Under consideration for publication in Formal Aspects of Computing

Continuous KAOS, ASM, and Formal Control System Design Across the Continuous/Discrete Modeling Interface: A Simple Train Stopping Application

Richard Banach[1][1], Huibiao Zhu[2][2] Wen Su[2], Runlei Huang[3]

[1]School of Computer Science, University of Manchester, Oxford Road, Manchester, M13 9PL, U.K.
banach@cs.man.ac.uk,
[2]Software Engineering Institute, East China Normal University, 3663 Zhongshan Road North, Shanghai 200062, P.R. China.
{hbzhu,wensu}@sei.ecnu.edu.cn,
[3]Alcatel-Lucent Shanghai Bell, 388 Ningqiao Road, Pudong Jinqiao, Shanghai 201206, P.R. China.
runleihuang@alcatel-sbell.com.cn

Abstract. A very simple model for train stopping is used as a vehicle for investigating how the development of a control system, initially designed in the continuous domain and subsequently discretized, can be captured within a formal development process compatible with standard model based refinement methodologies. Starting with a formalized requirements analysis using KAOS, an abstract model of the continuous system is created in the ASM formalism. This requires extensions of the KAOS and ASM formalisms, capable of dealing with quantities evolving continuously over real time, which are developed. After considering how the continuous system, described as a continuous control system in the state space framework, can be discretized, a discrete control system is created in the state space framework. This is re-expressed in the ASM formalism. The rigorous results on the relationship between continuous and discrete control system models that are needed to establish provable properties of the discretization, then become the ingredients of a retrenchment between continuous and discrete ASM models, and are thus fully integrated into the formal development. The discrete ASM model can then be further refined towards implementation.

Keywords: Continuous KAOS, Continuous ASM, Control Systems, Rigorous Design, Refinement, Retrenchment, Continuous Modeling, Discrete Modeling, Train Control.

Correspondence and offprint requests to: Richard Banach, School of Computer Science, University of Manchester, Oxford Road, Manchester, M13 9PL, U.K. email: banach@cs.man.ac.uk

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

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

1. Introduction

Conventional model based formal refinement technologies (see for example [dRE98, DB01, SS98, Abr96, PST96, WD96, Abr10]) are based on purely discrete mathematical and logical concepts. These turn out to be ill suited to modeling —still less, to formally developing— applications whose usual models are best expressed using continuous mathematics. Nevertheless, many such applications, control systems in particular, are these days implemented using digital techniques, despite being designed in the continuous domain. To the extent that such systems can be high consequence (eg. avionics systems, nuclear power control systems, weapons command and control systems, automated public transport systems, medical instrument systems), the dependability to be gained by utilizing formal techniques during their development is of course a highly desirable addition to the development process [BH99a, BH99b, Hal90, BH95, Hal07]. This raises a dilemma: the conflict between the acceptance of the intrinsic desirability of using formal techniques, and the recognition that there are significant technical obstacles to their direct application in the natural problem domain. In practice this usually results in some sort of "avoiding the problem" for the majority of cases.

In the vast majority of scenarios, engineering rules of thumb are used to guide the implementation of a continuous control design by a discrete controller (i.e. one which reacts to inputs and determines outputs at regularly occurring instants corresponding to the sampling frequency, instead of continuously). Such implementations are evaluated principally by testing [Bro10], the more so when the application is not viewed as being of high criticality.

Even in relatively critical applications, such as cruise control for vehicles, present day energy saving considerations place pressure on the design to reduce the sampling frequency to the minimum that will suffice for the application, thus forcing the sampling frequency to vary according to real-time system parameters [But]. Again, the evaluation of such designs is empirical and heuristic.

In more critical applications, what usually happens is that even if formal development techniques are utilized, their deployment takes place only after the design and modeling process has crossed the continuous to discrete watershed [Mey]. This is, of course, better than nothing, but it is hardly ideal.

In the light of the above considerations, and in view of increasing pressure to optimise design parameters including sampling frequency, it would clearly be of benefit to the development of this kind of system, if the rigour of formal techniques could be applied also to the continuous to discrete design transformation, supplementing the informal techniques routinely used.

In this paper, we tackle the mismatch between continuous modeling and discrete development techniques, head on. Although the traditional discrete development technique of choice, model based refinement (in one or other of its guises), is too exacting as regards how close a system model has to be to its successor in the development process for refinement to have the capacity to straddle the continuous to discrete demarcation line (at least in the most general case), a judicious weakening of it, retrenchment, proves to be adaptable enough to do the job.

Retrenchment, being an intrinsically weaker notion than refinement, thus possessing weaker generic properties, is best only used where refinement is inapplicable, or where the application of refinement would be so unnatural that it would risk derailing the development strategy. Accordingly, the ideal development methodology (from our perspective) is to combine refinement steps (where these can be made to work convincingly), with retrenchment steps (in those parts of the development where refinement will not work well). The whole process needs to be consistent of course. This consistency is handled by the Tower theorems [BJ, Jes05], which show how retrenchment and refinement steps can coexist in a consistent whole.

In this paper we tackle the continuous to discrete issue, by taking a simple running example, one that can be solved fully by analytic means in both the continuous and discretized domains, and tracing it through a full scale formal development process. This not only shows how the discretization problem may be tackled *per se*, but also exibits the entire development process end-to-end, showing how the various technical ingredients fit together.

Thus we start with requirements analysis. In the context of our simple example, this entails eliciting the requirements in the continuous domain. Proceeding from top level requirements, we engage in a requirements refinement activity, albeit a rather simple one suited to the context of our example. So the top level requirements get refined to a model in the form of a continuous control problem. At this point our treatment of discretization kicks in. We remodel the continuous control problem to a discretization of it as a discrete control problem, and derive a description of the discretization step via a suitable retrenchment. Now firmly in the discrete world, we finally illustrate how a more complex development in this style would go, by refining further the discrete control problem towards implementation.

Discretizations in general are very challenging as regards the obtaining of precise and reliable quality metrics for their behaviour. This explains the overwhelming preponderance of informal and heuristic approaches to their use mentioned above. Some progress on this front can be made by looking at very simple models that can be solved exactly in both discrete and continuous domains. The comparison of exact (and thus reliable) calculations of both continuous and discretized behaviours, at least on a limited range of examples, gives a reasonable idea of how things might go in a

more general class of less tractable instances. So, as well as exibiting the whole development process in this paper, we focus on a very simple example in order to also illustrate these more qualititative aspects, based on reliable evidence.

Our example will be based on an extremely simple model for train stopping, omitting all the details that make automatic train control highly non-trivial. The extreme simplicity of the example, all aspects of which can be solved analytically in both continuous and discretized domains, enables us to monitor the formal aspects of the development in detail, but yet within a relatively limited space. A much more complex case study is tackled in [BZSW12b, BZSW].

The rest of the paper is organized as follows. We start in Section 2 by reviewing the nature of related work in the literature, concluding that the approach we are promoting is a distinct contribution. The next few sections develop the train stopping case study informally. Section 3 develops both the continuous and discretized versions in the manner of an elementary dynamics problem, showing how the various quantities that enter the system description are related to each other. All the calculations are exact thus far. Section 4 recasts these insights as control problems. Section 5 then analyzes the relationship between the continuous control problem and its discretized counterpart, using rigorous results on differential equations covered in the Appendix. These are checked against the exact calculations, and the value of such corroboration is discussed from an engineering viewpoint. This completes the informal development of the case study.

In Section 6 we stand back and give an overview of how we intend to formalize the preceding. In Section 7 we examine the KAOS formal requirements engineering methodology. The discrete version of this is reviewed, and a substantial detour extends this to properly cover continuously varying quantities. The KAOS modeling ends with an operationalized description, which is cast in the ASM framework. The continuous extension of KAOS enables a continuous extension of the ASM formalism to be made, in order that the continuous aspects of the models developed earlier can be formally operationalized. The case study is therefore revisited in Section 8, where it is redeveloped as far as a formal ASM continuous model. Section 9 presents the corresponding formal ASM discretized model. Section 10 reviews ASM refinement and retrenchment in the discrete world, and gives the extension for the continuous world, which follows straightforwardly. Section 11 discusses the partitioning of the integrated discretized model into separate plant and controller submodels, and argues that in the general case, the recombination of these ought to be a complete refinement of the integrated model — similar remarks would also apply to the continuous model. The partitioning into submodels confirms that the variables that were involved in the earlier analysis of the relationship between continuous and discretized models were the correct ones, and this supports the formalization of that relationship between the models as a retrenchment in Section 12 — further comments are also made regarding the corroboration between exact and formal results. We are now firmly in the discrete world. Section 13 develops the formal ASM controller submodel further towards implementation via refinement, as would be done in conventional formal developments of control systems. Since refinement is well understood in the purely discrete world, the refinement that is given is relatively simple, and is mainly for purposes of illustration. In Section 14 we cover a number of issues, which, although not on the critical path of the work done earlier, nevertheless impinge significantly on it. Section 15 concludes, suggesting how the ideas developed in the paper may be extended into wider engineering practice.

2. Related Work

In this section we describe the context of our work, particularly as it relates to the hybrid systems field, before plunging into the details of our own approach later.

For a long time, the relationship between continuous and discrete transition systems has been a topic for investigation in the hybrid systems arena. Earlier work includes [ACHH93, Hen96, AD94, He94]. Also, the series *International Conference on Hybrid Systems: Computation and Control*, has been the venue for a large amount of research in this area. More recent references are [Tab09, Pla10b]. Much of this work is connected with the design and development of embedded systems in which digital computation (which is obviously discrete) is interfaced to the physical world (which is obviously continuous). Much more recently, the newly coined term “Cyber-Physical Systems” covers embedded systems in which the computational component is typically much more distributed among a network of processing nodes than in the typical single-processor embedded systems of the past. Controller applications are a major area of interest in these areas.

Compared to the majority of work in the hybrid systems and related areas, our own work features a difference in overall aim, which subsequently leads to technical differences — we describe this now. Hybrid systems are dynamical systems that mix smooth, continuous transitions with discrete, discontinuous ones. The major focus for existing work has been the automatic verification of properties of such systems. Obviously, such verification demands the representation of the systems in question in discrete and finite terms, whether by means of an explicitly constructed finite state space (which can be manipulated directly), or a state space whose individual states arise via the symbolic representa-

tion of a much less tractable state space of a previously constructed underlying system (the space of symbolic states then being manipulated symbolically of course).

The main tool for bringing an intractable state space within the scope of computable techniques is the construction of a suitable equivalence relation. Regions of the state space are gathered into equivalence classes, and a representation of these equivalence classes (whether as individual elements in a simple approach, or as symbolic expressions that denote the equivalence class in question) constitutes the state space of the abstraction. Transitions between these states are introduced to mirror the behaviour of the underlying system. The properties of interest can then be checked against the abstract system. For instance, properties that can be expressed as reachability properties fall within the scope of model checking approaches that are applied to the abstraction.

Of course what has been constructed thereby is a (bi)simulation, and a major strand of hybrid systems research is the investigation of such (bi)simulations. The same remarks apply when there is an external control applied to the systems.

One disadvantage of the above approach is the frequent reliance on brittle properties of the studied systems. Put most simply, a number of techniques rely on the parameters of the problem falling within a subset of measure zero of the parameter space. Of course a real system can never hit such a small target in any reliable way. Equally, the simulation relations studied can also be just as brittle — and even if a simulation built upon such a foundation appears to work well for a while, inevitably the difference between the constucted system and the ideal system will lead to increasingly visible departures from the mathematically exact behaviour. To alleviate this, and to address other issues of interest, the notion of *approximate (bi)simulation* has been studied in recent years ([Tab09] gives a good introduction). Here, instead of defining the simulation relation $R(u,v)$ between an abstract state u and a concrete state v as a simple predicate on states, it is defined via a distance function $\mathbf{d}$ as $R_\varepsilon(u,v) \equiv \mathbf{d}(f(u),v) \leq \varepsilon$, where f is a precise relationship between the two state spaces which is in some sense "semantically natural" (what this means in practice is that f typically captures the kind of ideal relationship between the state spaces that the brittle approaches just discussed rely on). For bisimulation you need a symmetrical arrangement of course.

(Bi)simulation depends on assuming the appropriate relation between the two before-states and re-establishing it in the after-states of suitable pairs of transitions. To preserve a relationship based on distance, the dynamics needs to be inherently *stable*. The obvious centre of attention thus becomes stable control systems, normally *linear* stable control systems, because of their calculational tractability. These are discussed in very many places, eg. [Oga08, DB10, DTB97, DH95, Ahm06, Son98, Bar75, AM06].

In a stable system all trajectories converge to a single point, so the distance between two trajectories decreases monotonically; hence a simulation relation based on an *a priori* specified distance between trajectories is maintained. But although most systems are designed to be stable in this sense, some are not, and there can be parts of a system phase space in which trajectories diverge rather than converge, without this rendering the system useless. Below, we treat in detail a very simple example which happens to be unstable in the sense just discussed. We know it is not stable because we solve it exactly.

