined by updates of the form:

$$f(t_1, \ldots, t_n) := t \tag{1}$$

In (1), the $t_1, \ldots, t_n, t$ are terms evaluated in the current state, i.e. with respect to the current definitions of all the elements of the algebras (static and dynamic); with these values, the dynamic function f at the element of its domain consisting of the values of the tuple $t_1, \ldots, t_n$ is redefined to be the value of t. If f is a nullary dynamic function, then (1) corresponds to updating a variable.

A basic ASM *transition rule* is a construct of the form:

if *Condition* **then** *Updates* (2)

where the *Updates* are as in (1), and *Condition* evaluates to a truth value. In practice, the basic form in (2) is enhanced to improve readability by admitting various syntactic sugars: the usual elaborations of the conditional; a **forall** x **with** *cond Rule* form for iterating *Rule* over a collection ranged over by x (with x constrained by *cond*); and a **choose** x **with** *cond Rule* form to allow *Rule* to be nondeterministic. Below, we only need rules of the form:

OP(**in** is, **out** os) =
if $guard(xs, is)$ **then**
choose xs', os **with** $rel(xs', xs, is, os)$
do $xs, os := xs', os$ (3)

In (3) the rule's name is OP, and we have (read-only) inputs **in** is and (write-only) outputs **out** os in the signature of OP. For us, a basic ASM is a finite set of such rules.

We say that a rule like (3) is enabled if its *guard* evaluates to true. In most model based development formalisms, given a state of the model, i.e. a valuation that maps each variable to a value in its type, progress is made by selecting *one* of the enabled rules and executing it. The ASM policy though, is that *all* enabled rules are selected, and their updates are performed in parallel. So the sets of enabled rules that arise must define *consistent* sets of updates. If not, execution aborts. A *run* of an ASM system thus starts at an initial state, and continues via a succession of state changes, defined by maximal sets of enabled rules that define consistent update sets.

2.2 Continuous ASM Models

We extend the framework above to the continuous world by first examining the semantic domain. Looking ahead a little, we will be using differential equations

(DEs), and therefore, for mathematical consistency, we need to be precise about the semantic domain with respect to which the DEs will be interpreted.

For simplicity, we restrict to the case in which the states are given by valuations of the tuple of variables of the model, i.e. functions from the tuple of variables to the tuple of the variables' types. To extend such models smoothly to include continuously varying phenomena, we partition the variables into two kinds: **mode variables**, whose types are discrete sets, and which only change discontinuously, and **pliant variables**, whose types include topologically dense sets, and which are permitted to evolve both continuously and via discrete changes. In our terminology, discrete ASM just uses mode variables.

We model time as a left-closed interval $\mathcal{T}$ of the reals $\mathbb{R}$, with a finite left endpoint for the initial state, and with a right endpoint which is either finite (and right-open) or infinite, depending on whether the dynamics is finite or infinite. Now, the values of all variables become functions of $\mathcal{T}$.

For a mode variable v, the function is a piecewise constant function, which is constant on each element of a sequence of left-closed right-open intervals. Thus the behaviour of v partitions $\mathcal{T}$ into a sequence of left-closed right-open intervals, $\langle [t_{v,0} \ldots t_{v,1}), [t_{v,1} \ldots t_{v,2}), \ldots \rangle$, on each piece of which the behaviour is constant.

For a pliant variable x, the permitted behaviours are piecewise continuous.[1] These pieces again partition $\mathcal{T}$ into a sequence of left-closed right-open intervals, $\langle [t_{x,0} \ldots t_{x,1}), [t_{x,1} \ldots t_{x,2}), \ldots \rangle$, on each piece of which the behaviour undergoes no discontinuities.

Putting together all the behaviours of all the variables that participate in defining a particular execution of a system, yields a sequence of left-closed right-open intervals, $\langle [t_0 \ldots t_1), [t_1 \ldots t_2), \ldots \rangle$, which is the coarsest partition of $\mathcal{T}$ into such intervals where all discontinuous changes of all the variables of the system during that execution take place at a boundary point t_i.

We note at this juncture that our formulation is by no means the first work on the ASM formalism to consider the notion of time *per se*. In this context we could mention the earlier work in [26, 12, 9, 23, 24, 42, 16, 45] for example. While all of these are concerned with time, in all of them the concern is with pure timing, i.e. there is no continuously varying behaviour. So in our formulation, states in all of these works are piecewise constant functions of time.

In a typical interval $[t_i \ldots t_{i+1})$, mode variables will be constant, but pliant variables will change in a continuously varying manner. However, mere continuity still allows for a very wide range of mathematically pathological behaviours.[2] To constrain these, we make the following restrictions and recommendations:

I Zeno: there is a constant δ_{Zeno}, such that for all i needed, $t_{i+1} - t_i \geq \delta_{\mathsf{Zeno}}$. N. B. Since the presence or absence of Zeno behaviour is usually a global property of a system's reachability relation, this point must be regarded as a recommendation rather than a restriction that is statically enforceable.

[1] We mention below that actually, we need *absolute* continuity, not mere continuity alone.

[2] Texts on mathematical analysis are usually replete with relevant examples.

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

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

It is recognised that ASM types can be mathematically complex entities. Therefore it is intended that **I-III** above apply to variables with as general a type as might be needed, provided that the concepts required in **I-III** (such as left/right limits, initial value problem, Lipschitz constants, uniform boundedness, measurability) make sense for them. That said, in the overwhelming majority of cases, the conventional real type $\mathbb{R}$ is sufficient, so we do not consider more complicated possibilities in this paper.

With **I-III** in place, the behaviour of every pliant variable is piecewise *absolutely* continuous [41, 52], with the variation being described by a suitable differential equation.

Accompanying the distinction between mode and pliant variables, is a distinction between mode and pliant transitions. Mode transitions are just like conventional ASM transitions in that they record a discrete transition from before-values to after-values of the mode variables, albeit that these are the values of piecewise constant functions of time. A rule for a mode transition Op can be written using familiar ASM notation:

$$
\begin{array}{l}
\textsc{Op}(\textbf{in}\ \overrightarrow{is}, \textbf{out}\ \overleftarrow{os}) \ = \\
\textbf{if}\ guard(\overrightarrow{xs}, \overrightarrow{is})\ \textbf{then} \\
\textbf{choose}\ \overleftarrow{xs'}, \overleftarrow{os}\ \textbf{with}\ rel(\overleftarrow{xs'}, \overrightarrow{xs}, \overrightarrow{is}, \overleftarrow{os}) \\
\textbf{do}\ xs, os \ := \ \overleftarrow{xs'}, \overleftarrow{os}
\end{array}
\tag{4}
$$

In (4), the overarrows are semantic decorations. These are not part of the syntax, but are included for clarity to indicate which limiting value for a variable (selected from its behaviour as a function of time) is to be taken as being referred to in the rule. This needs to be understood since all runtime executions of Op take place at points of discontinuity in the temporal behaviour of (at least some of the) variables, because rules like Op are intended precisely to define such discontinuities. Note therefore that the choice of left limit for before-values and right limit for after-values (at a given transition point) makes (4) into the kind of instantaneous transition we would expect. Stripping off the overarrows from (4) yields the form one would write to describe a rule in a specific application.

In (4) we single out the inputs is and outputs os, (read-only and write-only respectively), while xs are the state variables (accessed in read/write man-

ner). Note the double decoration of the after-state variables $\overleftarrow{xs'}$. The prime corresponds to the usual syntactic decoration that one would expect to use in distinguishing before-states (unprimed) from after-states (primed), whereas the overarrow indicates the temporal semantic interpretation. Obviously, if the after-values for xs and os are available explicitly, the relevant expression can be assigned in the **do** clause, and the **choose** and **with** clauses can be omitted.

Pliant transitions do the corresponding job for pliant variables. While a mode transition is a single before-/after-value pair, a pliant transition is a family of before-/after-value pairs parameterised by the relevant time interval $[t_i \ldots t_{i+1})$. Moreover, instead of the change from before-values to after-values taking place instantaneously, the before-value can be understood to refer to the initial value at t_i (which, by **II**, equals the right limit at t_i), while the after-value refers to an arbitrary time in the open interval $(t_i \ldots t_{i+1})$, so the before-value and after-value are separated in time. To reflect the constraints that apply to pliant transitions, we write rules for them thus:

$$\begin{array}{l}\textsc{PliOp}(\textbf{in}\ is(t \in (t_{\mathrm{L}(t)} \ldots t_{\mathrm{R}(t)})), \textbf{out}\ os(t \in (t_{\mathrm{L}(t)} \ldots t_{\mathrm{R}(t)}))) \stackrel{\mathrm{c}}{=} \\ \textbf{if}\ IV(xs(t_{\mathrm{L}(t)})) \wedge guard(xs(t_{\mathrm{L}(t)}))\ \textbf{then} \\ \textbf{with}\ rel(xs, is, os, t) \\ \textbf{do}\ xs(t), os(t)\ :=\ \textbf{solve}\ DE(xs(t), is(t), os(t), t)\end{array} \tag{5}$$

In (5), the symbol $\stackrel{\mathrm{c}}{=}$ signals the presence of a rule for a pliant transition, distinguishing it from the instantaneously executed kind. The notations $t_{\mathrm{L}(t)}$ and $t_{\mathrm{R}(t)}$ refer to the beginning and end, respectively, of the time interval during which PliOp executes. Of course, the values of these cannot be known statically, even disregarding the fact that different invocations of the rule at runtime will require different values for $t_{\mathrm{L}(t)}$ and $t_{\mathrm{R}(t)}$. Therefore all explicitly given references in (5) to variables' time dependencies, and to $t_{\mathrm{L}(t)}$ and $t_{\mathrm{R}(t)}$ values, are semantic decorations, included for readability, and indicated by the shading. They do not form part of the syntactic form of the rule, and so "$t \in (t_{\mathrm{L}(t)} \ldots t_{\mathrm{R}(t)})$" in the declaration of the input "**in** $is(\ldots)$" is redundant (similarly for the output), as is "(t)" in occurrences of "$xs(t)$", etc. In this paper, we have opted to retain the "$var(t)$" way of referring to the time dependent behaviour of pliant variables inside pliant rules, for improved readability.

Given a specific execution of the system, which generates a specific partition of $\mathcal{T}$, for an arbitrary t, we define $\mathrm{L}(t) = \max\{i \mid t_i \leq t\}$ and $\mathrm{R}(t) = \min\{i \mid t_i > t\}$ (with obvious default for an infinite last interval) which yields the indexes in the partition of $\mathcal{T}$ relevant to the subinterval of $\mathcal{T}$ to which t belongs. This is consistent with the notations $t_{\mathrm{L}(t)}$ and $t_{\mathrm{R}(t)}$ in (5). These devices allow us to refer to the beginning and end of the interval during which the pliant event runs in a generic manner in our meta level discussions. They also permit, despite what has been said above, rules like PliOp to refer to *relative time* from the beginning of the execution of a transition specified by PliOp, by using expressions like $(t - t_{\mathrm{L}(t)})$. This is useful within clauses such as $rel(xs, is, os, t)$, for example. Obviously, fresh syntactic sugar could be introduced to handle this, if desired.