As our account so far indicates, in the majority of the literature it is the case that the abstract discrete approximation to a given continuous system is manufactured from it (eg. by constructing the equivalence classes discussed above). In practice this process can be computationally very expensive.[3] It is one thing to be able to generically prove in a page or two that a particular kind of discrete approximation to a continuous or hybrid system exists, and quite another to actually build that approximation in a specific case. The reason is that the typical generic proof relies heavily on existential quantification, and this translates, in a practical case, to some form of search — the discrete approximation is not so much calculated as found. In such a case, one can legitimately ask whether working with the original system to verify the properties of interest has significantly different complexity to building the discrete approximation and then verifying the properties of interest against it. Often the verification strategy in the discrete approximation relies on properties inherited from the original system anyway, such as the preservation of the convexity of subsets of the phase space along the system dynamics.

In contrast to the above, our aim is rigorous development of a system towards a concrete implementation starting from an abstraction, rather than buiding an abstraction from a concrete system in order to verify properties of the concrete system. In fact, our starting point is the same as for the previous approach: the continuous system; but the orientation is different. Continuous systems are implemented using devices (eg. sensors and actuators) which, even if they themselves act continuously, are managed using digital (i.e. discrete) controllers. If we thus view the continuous system as the idealisation (which, given that design most often starts by considering the continuous system, is fair

[3] This is in stark contrast to the case of abstracting a system which is discrete to start with. There, the original system is typically described using first order logic formulae (or similar), and the abstraction process usually depends on forgetting case distinctions, resulting in simpler formulae and reducing complexity.

enough), the implementation strategy necessitates the crossing of the continuous/discrete divide, imposing a significant challenge to refinement technologies — a challenge which we overcome by using retrenchment in this paper.

Turning a continuous system into a discrete system generates complexity. This is true regardless of whether one is generating a discrete system for verification or for implementation, although the nature of the two processes is different. Not only do continuous systems correspond well to observed reality (at least in the classical physics sphere), but the limit processes that underpin continuous mathematics clean up the properties of continuous mathematics very well, leading to calculational tractability where there would be none in a closely related discrete counterpart. This explains the strong impulse to do initial design in the continuous domain. Although the majority of related work aims for an abstraction, as described, there are some works that are relevant to the development problem; we can mention [DHR05, Sta02].

In our approach to implementation, we take an "off the shelf" attitude to discretization. What this means is that rather than attempting to construct the discrete concrete counterpart of an abstract continuous system by some *ad hoc* method that might fit what is available via refinement and simulation theory (which would roughly correspond to what is discussed above for verification purposes), we take (at least a surrogate for) what is actually done in engineering practice, and instrument it as well as we can using refinement and retrenchment techniques, the aim being to arrive at quantitative estimates for the quality of this approach to implementation.

What is actually done in practice is adequately represented by a "zero order hold" approach to discretization. In this, a digital controller wakes at regular intervals, reads the current values available from the relevant sensors, and on the basis of these values and any relevant internal state, calculates the values to be output to the relevant actuators. The "zero order" aspect indicates that these output values are then "held" constant throughout the succeeding time interval.[4]

This is the sense in which our approach is closer to conventional engineering practice. Of course this approach and the abstraction approach discussed earlier are not necessarily mutually exclusive: the parameters of the zero order hold approach may fall within the range of parameters of a discrete approximation extracted by analysis of the original system, and *vice versa*.

Finally, our approach is expressed via retrenchment, one consequence of which is that our analysis is not confined to the purely stable case. In effect, the greater expressiveness of retrenchment permits (the analogue of) the approximate simulation relation mentioned above, to increase its permitted margin of error, as well as to decrease it, which makes for greater convenience in practice.

3. Simple Train Stopping, Continuous and Discrete

In this section we introduce our running example for the paper. The discussion remains at a fairly informal level for the time being.

Our target application domain is control problems in the railway sphere. In this paper we have train stopping as a specific case study. Of course, in reality, train position control is a complex problem [SYW+11, SAZH11, IEE], relying on the co-operation of many mechanisms to achieve a reliable outcome. We do not have the space here to deal with all these aspects and their subtle interactions. Rather, our aim is to describe how model based refinement and retrenchment can be deployed to add dependability to the development of such a complex scenario, and to illustrate this, we stick to a problem that is relatively trivial from the technical standpoint: stopping the train in a relatively straightforward way. Moreover, we focus on a key technical aspect of this simple problem, namely how to capture quantitative information about the transition from the continuous version of the problem to a discrete counterpart (the zero order hold version, as discussed above), and how to integrate that knowledge into a model based refinement style formal development.

Our scenario is thus as follows. Suppose a train, of mass M, is traveling at its cruise velocity V, when it needs to stop. In reality, the profile of the stopping dynamics will be determined by a large number of factors, ranging from the timeliness of the stopping, via energy considerations, to passenger comfort and usability. For us, simplicity is the keystone, since a proper evaluation of our results will rest on having exact calculations available in our case study, to enable comparison with more generic but inevitably more approximate techniques.

The ultimate in simplicity is constant deceleration, and this could even be said to be an entirely plausible proposal regarding energetics and usability considerations. Unfortunately, the zero order hold discretized approximation to a constant-valued function is ... the same constant-valued function, which would completely trivialise our case study.

[4] Higher order holds attempt to follow the anticipated evolution of the ideal continuous control more closely, by estimating the derivative(s) of the sensor signals using internal controller state, and instructing the actuators to follow a suitable low order polynomial behaviour during the succeeding time interval.

So we choose the next simplest thing: we assume that a linearly increasing deceleration rate at (with a constant) is appropriate. To bring the train to a standstill in this way, a force $F = -Mat$ (where t is time) has to be applied, by Newton's Law. Now we hit a snag, namely that passengers get on and off the train at various stations, causing M to vary. However, we note that passengers are unable to get on and off the train *in between* stations, so the work expended by the train in achieving cruise velocity since the last station, which is monitored by the train propulsion system, enables the propulsion management system to calculate the current value of M. So we will assume that M is known. Now, since the relationship between force and acceleration is purely proportional, with a constant coefficient of proportionality (per individual stopping episode), for the purpose of simplifying our case study, we can focus on just the kinematic aspects, and so we will neglect mass and force aspects for the rest of the paper.

To minimize notational clutter coming from excessive generality, for the rest of the paper, we will suppose that there is a single stopping episode, which starts at time 0 and at x position 0, and which thus ends at some time which we call T_{Stop}, with the train having traveled to position $x = D$.

The following subsections elaborate the details of this in the continuous and zero order hold discretized scenarios. It is worth noting here that the formulae that follow might be representative of the kind of exploratory calculations done in the earliest stages of design, where the broad parameters governing the application are identified, and the constraints that hold between them are recognised. Thus, in both the continuous and discretized cases, the problem emerges as a number of constraints that have to hold between the problem parameters, and the system designers are then free to adjust or manipulate these parameters at will to attain their design goals, provided that the identified constraints are maintained. In other words, the following calculations serve to define the continuous and discretized design spaces, and our goal in this paper is to explore how these spaces relate to each other.

3.1. Continuous Train Stopping

For the continuous scenario, a cursory knowledge of elementary kinematics is enough to reveal that under linear deceleration, the deceleration, distance and stopping time are linked. Representing time derivatives with a dot, if v is the velocity, then we know that

$$\dot{v} = -at \tag{1}$$
$$v(0) = V \tag{2}$$
$$v(T_{Stop}) = 0 \tag{3}$$

Regarding the distance traveled x, we know that

$$\dot{x} = v \tag{4}$$
$$x(0) = 0 \tag{5}$$
$$x(T_{Stop}) = D \tag{6}$$

Integrating these, rapidly brings us to

$$V = \frac{1}{2} a T_{Stop}^2 \tag{7}$$
$$D = V T_{Stop} - \frac{1}{3!} a T_{Stop}^3 \tag{8}$$

Both (7) and (8) feature a. Substituting the a value from (7) into (8) gives

$$D = \frac{2}{3} V T_{Stop} \tag{9}$$

which is the required relationship between D and V and T_{Stop}. The value of a links to these via (7)

$$a = \frac{2V}{T_{Stop}^2} \tag{10}$$

3.2. Discrete Train Stopping

For the discretized scenario, the situation is more complicated. As we said before, in a zero order hold discretization, at regular evenly spaced moments during the dynamics, the digital control system takes note of any inputs coming from

the sensors in the environment, and outputs control values to relevant actuators in the environment. In between these regular moments the control system is idle, ignoring inputs and making no change to outputs. Consequently, since the environment (being part of the real world) always behaves as a continuous system, it receives a different control signal from that which it would if the control system were also behaving continuously. Understanding the consequences of this difference in a way that can be integrated into an end-to-end formal development process is the main objective of this paper. The importance of the task follows from the fact that such zero order hold strategies are very common implementation strategies in the control systems of today.

We design the discretization as follows. As before, the train, traveling at velocity V, needs to come to a standstill after a time T_{Stop}, having gone a distance D_D.[5] Instead of doing so continuously though, it will do it in a number of discrete episodes. For this purpose, let us assume that T_{Stop} is divided into N short periods, each of length T, so that

$$T_{Stop} = NT \tag{11}$$

Since our discretization scheme is based on a zero order hold, in which the same control input value is maintained throughout an individual time period, the counterpart of the linear deceleration rate at of the continuous treatment will be a piecewise constant deceleration, with the constant deceleration rate increasing in magnitude by an additional multiple of a constant a_D after each time interval of length T.

Calling the discretized velocity variable v_D, we have for the acceleration

$$\dot{v}_D(t) = -ka_DT \tag{12}$$

where

$$k = \left\lceil \frac{t}{T} \right\rceil \tag{13}$$

so that k takes values in the range $1 \ldots N$. If we set, for a general t,

$$\delta t_k = t-(k-1)T = t - \left\lfloor \frac{t}{T} \right\rfloor T \tag{14}$$

then recalling that the initial velocity is V, provided $(k-1)T < t < kT$, the velocity during the k'th period is

$$v_D(t) = V - a_DT^2 - 2a_DT^2 - \ldots - (k-1)a_DT^2 - ka_DT\delta t_k \tag{15}$$

We retain the constraint that the final velocity is zero, whence we derive

$$\begin{aligned} V &= a_DT^2 + 2a_DT^2 + \ldots + Na_DT^2 \\ &= \frac{1}{2}a_DT^2N(N+1) \end{aligned} \tag{16}$$

Knowing the velocity, we can integrate again. Calling the displacement in the discretized world x_D, the contribution to x_D during the period $(k-1)T < t < kT$ comes out as

$$(V - a_DT^2 - 2a_DT^2 - \ldots - (k-1)a_DT^2)\delta t_k - \frac{1}{2}ka_DT\delta t_k^2 \tag{17}$$

Thus for the total distance we find

$$\begin{aligned} D_D = {} & VT - \frac{1}{2}a_DT^3 + \\ & VT - a_DT^3 - \frac{1}{2}2a_DT^3 + \\ & VT - a_DT^3 - 2a_DT^3 - \frac{1}{2}3a_DT^3 + \\ & VT - a_DT^3 - 2a_DT^3 - 3a_DT^3 - \frac{1}{2}4a_DT^2 + \\ & \ldots \\ & VT - a_DT^3 - 2a_DT^3 - \ldots - (N-1)a_DT^3 - \frac{1}{2}Na_DT^3 \end{aligned} \tag{18}$$

[5] We will use a subscript 'D' to indicate quantities in the discretized model that differ from their continuous counterparts.

Summing this vertically, we get

$$D_D \;=\; NVT - a_D T^3 \sum_{k=1}^{N-1} (N-k)k - \frac{1}{2} a_D T^3 \sum_{k=1}^{N} k \tag{19}$$

Applying the summation formulae

$$\sum_{k=1}^{N} k \;=\; \frac{1}{2} N(N+1) \qquad \text{and} \qquad \sum_{k=1}^{N} k^2 \;=\; \frac{1}{6} N(N+1)(2N+1) \tag{20}$$

and bearing in mind (11), we reduce (19) to

$$D_D \;=\; VT_{Stop} - \frac{1}{12} a_D T^3 (2N^3 + 3N^2 + N) \tag{21}$$

Both (16) and (21) feature a_D. Substituting the a_D value from (16) into (21) gives

$$D_D \;=\; VT_{Stop} \left[1 - \frac{2N^2 + 3N + 1}{6N^2 + 6N} \right] \;=\; \frac{2}{3} VT_{Stop} \left[1 - \frac{1}{4N} \right] \tag{22}$$

We see that (22) for D_D contains an $O(1/N)$ correction compared with (9) for D (assuming we keep V and T_{Stop} the same). This is because we have an extra constraint generated by the requirement that T_{Stop} is an integral multiple of T, making the problem overconstrained if we wished D and D_D to be the same.

It is hard not to notice how much more complicated the above is compared with (1)-(9). It is always so with discrete systems — hence the strong desire to model systems in the continuous domain.

4. Train Stopping as a Control Problem, Continuous and Discrete

In this section, we summarise how the informal considerations of the last section fit into a standard control engineering framework, in both the continuous and discrete domains.

4.1. The Continuous Control Problem

At the introductory level, control theory is usually developed in the frequency domain [Oga08, DB10, DTB97, DH95], because of the relative simplicity and perspicuity of the design techniques in that domain. However, for results sufficiently rigorous to interface to conventional formal techniques (where there is no routinely available notion of frequency domain), we need to go to the state space formulation favoured by more mathematically precise treatments of control [Ahm06, Son98, CLSW97, Cla87, Bar75].

In the state space picture, the system consists of a number of state variables, and their evolution is governed by a corresponding number of first order ordinary differential equations (FO ODEs). In fact, many ODE texts routinely consider control problems [Wal98, Chi06]. State variables and differential equations mirror the states and transition systems of model based refinement formalisms sufficiently closely that a connection can be made between them.

Regarding our specific example, most of what we need has already been written down in the previous section, so we just need to repackage it in a suitable way.

To use the first order framework in our example, the state has to consist of both the position $x(t)$ and the velocity $v(t)$. So we get the state vector

$$\boldsymbol{x}(t) \;=\; \begin{bmatrix} x(t) \\ v(t) \end{bmatrix} \tag{23}$$

The dynamics of the system is captured in the equation

$$\dot{\boldsymbol{x}}(t) \;=\; \begin{bmatrix} \dot{x}(t) \\ \dot{v}(t) \end{bmatrix} \;=\; \boldsymbol{f}(\dot{x}(t), u(t)) \;=\; \begin{bmatrix} v(t) \\ u(t) \end{bmatrix} \tag{24}$$

where

$$u(t) \;=\; -at \tag{25}$$

is the external control signal in the terminology of the Appendix.

We also have the initial condition

$$\boldsymbol{x}(0) \;=\; \begin{bmatrix} 0 \\ V \end{bmatrix} \tag{26}$$

The solution to the control problem then proceeds to yield the data that we already described in Section 3.1.

4.2. The Discrete Control Problem

To genuinely implement a continuous control model, such as the one described above, requires analogue apparatus. In the highly digitized world of today, hardly any such systems are built. Instead, continuous control designs are discretized, and it is the corresponding digital control systems that are implemented.

The digital approach to control has many parallels with the continuous case. Particularly in the frequency domain, many techniques are very similar to their continuous counterparts, the main difference being the use of the *z*-transform rather than the Laplace transform. The state based picture too boasts many parallels, with first order difference equations replacing first order differential equations [FV09, FPW96, Par96, Kuo92].

In the control literature, one finds many ways of discretizing continuous designs (see *loc. cit.*). While these typically yield perfectly good results in practice, the evaluation of the relationship between the resulting discrete control system and the original continuous design is often based on *ad hoc* engineering rules of thumb — these rules of thumb are themselves typically based on frequency domain considerations. Such criteria fall far short of the kind of precision and detail needed for a good fit with the kind of model based formal development techniques of interest in this paper. As a consequence, when model based formal development techniques are used to support the implementation of the discrete control system counterpart of some continuous design, the formal modeling inevitably starts already in the discrete domain. Obviously this yields a weaker kind of formal support for the process than if the formal modeling had started earlier, at the continuous design stage, and was integrated into all the subsequent design steps, including the change from continuous to discrete modeling.

Despite the fact that in digital control systems the control is recomputed at regularly spaced points of time rather than continuously, the control is still exercised in the real continuous world, using devices that hold the outputs until the next sampling instant. As noted, this leads to relationships between the real continuous world behaviour of the discretized control, the discretized control itself, and the continuous control that it is supposed to emulate, that are rather complex in general. To avoid the heuristic nature of typical engineering approaches to this issue, and to retain the precision needed to integrate solidly with model based formal development techniques, there is no option but to analyze the discretized model in the continuous world. Thus the problem becomes a variant of the continuous control problem we saw before, with the characteristics of the zero order hold built directly into the model.

When we recast the discretized model of Section 3.2 as an initial value first order system, we find a structure very similar to what we had before. The state vector is

$$\boldsymbol{x}_D(t) \;=\; \begin{bmatrix} x_D(t) \\ v_D(t) \end{bmatrix} \tag{27}$$

and the dynamics of the system is captured in the equation

$$\dot{\boldsymbol{x}}_D(t) \;=\; \begin{bmatrix} \dot{x}_D(t) \\ \dot{v}_D(t) \end{bmatrix} \;=\; \boldsymbol{f}(\dot{x}_D(t), u_D(t)) \;=\; \begin{bmatrix} v_D(t) \\ u_D(t) \end{bmatrix} \tag{28}$$

where

$$u_D(t) \;=\; \dot{v}_D(t) \;=\; -ka_DT \tag{29}$$

as given by (12), is the external control signal.

We also have the initial condition

$$\boldsymbol{x}_D(0) \;=\; \begin{bmatrix} 0 \\ V \end{bmatrix} \tag{30}$$

Note in particular, that the control functional $\boldsymbol{f}$ is the same in both the continuous and discretized cases.

5. Rigorous Bounds on the Continuous and Discrete Systems

We already noted that engineering discretization schemes are usually based on tried and tested rules of thumb. While these might be perfectly good enough in practical applications, there is, unfortunately, no universally accepted way of translating "perfectly good enough in practice" into a judgement in a formal model based framework. Moreover, the situation is compounded by the fact that in the majority of real world engineering situations, there is a good deal less knowledge about the detailed behaviour of the system than we have for our case study.

Therefore, one of the wider objectives of this paper is to illustrate how to make a judgement about the discretization of a control problem, which firstly: is sufficiently rigorous and precise that it can be incorporated into a formalized development method, and secondly: is sufficiently generic that the principles of the technique used can be applied to a wide variety of control situations. The additional fact, that in the case of our specific case study we can solve both the continuous and discretized versions exactly, ensures that in this case, we can evaluate the quality of the generic judgement reliably. When the difference between the continuous and discretized dynamics turns out to be of a lower order in the discretization parameter than the leading behaviour (as it does in our case), then that behaviour proves to be typical for a wide class of similar systems, and the comparison with the exact calculations becomes trustworthy as an indication of the quality of the generic approach. We return to this point later.

5.1. A Generic Bound

There are many different kinds of control problem, and many different things one can say about them. In this paper we focus on one specific result concerning generic control problems, which can be applied to quantify the continuous to discretized modeling transformation. This result states a rigorous quantitative estimate of the difference between the behaviours of two instances of a control problem which differ solely in the input control signal.

The mathematical facts upon which this quantitative estimate are based are developed in the Appendix. Using material from the non-smooth analysis and mathematical control theory literatures, a result is proved that relates the maximum difference between the final states attained when a given continuous control system is subjected to two different input control signals over the same time interval, having been started in the same initial state. We apply this in the case that one of the input control signals is the control signal for our original continuous control problem, and the other input control signal is the control signal for a zero order hold discretized approximation to it. We note that the continuous and discretized versions of our case study ((23)-(26) and (27)-(30) respectively), with initial states (26) and (30), and acting over the time interval from 0 to T_{Stop}, characterize just such a scenario, since (24) and (26) differ from (28) and (30) only in the use of u_D rather than u among the independent variables.

The main result of the Appendix is formula (146). When we substitute our state variables $\boldsymbol{x}$ and $\boldsymbol{x}_D$, and our input control signals u and u_D into (146), it states that there is a constant $K2$ such that

$$||\boldsymbol{x}^u - \boldsymbol{x}_D^{u_D}|| \;\leq\; K2||u - u_D||_2 \tag{31}$$

We now flesh out what this means for our little case study.

In (31), $||\boldsymbol{x}^u - \boldsymbol{x}_D^{u_D}||$ is the $\mathcal{L}^\infty$ norm of $\boldsymbol{x}^u - \boldsymbol{x}_D^{u_D}$, or, in plain English, the maximum value over the interval $[0 \ldots T_{Stop}]$ attained by the difference between continuous and discrete values of any state component; or, in plainer English still, the greater of: the maximum value over the interval $[0 \ldots T_{Stop}]$ of $|x(t) - x_D(t)|$; and the maximum value over the interval $[0 \ldots T_{Stop}]$ of $|v(t) - v_D(t)|$. In other words, it is the greatest difference between corresponding components of the two states attained anywhere in the interval of interest.[6]

Likewise, $||u - u_D||_2$ is the $\mathcal{L}^2$ norm of $u - u_D$, or, in plain English, the root integrated square difference between u and u_D, calculated over the interval $[0 \ldots T_{Stop}]$ by a formula like (151). Finally, $K2$ is a constant.

We next consider the values of the quantities on the right hand side of (31) in order to obtain a bound for the value of the left hand side.

Referring to (31), the Appendix furnishes an explicit value for the constant $K2$, namely

$$K2 \;=\; e^{K_f}||k_u||_2 \tag{32}$$

In (32) K_f is $k_f\, T_{Stop}$, where k_f is the $\mathcal{L}^\infty$ norm of $\boldsymbol{f}_{\boldsymbol{x}}$, or, the absolute maximum value (over the interval $[0 \ldots T_{Stop}]$) of the Lipschitz constant governing the variation of the control law $\boldsymbol{f}$ with respect to the state. In our case, the Lipschitz

[6] Here, and in the rest of the paper, we compare state components such as position and velocity "on the nose" for simplicity. In a more realistic calculation, there would be scale factors and dimensionality considerations which would affect the details. We discuss these in Section 14.3.

constant reduces to the maximum value of the partial derivative of (any (linear combination of) component(s) of) the control law $\boldsymbol{f}$ with respect to (any (linear combination of) component(s) of) the state. In our application, the form of the control law is

$$\boldsymbol{f}(v(t),u(t)) \;=\; \begin{bmatrix} v(t) \\ u(t) \end{bmatrix} \tag{33}$$

and it is clear that there is only one component of $\boldsymbol{f}$ with a non-zero partial derivative with respect to either x or v, namely the first

$$\frac{\partial \boldsymbol{f}_1}{\partial v} \;=\; 1 \tag{34}$$

With this, the first factor of (32) is just $e^{T_{Stop}}$.

Regarding the second factor, $||k_u||_2$ is the root integrated square value of the Lipschitz constant governing the variation of the control law with respect to the input control signal. In our case, the Lipschitz constant reduces to the maximum value of the partial derivative of (any component of) the control law $\boldsymbol{f}$ with respect to the input control signal u. Again there is only one component of $\boldsymbol{f}$ with a non-zero partial derivative with respect to u, namely the second

$$\frac{\partial \boldsymbol{f}_2}{\partial u} \;=\; 1 \tag{35}$$

so the root integrated square reduces to $\sqrt{T_{Stop}}$. So we get

$$K2 \;=\; e^{T_{Stop}}\sqrt{T_{Stop}} \tag{36}$$

Turning to the second factor on the right hand side of (31), $||u-u_D||_2$, we recall that we know explicitly what u and u_D are from our earlier calculations. From (25) we know that

$$u(t) \;=\; -at \tag{37}$$

and from (29) we know that

$$u_D(t) \;=\; -ka_DT \tag{38}$$

where, from (7)

$$a \;=\; \frac{2V}{T_{Stop}^2} \tag{39}$$

and from (16)

$$a_D \;=\; \frac{2V}{T_{Stop}^2(1+1/N)} \tag{40}$$

Now (37) shows that $u(t)$ decreases linearly, and (38) shows that $u_D(t)$ is a staircase function, decreasing in equal sized steps in the vicinity of $u(t)$.

Let us write $\overleftarrow{q(t)}$ to denote the right limit of a function $q(\tau)$ at t; i.e. the limit as τ approaches t from values $\tau > t$ (assuming such a limit exists). Similarly, let $\overrightarrow{q(t)}$ denote the left limit. Then it is clear from (37) and (38) that $\overleftarrow{u(0)} = 0$ and $\overleftarrow{u_D(0)} = -a_DT$, so that $\overleftarrow{u(0)} - \overleftarrow{u_D(0)} = a_DT$. It is also clear from (37) and (38) that $\overrightarrow{u(T_{Stop})} = -aT_{Stop}$ and $\overrightarrow{u_D(T_{Stop})} = -Na_DT = -a_DT_{Stop}$, so that $\overrightarrow{u(T_{Stop})} - \overrightarrow{u_D(T_{Stop})} = (a_D-a)T_{Stop} = a_DT_{Stop}[1-(1+1/N)] = -a_DT_{Stop}/N = -a_DT$. Since the staircase has equal sized steps, it is evidently the case that the staircase $u_D(t)$ ranges around $u(t)$ within a bound a_DT.

$$|u(t)-u_D(t)| \;\le\; a_DT \tag{41}$$

This furnishes a suitable overestimate for the root integrated square difference between $u(t)$ and $u_D(t)$ as follows

$$||u-u_D||_2 \;\le\; \sqrt{\int_{t=0}^{T_{Stop}} [a_DT]^2 dt} \;=\; a_DT\sqrt{T_{Stop}} \tag{42}$$

Substituting all the values we have obtained into (31), we get

$$||\boldsymbol{x}^u - \boldsymbol{x}_D^{u_D}|| \;\leq\; e^{T_{Stop}}\sqrt{T_{Stop}} \times a_D T \sqrt{T_{Stop}} \;=\; e^{T_{Stop}} a_D T T_{Stop} \tag{43}$$

We see that despite the potential for the deviation between $u(t)$ and $u_D(T)$ to grow exponentially with the size of the time interval, a possibility exacerbated by our rather crude bound (42), it is always possible to reduce it by an arbitrary amount by making the discretization fine enough.[7]

5.2. Corroboration: Comparing Generic Bounds with Exact Results

In our case study, as well as the generic facts discussed and instantiated in the preceding section, we also have the fact that the (deliberately intended) extreme simplicity of our control models, in both continuous and discretized domains, allows for an exact solution. This gives us additional and independent confirmation of the approach we are advocating in this paper.