For a specific execution of PliOp, the inputs is and outputs os are continuously absorbed from and emitted to the environment over the open interval $(t_{\mathrm{L}(t)} \ldots t_{\mathrm{R}(t)})$, as indicated in the signature. (Both must be absolutely right continuous.) Note that the initial values IV and guard $guard$ depend only on the before-value of the state,[3] and not on the input, whereas rel, which expresses any additional constraints that must hold beyond the differential equation DE itself, can depend on all state and input values from the start of the interval $t_{\mathrm{L}(t)}$ up to the current time t. The assignment in (5) says that the after-state and output at t should satisfy the differential equation DE (as well as rel). As for the instantaneous case, if the continuous functions of t to be assigned to xs, os are known explicitly, we can omit the **with** and/or **solve** clauses as appropriate, and just assign xs, os to the relevant expression.

As mentioned earlier, pliant variables can undergo instantaneous discontinuous transitions as well as continuous ones. For such transitions, the structure in (4) is sufficient. We continue to call instantaneous transitions involving both kinds of variable **mode transitions**, introducing the term **pure mode transitions** for the former kind.

A continuous ASM ruleset is **well formed** iff:

- 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).
- Every enabled pliant transition is feasible, i.e. has a time-indexed family of after-states, and EITHER:
  - (i) During the run of the pliant transition a mode transition becomes enabled. It preempts the pliant transition, defining its end. ORELSE
  - (ii) During the run of the pliant transition it becomes infeasible: finite termination. ORELSE
  - (iii) The pliant transition continues indefinitely: nontermination.

A **run** of a continuous ASM system starts with a mode transition which assigns the initial state of all system variables, and then, pliant transitions alternate with mode transitions. The last transition (if there is one) is a pliant transition (whose duration may be finite or infinite). We thus see that the sequence t_i of times at which discontinuities take place, emerges as the sequence of times at which the first possible preemptions of the pliant transitions by the enabling of mode transitions arises.

3 Example: A Bouncing Tennis Ball

To illustrate our formalism, we consider an idealised tennis rally, in which a pointlike tennis ball of unit mass is being hit back and forth over the tennis net, which is of height N. Let the horizontal and vertical components of the ball's

[3] Normally, we would expect IV to depend on the pliant variables and $guard$ to depend on the mode variables, but there is no need to insist on this formally.

velocity be vx and vy, positive for rightwards and upwards motion. Suppose horizontal and vertical positions are measured from the bottom point of the net, positive for rightwards and upwards displacements, and for the tennis ball, these are px and py.

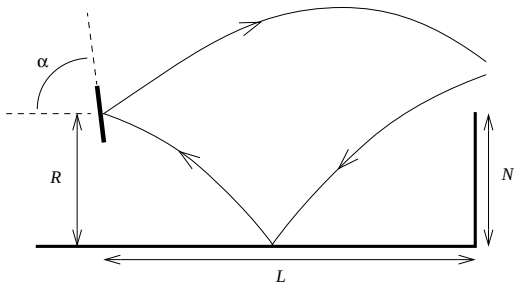

Fig. 1. A single shot in a tennis rally.

We consider a single shot in the rally. As illustrated in Fig. 1, the ball appears from the right, with velocity $(vx_{\text{in}}, vy_{\text{in}})$ say (both vx_{in} and vy_{in} being negative), bounces once, and then on its continuing path encounters the player's racquet at height R, having travelled a horizontal distance L. After striking the ball, the racquet gives it a velocity $(vx_{\text{out}}, vy_{\text{out}})$. We can model this scenario using continuous ASM rules as follows. The free flight of the ball is governed by a pliant rule:

FLIGHT $\stackrel{c}{=}$
if $py > 0$ **then with** $py \geq 0$
do $px(t), py(t), vx(t), vy(t) :=$ **solve** $[\mathcal{D}px, \mathcal{D}py, \mathcal{D}vx, \mathcal{D}vy] = [vx, vy, 0, -g]$

In FLIGHT we see the usual equations of Newtonian motion for a point mass in first order row-vector form. We use the symbol $\mathcal{D}$ to denote the time derivative in "program-like" situations; g is the acceleration due to gravity. We check, and continually enforce, the constraint that py is non-negative — the ball is not allowed to penetrate the surface of the tennis court. This one rule is enough for all three free-flight episodes of our scenario.

The interactions of the ball with the ground and with the racquet require some mode rules. The simplest is the bounce off the tennis court surface. The following rule will do.

BOUNCE $=$ **if** $py = 0 \wedge vy < 0$ **then** **do** $vy := -c\,vy$

Rule BOUNCE assumes that the motion of the pointlike tennis ball in the horizontal direction is unaffected by the bounce, but that the vertical component is reflected, and scaled down by the coefficient of restitution c (where we have $0 < c < 1$).

While the modelling of the bounce can be said to be reasonably realistic, we simplify the interaction with the racquet fairly dramatically, by assuming that the racquet has infinite mass and is infinitely stiff. In this case, the encounter between the ball and racquet can be modelled just like a bounce, i.e. the normal component of the relative velocity is reflected modulo the coefficient of restitution c, and the tangential component remains unaffected.

To model this properly, we need position and velocity variables for the racquet; let these be rpx, rpy, rvx, rvy respectively, and suppose that these variables refer to the precise point of impact on the racquet of the ball. Suppose that at the moment of impact, the racquet is inclined at an angle α to the horizontal, as in Fig. 1. Then a mode rule that will fulfill our requirements is the following.

RACQUET =
if $py > 0 \wedge py = rpy \wedge px < 0 \wedge px = rpx \wedge vx < 0 \wedge vx.rvx + vy.rvy < 0$ **then**
do
$vx := -(vx - rvx)(\cos^2(\alpha) + c\sin^2(\alpha)) + (vy - rvy)(1-c)\cos(\alpha)\sin(\alpha) + rvx,$
$vy := (vx - rvx)(1-c)\cos(\alpha)\sin(\alpha) - (vy - rvy)(\sin^2(\alpha) + c\cos^2(\alpha)) + rvy$

The guard of RACQUET checks that the ball is above the ground and to the left of the net, and that the ball and racquet are in the same place. The final conjunct of the guard is the inner product of the racquet and ball velocities. Insisting that it is negative ensures that the racquet strikes the ball in such a way that there is a component of the resulting velocity that is opposed to the ball's previous motion — which, if the ball is travelling as we would expect, towards the left, ensures that the ball will travel towards the net after the impact. Beyond that, explaining the assignments to vx and vy in the rule takes us deeper into classical mechanics than is appropriate here, so the details are relegated to an appendix.

Our model is completed with an INIT rule to assign appropriate initial values to all the variable. We do not write it down.

4 Formal Semantics

The account of the Continuous ASM in Section 2 was intended to give a picture of our formalism that is conceptually easy to grasp and is clear enough for model building, relying to some extent on the reader's intuition and experience to fill in any gaps (e.g. positing *ab initio* the sequence t_i of times at which discontinuities take place). In this section we give a summary of the formal operational semantics of our formalism. In order to not waste large amounts of space on repeating routine material, we rely heavily on existing work: on [13] (especially Chapter 2.4) for conventional ASM semantics; and on [49] (especially Chapter III §10) for differential equations in the sense of Carathéodory. Given these trusted foundations for discrete and continuous update respectively, the issues we must be most careful about are the handovers between mode and pliant transitions. We discuss these further after presenting the semantics.

One thing that we have not explicitly mentioned hitherto, is that we have been assuming that the system being discussed is defined monolithically, i.e. as a single indivisible syntactic unit. This is in accord with the automata-centric view taken in the majority of work on hybrid systems in the literature (see Section 8). However, in rule based formalisms (such as ASM), it is quite common to compose systems out of smaller subsystems — in the ASM case, the simultaneous execution of *all* enabled rules at each step provides a very simple semantics for composing subsystems that just aggregates the subsystems' rulesets.[4]

In this regard, the semantics we sketched in Section 2.2 is inadequate. For one thing, we spoke (almost exclusively) of transitions, and did not explore in detail how they might be related to ASM rules, except that intuitively it is clear that

[4] Dually, one can approach the same issue by decomposing simpler abstract systems into collections of smaller, more detailed subsystems, as happens in Event-B for instance.

rules should *specify* transitions. This also sidesteps the scheduling convention just mentioned. For another thing, we did not consider whether insisting that the system as a whole engaged in the alternation of mode and pliant transitions as we described them, made sense when the system is not monolithic.

The latter point raises an issue not present in the usual discrete world. In the discrete world, when an update is made to some system variables, any variables not mentioned in the syntactic description of the update, conventionally remain at their existing value. This coincides with the natural real time behaviour of variables that have piecewise constant values over time. So there is no observable distinction between leaving such a variable unaltered (to pursue its natural temporal evolution) on the one hand, and updating it to remain at the same constant value on the other hand. The former view is appropriate if the variable belongs to a different subsystem which is unaware of the ongoing update, while the latter view is appropriate if the variable belongs to the system being currently updated, but no change in its value is required.

In the continuous world, in which the values held in system variables may vary in a non-piecewise constant manner over time, the distinction between these two views can become apparent. If a variable that belongs to the subsystem currently being updated (via a pliant transition that is about to start) is not mentioned in the syntactic description of the update, then the policy that its value remains constant throughout the ensuing interval of time during which the new pliant transition will act, represents a specific design decision about the semantics of the current subsystem.

While it might be possible to justify such a design decision on requirements grounds when the variable belongs to the system being updated, the same design decision can seem very unnatural when the variable in question belongs to a *different* subsystem, in which its behaviour is being governed by a pliant transition that started in that subsystem earlier, and which demands some non-constant behaviour for the variable. Then, the idea that that behaviour is suddenly overridden by a constant behaviour that "appears out of nowhere" (from the point of view of that other subsystem) is very counterintuitive. So it is highly preferable that such variables be allowed to continue with their pre-existing behaviour.

Taking this latter view complicates the semantic picture a little. On the face of it, the definition of the sequence of times t_i at which discontinuities take place becomes more problematic — the sequence that is "naturally" seen by one subsystem need not coincide with the sequence that is "naturally" seen by another subsystem. Additionally, specifying the moments at which mode transitions arise, and their scope, as well as determining the scope of pliant transitions, requires more care. Deciding what "subsystem" refers to, and how to handle it in the context of a rule system based formulation, also requires care.