Both the continuous and discrete models "run" for the same amount of time, T_{Stop}, so we can compare the difference in the states reached by the continuous and discrete models after the elapse of T_{Stop}. Furthermore, we can make that comparison by using both the exact values following from the exact solutions, and also the more approximate estimates yielded by the generic calculations.

The states themselves consist of two components, the displacements and the velocities.

Regarding the velocities, both models come to an exact standstill after T_{Stop}. Consequently both $v(T_{Stop})$ and $v_D(T_{Stop})$ are zero, so that $|v(T_{Stop}) - v_D(T_{Stop})| = 0$, and any positive upper bound is bound to be sound. So (43), which gives the overestimate $e^{T_{Stop}} a_D T T_{Stop}$ for $|v(T_{Stop}) - v_D(T_{Stop})|$ is correct regarding the velocities, but in an unsurprising way.

Regarding the displacements, the quantization of T_{Stop} in the discrete case, leads to the continuous and discrete dynamics stopping at slightly different places, D and D_D respectively, which we calculated earlier. On the basis of (9) and (22), and using (40), we can calculate the exact difference:

$$\begin{aligned} |x(T_{Stop}) - x_D(T_{Stop})| \;&=\; \frac{2}{3}\frac{V T_{Stop}}{4N} \;=\; \frac{1}{2} a_D T_{Stop}^2 \left(1 + \frac{1}{N}\right) \frac{T_{Stop}}{6N} \\ &=\; \frac{1}{12} a_D T T_{Stop}^2 \left(1 + \frac{1}{N}\right) \end{aligned} \tag{44}$$

On the other hand, (43) again gives the estimate $e^{T_{Stop}} a_D T T_{Stop}$ for this quantity. Thus the exact value falls within the bounds of the estimate, as it should, if and only if (after cancelling the common factor $a_D T T_{Stop}$):

$$\frac{T_{Stop}}{12}\left(1 + \frac{1}{N}\right) \;\leq\; e^{T_{Stop}} \tag{45}$$

Since a linear function of T_{Stop} of slope less than 1 can never catch an exponential function of T_{Stop} with coefficient 1, (45) is obviously true, and we have our corroboration.

5.3. Corroboration as Engineering Evaluation

A comparison between exact results and a generic estimate that confirms that the exact results lie within the bounds permitted by the generic estimate is of course a good thing, and is very reassuring. But in a sense it does no more than to confirm that a portion of classical mathematics, carried out without errors, is consistent, and in that sense it is unsurprising.

A more useful byproduct of such a comparison between exact and generic results addresses engineering concerns. In general, genuine engineering applications will not exhibit the simplicity of our small example. Therefore there is no realistic possibility of deriving an exact solution as we did above. In genuine engineering situations therefore, we are left with more widely applicable generic techniques. However, such generic techniques yield only an approximation to the truth. For engineering purposes, we require some measure of the quality of the approximation involved, in order to be able to gauge the nature of the engineering compromise being made when the generic techniques are being relied

[7] Making the discretization finer increases N, and since $T_{Stop} = NT$, this decreases T, which in turn decreases the right hand side of (43).

on. For this, a comparison with a selection of reliably derived exact solutions is of inestimable value. A good selection of exact solutions will give a good idea of the character of the typical difference between an exact solution and a generic estimate, and this can be used to inform the wider engineering process.

In the case of our small example, if we keep the quantity $a_D T$ constant for the sake of argument, we see that the exact solutions to the continuous and discretized problems yield a difference between the respective displacements D and D_D which is quadratic in T_{Stop}, by (44). Under the same assumption that $a_D T$ is constant, the generic estimate (43) gives a value which is more than exponential in T_{Stop}.

Now, no one with an eye to the precision of engineering design will be too impressed by a general purpose estimate which is exponential, when it is known that the true value is quadratic. In this sense the estimate we derived is rather poor. However, the aim of this paper is not necessarily to derive the best possible generic estimate, but to illustrate how the various components of the the whole of the development process fit together, while keeping the size of all the constituent pieces reasonable. For this, the generic estimate that we have derived, being relatively simple in form, even if it is not outstanding quantitatively, is adequate.

Engineering systems in general and control systems in particular, come in many varieties, as do the mathematical results about such systems that can be brought to bear on their analysis. It is possible to do considerably better as regards the quality of the generic estimates that can be applied to such systems, but that comes at the price of derivations of increased complexity. We will pursue such improved techniques in other publications, the aim being to obtain results that scale and compose in a considerably better manner. Nevertheless, the value of the present treatment is to show that whatever the details of the generic estimates obtained, the facts at its interface will be of the same character as those we deal with in the present paper. Therefore, once one has obtained such improved generic results, they can be inserted into the kind of development that we are illustrating here unproblematically.

6. Outline: A Control System Design Process

In this section we start the process of integrating the modeling given previously into a formal development methodology, by giving an overview. The detailed exposition of the various compoments of the process are given in the technical sections to follow. Although we envisage that the overall shape of the process that we describe would persist in any practical realisation of it, individual aspects could of course be addressed in different ways, according to the needs and pre-existing practices of a particular engineering environment. In that sense, what we describe is a proof of principle.

The start of any engineering solution is the identification of the requirements. Since we aim at the development of control *software*, we address the requirements engineering task from a perspective aimed primarily at the development of software solutions. And since we aim at the use of techniques that permit rigorous mathematical checking of their content throughout our process, we use a requirements engineering methodology with full formal backup. A good candidate for this is the KAOS technique [vL09, Let01], in which stylized natural language statements of *requirement goals* map smoothly into temporal logic statements, enabling formal analysis to be carried out from the beginning, enabling goal decomposition and refinement to be done in a rigorous way, and ultimately enabling operations to be extracted from the lowest level goal set — these being guaranteed to implement the identified goals.

Once we have suitable operations, these can be translated into a model based formal development formalism. Again there are many candidates for this, tending to differ from one another in rather specific technical details. Although most could be used for our task, we choose the ASM framework [BS03, Bör03]. There are three main reasons for our choice. Firstly it is because ASM has a scheduling policy for the application of ASM rules that fits nicely with the KAOS scheduling policy for KAOS operations. Secondly it is because ASM is formulated in a very flexible way, making the use of the kind of data types that can arise in continuous applications relatively unproblematic, at least in principle.[8] Thirdly it is because the refinement theory of ASM stresses a fairly general setup, in which an arbitrary number of steps at the abstract level of a refinement can be made to simulate an equally arbitrary number of steps at the concrete level — abstract paths are refined to concrete paths. In a discretization situation, where a continuous control activity is made to correspond to a large number of smaller steps, this built in flexibility of ASM regarding the numbers of steps that must correspond at the two levels being related, is one less thing to worry about.

Now, it is necessary to recall that both KAOS and ASM are designed for systems where transitions are discrete, with each current (non-final) system state in a system run having a well defined successor. Of course this does not fit well with the continuous behaviour we need to model, where there is a continuum of states and no state has a well defined successor state. In the relevant detailed sections we will deal with the modifications to both KAOS and ASM

[8] In fact we will not need to make great use of this capacity in our case study due to the extreme simplicity of the control problem we address.

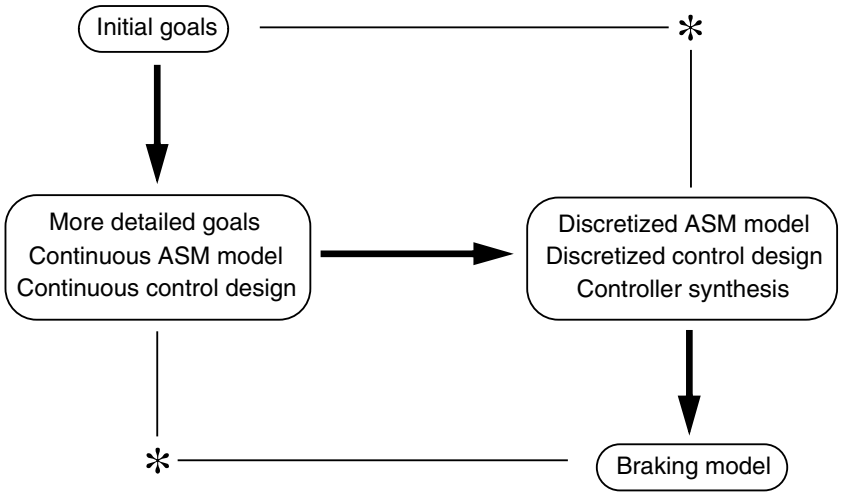

Fig. 1. An overview of the whole development, starting with the most abstract goals for train stopping, proceeding through explicit continuous and discretized deceleration models, including discrete level controller synthesis, and ending with the lowest level braking model. Vertical arrows are (perhaps sequences of) model based refinements, each underpinned by a theoretical framework offering robust guarantees. Horizontal arrows are retrenchments, underpinned by theory offering weaker guarantees, but suitable for relating models too different to be tied together by refinement.

that are needed, both for modeling in the continuous domain, and for defining suitable notions of refinement. Although we do not use full formal machinery for the modifications, we will pitch the discussion at a level of detail appropriate to the needs of the rest of the paper. For now, we take it for granted that it can be done.

Returning to our development, so far, the formal modeling and refinement has captured the development of the continuous control model down to the level where a continuous operational behaviour has been specified in ASM. At this point, we need to formalize the continuous to discretized modeling transformation. For this, the notions of refinement just mentioned prove too demanding. The need for the dynamics to re-establish a fixed retrieve relation that provides the sole relationship connecting the two models (this being the typical demand of refinement frameworks) is too restrictive for applications in which some appropriate notion of "abstract and concrete states being close enough" can grow as well as shrink. For a fairly generic approach to these more flexible modeling transformations we use retrenchment, whose principal proof obligation (PO) provides greater scope for accommodating issues relating to the two models that do not fit refinement, via the presence of additional relations in the PO.

The quantitative differences between continuous and discretized models discussed in the previous section can be accommodated in suitable retrenchments. We could design a retrenchment to formalize the exact results about the continuous and discretized models, but the technique it embodied would apply to just that example and nothing else. Instead, we concentrate on formalizing the generic approach. This, more generic approach, has the potential for being applied to a whole class of similar systems. The fact that these generic results are mathematically derived, (a fact supported by comparison with exact calculations, above) implies the provability of the corresponding retrenchment PO. In one sense, the demonstration of this formal route, starting from the continuous world and moving into the discretized world, is the main contribution of this paper.

The way that we construct our discretized model is intended to make the extraction of a corresponding conventional discrete event controller a relatively trivial task. So the next step in the development is the identification of a plant submodel (taking care of train variables), and of a controller submodel (taking care of control variables). Their composition is equivalent to the original discretized model. Once we have completed this, we can further refine the now discrete ASM controller model by conventional means.

Therefore, the discretized model arrived at earlier is refined to a relatively trivial model of train braking, this being purely to illustrate the point that refinement now proceeds unsurprisingly. The outcome of this refinement step constitutes the culmination of our development process. The whole process may be summarized in Fig. 1, which is an instance of the Tower Pattern [BJ, Jes05]. Refinements are drawn as vertical arrows while retrenchments are drawn as horizontal arrows.

The left hand side of the diagram depicts the continuous world, while the right hand side depicts the discrete world. The bold arrows trace our path from system model to system model. Thus our process starts at the top left

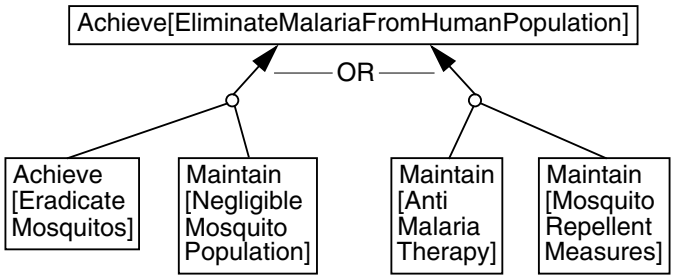

Fig. 2. A hypothetical one level KAOS goal refinement, decomposing a higher level goal into a disjunction of two possible ways of satisfying it by realising a pair of lower level goals.

hand corner, with the abstract requirements. These are then refined to the continuous ASM model. Then we retrench to the discretized ASM model, from which we extract the discrete event model. Finally, we refine further to the braking model.