Our semantics takes these considerations into account. It defines the behaviours of a set of rules $\mathcal{R}$, much as one would do for a monolithic system. However, we allow for the fact that $\mathcal{R}$ may itself be made of the union of one or more constituent sets of rules. We do this by: (i) allowing for several INITial rules (which must, of course, be consistent, originating from different

Table 1. Notations utilised in the semantics

Notation	Explanation
$\mathcal{T}$	time interval, duration of the dynamics
U^{var}	type for variable var
$\mathcal{R}$	set of rules
$\mathcal{S}$	semantics of $\mathcal{R}$, a set of system traces
ζ_{var}	system trace of var , $\zeta_{var} : \mathcal{T} \to U^{var}$
$PlRl(pli, t)$	pliant rule for variable pli that ζ_{pli} obeys at time t
$InitUDS$	set of consistent update sets for the initial rules of $\mathcal{R}$
$PliRsEN$	set of enabled pliant rules of $\mathcal{R}$ (at any execution of step [**5**] of the semantics)
$PliRsCT$	set of pliant rules that are to continue preceding execution (at any execution of step [**6**] of the semantics)
$PliREM$	set of remaining pliant rules (not in $PliRsEN \cup PliRsCT$ at any execution of step [**7**] of the semantics)
$MoRs$	set of non-INIT mode rules enabled at a preemption point (at any execution of step [**12.2**] of the semantics)
$MoRsUDS$	set of consistent update sets for the rules in $MoRs$ (at any execution of step [**12.3**] of the semantics)

constituent rule subsets), (ii) having a preemption mechanism that allows pliant rules to continue past a preemption point (when this is appropriate) as well to be preempted (when that is appropriate), using rule-variable dependencies to determine which course of action to apply after any mode transition. This gives a simple syntax-independent semantics for composition. With these thoughts in mind, the semantics is given in the following sections.

4.1 Semantic Context

We start with a number of contextual observations and definitions. Table 1 summarises the specialised notations introduced during the course of the technical details.

[A] 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 universe of values (or type) U^{var}. If var is pliant, then U^{var} is $\mathbb{R}$. (N. B. Earlier we were more lax concerning the types of pliant variables. Now we will be more specific, recognising that, in practice, more complex types that are of interest can be constructed from $\mathbb{R}$ anyway.)

[B] The semantics is given for $\mathcal{R}$ which is a set of rules. $\mathcal{R}$ contains one or more distinguished INITial rules. Each INIT rule has a guard which is either "true" or "$t = t_0$".

[C] Time is a distinguished variable (read-only, never assigned by rules). All other variables have interpretations which are functions of an interval of time

starting at t_0. (See **[E]**.) As well as directly referring to the time variable, time may be handled indirectly by using clock variables. Their values may be assigned by mode rules, and their rates of change with respect to time may (during use in pliant rules) be specified directly, or defaulted to unity.

[D] $\mathcal{R}$ consists of mode rules and pliant rules. A mode rule (e.g. (4)), is *enabled* iff, under the current valuation of the system variables, the value of the *guard* of (4) lies in the topological closure of the true-set of the *guard*. A pliant rule (e.g. (5)), is *enabled* iff $IV \wedge guard$ evaluates to true under the current valuation of the system variables. A variable is *governed* by a mode rule iff it is assigned by that rule. A pliant variable is *governed* by a pliant rule iff it appears in the left hand side of the DE of the rule, or is directly assigned in the rule, or is constrained in the **with** clause.

[E] The semantics of $\mathcal{R}$ 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 variable var. If $\mathcal{S}$ is empty we say that the semantics of $\mathcal{R}$ is VOID.

[F] In order that the evolution of each pliant variables is suitably managed, an additional data structure is needed. For each pliant variable pli, the function $PlRl(pli, t)$ returns the pliant rule that the interpretation ζ_{pli} of variable pli is obeying at time t.

[G] The set of traces $\mathcal{S}$ is constructed by the step by step process below, which describes how individual system traces are incrementally constructed.[5] 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. Whenever an ABORT is encountered, the current trace-so-far is abandoned, and eliminated from the semantics $\mathcal{S}$, of $\mathcal{R}$.

4.2 Operational Semantics

In the context of the assumptions **[A]**-**[G]** above, the operational semantics of the Continuous ASM can be given as follows.

[1] Let $i := 0$ (where i is a meta-level variable).
[2] Let $InitUDS$ be the set of consistent update sets for the collection of initial rules of $\mathcal{R}$. If $InitUDS$ is empty then VOID. Otherwise, CHOOSE an update set from $InitUDS$ and assign all variables accordingly, thereby interpreting their values at time t_0. (N. B. We assume that all system variables acquire an initial value in this manner.)
[3] If any non-INIT mode rule is enabled when the variables have the values at t_i then ABORT.
[4] If no pliant rule from $\mathcal{R}$ is enabled then ABORT.

[5] N. B. The process is not intended to be executable. All traces-so-far are intended to be explored simultaneously.

[5] Let $PliRsEN$ be the set of enabled pliant rules from $\mathcal{R}$.

[5.1] If any pliant variable occurs in the left hand side of the DE (or direct assignment) of more than one rule in $PliRsEN$ (or more than once in the left hand side of the DE in the same rule), then ABORT.

[6] If $i = 0$ let $PliRsCT = \varnothing$. Otherwise, let $PliRsCT$ be the set of pliant rules from $\mathcal{R}$, such that: $PliRsCT$ is maximal; no rule in $PliRsCT$ is in $PliRsEN$; no variable governed by any rule in $PliRsCT$ is governed by any rule in $PliRsEN$; for every rule PLIRLCT in $PliRsCT$, for every pliant variable pli governed by PLIRLCT, $\overrightarrow{PlRl(pli, t_i)} =$ PLIRLCT; for every rule PLIRLCT in $PliRsCT$, for every mode variable v which occurs in the *guard* of PLIRLCT, $\overrightarrow{v(t_i)} = v(t_i)$.

[7] Let $PliREM$ consist of any pliant variables pli that are not governed by any rule in either $PliRsEN$ or $PliRsCT$. If $PliREM$ is nonempty, then ABORT.

[8] If there does not exist a $t_{\text{NEW}} > t_i$ such that there is a simultaneous solution of all the DEs and direct assignments in the rules in $PliRsEN \cup PliRsCT$ in the left-closed, right-open interval $[t_i \ldots t_{\text{NEW}})$, using as initial values the variable values and right limits of inputs and outputs at t_i, and such that the *rel* predicates also evaluate to true in the interval $[t_i \ldots t_{\text{NEW}})$, then ABORT.

[9] Otherwise, CHOOSE a simultaneous solution as in **[8]**, and let t_{MAX} be maximal such that $t_{\text{MAX}} > t_i$ and this solution is defined in the interval $[t_i \ldots t_{\text{MAX}})$.

[9.1] For all pliant variables pli, for all $t \in [t_i \ldots t_{\text{MAX}})$, let $PlRl(pli, t)$ be the rule governing the behaviour of pli. (N. B. This assignment is total by **[7]** and unambiguous by **[5.1]**.)

[9.2] For every mode variable, extend its value at t_i to a constant function in the interval $[t_i \ldots t_{\text{MAX}})$.

[10] If no non-INIT mode rule is enabled at any time t_{NEXT} in the open interval $(t_i \ldots t_{\text{MAX}})$, or no non-$Init$ mode rule is enabled by the left-limit values of the state variables at time t_{MAX} in the case that these left-limit values exist and are finite at t_{MAX}, then TERMINATE.

[11] Let $i := i + 1$.

[12] Let t_i be the smallest time t_{NEXT} at which some non-INIT mode rule is enabled in **[10]**.

[12.1] Discard the interpretation of all variables, and the definition of $PlRl$, in the interval $[t_i \ldots t_{\text{MAX}})$.

[12.2] Let $MoRs$ be the set of non-INIT mode rules that are enabled when all variables *var* are interpreted as the left-limit values at t_i, i.e. as $\overrightarrow{var(t_i)}$.

[12.3] Let $MoRsUDS$ be the set of consistent update sets for the rules in $MoRs$. If $MoRsUDS$ is empty then ABORT. Otherwise, CHOOSE an update set from $MoRsUDS$ and assign all the updated variables accordingly, thereby interpreting their values at time t_i.

[12.4] For all other variables *var*, interpret their values at time t_i to be their left-limit values at t_i, i.e. to be $\overrightarrow{var(t_i)}$.

[13] Goto **[3]**.

4.3 Mode-Pliant and Pliant-Mode Handovers

Before commenting further, we make some observations on the consistency of the above definition. As noted earlier, we can take certain things for granted, such as well definedness of mode transitions via ASM update semantics, and the existence of solutions to differential equations. The key remaining points then, are whether the handovers from pliant to mode transitions, and those from mode to pliant transitions, are well defined.

We observe that the handover from pliant to mode transitions is trouble-free as follows. Since the set of values at which any mode rule becomes enabled is closed (being given by the closure of the true-set of the *guard* of the rule, by [**D**]), and since the system trajectory is a continuous function during any interval in which a pliant rule is active, if the system trajectory meets the closure at all during such an interval, it first meets it at some specific time point. Since there are only finitely many rules, the minimum of these points is a unique well defined time point, and so t_i in [**12**] emerges as this minimum. Thus the earliest moment that a mode transition becomes enabled during a pliant transition is a well defined time point, and the time at which the pliant transition is preempted is well defined, from which a consistent set of mode updates is derived, by [**12**], [**12.1**], [**12.2**], [**12.3**].

We argue that the handover from mode to pliant transitions is also consistent. Firstly, upon completion of a mode transition, some pliant rules will (typically) be enabled, [**5**]; these are required to be unambiguous and consistent by [**5.1**]. Secondly, these rules need not govern *all* the pliant variables of the whole system. By [**6**], if there were pliant rules contributing to the pliant transition that was just preempted, which govern variables disjoint from those governed by the first case, they are permitted to continue — we might term this figurative interruption and resumption a "virtual skip". Thirdly, the former two measures may still not take care of *all* pliant variables, since there is no requirement for pliant rules and the sets of variables that they govern to dovetail neatly together. If there are any pliant variables left over, [**7**] ensures that the run is ABORTed.

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.

4.4 Multiple Subsystems

We return to the questions that were raised earlier concerning the definition of, and interaction between, subsystems that coexist within a single encompassing system. We examine how the formal semantics above helps to address these, and we tie up the semantic loose ends.

To start with, we would normally expect that a separate subsystem would control (i.e. have write access to) an exclusive set of variables. We therefore take that as a fundamental principle.[6]

[6] We can imagine that write access to some variable might, exceptionally, be shared by more than one subsystem, but under such circumstances a suitable protocol will

The next basic insight comes from [13], which promotes a perspective in which a system's variables are either *monitored* or *controlled*. Controlled variables are written to by the system, whereas monitored variables are merely read. For the latter, it is assumed that the environment supplies the values that are read, but aside from the condition that the values of monitored variables should be stable when read, no further restriction is placed on them. Thus, there is nothing to prevent their values from being supplied by another ASM system, the original system and its environment thus becoming two subsystems of a larger system. In other words, the conventional definition of an ASM system is intended to enable it to play the role of subsystem, essentially without modification.

For our purposes, we add a couple of observations to the above picture to make it suit the Continuous ASM situation. Firstly, since pliant variables' values will change continuously in general, we can modify "stable when read" to "reliably readable when needed", to avoid any possible confusion. Secondly, we emphasise that in the context of a system comprising several subsystems (i,e, one constructed via the composition of the subsystems' rulesets), each writable variable is written to by the rules belonging to exactly one of the subsystems, and no rule (of the whole system) writes to the writable variables of more than one of the subsystems. With these simple structural restrictions in place, the semantics of a system consisting of the composition of multiple subsystems is simply given by aggregating all of the rules of all of the subsystems in the usual way, and processing them according to the single system semantics given above.

Of course, a non-VOID semantics for subsystem $A1$ which assigns a variable $x1$ while reading variable $x2$ which belongs to subsystem $A2$, and a non-VOID semantics for subsystem $A2$ which assigns a variable $x2$ while reading variable $x1$, does not guarantee a non-VOID semantics for the entire system consisting of $A1$ together with $A2$, since there may not be values for $x1$ and $x2$ that *simultaneously* satisfy all the constraints imposed by the two subsystems.

The second proviso above also acts in concert with the stipulations of the formal semantics to ensure that, in a multi-subsystem system, each preemption point is caused by an identifiable subset of the subsystems,[7] and upon completion of the preemption, an identifiable subset of the subsystems embarks on new pliant behaviour, with the remainder resuming the pliant behaviour they were executing previously. (N.B. The two subsets need not be the same.)

The last point brings us to the issue of the how the indexing of mode and pliant transitions works in a system conceptually divided into separate subsystems. We see that the semantics defines a global indexing, which is a strict sequentialisation of the mode transitions of the entire system, regardless of which subsystem they might arise from. From the vantage point of any given subsystem, only a subset of these mode transitions might be "visible", but this amounts to simply re-indexing the mode transitions if we want to describe the system dynamics

---

have be in place to prevent race conditions, such as in the case of familiar mutual exclusion protocols [40, 31]. We do not consider such cases here.

[7] For simplicity, we permit simultaneous preemption by more than one subsystem, even if it would be a little impractical in reality.

from that subsystem's viewpoint. Allied to this is the fact that if a remote subsystem undergoes a mode transition of which a given subsystem is unaware, some values being read by the local subsystem might still undergo discontinuous change in the midst of a pliant transition (of the local subsystem). This discontinuity causes no discomfort, since we understand differential equations in the sense of Carathéodory. Provided that the right hand side of each ODE in the system has the uniformly bounded Lipschitz property in the system variables, and remains measurable over time, it is guaranteed that a solution exists and is absolutely continuous.

Lastly, a note on Zeno behaviour. Nothing in the semantics that we have discussed precludes it. Therefore the semantic model does not of itself guarantee the recommendation **I** of Section 2.2. As we remarked, Zeno-freeness normally depends on global reachability, so our view is that if a system model is capable of exhibiting Zeno behaviour, then there is potentially something wrong with the model, and, depending on circumstances, the model ought to be improved to remove or mitigate those aspects that lead to it.[8]

5 The Tennis Ball Revisited

What is interesting about the tennis ball example is to consider how the formal semantics of Section 4 views the behaviour of the tennis ball system. We start by noting that the guards of the various mode rules in the tennis ball system all featured strict inequalities. However, in order that in any given run, the times at which mode events occur are well defined, the runtime interpretation of mode events' guards is via the closure regions of their true-sets. In other words, the strict inequalities of guards are reinterpreted non-strictly. This gives rise to some interesting effects, which we comment on now.

One interesting effect concerns the constraint $py > 0$ in the guard of Racquet. If this is replaced by $py \geq 0$, then the scenario is possible in which $py = 0$ becomes true at the precise time that the ball strikes the racquet. In this case both Bounce and Racquet are enabled. If Bounce runs first, then Racquet will be enabled immediately afterwards, and the run will be aborted by point [**3**] of the formal semantics. If Racquet runs first, then Bounce will be enabled immediately afterwards, and the run will also be aborted. These aborts are typical of the "cleaning up" that the semantics performs when the rules do not neatly conform to the requirements of the strucure of runs that we have demanded. This also supports the view that one should design *and reason about* systems using such guards etc. as most eloquently address the needed system requirements. Regarding behaviours at awkward boundary cases which arise because of the semantics, even if the semantics does not abort, behaviours may be forced that could justifiably be regarded as anomalous.

As an example of this consider the flight of the ball after the racquet strike, as it approaches the net. If the path of the ball is low, it will hit the net after

[8] This could depend on requirements. Zeno behaviour may sometimes be tolerable and sometimes not.

having travelled a horizontal distance L from the point of impact with the racquet. Assuming the racquet strikes the ball so that it has an upward velocity component, let us consider increasing the velocity with which the racquet strikes the ball. As this increases, the ball will typically hit the net at points higher and higher up. Eventually, the top of the net will be reached. If the net is modelled as a vertical line, closed at the top (i.e. using a definition such as $0 \leq net\_y \leq N$), then the family of ball trajectories including these net impacts, generated by the above rules will behave smoothly as the limit of the top of the net is reached.

By contrast, keeping the same net definition, consider the case when the ball has a lot of energy on its return towards the net. Then it will fly over the net towards the right. Now consider reducing the ball's energy gradually. At the limit point, i.e. when the flight of the ball touches the net at height N, there will be a discontinuity in the family of ball trajectories. At the limit point, instead of the ball flying over the net, it will hit the net and drop leftwards. Assuming that the net is modelled as we said, and that we have a mode rule to model the impact with the net, this anomalous limiting behaviour of the family of trajectories in which the ball flies over the net will be generated by the semantics.

By contrast, if we model the net as $0 \leq net\_y < N$, then the anomaly would be generated the other way round, as the point of impact with the net rose.

Is the existence of either anomalous limit harmful? We argue that it is not. The anomaly exists at a single set of values for the system parameters. Viewed from the perspective of the system as a whole, this is a set of measure zero. In engineering terms, it is something which cannot be observed since the slightest departure from the specific parameter values causes a change in behaviour — only behaviours that are modelled by systems in which the behaviours are robust over parameter sets of non-zero measure can play a role in real life, so the existence of relatively isolated anomalous behaviours does no harm. These relatively isolated anomalous behaviours are the price that one sometimes has to pay for being able to model at an idealised level. And although such idealised models are clearly unrealistic to a degree, the clarity they can bring to high level system conceptualisation makes the price one that is worth paying.

A further set of interesting behaviours arises if we allow the coefficient of restitution parameter c, to vary. Under normal circumstances, one would expect a single bounce of the ball before the racquet returns the ball over the net, this being what is allowed by the rules of tennis. If, however, we consider a succession of further bounces, more interesting things can occur.

We assume that the racquet stays at a horizontal distance L away from the net. Then the number of bounces that the ball can experience depends on the value of c. Provided there is at least one bounce, then by adjusting the value of c we can increase the number of subsequent bounces arbitrarily. To see this we observe that the (vertical part of the) kinetic energy reduces by a factor of c^2 on every bounce. So the height of the parabolic flight segment after every bounce also reduces by a factor of c^2, and so does its width, which corresponds to the horizontal distance travelled during that parabolic flight segment. So, aside from the initial part in which the ball comes over the net, the total horizontal distance

travelled is proportional to $\sum_{k=1}^{\infty} c^{2k} = c^2(1-c^2)^{-1}$. As c reduces to zero, this approaches c^2, which also approaches zero.

So the total additional horizontal distance travelled after the first bounce can get arbitrarily small, and thus the ball may never reach the racquet while still in the air. What we see here is an example of a Zeno effect. To absorb all the initial vertical kinetic energy takes an infinite number of bounces.

What happens afterwards? Arguing physically (though still in a highly idealised way), since the vertical and horizontal elements of the kinetic energy are decoupled, after the vertical kinetic energy has been absorbed by the bounces, the horizontal kinetic energy remains, so the ball rolls along the ground with velocity vx_{in}. Arguing according to the semantics of Section 4, if c is small enough for this behaviour to ensue, then the single system run allowed by a fixed set of system parameters never gets past the limiting Zeno point of the behaviour described. This is because the semantic construction in Section 4 only allows for a number of steps that is indexable by the naturals. Going beyond the Zeno point would require a transfinite construction, which we have not explored in this paper. Such constructions, while possible, would always be unphysical to a greater or lesser extent, so are of limited value for application modelling.

What we have just been discussing, illustrates in a very clear way the difficulties inherent in the Zeno recommendation of Section 2. For one set of parameters, the Zeno effect is absent, and the model behaves in an exemplary way. For another set of parameters, looking not much different from the first, the behaviour is completely different, exhibiting Zeno effects. (And, of course, there is the boundary scenario, in which the limit of the Zeno behaviour reaches exactly the distance L, which we didn't explore.)

5.1 The Tennis Ball as a Multiple System

Let us contemplate our tennis ball example from the vantage point of multiple subsystems. Taking the rules we wrote at face value, there is no sensible possibility of partitioning the system, since its simplicity dictates that all the rules modify all all variables, more or less.

However, we can consider enlarging the system, for example by adding a television camera that follows the flight of the ball.[9] This would have read access to the ball's dynamical variables, but not write access. In this case, the TV camera's variables and rules would reside in a separate subsystem from those of the tennis ball itself. Depending on the detail of its model, the ball's dynamical variables would, via read access, determine which pixels of the camera's CCD sensor changed in response to the ball's flight, etc.

An alternative approach might note that we have focused on building a system for the left hand side of the tennis court. We could thus contemplate building a complementary system to cover the right hand side of the court, subsequently composing the two subsystems to get a system covering the entire court.

[9] A sports programme that genuinely did this would make viewers dizzy, but we can tolerate the idea of it for the sake of the example.

However, there are problems with this approach, should we attempt to construct the system described. The ball would alternately be found, first in one half of the court, and next in the other. This means that the ball's behaviour would not be the responsibility of a single subsystem. This flies in the face of the restriction made at the beginning of Section 4.4, to ensure each variable is updated by a single subsystem.

The motivations for imposing such a restriction differ between mode and pliant variables. For mode variables, the fact that a mode variable can retain its value indefinitely, without special supervision, until an update changes the value, reduces the problem of its semantics' consistency to well understood questions of mutual exclusion, so that a single agent has authority to update the variable at any moment. The restriction of updates to a variable to a single subsystem is therefore just the simplest incarnation of this policy.

For pliant variables, the situation is different. A pliant variable's semantics demands that its value needs to be supervised at all times, by a differential equation for example. This corresponds to the physical reality that the laws of nature hold at all times, and therefore that any description of a physical process must adhere to the same principle. If responsibility for ensuring this is divided among a number of subsystems, the challenge of verifying that it is met becomes the harder. In particular, the description must be continuous and unbroken over time. This is easiest to ensure if all updates to the pliant variable are contained in one subsystem.

In [7] there is a much more extensive discussion of the implications of physical law for language systems intended for the definition and description of cyber-physical systems.

6 Continuous ASM Refinement

Now we develop our Continuous ASM framework to encompass refinement of Continuous ASM models. We start by describing the usual ASM refinement formulation, appropriate to pure mode transitions, and then show how to extend this to encompass the new kinds of transition.

6.1 The Discrete Case

In general, to prove a conventional ASM refinement, we verify so-called (m, n) diagrams, in which m abstract steps simulate n concrete ones in an appropriate way. This means that there is nothing that the n concrete steps can do that is not suitably reflected in m appropriately chosen abstract steps, where both m and n can be freely chosen to suit the application. It will be sufficient to focus on the refinement proof obligations (POs) which are the embodiment of this policy. The situation for refinement is illustrated in Fig. 2, in which we suppress input and output for clarity.