As well as the models we discuss directly, there are other (nameless) models in the diagram, drawn as big asterisks. These are sure to exist by virtue of the Tower theorems [BJ, Jes05]. They enable the whole development route to be reworked in terms of purely continuous or purely discrete models, with retrenchments connecting the two sides. Of course not all the useful characteristics of models of one kind are present, or capable of being represented, in models of the other kind, which is precisely why the original development includes both kinds of model. This observation also explains why such a reworking, though possible, would not in fact be particularly beneficial in practice. Nevertheless, the schematic structure that Fig. 1 offers, is helpful in clarifying the conceptual framework into which this extended model building activity fits.

7. Formally Modeling Requirements

The objective of our design process is to start with the highest level requirements, and follow the development of the design right down to (or at least quite close to) implementation, in a completely rigorous manner. In this section, we focus on how we can model the high level requirements in the KAOS requirements engineering methodology [vL09, Let01]. The main virtue of KAOS is that all design goal expressions in KAOS, correspond in principle to formulae in a (specific) temporal logic [Koy92], thus formalizing the design from the earliest possible moment. In the next subsection we give an overview of traditional KAOS, intended for discrete event systems. In the following subsection, we extend this to accommodate continuous time and continuously varying behaviour.

7.1. Discrete Event KAOS

The KAOS approach to requirements engineering is via the concept of *goal*. Goals are desirable system properties which, as a result of the requirements engineering process, get refined to collections of subgoals. We give a brief illustration. In Fig. 2 there is a one step goal refinement of a strategy for eliminating malaria from some hypothetical Far Eastern location.[9] We see that a typical goal refinement step is the development of an additional layer in an AND/OR tree. Each disjunctive branch of the new layer of the tree aggregates a collection of subgoals that together can guarantee the parent goal. Equally, each of the alternative disjunctive branches provides a different strategy for satisfying the parent goal. In the KAOS philosophy, exploring different alternative strategies for fulfilling a goal is a vitally important ingredient of the process of identifying the best way to achieve the overall goals of the development. KAOS explicitly discourages the possibility of overcommitment to unwise choices early on in the requirements gathering process, so the contemplation of incompatible alternative strategies is explicitly encouraged. Still, that aspect is of less concern to us in this paper, so we will ignore these disjunctive possibilities in the remainder of the paper.

The goal refinement process continues until a level is reached at which the goals can be directly *operationalized*. A goal is regarded as operationalized if it can be made to correspond to a state changing operation of some state transition system. For this to make sense in the KAOS context, the state space has to consist of the cartesian product

[9] The authors hasten to add that ECNU, though blessed with mosquitos at certain times of the year, is not malarial.

of the domains of the state variables of the system (at that level of abstraction), and a major concern of the KAOS philosophy is to ensure that for every goal which is operationalized, we can identify a unique *agent* which is responsible for carrying out the state change operation, and that the agent in question actually has the capability of reading all the state variables needed to confirm the before-state of the state transition, and has the capability of writing all the state variables needed to enforce the after-state of the state transition. Because of the technical focus of this paper, these agent-centric concerns are somewhat in the background for most of the paper, although they resurface in another guise in Section 11.

During the KAOS requirements engineering process, the requirements engineer discusses prospective requirements with the client. Behind the scenes, the requirements that are formulated are captured formally using formulae of Linear Temporal Logic (LTL) in the style of [Koy92]. We look at this now.

We write $\mathbb{N}^\bullet$ to denote a proper or improper prefix of the naturals $\mathbb{N}$, i.e., depending on the context, $\mathbb{N}^\bullet$ is either $\langle 0 \ldots \mathbb{N}_{\mathrm{MAX}} \rangle$, or all of $\mathbb{N}$. In the former case, $\mathbb{N}_{\mathrm{MAX}}$ also depends on the context, and is always assumed to be big enough to enable all the relevant behaviour to be described, yet small enough that we do not have to explicitly describe a pointless infinite family of stuttering steps at the end of a finite behaviour.

A typical KAOS LTL formula is evaluated on total histories of the state variables (at the given level of abstraction). A history σ maps $\mathbb{N}^\bullet$ to valuations of the tuple of variables relevant at the given level of abstraction, $\sigma : \mathbb{N}^\bullet \times vs \to DS$, where *vs* is the tuple of variables, and *DS* is the cartesian product of their domains. Supplementing the history σ is a notion of "*now*", where $now \in \mathbb{N}^\bullet$, and the notation (σ, i) indicates that we are dealing with σ in a context where *now* is *i*. If *now* is unstated and is not clear from the context, it defaults to 0.

Given a history σ and a value for *now*, KAOS LTL formulae are interpreted as one would expect. Thus, variables are interpreted with respect to the valuation referred to by *now*. The propositional connectives $\neg, \wedge, \vee, \to, \leftrightarrow$ are interpreted in the usual way, as are the universal and existential quantifiers $\forall, \exists$ when ranging over (factors of the product space) *DS*. Quantification over the *now* index is handled by the temporal operators, and KAOS LTL features all the usual suspects, namely: *eventually*, *always*, *next*, *formerly*, *hitherto*, *previous*; $\Diamond, \Box, \bigcirc, \blacklozenge, \blacksquare, \bullet$. Their semantics are formally defined as follows.

$$(\sigma, i) \vDash \Diamond P \quad \textbf{iff} \quad (\sigma, j) \vDash P \quad \text{for some } j \geq i \tag{46}$$

$$(\sigma, i) \vDash \Box P \quad \textbf{iff} \quad (\sigma, j) \vDash P \quad \text{for all } j \geq i \tag{47}$$

$$(\sigma, i) \vDash \bigcirc P \quad \textbf{iff} \quad (\sigma, i+1) \vDash P \quad \text{and } i < \mathbb{N}_{\mathrm{MAX}} \text{ (if applicable)} \tag{48}$$

$$(\sigma, i) \vDash \blacklozenge P \quad \textbf{iff} \quad (\sigma, j) \vDash P \quad \text{for some } j \leq i \tag{49}$$

$$(\sigma, i) \vDash \blacksquare P \quad \textbf{iff} \quad (\sigma, j) \vDash P \quad \text{for all } j \leq i \tag{50}$$

$$(\sigma, i) \vDash \bullet P \quad \textbf{iff} \quad (\sigma, i-1) \vDash P \quad \text{and } i > 0 \tag{51}$$

Beyond the basic temporal primitives, KAOS supports two kinds of generalization. Firstly there are the binary temporal operators: *until*, *unless-later*, *since*, *unless-earlier*; U, W, S, B, with semantics given as follows.

$$(\sigma, i) \vDash P \,\mathsf{U}\, Q \quad \textbf{iff} \quad (\sigma, j) \vDash Q \text{ for some } j \geq i \quad \textbf{and} \quad (\sigma, k) \vDash P \text{ for all } k : i \leq k < j \tag{52}$$

$$(\sigma, i) \vDash P \,\mathsf{W}\, Q \quad \textbf{iff} \quad (\sigma, i) \vDash \Box P \quad \textbf{or} \quad (\sigma, i) \vDash P \,\mathsf{U}\, Q \tag{53}$$

$$(\sigma, i) \vDash P \,\mathsf{S}\, Q \quad \textbf{iff} \quad (\sigma, j) \vDash Q \text{ for some } j \leq i \quad \textbf{and} \quad (\sigma, k) \vDash P \text{ for all } k : j < k \leq i \tag{54}$$

$$(\sigma, i) \vDash P \,\mathsf{B}\, Q \quad \textbf{iff} \quad (\sigma, i) \vDash \blacksquare P \quad \textbf{or} \quad (\sigma, i) \vDash P \,\mathsf{S}\, Q \tag{55}$$

Secondly, there are (both relative and absolute) deadline-enhanced versions of the quantified primitives in (46), (47), (49), (50) and (52)-(55). For these, we need to discuss the relationship between real time and the $\mathbb{N}^\bullet$-valued history index introduced above.

We assume that there is a monotonically increasing function $\mathsf{Time} : \mathbb{N}^\bullet \to RealTime$ which maps each index to a physical time. For brevity, in future we write:

$$t_i = \mathsf{Time}(i) \tag{56}$$

To avoid "Zeno" situations, we assume that for each system model that we develop, there is an appropriate system-specific global lower bound δ_{Zeno} for the length of the *RealTime* interval that a pair of successive indexes can correspond to:

$$\forall i \in \mathbb{N}^\bullet \bullet t_{i+1} - t_i \geq \delta_{\mathsf{Zeno}} \tag{57}$$

With this, the relative primitives: *P not later than d from now*, *P constantly till d from now*, *P no earlier than d before now*, *P constantly since d before now*, *P constantly till Q no later than d from now*, *P since Q no earlier than d*

before now; written $P \Diamond_{\leq d}$, $P \Box_{\leq d}$, $P \blacklozenge_{\leq d}$, $P \blacksquare_{\leq d}$, $P \mathsf{U}_{\leq d}\ Q$, $P \mathsf{S}_{\leq d}\ Q$ respectively, are given by adding the clause $|t_i - t_j| \leq d$ (under the relevant quantifier) to the right hand sides of (46), (47), (49), (50), (52), (54). Regarding (53) and (55), we just replace the occurrences of the temporal primitives in their definitions by their deadline-enhanced versions.

As for the absolute deadline-enhanced versions, the subscript $\leq d$ on the temporal primitives is replaced by $\leq T$, where T is a reference to an absolute time, and the $|t_i - t_j| \leq d$ in their definitions is replaced by $t_i \leq T$ for those cases in which $j \geq i$ occurs, and by $T \leq t_i$ for those cases in which $j \leq i$ occurs.

KAOS LTL also introduces the "strong" versions of the propositional connectives $P \rightarrow Q$ and $P \leftrightarrow Q$, written $\Rightarrow$ and $\Leftrightarrow$, which are universally quantified over time:

$$P \Rightarrow Q \ \equiv\ \Box(P \rightarrow Q) \qquad \text{and} \qquad P \Leftrightarrow Q \ \equiv\ \Box(P \leftrightarrow Q) \tag{58}$$

Turning now from the basic language, to KAOS goals themselves, goals are normally constructed to conform to specific *patterns* of temporal behaviour. KAOS stresses the Achieve, Cease, Maintain, Avoid patterns, which are typically given by expressions such as:

Achieve:	$P \Rightarrow \bigcirc Q$	$P \Rightarrow \Diamond Q$	$P \Rightarrow \Diamond_{\leq d} Q$			(59)
Cease:	$P \Rightarrow \bigcirc \neg Q$	$P \Rightarrow \Diamond \neg Q$	$P \Rightarrow \Diamond_{\leq d} \neg Q$			(60)
Maintain:	$P \Rightarrow Q$	$P \Rightarrow \Box Q$	$P \Rightarrow \Box_{\leq d} Q$	$P \Rightarrow Q \mathsf{W} R$	$P \Rightarrow Q \mathsf{W}_{\leq d} R$	(61)
Avoid:	$P \Rightarrow \neg Q$	$P \Rightarrow \Box \neg Q$	$P \Rightarrow \Box_{\leq d} Q$	$P \Rightarrow \neg Q \mathsf{W} R$	$P \Rightarrow \neg Q \mathsf{W}_{\leq d} R$	(62)

Goals can also be further categorized as e.g. *Satisfaction*, *Safety*, *Security*, *Information*, *Accuracy* goals, etc.

Goal refinement is formally defined as follows. A set of subgoals $G_1 \ldots G_n$ refines a goal G in the context of a domain theory *Dom* (intended with a particular application in mind) iff the following hold:

$$G_1 \ldots G_n, Dom \vDash G \qquad \text{(completeness)} \tag{63}$$

$$G_1 \ldots G_n, Dom \nvDash \mathsf{false} \qquad \text{(consistency)} \tag{64}$$

$$\forall i \bullet (\textstyle\bigwedge_{j \neq i} G_j, Dom \nvDash G) \qquad \text{(minimality)} \tag{65}$$

The completeness criterion (63) states that $G_1 \ldots G_n$ and *Dom* are sufficient to establish the parent goal G. The consistency criterion (64) demands that $G_1 \ldots G_n, Dom$ are not contradictory. And the minimality criterion (65) ensures that none of $G_1 \ldots G_n, Dom$ is superfluous.

Ultimately, the goals have to be *operationalized*, i.e. turned into collections of operations that can be performed by individual (software or environment) agents, acting on the variables relevant to the operationalized level of abstraction. As noted earlier, the customer-focused side of the KAOS methodology emphasises the importance of identifying agents (for the application-to-be), such that every operation is the unique responsibility of a well defined and unique agent, which has the ability to monitor all the variables involved in the operation, and which also has the exclusive responsibility for controlling the variables updated by the operation.