In Fig. 2 the refinement relation $R_{\mathrm{A,C}}$ (also often referred to as the gluing relation) between abstract and concrete states, holds at the beginning and end

of the (m, n) pair. This permits us to abut such (m, n) diagrams, by identifying the last (abstract and concrete) states of one (m, n) diagram, with the first (abstract and concrete respectively) states of the next, and thereby to create relationships between abstract and concrete runs in which $R_{\mathrm{A,C}}$ is periodically re-established. (N. B. In much of the ASM literature, the main focus is on an *equivalence*, usually written $\equiv$, between abstract and concrete states. This is normally deemed to contain a "practically useful" subrelation $R_{\mathrm{A,C}}$, chosen to be easier to work with. The approach via $R_{\mathrm{A,C}}$ will be the focus of our treatment, and is also focus of the KIV [30] formalization in [43, 44].)

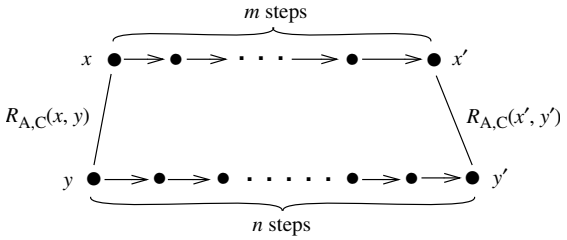

Fig. 2. An ASM (m, n) diagram, showing how m abstract steps, going from state x to state x' simulate n concrete steps, going from y to y'. The simulation is embodied in the refinement relation $R_{\mathrm{A,C}}$, which holds for the before-states of the series of steps $R_{\mathrm{A,C}}(x, y)$, and is re-established for the after-states of the series $R_{\mathrm{A,C}}(x', y')$.

The first PO is the initialization PO:

$$\forall\, y' \bullet CInit(y') \Rightarrow (\exists\, x' \bullet AInit(x') \wedge R_{\mathrm{A,C}}(x', y')) \tag{6}$$

In (6), it is demanded that for each concrete initial state y', there is an abstract initial state x' such that $R_{\mathrm{A,C}}(x', y')$ holds.

The second PO is correctness. The PO is concerned with the verification of (m, n) diagrams. For this, we have to have some way of deciding which (m, n) diagrams are sufficient for the application. In practice, this is part of the design process, so let us assume that this has been done. Let $CFrags$ be the set of fragments of concrete runs that we have previously determined will permit a covering of all the concrete runs of interest for the application. Using :: to denote concatenation, we write $y :: ys :: y' \in CFrags$ to denote an element of $CFrags$ starting with concrete state y, ending with concrete state y', and with intervening concrete state sequence ys. Likewise we write $x :: xs :: x' \in AFrags$ for abstract fragments. Let is, js, os, ps denote the sequences of abstract inputs, concrete inputs, abstract outputs, concrete outputs, respectively, belonging to $x::xs::x'$ and $y::ys::y'$ and let $In_{\mathrm{AOPS,COPS}}(is, js)$ and $Out_{\mathrm{AOPS,COPS}}(os, ps)$ denote suitable input and output relations. Then the correctness PO reads:

$$\begin{aligned}
&\forall\, x, is, y, ys, y', js, ps \bullet y :: ys :: y' \in CFrags \wedge \\
&R_{\mathrm{A,C}}(x, y) \wedge In_{\mathrm{AOPS,COPS}}(is, js) \wedge \mathrm{COPS}(y :: ys :: y', js, ps) \Rightarrow \\
&\quad (\exists\, xs, x', os \bullet x :: xs :: x' \in AFrags \wedge \\
&\qquad \mathrm{AOPS}(x :: xs :: x', is, os) \wedge R_{\mathrm{A,C}}(x', y') \wedge Out_{\mathrm{AOPS,COPS}}(os, ps))
\end{aligned} \tag{7}$$

In (7), it is demanded that whenever there is a concrete run fragment of the form $\text{COPS}(y :: ys :: y', js, ps)$, carried out by a sequence of concrete operations[10] COPS, with state sequence $y :: ys :: y'$, input sequence js and output sequence ps, such that the refinement and input relations $R_{\text{A,C}}(x, y) \wedge In_{\text{AOPS,COPS}}(is, js)$ hold between the concrete and abstract before-states and inputs, then an abstract run fragment $\text{AOPS}(x :: xs :: x', is, os)$ can be found to re-establish the refinement and output relations $R_{\text{A,C}}(x', y') \wedge Out_{\text{AOPS,COPS}}(os, ps)$.

The ASM refinement policy also demands that non-termination be preserved from concrete to abstract. We retain this in our extension of the formalism for when it is needed.

Assuming that (6) holds, and that we can prove enough instances of (7) to cater for the application of interest, then the concrete model is a **correct refinement** of the abstract model. In a correct refinement, all the properties of the concrete model (that are visible through the refinement and other relations), are suitably reflected in properties of the abstract model (because of the direction of the implication in (7)). If in addition, the abstract model is also a correct refinement of the concrete model (using the converses of the same relations), then the concrete model is a **complete refinement** of the abstract model. In a complete refinement, all relevant properties of the abstract model are also present in the concrete model (because of the direction of the implication in the modified version of (7)). Therefore, to ensure that the complete set of requirements of an intended system is faithfully preserved through a series of refinement steps, it is enough to express them all in a single abstract model, and then to ensure that each refinement step is a complete refinement.

6.2 The Continuous Case

The preceding was formulated for the discrete world. However, to extend it to the continuous world is not very hard. The essence of the approach is to reinterpret the run fragments $\text{AOPS}(x :: xs :: x', is, os)$ and $\text{COPS}(y :: ys :: y', js, ps)$ appearing in (7) in a way that yields a natural extension of the discrete case.

In the discrete context, such a notation refers to a sequence of states, i.e. a map from some natural number indexes to state values. In a context including real time, the analogue of this is a function from an interval of time to state values, which is piecewise constant. More precisely, the interval of time in question will be a finite closed interval $[t_A \ldots t_B]$, where $t_A < t_B$. Such an interval corresponds to a typical left-closed right-open interval $[t_A \ldots t_B)$ on which the function is piecewise constant *plus* the right endpoint t_B. The interval $[t_A \ldots t_B)$ itself is partitioned into a finite sequence of left-closed right-open subintervals, on each piece of which the function is constant (as seen in the semantics of mode variables in Section 4).

The purpose of the right endpoint t_B, is to record the after-state of the last mode transition, so that it can be identified with the initial state of a successor

[10] We define an operation as a maximal enabled set of rules — provided its updates are consistent. Enabled inconsistent updates cause abortion of the run, as usual in ASM.

function, when (m, n) diagrams are abutted. Referring to Fig. 2, we can view the rightward pointing arrows (both abstract and concrete) as the constant functions on non-empty left-closed right-open subintervals (with the blobs at their tails representing the leftmost values), and the final blob (both abstract and concrete) representing the isolated value at the right closure of the entire interval.

We allow an exception to this convention when $t_B = \infty$. In that case the last subinterval of $[t_A \ldots t_B)$ is of infinite length, corresponding to a nonterminating final transition, and there is no isolated right endpoint, and no abutting of an (m, n) diagram featuring this kind of final subinterval to any successor.

The obvious generalisation of this for the framework of Continuous ASM is to use piecewise absolutely continuous functions from intervals of time to state values. These would be defined on finite closed intervals $[t_A \ldots t_B]$, with $t_A < t_B$. As above, such an interval would partition into one or more left-closed right-open subintervals on each of which the state function is absolutely continuous and without internal discontinuities, *plus* an isolated state at the right endpoint t_B, included to allow identification with the initial state of a successor function.

In this context, $\text{AOPS}(x :: xs :: x', is, os)$ at the abstract level and $\text{COPS}(y :: ys :: y', js, ps)$ at the concrete level, each consist of an alternating sequence of pliant and mode transitions (starting with pliant and ending with mode — unless the $t_B = \infty$ exception applies, and the last transition is pliant too).

Similar principles apply to inputs and outputs. Mode inputs and outputs are mapped to the time instant at which the after-state is established, while pliant inputs and outputs are mapped to the (left- and right-) open interval during which the pliant transition runs. We thereby derive an interpretation for the notation used in (7) appropriate for the current context. Thus $R(x, y)$ becomes a predicate about the earliest abstract and concrete state values referred to by the state functions mentioned, while $R(x', y')$ refers to the latest state values.

In this way, (7) continues to define refinement in the Continuous ASM context. The piecing together of (m, n) diagrams to build an abstract simulation of a concrete run, now reduces to the identification of the latest (abstract and concrete) state values reached by one (m, n) diagram, with the earliest (abstract and concrete) state values of its successor (m, n) diagram, in the way indicated above for the discrete case.

Given the above, it is instructive to point out what the PO (7) *does not* demand. We have already said that the ASM PO does not mention states that are internal to the length m and length n fragments that occur in a given (m, n) diagram. This frequently simplifies the relations R, In, Out etc., that capture the relationship between abstract and concrete worlds — the policy is particularly useful when it is easy to predict that the systems are *guaranteed* to schedule their steps in the particular way exploited in a given (m, n) diagram.

Finally, there is no explicit mention of time in (7). In particular there is nothing in (7) that indicates, in any relative way, how time is expected to progress during the respective length m and length n fragments — the abstract and concrete systems are free to progress according to their own notions of time.

Such aspects give the ASM POs great flexibility. Designers can define relationships between systems in the most practically useful way, a perspective the the ASM philosophy promotes. The appropriateness of the policy adopted for a given development becomes a matter for the wider requirements arena.

6.3 Continuous ASM Refinement and Multiple Subsystems

It is clear that refinement, as thus defined, suits a monolithic semantics — the correctness PO implicitly speaks of the state and I/O spaces in their entirety, and makes no concession to the subsystem issues debated in detail in Section 4.4. We comment on this now.

Suppose we have two subsystems $A1$ and $A2$. Taking $A1$ in isolation, its semantics is given in Section 4, on the understanding that any external values needed (e.g. from $A2$) appear as values of free variables of $A1$ that are "reliably readable when needed". On the other hand, viewing the system as a whole, forces the same Section 4 semantics to address the whole system, and to supply values of variables for both $A1$ and $A2$ simultaneously and consistently. Additionally, concerning the effect of a single rule, the ASM rule firing policy (namely that any enabled rule executes), also allows us to largely ignore whether the rule is being executed by a monolithic system, or by one of its subsystems.[11] We observed already though, that a non-VOID semantics for $A1$ and a non-VOID semantics $A2$, do not in themselves guarantee a non-VOID semantics for the combination of $A1$ and $A2$, since a globally consistent assignment of values to variables does not follow from individually consistent partial assignments.

The last observation makes clear that things get more complicated when refinement is considered. Suppose abstract subsystem $A1$, with variables $x1$, $is1$, $os1$, is refined by concrete subsystem $C1$, with variables $y1$, $js1$, $ps1$, using relations $R_{\text{A1,C1}}$, $In_{\text{A1OPS}_d,\text{C1OPS}_d}$ and $Out_{\text{A1OPS}_d,\text{C1OPS}_d}$, where d indexes over the (m,n) diagrams of the $A1$ to $C1$ refinement. Suppose abstract subsystem $A2$, with variables $x2$, $is2$, $os2$, is refined by concrete subsystem $C2$, with variables $y2$, $js2$, $ps2$, using relations $R_{\text{A2,C2}}$, $In_{\text{A2OPS}_d,\text{C2OPS}_d}$ and $Out_{\text{A2OPS}_e,\text{C2OPS}_e}$, where e indexes over the (m,n) diagrams of the $A2$ to $C2$ refinement.