The technical side of a KAOS operation is defined using a number of predicates on states: the *domain preconditions* DomPre, the *domain postconditions* DomPost, the *required preconditions RPr* $\in$ ReqPre, the *required triggers RTr* $\in$ ReqTrig, and the *required postconditions RPo* $\in$ ReqPost. These predicates contribute to the semantics of an operation *Op* in the following manner.

Since each operation *Op* is essentially a description of a set of state transitions similar to those appearing in any typical model based refinement formalism, the theory of operationalization amounts to establishing the connection between that view and the temporal logic view of goals. We review the essential facts [vL09, Let01].

An operation *Op* is given a temporal semantics via the formula $[|\,Op\,|]$ in (66) below. This formula just describes the raw updates to state variables accomplished by the operation at two adjacent ticks via DomPre and DomPost, all without concern for *when* the operation occurs. The remaining predicates, *RPr*, *RTr*, *RPo*, are given a temporal semantics via the formulae $[|\,RPr\,|]$, $[|\,RTr\,|]$, $[|\,RPo\,|]$, defined in (67)-(69), in which $(\forall *)$ indicates quantification over all free state variables. (N. B. For (66)-(69), we favour the slightly simpler formulation in [vL09] over that in [Let01].)

$$[|\,Op\,|] \ \equiv\ \mathrm{DomPre}(Op) \wedge \bigcirc \mathrm{DomPost}(Op) \tag{66}$$

$$\textbf{If}\ \ RPr \in \mathrm{ReqPre}(Op)\ \ \textbf{Then}\ \ [|\,RPr\,|] \ \equiv\ (\forall *)\ [|\,Op\,|] \Rightarrow RPr \tag{67}$$

$$\textbf{If}\ \ RTr \in \mathrm{ReqTrig}(Op)\ \ \textbf{Then}\ \ [|\,RTr\,|] \ \equiv\ (\forall *)\ \mathrm{DomPre}(Op) \wedge RTr \Rightarrow [|\,Op\,|] \tag{68}$$

$$\textbf{If}\ \ RPo \in \mathrm{ReqPost}(Op)\ \ \textbf{Then}\ \ [|\,RPo\,|] \ \equiv\ (\forall *)\ [|\,Op\,|] \Rightarrow \bigcirc RPo \tag{69}$$

The meaning of (67)-(69) can be described in the following terms.

The required preconditions $RPr \in \text{ReqPre}(Op)$, act as guards. In other words, the operation cannot take place unless all its guard are true in the before-state. So (67) defines $[\![RPr]\!]$ to be the LTL formula that says that it is always ($\Rightarrow$) the case that $[\![Op]\!]$ implies the truth of every $RPr \in \text{ReqPre}(Op)$.

The required triggers $RTr \in \text{ReqTrig}(Op)$ strengthen the required preconditions by forcing execution of the operation. As soon as any required trigger holds, the operation must execute. So (68) defines $[\![RTr]\!]$ to be the LTL formula that says that it is always the case that the truth of any $RTr \in \text{ReqTrig}(Op)$ implies $[\![Op]\!]$.

Finally, the required postconditions $RPo \in \text{ReqPost}(Op)$ take care of any other conditions that must hold for the operation which are not already dealt with earlier. So (69) defines $[\![RPo]\!]$ to be the LTL formula that says that it is always the case that $[\![Op]\!]$ implies the truth of every $RPo \in \text{ReqPost}(Op)$ in the after-state.

A history σ contains only valid occurrences of operation Op iff all occurrences of Op satisfy $[\![Op]\!]$, and furthermore, $[\![RPr]\!]$, $[\![RTr]\!]$, $[\![RPo]\!]$ all hold on σ.

A collection of operations $Op_1 \ldots Op_n$ operationalize a goal G iff the following criteria hold:

$$[\![Op_1]\!] \ldots [\![Op_n]\!] \vDash G \qquad \text{(completeness)} \tag{70}$$

$$[\![Op_1]\!] \ldots [\![Op_n]\!] \nvDash \mathsf{false} \qquad \text{(consistency)} \tag{71}$$

$$G \vDash [\![Op_1]\!] \ldots [\![Op_n]\!] \| \qquad \text{(minimality)} \tag{72}$$

The similarities and differences between (63)-(65) and (70)-(72) are instructive. There is no domain theory in (70)-(72), since at that level, the environment is itself assumed to be modeled by operations and invariants at an appropriate level of detail. Beyond that, (70)-(71) closely parallels (63)-(64). However (72) differs from (65) because, while minimality for goal refinement merely requires that the refinement does not introduce superfluous goals, minimality for operationalization requires that the behaviours of the goal are sufficient to infer the capabilities of the operations. Thus while a goal may be a *proper under*specification of its refinement — in keeping with the desire to allow the smooth introduction of additional design detail during refinement, an operationalization of a goal must be *equivalent* (in temporal terms) to the goal that it operationalizes, since all required design desiderata should have been catered for in the goal graph, and the job of operationalization is just to implement what has been made visible there, and not to sneak in further design by the back door.[10]

Once the goals have reached a level of detail at which operationalization is possible, they can be translated into operations in a model based refinement framework. As mentioned earlier, although more or less any such framework will do, in this paper we will use the ASM framework [BS03, Bör03], following [Ban10], since it offers several features that are particularly convenient for us. Among these is the scheduling policy for ASM operations, which melds closely with the scheduling policy for KAOS operations. Stated informally, the KAOS policy states: "all operations with a true trigger execute in the next step", whereas the ASM policy states: "all rules with a true guard execute in the next step" — this alignment between scheduling policies enables a simple 1-1 translation to be set up from KAOS operations to ASM operations which is explored in [Ban10].

This synergy between KAOS and ASM operation scheduling is to be contrasted with the situation for most other model based refinement frameworks. The scheduling policy for these can typically be stated as: "exactly one operation (from amongst all the currently enabled ones) executes in the next step". Such a policy, although not immediately aligned with the KAOS policy, may yet be reconciled with it at the cost of precompiling the scheduler into the bodies of operations (not to mention creating combined operations when necessary). Unfortunately this has to be done on an application by application basis. While perfectly doable (see e.g. [MGL08]), it would be an unnecessary distraction for us.

A further advantage of using ASM is the flexibility of using the data types it offers, which can be very general and abstract. This is a genuine boon in the context of continuous time phenomena, where the typical discrete data types offered by most refinement frameworks would prove too restrictive unless the type framework were suitably augmented. Yet another advantage of ASM is the fact that ASM refinement stresses the refinement of *paths* in the abstract model to *paths* in the concrete model. This will prove very useful to us — we will go into the details later.

With the above understood, and accepting that we will discuss ASM more extensively in Section 10, a straightforward translation of a generic KAOS operation Op into its corresponding ASM rule Op reads as in (73). ASM rules can be understood as guarded commands, and at each transition, all rules with true guards must fire (and must of course prescribe a consistent set of updates, or else the execution aborts). Conventionally, ASM rules have parameters that distinguish different static instantiations. For later convenience, we will extend this by singling out the pure input variables **in** *is*, and pure output variables **out** *os* of the rule, and placing them in the rule signature too. The guard is

[10] We see in this policy the stress (in KAOS) on modeling everything via the goals, whereas purely technically, we could imagine a more flexible interplay between modeling in the goal and operational spheres.

contained in the **if** clause, and when it is true, the update to the variables *vs* and outputs *os* is performed using the after-values vs' and os' nominated in the **choose** clause and specified in the **with** clause.

$$
\begin{array}{l}
Op(\textbf{in}\ is, \textbf{out}\ os)\ = \\
\textbf{if}\ \ \mathrm{DomPre}(Op)(vs,is)\ \ \textbf{and}\ \bigwedge_{RPr\in \mathrm{ReqPre}(Op)} RPr(vs,is)\ \ \textbf{and}\ \bigvee_{RTr\in \mathrm{ReqTrig}(Op)} RTr(vs,is) \\
\textbf{then} \\
\quad \textbf{choose}\ \ vs', os' \\
\quad \textbf{with}\ \ \mathrm{DomPost}(Op)(vs',os')\ \ \textbf{and}\ \bigwedge_{RPo\in \mathrm{ReqPost}(Op)} RPo(vs',vs,is,os') \\
\quad \textbf{do}\ \ vs\ :=\ vs', \\
\qquad os\ :=\ os'
\end{array}
\tag{73}
$$

7.2. Extending Discrete Event KAOS for Continuous Time

Thus far, the behaviours describable using KAOS, though embedded in real time via the *Time* function, are nevertheless discrete state behaviours. Thus, state histories, described hitherto as sequences, can be extended to functions of time which are piecewise constant. Similarly, once a state transition has taken place (which we assume to be an instantaneous process), the state variables hold their values until the next state transition. All this applies equally to the ASM operationalisation. We now introduce machinery that deals with state functions of time explicitly, and we extend the picture to also include behaviours involving continuously varying quantities.

Henceforth, all histories will be regarded as functions of real time, which is modeled using a segment of the real numbers. Within real time, we continue to identify the intervals defined by the points $\{t_i = \mathsf{Time}(i) \mid i \in \mathbb{N}^\bullet\}$, and we regard real time as being split up into the sequence of left-closed right-open intervals $\langle [t_0 \ldots t_1), [t_1 \ldots t_2), \ldots \rangle$, where, if $\mathbb{N}^\bullet = \mathbb{N}$ then the sequence extends indefinitely, and if $\mathbb{N}_{\mathrm{MAX}}$ is finite then the sequence ends with the semi-infinite interval $[t_{\mathbb{N}_{\mathrm{MAX}}} \ldots \infty)$.

We first distinguish between, on the one hand, variables whose behaviours over time are the piecewise constant extensions of existing $\mathbb{N}^\bullet$-indexed variables, which we call **mode variables**, and on the other hand, variables whose behaviours over time are more general functions of time, which we call **pliant variables**.

The inclusion of variables that vary continuously, but without other restrictions, admits behaviours that are pathological in all sorts of ways. We wish to avoid this, so in addition to the Zeno restriction (57), we will assume that:

- For every variable x, and for every time t, the left and right limits (i.e. the limits (from earlier times) of $x(t-\delta)$ and (from later times) of $x(t+\delta)$ respectively, with δ positive and with δ tending to 0), written $\overrightarrow{x(t)}$ and $\overleftarrow{x(t)}$ respectively, are well defined, and for every t, the value of x at time t, is equal to the right limit, i.e. $(\forall t \bullet x(t) = \overleftarrow{x(t)})$.[11] (74)
- The behaviour of every variable x in the interval $[t_i \ldots t_{i+1})$ is given by the solution of an initial value problem of ordinary differential equations (ODEs) that is well posed over the interval, where the right hand side φ of the ODE satisfies: (a) regarding the variation of φ with respect to variables, there are Lipschitz constants for φ which are uniformly bounded during the interval; (b) regarding the variation of φ with respect to time, φ is measurable during the interval. (Such problems can be handled by the techniques used in the Appendix, and yield solutions satisfying the continuity criteria in (74)). (75)

We observe that for a mode variable, its value during an interval $[t_i \ldots t_{i+1})$ is a solution to a well posed initial value problem in a trivial way (the time derivative of the variable being zero throughout). Moreover, the behaviour of a pliant variable x over the whole of the period of time of interest, breaks up into a sequence of abutting left-closed right-open time intervals, on each of which x is absolutely continuous, and which together make up a total function from a segment of *RealTime* to the type of x. And although $x(t_i) = \overleftarrow{x(t_i)}$ always holds, it will not be the case that $x(t_{i+1})$ equals $\overrightarrow{x(t_{i+1})}$ if x is discontinuous at t_{i+1}.

The earlier treatment of discrete event KAOS can now be integrated more formally with the continuous time framework as follows. The variables for which we have valuations at the individual indexes $i \in \mathbb{N}^\bullet$, map to mode variables whose values are constant over the intervals $[t_i \ldots t_{i+1})$. The state transitions between the before-values $x(i)$

[11] Right limits are always well defined since time is unbounded from above, but time is bounded from below, by $\mathsf{Time}(0)$, so for every variable x, we define $\overrightarrow{x(t_0)}$ to be equal to $\overleftarrow{x(t_0)}$.

and after-values $x(i+1)$ (in the earlier picture), map to transitions from the left limits $\overrightarrow{x(t_{i+1})}$ to the right limits $\overleftarrow{x(t_{i+1})}$ of the associated mode variable's value, as time moves through t_{i+1}, taking place in an instantaneous manner at t_{i+1}. We call these instantaneous mode variable transitions **mode transitions**.