Now, even if there is a non-VOID semantics for the combination of $A1$ and $A2$, there is no guarantee of a non-VOID semantics for the combination of $C1$ and $C2$. Furthermore, even if there is a non-VOID semantics for the combination of $C1$ and $C2$, there is no *a priori* guarantee that the non-VOID semantics for the $C1$ and $C2$ combination is a refinement of the non-VOID semantics of the combination of $A1$ and $A2$.

The root cause of these problems is the presence of the existential quantifiers in the conclusion of (7), since a conjunction of existential quantifications does not imply the existential quantification of the conjunction, as would be needed

[11] Contrast that with the case in which only one enabled rule is chosen to execute at a time. Then, whether a single rule executes, or a single rule *per subsystem* executes (and how this is reflected in observable effects), has a significant impact on the semantics and becomes very visible to the environment.

if refinement of the combined system were to follow from the refinements of the subsystems individually.

Moreover, even contemplating the $C1$ and $C2$ combination as a refinement of the combination of $A1$ and $A2$ raises difficulties, since the lexical scope of (7) reaches beyond just the state variables of the abstract and concrete systems, to include any read-only input variables and write-only output variables (via $In_{\text{A1Ops,C1Ops}}$ and $Out_{\text{A1Ops,C1Ops}}$ respectively for $A1$, for example). If some of the read-only input variables or write-only output variables of one subsystem are identified with state variables of the other, the form of (7) itself would have to be adapted to reflect this, depending on the context.

Further difficulties in eliciting a refinement of $A1$ and $A2$ to $C1$ and $C2$ from individual refinements of $A1$ to $C1$ and $A2$ to $C2$ come from the fact that these individual refinements need not use (m, n) diagrams that are necessarily congruent. Thus the 'shape' of the diagram used at a particular point of the $A1/C1$ execution (in terms of the number of steps and their durations at the two levels of abstraction) need not coincide with the 'shape' of the diagram used at the corresponding point of the $A2/C2$ execution. All this notwithstanding the fact that the two separate refinements do not have to agree about the way that time itself progresses in their constituent models.

In the face of all the difficulties pointed out, there are two approaches that make sense. The first approach is to leave the resolution of all the issues that come up, case by case, to individual application developments. Most often, an individual application will be characterised by features that reduce most of the points raised to trivialities, and by other features that indicate the way to resolve those that remain in ways that are relatively convincing and evident from the structure of the application.

The second approach is to simplify matters drastically, until a point is reached at which the difficulties pointed out are sufficiently reduced that a relatively tractable generic formulation results. We illustrate what can be done using a simple example of this approach.

To start with, we make a number of restrictions, and we argue for their sufficiency as a scheme for combining two subsystems (having a restricted structure) below.[12] Thus let $A1$ be refined to $C1$ and $A2$ be refined to $C2$, and let us assume the other notations introduced above. We will refer to the combination of $A1$ and $A2$ as A, with state variables x, and the combination of $C1$, and $C2$ as C, with state variables y. The sought for refinement from A to C will be described by a refinement relation $R_{\text{A,C}}$, which we define in terms of the *per subsystem* ones already introduced.

In less technical terms, what follows can be seen as the opening of the lexical scopes of the separate name spaces of the $A1$ and $A2$ systems, and the creation of the name space of A via their union. This allows name capture of identical identifiers. Similarly for $C1$ and $C2$, yielding C. To then get a valid A to C refinement requires a number of additional compatibility properties to hold, so that the desired refinement can be proved.

(1) Time is deemed to progress at the same rate in all models of the construction.
(2) For simplicity, we assume that none of $A1$, $A2$, $C1$, $C2$ have any I/O.
(3) The state variables $x1$ of $A1$ partition into $x1_1$, $x1_2$. The state variables $C1$ $y1$ partition into $y1_1$, $y1_2$. The state variables $x2$ of $A2$ partition into $x2_1$, $x2_2$. The state variables $C2$ $y2$ partition into $y2_1$, $y2_2$.
(4) The following pairs of variables are identical: $x1_2 \equiv x2_1$ ($\equiv x_{12}$); $y1_2 \equiv y2_1$ ($\equiv y_{12}$). There are no other variable clashes.
(5) The refinement relations of the $A1/C1$ and $A2/C2$ refinements decompose as follows, being nontrivial on only the variables mentioned. $R_{\mathrm{A1,C1}}(x1, y1) \equiv R^{1_1}_{\mathrm{A1,C1}}(x1_1, y1_1) \wedge R^{1_2}_{\mathrm{A1,C1}}(x1_2, y1_2)$; $R_{\mathrm{A2,C2}}(x2, y2) \equiv R^{2_1}_{\mathrm{A2,C2}}(x2_1, y2_1) \wedge R^{2_2}_{\mathrm{A2,C2}}(x2_2, y2_2)$. $R^{1_2}_{\mathrm{A1,C1}}(x1_2, y1_2) = R^{2_1}_{\mathrm{A2,C2}}(x2_1, y2_1) \equiv R^{12}_{\mathrm{A2,C2}}(x_{12}, y_{12})$. We define $R_{\mathrm{A,C}}(x, y) \equiv R^{1_1}_{\mathrm{A1,C1}}(x1_1, y1_1) \wedge R^{12}_{\mathrm{A2,C2}}(x_{12}, y_{12}) \wedge R^{2_2}_{\mathrm{A2,C2}}(x2_2, y2_2)$.
(6) There is a relation $\rho_{1,2}$ from the (m, n) diagrams of the $A1/C1$ refinement to the (m, n) diagrams of the $A2/C2$ refinement. It satisfies the following conditions. (i) For every (m, n) diagram of the $A1/C1$ refinement featuring a given behaviour of $x1_2$ in $A1$ and $y1_2$ in $C1$ (over the duration of the diagram), there is a (m, n) diagram of the $A2/C2$ refinement featuring an identical behaviour of $x2_1$ in $A2$ and $y2_1$ in $C2$ (over the identical duration), and the pair of (m, n) diagrams is in $\rho_{1,2}$. (ii) As for (i), but directed from the $A2/C2$ refinement to the $A1/C1$ refinement using the converse of $\rho_{1,2}$. (iii) $\rho_{1,2}$ is universal on all $A1/C1$ and $A2/C2$ (m, n) diagram pairs having a given common $x1_2/y1_2$ ($\equiv x2_1/y2_1$) behaviour.
(7) For each pair of $\rho_{1,2}$-related (m, n) diagrams, we construct an (m, n) diagram of the A/C refinement as follows. The execution fragment of A is the conjunction of: the execution fragment of $A1$ on $x1_1$, the execution fragment of $A1$ (or $A2$) on x_{12}, and the execution fragment of $A2$ on $x2_2$. The execution fragment of C is the conjunction of: the execution fragment of $C1$ on $y1_1$, the execution fragment of $C1$ (or $C2$) on y_{12}, and the execution fragment of $C2$ on $y2_2$. (And the refinement relation satisfied at the beginning and end of the constructed (m, n) diagram is $R_{\mathrm{A,C}}$.

Theorem 1. *Let system $A1$ have refinement $C1$ and system $A2$ have refinement $C2$, as described. Let systems A and C be as constructed above, and suppose points (1)-(7) above hold. Then*

1. *The (m, n) diagrams constructed for A and C in (7) are valid (m, n) diagrams, in that the abstract and concrete execution fragments are related via the $R_{\mathrm{A,C}}$ refinement relation.*
2. *The collection of (m, n) diagrams for A and C thereby constructed yields a Continuous ASM refinement from A to C.*

Proof: To show claim 1., we consider a typical constructed (m, n) diagram, created by fusing an $A1/C1$ (m, n) diagram with a corresponding $A2/C2$ (m, n)

[12] In the sequel, we refer to manipulations on relations via the logical operations on the logical definition of their bodies, for simplicity — e.g., (set theoretic) intersection of relations (over the same signature) is expressed via conjunction.

diagram. The $C1$ execution fragment starts in a $C1$ state that is related by $R_{\mathrm{A1,C1}}$ to the starting state of the $A1$ execution fragment. Likewise for $C2$ and $A2$. Composing the two concrete execution fragments in parallel while fusing the two identical behaviours of $y1_2$ and $y2_1$, yields an execution fragment of C that starts in a state which is related by $R_{\mathrm{A,C}}$ to the starting state of A, and because the two concrete execution fragments have the same duration as a consequence of being related by $\rho_{1,2}$, they end simultaneously, in states that are $R_{\mathrm{A1,C1}}$ and $R_{\mathrm{A2,C2}}$ related respectively to end states of the corresponding abstract execution fragments (whose identical $x1_2$ and $x2_1$ behaviours have also been fused), reestablishing $R_{\mathrm{A,C}}$ for the after-state of the constructed (m, n) diagram.

To show claim 2., we work by induction on an arbitrary execution of C. The initial C state decomposes into a $C1$ initial state fused with a $C2$ initial state. The combination of these is related by $R_{\mathrm{A,C}}$ to an A initial state, similarly decomposed and fused. For the inductive hypothesis we assume that the execution of C has been simulated, using a succession of the constructed (m, n) diagrams, reaching a concrete state $y \equiv (y1_1, y_{12}, y2_2)$ which is related by $R_{\mathrm{A,C}}(x, y)$ to an abstract state $x \equiv (x1_1, x_{12}, x2)$.

Consider the concrete execution continuing from y. The $(y1_1, y1_2)$ part of the initial portion of it is a $C1$ execution fragment that is simulated by an $A1$ execution fragment via an (m, n) diagram of the $A1/C1$ refinement. The $y1_2 \equiv y_{12}$ part of the $C1$ execution fragment is common to the $(y2_1, y2_2)$ part of the concrete execution continuing from y, i.e. common to a $C2$ execution fragment that is simulated by an $A2$ execution fragment via an (m, n) diagram of the $A2/C2$ refinement. By (6).(i) and (6).(ii), we can choose the $A2$ execution fragment to have the same x_{12} behaviour exhibited by the $A1$ execution fragment. Therefore, by (6).(iii) we can fuse the two (m, n) diagrams to give an (m, n) diagram of the A to C refinement that simulates the concrete execution continuing from y. By (5) we easily derive that the refinement relation satisfied at the end of the (m, n) diagram is R. □

The preceding constitutes a basic generic result of the kind being sought. We can imagine many variations on a result like this. For instance, we could involve inputs and outputs. Alternatively, we could insist that some of the R relations (and/or their I/O analogues) were *functions* from concrete to abstract. As another option we could relax the independence of the various relations on shared vs. unshared variables in various ways. And so on.

It is now evident that the all the mechanisms involved in combining the $A1$ to $C1$ refinement with the $A2$ to $C2$ refinement to get the A to C refinement —whether as described above, or via the generalisations suggested— centre on manipulation of the name spaces of the individual subsystems. In principle, these act as lexical binders, fixing the meaning of each identifier within the context of that subsystem. The objective of the manipulation is to then open these name spaces, in order to permit name capture of the free identifiers inside, which then become variables shared across the larger system. The ASM approach of allowing an individual subsystem's behaviour to be influenced by monitored variables —whose updates need not be defined within the subsystem— allows each sub-

system to have enough available behaviours, that the behaviours required for shared variable cooperation are available, and can be specified within a larger system by the name capture technique.

7 Refinement and the Tennis Ball

We now expand our tennis ball example to illustrate the potential for our framework to express the inclusion of design detail via refinement. However, rather than developing a more elaborated version of the previous model —which typically would entail the introduction of copious quantities of technical detail— we develop an *abstraction* of the model of Section 3, and we argue that our original model arises as a refinement of the new one via the ASM refinement mechanism.

Thus, suppose the rally was taking place in a court surrounded by a fence of height H. One approach to the design might be that the fence is sufficiently high that no player can hit the ball out of the court. Of course, to enforce such a restriction absolutely might well be too demanding in a realistic setting, but we can pursue it in our idealised scenario anyway.

From this perspective, the only property of the ball that we need to care about is its total energy. That determines the maximum height it can reach via the conversion of all that energy into potential energy via the law of conservation of energy. In fact, since the ball's kinetic energy arises from the inner product of the velocity vector with itself, and the vertical and horizontal components of the velocity are orthogonal, we can identify the vertical energy (arising from the square of the vertical component of the velocity) as a separately conserved quantity, and it is only that energy that is available to be converted into potential energy as the ball flies upwards. Therefore, aside from an abstract INIT rule, we can model the path of the ball in abstract terms using the following abstract pliant event, in which, for clarity, the subscript A distinguishes abstract variables from their former (now concrete) counterparts:

$\textsc{Flight}_A \stackrel{c}{=}$
choose $px'_A(t), py'_A(t), vx'_A(t), vy'_A(t)$ **with** $\frac{1}{2}{vy'}^2_A(t) + g\, py_A(t) \leq E_{max}$
do $px_A(t), py_A(t), vx_A(t), vy_A(t) \;:=\; px'_A(t), py'_A(t), vx'_A(t), vy'_A(t)$

In $\textsc{Flight}_A$, we use the direct assignment form of a pliant rule to allow the dynamics of the abstract tennis ball to evolve arbitrarily, subject only to the constraint that its vertical energy, $\frac{1}{2}{vy'}^2_A(t) + g\, py_A(t)$ remains within E_{max}. In such direct assignment pliant rules, it is tacitly assumed that the behaviours of the variables are only ever assigned to (piecewise) absolutely continuous functions of time. This restriction implies that the derivatives of these functions exist in the Carathéodory sense, and thus the semantics of such direct assignment cases falls within the scope of the previously given differential equation semantics when we interpret a direct assignment $z := \Theta$ via differentiation, i.e., $\mathcal{D}z := \mathcal{D}\Theta$. (We observe that if Θ has discontinuities, then these can be handled via the "virtual skip" mechanism discussed earlier.) With the $\textsc{Flight}_A$ rule

in place, it is now easy to build an (m, n) diagram to show the refinement of FLIGHT$_A$ to some of the behaviours we discussed in the previous section.

Firstly, we model the passage of time in the same way in our two systems. Secondly, the m of our (m, n) diagram will be 2: i.e. an execution of the abstract INIT rule, followed by a single execution of FLIGHT$_A$ at the abstract level. Thirdly, the n of our (m, n) diagram will be covered by two broad cases: it will be 6 for the normal dynamics case discussed in Section 3; and it will be $6 + 2k$ (with $k \geq 1$) for the "approaching Zeno" cases.

For the normal dynamics case, the sequence of 6 steps consists of INIT, FLIGHT, BOUNCE, FLIGHT, RACQUET, FLIGHT. For the "approaching Zeno" cases, it consists of INIT, FLIGHT, then k repetitions of (BOUNCE, FLIGHT), and then RACQUET, FLIGHT. The Zeno case itself would correspond to FLIGHT, followed by an infinite number of repetitions of (BOUNCE, FLIGHT). But we do not regard that as a proper (m, n) diagram because of the infinite number of steps.

To qualify as ASM refinements, we need to make explicit the equivalence R that such (m, n) diagrams preserve. For this, we observe that provided that the parameters of the earlier model are confined (in the static algebra within which the dynamics takes place), to values that limit the vertical energy of the ball appropriately, then R can taken to be a partial identity relation between abstract and concrete states, being defined as the identity on those states which have a vertical energy that does not exceed the specified maximum. The fact that the equivalence is preserved is proved by the observation that the concrete dynamics permitted by the explicit model of Section 3 is simply one of the arbitrary behaviours allowed in the abstract model (provided that the energy of the concrete model remains suitably constrained).

8 Related Work

The framework we described above is similar to many ways of formulating hybrid systems present in the literature. We comment on some aspects of that here. Earlier work includes [4, 5, 27, 32]. Shortly after these works were published, there appeared a spate of other papers, such as [32, 20, 21]. Much further activity ensued, too much to be surveyed comprehensively. A large proportion of it is described in the *Hybrid Systems: Computation and Control* series of international conferences. Slightly later formulations include [33, 10, 28]. Many of these earlier approaches, and especially the tools that support the relevant methodologies are surveyed in [14]. A less old theoretical overview is to be found in [48].

The majority of these works take an automata-theoretic view of hybrid systems. Thus, they have named states for the discrete control, within each of which, continuous behaviour evolves. This continues until the next preemption point arrives, triggered by the guard condition of the next discrete state becoming true. We achieve a similar effect via our mode and pliant operations. This relatively small degree of difference is in fact reassuring, since, in attempting to describe physical behaviour we have little leeway: the physical world is as it is and all descriptions must conform to it.

From our point of view, the capabilities of most of these systems are rather similar, except in those cases where the expressivity of the continuous part has been deliberately curtailed in order to get greater decidability, e.g. the pioneering [28] where continuous behaviour is linear in time. The focus on decidability is pursued vigorously in the literature. The survey [19] is a contemporary overview of reachability analysis of hybrid systems, and discusses many sublanguages of the general hybrid framework, restricted so that one or other variation of the notion of reachability is decidable for them.

The general hybrid framework is so expressive, that its undecidability is relatively self-evident, even if attention has to be paid to the details in order to model a two counter machine, which is the usual route to the result. The consequence of this is that unbounded state values are needed, or the state space will have accumulation points. While these are fine theoretically, both are unrealistic from an engineering standpoint, since engineering state spaces have both a finite size, and a limited accuracy.

The absence of the automata-theoretic structure in our approach simplifies the description of systems somewhat. All aspects become expressible in a relatively recognisable "program-like" syntax. The separation of discrete transitions from continuous ones also chimes with our other goal, of developing a hybrid formalism as a clean extension of an existing discrete formalism, syntactically and semantically. This also allows for different kinds of mathematical reasoning, relevant to the two worlds, to be cleanly separated on a *per rule* basis.

One difference between these approaches and ours, is the greater attention we have paid to the general semantics of differential equations. Issues of noise aside, classical physics is invariably defined in these terms, so we took that as basic. Many of the approaches above sidestep the issue by merely positing the existence of a *continuous flow* over the time interval between two discrete transitions. The equivalent of that for us would have been to take the criteria at the end of section 2.2 as part of our formalism's *definition*, rather than as *properties* to be demonstrated on the basis of that definition. We argued for the truth of these on the basis of "off the shelf" mathematics in Section 4.

Properly controlling the continuous behaviour is just as important as properly defining the discrete, of course. Innocent looking conditions, such as merely requiring the right hand side of a DE to be continuous (c.f. [6]), can, strictly speaking, be unsound.[13]

The way to avoid problems, is to restrict the form of the allowed differential equations to cases whose properties are known. The results surveyed in [19] give many examples of this kind. Among these are several that incorporate the standard textbook results on linear and non-linear DEs, long included in computer algebra systems like Mathematica [35] or Maple [34]. A more general approach to DEs is taken in [37, 38]. In our case, we have broadened the class of allowed

[13] The standard counterexample that mere continuity of the right hand side admits is $\mathcal{D}\, x = x^2$. This has a solution $x(t) = (a - t)^{-1}$ (for some constant of integration a), which explodes at $t = a$. Such counterexamples are very familiar in the differential equations literature, typically being surveyed in the opening pages of standard texts.

differential equations quite a bit, to maximise expressivity, relying on the "off the shelf" mathematics mentioned above to suply solutions, where they are available. (Of course, only a tiny fraction of the DEs that one can write down have solutions that one can write down [39].)

9 Conclusions

In the preceding sections we first reviewed traditional discrete ASM, founding it on a discussion of basic ASM rules, and then we embarked on an extension that would allow a convincing description of the continuous phenomena inherent in hybrid and cyber-physical systems. Our strategy was based on deciding on a simple semantic domain first, centered on piecewise absolutely continuous functions of time that were solutions of well posed initial value problems of ordinary differential equations. We then arranged the syntax and its formal semantics to map cleanly onto it. The benefits of this included the fact that the behaviour of every variable could be fully described by a straightforward function: from a semi-infinite or finite interval of time to its type, and satisfying the properties mentioned. Of the many available ways of formulating continuous phenomena within applied mathematics, this semantic domain covers the vast majority of the problems that arise in practice, and is ultimately behind most formulations of hybrid and cyber-physical systems, which are so intensively studied today [48, 37, 1, 36, 47, 46], [15, 55, 56, 51, 8, 53, 27].

The formal semantics was then described, in sufficient detail that a fully rigorous technical definition could be elaborated from it if desired. We did not go into the full details however, since so much of that could be straightforwardly taken from quite standard sources. After that we considered refinement, and having reviewed discrete ASM refinement, we formulated continuous ASM refinement as a minimal extension of the discrete case. Our various discussions of semantics were complemented by discussions of issues surrounding compositionality and multi-subsystem systems, in the light of the formulation given. Accompanying this, we gave a simple illustration of the formalism in an example involving the flight of a tennis ball. Despite the apparent simplicity, this example nevertheless provided an opportunity to discuss further technical issues that arise when we model hybrid systems in a clean way, for example as Zeno effects. We illustrated the formulation of Continuous ASM refinement by showing an abstraction, illustrating how very general properties could be specified in our formalism, and could then be refined to more specific behaviours. In future work, we intend to use our formulation to explore larger, more complex case studies.