At t_0 we have the initialisation, which assigns the intial values of all the system model's variables, and which we regard as a mode transition with true guard for technical simplicity — the initial values of the mode variables persist till t_1. Likewise, if $\mathbb{N}_{\mathrm{MAX}}$ is finite and t_i is $t_{\mathbb{N}_{\mathrm{MAX}}}$, then t_{i+1} is ∞ and there is no transition through t_{i+1} — then the final values of the mode variables persist from $t_{\mathbb{N}_{\mathrm{MAX}}}$ to ∞.

In this manner, the operationalization given before can be recast as follows, where the interpretation of the various formulae occurring in (76)-(79) is to be understood as referring to the specific times t_i for $i > 0$. (What this means in more detail is, for instance, that in (76), if $\overleftarrow{\bigcirc \mathrm{DomPost}(Op)}$ refers to the right limit of $\bigcirc \mathrm{DomPost}(Op)$ at t_{i+1}, then $\overrightarrow{\mathrm{DomPre}(Op)}$ refers to the left limit of $\mathrm{DomPre}(Op)$ at t_{i+1}, which is actually the value that $\mathrm{DomPre}(Op)$ acquires at t_i. And so on.)

$$[\![Op]\!] \;\equiv\; \overrightarrow{\mathrm{DomPre}(Op)} \wedge \overleftarrow{\bigcirc \mathrm{DomPost}(Op)} \tag{76}$$

$$\textbf{If}\;\; RPr \in \mathrm{ReqPre}(Op)\;\; \textbf{Then}\;\; [\![RPr]\!] \;\equiv\; (\forall *)\; [\![Op]\!] \Rightarrow \overrightarrow{RPr} \tag{77}$$

$$\textbf{If}\;\; RTr \in \mathrm{ReqTrig}(Op)\;\; \textbf{Then}\;\; [\![RTr]\!] \;\equiv\; (\forall *)\; \overrightarrow{\mathrm{DomPre}(Op)} \wedge \overrightarrow{RTr} \Rightarrow [\![Op]\!] \tag{78}$$

$$\textbf{If}\;\; RPo \in \mathrm{ReqPost}(Op)\;\; \textbf{Then}\;\; [\![RPo]\!] \;\equiv\; (\forall *)\; [\![Op]\!] \Rightarrow \overleftarrow{\bigcirc RPo} \tag{79}$$

Of course, (76)-(79) use the obvious extension of the $\rightarrow$ and $\leftarrow$ notations to expressions. We mention here that in the case of $\overleftarrow{\bigcirc RPo}$, where the **with** clause of (73) makes it clear that before-values as well as after-values can appear in the required postonditions RPo,[12] it is to be understood that references to before-values are to be interpreted as the corresponding left limits.

The translation into ASM of the operationalized goals in (73) follows the same pattern, where the times referred to are again t_i for $i > 0$:

$$\begin{array}{l} Op(\textbf{in}\ \overrightarrow{is}, \textbf{out}\ \overleftarrow{os}) \;= \\ \quad \textbf{if}\;\; \mathrm{DomPre}(Op)(\overrightarrow{vs}, \overrightarrow{is}) \;\; \textbf{and} \;\; \bigwedge_{RPr \in \mathrm{ReqPre}(Op)} RPr(\overrightarrow{vs}, \overrightarrow{is}) \;\; \textbf{and} \;\; \bigvee_{RTr \in \mathrm{ReqTrig}(Op)} RTr(\overrightarrow{vs}, \overrightarrow{is}) \\ \quad \textbf{then} \\ \quad\quad \textbf{choose}\;\; \overleftarrow{vs'}, \overleftarrow{os'} \\ \quad\quad \textbf{with}\;\; \mathrm{DomPost}(Op)(\overleftarrow{vs}, \overleftarrow{os'}) \;\; \textbf{and} \;\; \bigwedge_{RPo \in \mathrm{ReqPost}(Op)} RPo(\overleftarrow{vs'}, \overrightarrow{vs}, , \overrightarrow{is}, \overleftarrow{os'}) \\ \quad\quad \textbf{do}\;\; vs \;:=\; \overleftarrow{vs'}, \\ \quad\quad\quad\;\; os \;:=\; \overleftarrow{os'} \end{array} \tag{80}$$

Note that in (80) the overarrows are just semantic decoration on the previous formulation. Removing them recovers (73) obtained earlier, and no change in meaning is intended.

7.3. Pliant Variables and Pliant Transitions

Thus far, we have cleanly translated discrete event KAOS into a continuous time framework, and for the mode variables x involved, we gave a policy for replacing specific occurrences of x in the temporal formulae of operationalization by either $\overrightarrow{x(t)}$ or $\overleftarrow{x(t)}$, depending on whether the specific value was being used as a before-value or an after-value.

For pliant variables we have more to do. For each occurrence of a pliant variable x in a temporal formula, we need to decide how it is to be interpreted; whether as $\overleftarrow{x(t_i)}$ or as $\overrightarrow{x(t_i)}$, or indeed in some other way. For instance, in the operationalization formulae (76)-(79), some predicates (e.g. RPr and RTr) required a left limit, while others (e.g. RPo) required a right limit. We deliberately designed these definitions so that they would do duty also for pliant variables, so (76)-(79) can carry over without change to define instantaneous transitions for all variables. Regarding instantaneous transitions which take place at the times t_i and need to involve both mode and pliant variables, for simplicity we will

[12] Without *some* formula in (73) mentioning both before-values and after-values, it would be hard to express any correlation between before-values and after-values except by explicit enumeration.

continue to call them **mode transitions**, in contrast to **pure mode transitions**, which we define as involving mode variables alone.

For pliant variables, since their behaviour is non-constant in the interior of an interval $[t_i \ldots t_{i+1})$, we need new machinery, both to describe goals relating to their behaviour in the interior of an interval, and to describe the dynamics of their operationalizations in the interior of an interval.

Let t be a real time. We define $\mathrm{L}(t)$ and $\mathrm{R}(t)$ by:

$$\mathrm{L}(t) = \max\{i \mid t_i \leq t\} \qquad \mathrm{R}(t) = \min\{i \mid t_i > t\} \tag{81}$$

So $[t_{\mathrm{L}(t)} \ldots t_{\mathrm{R}(t)})$ is the unique interval that contains t. (N. B. The definition includes the case that $t_{\mathrm{R}(t)} = \infty$ if we accede to the convention that the minimum of an empty set of numbers is infinite.)

We now introduce the new temporal operators: *continuous eventually*, *continuous always*, *continuous formerly*, *continuous hitherto*; $\rhd$, $\Box$, $\lhd$, $\bullet$, with semantics given below. We retain the notation (σ, t) to refer to *now*, except that of course *now* may refer to an arbitrary point in real time.

$$\begin{aligned}
(\sigma,t) \vDash \rhd P \quad & \textbf{iff} \quad (\sigma, t_{\mathrm{L}(t)}) \vDash \overleftarrow{P} \ \textbf{ or } \ (\sigma, t_{\mathrm{R}(t)}) \vDash \overrightarrow{P} \ \textbf{ or} \\
& \quad (\sigma, t') \vDash P \ \text{ for some } t' : t_{\mathrm{L}(t)} < t' < t_{\mathrm{R}(t)} \qquad (82) \\
(\sigma,t) \vDash \Box P \quad & \textbf{iff} \quad (\sigma, t_{\mathrm{L}(t)}) \vDash \overleftarrow{P} \ \textbf{ and } \ (\sigma, t_{\mathrm{R}(t)}) \vDash \overrightarrow{P} \ \textbf{ and} \\
& \quad (\sigma, t') \vDash P \ \text{ for all } t' : t_{\mathrm{L}(t)} < t' < t_{\mathrm{R}(t)} \qquad (83) \\
(\sigma,t) \vDash \lhd P \quad & \textbf{iff} \quad \mathrm{L}(t) > 0 \ \textbf{ and } \big((\sigma, t_{\mathrm{L}(t)-1}) \vDash \overleftarrow{P} \ \textbf{ or } \ (\sigma, t_{\mathrm{R}(t)-1}) \vDash \overrightarrow{P} \ \textbf{ or} \\
& \quad (\sigma, t') \vDash P \ \text{ for some } t' : t_{\mathrm{L}(t)-1} < t' < t_{\mathrm{R}(t)-1} \ \big) \qquad (84) \\
(\sigma,t) \vDash \bullet P \quad & \textbf{iff} \quad \mathrm{L}(t) > 0 \ \textbf{ and } \ (\sigma, t_{\mathrm{L}(t)-1}) \vDash \overleftarrow{P} \ \textbf{ and } \ (\sigma, t_{\mathrm{R}(t)-1}) \vDash \overrightarrow{P} \ \textbf{ and} \\
& \quad (\sigma, t') \vDash P \ \text{ for all } t' : t_{\mathrm{L}(t)-1} < t' < t_{\mathrm{R}(t)-1} \qquad (85)
\end{aligned}$$

Note that the scope of the new operators is confined to a single interval.

With the new temporal operators, we can introduce new goal patterns: Reach, Leave, Adhere, Evade:

$$\begin{aligned}
&\text{Reach:} \quad P \rightarrow \rhd Q \qquad\qquad \text{Leave:} \quad P \rightarrow \rhd \neg Q \qquad (86) \\
&\text{Adhere:} \quad P \rightarrow \Box Q \qquad\qquad \text{Evade:} \quad P \rightarrow \Box \neg Q \qquad (87)
\end{aligned}$$

Note that in (86) and (87) we have only defined simple goal shapes, compared with the more complex possibilities in (59)-(62). Of course the more complex shapes could be defined here, but the nature of solutions to well behaved initial value problems such as those we are concerned with, implies that for more complex temporal behaviours, the qualitatively different regions of behaviour would usually be broken up into several pieces, individually defined. Note also that we restricted attention to future-oriented goals, which is more natural from the perspective of initial value problems, though the corresponding past-oriented goals would be easy enough to write down, just as they would be for the discrete case.

The predicates P and Q appearing in (82)-(87) will, in general, depend on t, i.e. P and Q are in fact time-indexed families of predicates over the system variables. To avoid the pathologies referred to earlier, we will henceforth assume that, regarding the variation with respect to time, they conform to the same Zeno restriction (57), and to well behavedness conditions like (74) and (75), where the latter are rewritten to refer to predicates instead of variables.

In the next section we rework our stopping case study in this framework, in preparation for which, we explore Adhere goals more deeply. A special case of the Adhere goal $P \rightarrow \Box Q$ is when P refers to the initial values of an initial value problem, and Q refers to its transition relation.[13] In more detail, suppose we have a system of ordinary differential equations, *ODEs*, for a collection of variables *vs* (with *ODEs* satisfying a set of technical restrictions as discussed in the Appendix, and the Zeno restriction (57) for the right hand sides of *ODEs*). Suppose, for t in the interval of interest $[t_{\mathrm{L}(t)} \ldots t_{\mathrm{R}(t)})$, we have that $ODEs(vs(t))$ holds almost everywhere, i.e. that $vs(t)$ solves *ODEs* in $[t_{\mathrm{L}(t)} \ldots t_{\mathrm{R}(t)})$. Then an equivalent statement to the one in the preceding sentence is that $P(t_{\mathrm{L}(t)}) \rightarrow \Box Q(t, t_{\mathrm{L}(t)})$ holds, where $P(t_{\mathrm{L}(t)})$ represents the initial state values at $t_{\mathrm{L}(t)}$, and $Q(t, t_{\mathrm{L}(t)})$ is the transition relation, connecting the initial state values at $t_{\mathrm{L}(t)}$ to the state values at an arbitrary $t \in [t_{\mathrm{L}(t)} \ldots t_{\mathrm{R}(t)})$. Moreover, as a consequence of satisfying the well behaved differential equations *ODEs*, the transition relation Q will satisfy both the Zeno restriction (57) and the well behavedness conditions (74) and (75).

[13] We now use the phrase "transition relation" in the sense of ordinary differential equations.

The transition relation Q is a family of pairs of states, parameterised by $t \in (t_{\mathrm{L}(t)} \ldots t_{\mathrm{R}(t)})$. Each individual pair of states in the family can be viewed as a state transition in its own right —comparable to an instantaneous mode transition— except that while the before-state is $vs(t_{\mathrm{L}(t)})$, which holds at $t_{\mathrm{L}(t)}$, the after-state is $vs(t)$, which holds at t, i.e. the after-state is not instantaneous with the before-state.[14] We call such time-parameterised smoothly evolving families of behaviours **pliant transitions**, to distinguish them from the mode transitions.

The preceding remarks constitute the theoretical underpinning for the operationalization (in an extended ASM framework), of the family of Adhere goals which have the form just described.

For operational purposes, the various predicates that specify a goal, which were earlier captured in DomPre, DomPost, ReqPre, ReqTrig, ReqPost, which were formalised in (66)-(69), and which were translated into ASM in (73), can be conveniently adapted to specify an initial value problem, i.e. to specify a pliant transition.