References

1. Summit Report: Cyber-Physical Systems (2008), http://iccps2012.cse.wustl.edu/_doc/CPS_Summit_Report.pdf
2. Abrial, J.R.: The B-Book: Assigning Programs to Meanings. Cambridge University Press (1996)

3. Abrial, J.R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press (2010)
4. Alur, R., Courcoubetis, C., Henzinger, T., Ho, P.H.: Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems. In: Proc. Workshop on Theory of Hybrid Systems. LNCS, vol. 736, pp. 209–229. Springer (1993)
5. Alur, R., Dill, D.: A Theory of Timed Automata. Theor. Comp. Sci. 126, 183–235 (1994)
6. Back, R.J., Petre, L., Porres, I.: Continuous Action Systems as a Model for Hybrid Systems. Nordic J. Comp. 8, 2–21 (2001), extended version of FTRTFT-00, LNCS 1926, 202-213.
7. Banach, R., Zhu, H.: Language Evolution and Healthiness for Critical Cyber-Physical Systems. J. Soft. Evol. and Proc. 2020;e2301, 24pp. (2021)
8. Barolli, L., Takizawa, M., Hussain, F.: Special Issue on Emerging Trends in Cyber-Physical Systems. J. Amb. Intel. Hum. Comp. 2, 249–250 (2011)
9. Beauquier, D., Slissenko, A.: On Semantics of Algorithms with Continuous Time. Tech. Rep. TR-LACL-1997-15, LACL, University of Paris-12 (1997)
10. 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, vol. 279, pp. 37–51. Springer, LNCIS (2002)
11. Börger, E.: The ASM Refinement Method. FACJ 15, 237–257 (2003)
12. Börger, E., Gurevich, Y., Rosenzweig, D.: The Bakery Algorithm: Yet another Specification and Verification. In: Börger (ed.) Specification and Validation Methods. Oxford University Press (1995)
13. Börger, E., Stärk, R.: Abstract State Machines. A Method for High Level System Design and Analysis. Springer (2003)
14. Carloni, L., Passerone, R., Pinto, A., Sangiovanni-Vincentelli, A.: Languages and Tools for Hybrid Systems Design. Foundations and Trends in Electronic Design Automation 1, 1–193 (2006)
15. Clarke, E., Zuliani, P.: Statistical Model Checking for Cyber-Physical Systems. In: Bultan, Hsiung (eds.) Proc. ATVA-11. LNCS, vol. 6996, pp. 1–12. Springer (2011)
16. Cohen, J., Slissenko, A.: Implementation of Timed Abstract State Machines with Instantaneous Actions by Machines with Delays. Tech. Rep. TR-LACL-2008-2, LACL, University of Paris-12 (2008)
17. Derrick, J., Boiten, E.: Refinement in Z and Object-Z: Foundations and Advanced Applications. Springer-Verlag UK (2001)
18. ESW: Embedded Systems Week Conferences
19. Fränzle, M. and Chen, M. and Kröger, P.: In Memory of Oded Maler: Automatic Reachability Analysis of Hybrid-State Automata. ACM SIGLOG News 6, 19–39 (2019)
20. Friesen, V., Nordwig, A., Weber, M.: Object-Oriented Specification of Hybrid Systems using UML, h and ZimOO. In: Proc. ZUM-98, vol. 1493, pp. 328–346. Springer, LNCS (1998)
21. Friesen, V., Nordwig, A., Weber, M.: Toward an Object-Oriented Design Methodology for Hybrid Systems. In: Object-Oriented Technology and Computing Systems Re-Engineering, pp. 1–15. Elsevier (1999)
22. Geisberger, E., Broy (eds.), M.: Living in a Networked World. Integrated Research Agenda Cyber-Physical Systems (agendaCPS) (2015), http://www.acatech.de/fileadmin/user_upload/Baumstruktur_nach_Website/Acatech/root/de/Publikationen/Projektberichte/acaetch_STUDIE_agendaCPS_eng_WEB.pdf

23. Graf, S., Prinz, A.: A Framework for Time in FDTs. In: Proc. FORTE-04. LNCS, vol. 1092, pp. 266–290. Springer (2004)
24. Graf, S., Prinz, A.: Time in Abstract State Machines. Fund. Inf. 77, 143–174 (2007)
25. Gratzer, G.: Universal Algebra. Springer (2008)
26. Gurevich, Y., Huggins, J.: The Railroad Crossing Problem: An Experiment with Instantaneous Actions and Immediate Reactions. In: Proc. CSL-95. LNCS, vol. 1092, pp. 266–290. Springer (1996)
27. He, J.: From CSP to Hybrid Systems. In: Roscoe (ed.) A Classical Mind, Essays in Honour of C.A.R. Hoare. pp. 171–189. Prentice-Hall (1994)
28. Henzinger, T.: The Theory of Hybrid Automata. In: Proc. IEEE LICS-96. pp. 278–292. IEEE (1996), Also http://mtc.epfl.ch/~tah/Publications/the_theory_of_hybrid_automata.pdf
29. HSCC: Hybrid Systems: Command and Control Conferences
30. Karlsruhe Interactive Verifier: http://www.informatik.uni-augsburg.de/lehrstuehle/swt/se/kiv/
31. Lynch, N.: Distributed Algorithms. Morgan Kaufmann (1996)
32. Lynch, N., Segala, R., Vaandrager, F., Weinberg, H.: Hybrid I/O Automata. Springer (1996)
33. Lynch, N., Segala, R., Vaandrager, F.a.: Hybrid I/O Automata. Information and Computation 185, 105–157, mIT Technical Report MIT-LCS-TR-827d.
34. Maple: http://www.maplesoft.com
35. Mathematica: http://www.wolfram.com
36. 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
37. Platzer, A.: Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics. Springer (2010)
38. Platzer, A.: Logical Foundations of Hybrid Systems. Springer (2018)
39. Polyanin, A., Zaitsev, V.: Handbook of Ordinary Differential Equations: Exact Solutions, Methods, and Problems. C.R.C. Press (2018)
40. Raynal, M.: Concurrent Programming: Algorithms, Principles and Foundations. Springer (2013)
41. Royden, H., Fitzpatrick, P.: Real Analysis. Pearson (2010)
42. Rust, H.: Hybrid Abstract State Machines: Using the Hyperreals for Describing Continuous Changes in a Discrete Notation. Tech. Rep. Proc. ASM-00, TIK Report 87, ETH Zurich (2000)
43. Schellhorn, G.: Verification of ASM Refinements Using Generalized Forward Simulation. JUCS 7, 952–979 (2001)
44. Schellhorn, G.: ASM Refinement and Generalizations of Forward Simulation in Data Refinement: A Comparison. Theor. Comp. Sci. 336, 403–435 (2005)
45. Slissenko, A., Vasilyev, P.: Simulation of Timed Abstract State Machines with Predicate Logic model Checking. JUCS 14, 1984–2006 (2008)
46. Stehr, M., Kim, M., Talcott, C.: Toward Distributed Declarative Control of Networked Cyber-Physical Systems. In: Yu, Liscano, Chen, Zhang, Zhou (eds.) Proc. UIC-10. LNCS, vol. 6406, pp. 397–413. Springer (2010)
47. Sztipanovits, J.: Model Integration and Cyber Physical Systems: A Semantics Perspective. In: Butler, Schulte (eds.) Proc. FM-11. Springer, LNCS 6664, p.1, http://sites.lero.ie/download.aspx?f=Sztipanovits-Keynote.pdf (2011), Invited talk, FM 2011, Limerick, Ireland

48. Tabuada, P.: Verification and Control of Hybrid Systems: A Symbolic Approach. Springer (2009)
49. Walter, W.: Ordinary Differential Equations. Springer (1998)
50. Wechler, W.: Universal Algebra for Computer Scientsts. Springer (1992)
51. White, J., Clarke, S., Groba, C., Dougherty, B., Thompson, C., Schmidt, D.: R&D Challenges and Solutions for Mobile Cyber-Physical Applications and Supporting Internet Services. J. Internet Serv. Appl. 1, 45–56 (2010)
52. Wikipedia: Absolute continuity.
53. Willems, J.: Open Dynamical Systems: Their Aims and their Origins. Ruberti Lecture, Rome (2007), http://homes.esat.kuleuven.be/~jwillems/Lectures/2007/Ruberti lecture.pdf
54. Woodcock, J., Davies, J.: Using Z, Specification, Refinement and Proof. Prentice Hall (1996)
55. Zhang, L., He, J.: A Formal Framework for Aspect-Oriented Specification of Cyber Physical Systems. In: Lee, Howard, Slezak (eds.) Proc. ICHIT-11. CCIS, vol. 206, pp. 391–398. Springer (2011)
56. Zhlke, L., Ollinger, L.: Agile Automaton Sysytems Based on Cyber-Physical Systems and Service Oriented Architectures. In: Lee (ed.) Proc. ICAR-11. LNEE, vol. 122, pp. 567–574. Springer (2011)

A The RAQUET Rule

We recall the RACQUET rule of Section 3.

RACQUET =
if $py > 0 \wedge py = rpy \wedge px < 0 \wedge px = rpx \wedge vx < 0 \wedge vx.rvx + vy.rvy < 0$ **then do**
$vx := -(vx - rvx)(\cos^2(\alpha) + c\sin^2(\alpha)) + (vy - rvy)(1-c)\cos(\alpha)\sin(\alpha) + rvx,$
$vy := (vx - rvx)(1-c)\cos(\alpha)\sin(\alpha) - (vy - rvy)(\sin^2(\alpha) + c\cos^2(\alpha)) + rvy$

To understand the assignments to to vx and vy in the above we use vector notation. So let $\mathbf{p}, \mathbf{v}, \mathbf{rp}, \mathbf{rv}$ be 2D vectors (in the plane of Fig. 1) corresponding to our earlier quantities. Let $\mathbf{r} = [-\cos(\alpha), \sin(\alpha)]$ be a unit vector pointing upwards along the line of the racquet, and let $\mathbf{r}^\perp = [\sin(\alpha), \cos(\alpha)]$ be a unit vector normal to the racquet, pointing towards the net.

We make a rigid Galilean transformation into the rest frame of the racquet, keeping the orientation of the racquet the same, but reducing its velocity to zero. In this frame of reference, the ball approaches the racquet with velocity $\mathbf{v} - \mathbf{rv}$. When the ball strikes the racquet, the tangential component of the ball's velocity remains the same, while the perpendicular component is reflected, and is reduced by the coefficient of restitution c. Resolving the velocity into these two components, the velocity before the collision is $[(\mathbf{v} - \mathbf{rv}) \cdot \mathbf{r}, (\mathbf{v} - \mathbf{rv}) \cdot \mathbf{r}^\perp]$, while the velocity after the collision is $[(\mathbf{v} - \mathbf{rv}) \cdot \mathbf{r}, -c(\mathbf{v} - \mathbf{rv}) \cdot \mathbf{r}^\perp] = [-(vx - rvx)\cos(\alpha) + (vy - rvy)\sin(\alpha), -c(vx - rvx)\sin(\alpha) - c(vy - rvy)\cos(\alpha)]$, where in the last expression, we have evaluated the dot products in the rectiliear frame of reference, since dot products are rotationally invariant. We can re-express this after-velocity in the rectilinear frame by applying a rotation matrix as follows:

$$\begin{bmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{bmatrix} \begin{bmatrix} -(vx - rvx)\cos(\alpha) + (vy - rvy)\sin(\alpha) \\ -c(vx - rvx)\sin(\alpha) - c(vy - rvy)\cos(\alpha) \end{bmatrix}$$

$$= \begin{bmatrix} -(vx - rvx)\cos^2(\alpha) + (vy - rvy)\cos(\alpha)\sin(\alpha) - \\ \qquad c(vx - rvx)\sin^2(\alpha) - c(vy - rvy)\cos(\alpha)\sin(\alpha) \\ (vx - rvx)\cos(\alpha)\sin(\alpha) - (vy - rvy)\sin^2(\alpha) - \\ \qquad c(vx - rvx)\cos(\alpha)\sin(\alpha) - c(vy - rvy)\cos^2(\alpha) \end{bmatrix}$$

Now, reversing the rigid Galilean transformation, we get the assignment given in RACQUET.