The initial values for the problem can be specified using a required trigger predicate; we rename it *RIn* (required initials) for this purpose. We can retain ReqPre to capture any other conditions that are needed in the guard of the pliant transition. The behaviour of the system in the interior of the interval of interest can be specified using a required postcondition predicate; we name it *RODEs* (required ODEs) in line with the preceding paragraphs. It states the differential equation to be obeyed by the pliant variables in the interior of the interval. We can retain ReqPost to capture any other conditions that need to be imposed on the pliant transition. The remaining KAOS predicates (domain preconditions and postconditions) can be defaulted to true. In this manner we arrive at the operationalized form of a pliant transition. Reinstating the inputs and outputs as we did for (73) and (80) we get:

$$
\begin{array}{l}
PliOp(\textbf{in}\ is(t \in (t_{\mathrm{L}(t)} \ldots t_{\mathrm{R}(t)})), \textbf{out}\ os(t \in (t_{\mathrm{L}(t)} \ldots t_{\mathrm{R}(t)}))) \stackrel{\mathrm{c}}{=} \\
\textbf{if}\ \ RIn(vs(t_{\mathrm{L}(t)}))\ \ \textbf{and}\ \bigwedge_{RPr \in \mathrm{ReqPre}(PliOp)} RPr(vs(t_{\mathrm{L}(t)})) \\
\textbf{then} \\
\quad \textbf{choose}\ \ t, vs'(t), os'(t) \\
\quad \textbf{with}\ RODEs(vs'(t), vs(t_{\mathrm{L}(t)}), os'(t), is(t), t)\ \ \textbf{and}\ \bigwedge_{RPo \in \mathrm{ReqPost}(Op)} RPo(vs'(t), vs(t_{\mathrm{L}(t)}), os'(t), is(t), t) \\
\quad \textbf{do}\ \ vs(t)\ :=\ vs'(t), \\
\qquad\ os(t)\ :=\ os'(t)
\end{array}
\tag{88}
$$

The symbol $\stackrel{\mathrm{c}}{=}$ in (88) signals that the *PliOp* being defined is a pliant transition, which exhibits qualitative differences compared with a mode transition.[15] In (88) we have made the time dependence of the various quantities of interest very explicit for expository clarity. In a specific concrete syntax (and in the treatment of our running case study below), we can choose to be briefer, recognising that the time dependence is implicit in the semantics. (Thus, for example, the restriction $t \in (t_{\mathrm{L}(t)} \ldots t_{\mathrm{R}(t)})$ is always implicit for all the variables, and does not hold just for the inputs and outputs *is* and *os*.)

In (88) we recognise a restatement of the discussion we had above. Thus, whenever we can find (a tuple of) values $vs'(t)$ that satisfies the differential equation *RODEs* at time t (which is equivalent to saying that $vs'(t)$ also satisfies the transition relation Q with initial state $vs(t_{\mathrm{L}(t)})$), then we are entitled to define the update of the state variables *vs* at time t to be the values $vs'(t)$.

A final definition addressing the end-to-end structure of system traces: we say that a collection of operationalized goals 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). (89)
- Every enabled pliant transition is feasible, i.e. has a time-indexed family of after-states, and EITHER: (90)
  - (i) During the run of the pliant transition a mode transition becomes enabled. Such a mode transition pre-empts the pliant transition, and defines the end of its family of after-states. ORELSE
  - (ii) During the run of the pliant transition it becomes infeasible, i.e. for some point in time, all the conditions stipulated cannot be satisfied simultaneously — finite termination. ORELSE
  - (iii) The pliant transition continues indefinitely — nontermination.

[14] This device, of viewing the smooth behaviour as a family of time-extended transitions, is technically much preferable to viewing it as a continuous succession of infinitesimal transitions arising from *ODEs*.

[15] In general, we have introduced different notations for the technical details concerning pliant transitions, since the reasoning used in dealing with them is typically of a different nature than that used for mode transitions. This can translate into different concrete syntax for automated tools, which can assist automated processing.

On the basis of the preceding definitions we can summarise the behaviour of a run of a well formed system over a time interval $[t_i \ldots t_{i+1})$ as follows. At t_i itself, there will be a mode transition that takes care of discrete changes to mode variables and discontinuous changes to pliant variables at t_i, operationalized as in (80). For the remainder, i.e. the open interval $(t_i \ldots t_{i+1})$, a pliant transition will relate the values of continuously varying pliant variables at various times to their right-limiting values at t_i, operationalized as in (88).

If there are any pliant variables that are not mentioned in the following pliant transition, they are assumed to remain constant at their t_i values during $(t_i \ldots t_{i+1})$. Conversely, for pliant variables that are required to remain constant at their t_i values during $(t_i \ldots t_{i+1})$, it is natural to omit them from the description (88). For such (temporarily) static variables, the operationalized pliant transition could obviously include the differential equations that equated their derivative to zero, but it would amount to notational clutter to insist on this. Instead we can envisage that there is a **notional pliant transition** that does exactly that, which we do not write down.[16]

As a consequence of the above, the complete behaviour of a run of a well formed system is just a sequence of such $[t_i \ldots t_{i+1})$ episodes, which starts with the initialisation at t_0.

One thing is immediately made clear by (89), (90), namely that the actual values of the time points t_i for a particular trace are determined dynamically as the system executes, rather than *a priori* (as we have implicitly tended to present it). The construction of an individual trace thus proceeds piece by piece, determining the t_i as it goes, as permitted by the possibilities offered by (89), (90). The set of successfully constructed traces constitutes the semantics of the system — any trace construction that cannot be successfully completed is not included in the semantics.

The summary of the semantics just given is sufficient for most practical purposes, but leaves a few small technical details unclear. For instance, how do we guarantee the conditions stipulated in (89) and (90)? How do we organise the pre-emption spoken of in (90)? A more precise treatment, also more geared towards distributed systems, is given in [BZSW].

8. Train Stopping Requirements in KAOS, and the Continuous ASM Stopping Model

In this section we revisit our earlier train stopping problem, and re-express the development of the requirements using the KAOS machinery of the previous section.

The top level requirement for our train stopping application is that it should be comfortable for the passengers (so the train should not come to a halt too abruptly), and timely (so that the train does not waste too much time slowing down). We can capture this in our top level KAOS goal:

Achieve[ComfortableTimelyTrainStopping] (91)

We can say two things at this point. Firstly, goal (91) will not immediately formalize since we have no idea yet what Comfortable or Timely might correspond to, formally. This does not matter straight away. It is good to explore ideas less formally early on, and commit to greater formality only when we are more sure of the right way forward. Secondly, once we have reached a level of detail at which we can be more confident of the formal properties, we can propagate these up the AND/OR goal tree using the relevant logical connectives, to give meaning to higher levels.

Next, we realise that the stopping phenomenon will require the initiation and termination of a "stopping episode". The initiation and termination will be discrete events, so we can refine the initial goal by:

Achieve[StopTrainInit] (92)

AND

Achieve[StopTrainFin] (93)

The goal tree so far gives a picture of our application in terms of its **modes**. These are very simple. Either the train is assumed to be in the process of stopping, or it is not. We can model this using a KAOS mode variable stopping of type BOOL. When it is true the train is in the process of stopping, and when it is false the train is proceeding normally or is stationary. The goal tree is given in Fig. 3. At this level of modeling, the two goals Achieve[StopTrainInit] and Achieve[StopTrainFin] correspond to the two KAOS LTL formulae:

[16] Alternatively, in a more complex semantics than is being sketched in this section, variables might be envisaged as belonging to various subsystems. Then, variables that are not mentioned (in a given pliant transition), might be viewed as belong to a different subsystem, and so might be expected to continue to evolve independently (instead of being forced to be constant). We do not consider such possibilities in this paper.

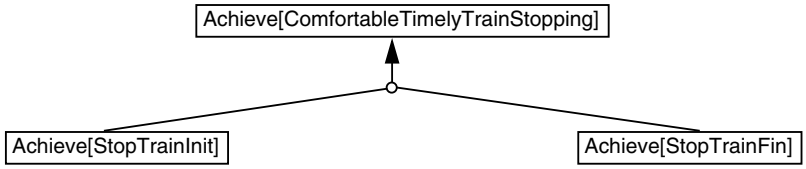

Fig. 3. A one level goal refinement of the high level Achieve[ComfortableTimelyTrainStopping] goal to a moded description using the subgoals Achieve[StopTrainInit] and Achieve[StopTrainFin].

$$\mathsf{stopping} = \mathsf{false} \Rightarrow \Diamond\ \mathsf{stopping} = \mathsf{true} \tag{94}$$

$$\mathsf{stopping} = \mathsf{true} \Rightarrow \Diamond\ \mathsf{stopping} = \mathsf{false} \tag{95}$$

Organizing a system's operation according to a set of operating modes is a well established and popular high level structuring strategy. See e.g. [Hei07, But96, RC04, SYW+11, SAZH11] and many more. Dotti at al. [DIRR09] develop a refinement method for moded systems.

Viewing a system purely through its modes gives a simple, but very incomplete picture of course. In formalisms like [DIRR09], transitions *between* modes are merely nondeterministic transitions, unaided by any information about the circumstances under which such transitions should take place. In a purely discrete system, such a description can be refined to include further discrete, "non-mode", variables, whose values determine when transitions between modes are enabled. However, when a physical system is involved, the enabledness of transitions between modes can be controlled also by continuously varying quantities, beyond the reach of an exclusively discrete description. Our own approach is to supplement the purely modal description of a system, by adding to the mode transitions the pliant transitions. The latter are able to cater for these more detailed enabledness issues in a proper manner.

To illustrate the point, we can re-examine the two Achieve goals in our case study. From our discussion so far, they are clearly operationalizable, so we refine them further by introducing additional KAOS variables accelerationrate, acceleration, velocity, displacement, t, the last of these representing time,[17] and all assumed to be of types suitable for translation into corresponding ASM variables. Thus, as regards the ASM variables, we have: *stopping*, *accelerationrate*, *acceleration*, *velocity*, *displacement*, *t*. We omit writing out the refined KAOS goals explicitly, and instead, directly present the ASM rules that embody their operationalizations. These in effect paraphrase the KAOS goals, while also paying attention to all the other variables that we have just introduced. Note that physically, *acceleration* is a pliant variable (and so is *accelerationrate* despite taking only discrete values in our case study). Rules *StopTrainInit* and *StopTrainFin* both operationalize mode transitions.

$$\begin{array}{l} \mathit{StopTrainInit} = \\ \textbf{if}\ \ \mathit{stopping} = \mathsf{false}\ \ \textbf{and}\ \ \mathit{displacement} = 0\ \ \textbf{and}\ \ \mathit{velocity} = V\ \ \textbf{and}\ \ t = 0 \\ \textbf{then do} \\ \quad \mathit{stopping}\ :=\ \mathsf{true}, \\ \quad \mathit{accelerationrate}\ :=\ -a, \\ \quad \mathit{acceleration}\ :=\ 0 \end{array} \tag{96}$$

$$\begin{array}{l} \mathit{StopTrainFin} = \\ \textbf{if}\ \ \mathit{stopping} = \mathsf{true}\ \ \textbf{and}\ \ \mathit{displacement} = D\ \ \textbf{and}\ \ \mathit{velocity} = 0 \\ \textbf{then do} \\ \quad \mathit{stopping}\ :=\ \mathsf{false}, \\ \quad \mathit{accelerationrate}\ :=\ 0, \\ \quad \mathit{acceleration}\ :=\ 0 \end{array} \tag{97}$$

Note that in (96) we have added the trigger condition $t = 0$. In the KAOS Achieve goal, there is no urgency about causing the goal to be fulfilled — it is defined by an *eventually* formula. However, the ASM translation we gave in (73) is *eager* — the rule fires as soon as its guard is true. So we added the trigger to permit a delay away from the initial time point; the same effect may be achieved by designing a more complex translation scheme into ASM (see [Ban10]). In reality of course, there would always be some real-world trigger (such as the train approaching a station) that would

[17] Unlike the other variables, time progresses of its own accord, so we do not write relations defining its progress. Instead the other variables are functions of time.

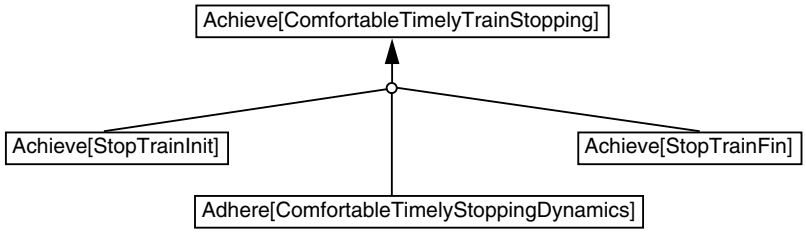

Fig. 4. A one level goal refinement of the high level Achieve[ComfortableTimelyTrainStopping] goal to the conjunction of the subgoals Achieve[StopTrainInit], Adhere[ComfortableTimelyStoppingDynamics] and Achieve[StopTrainFin].

cause the initiation of a stopping episode. Adding $t = 0$ also connects the subsequent treatment more closely to the earlier informal discussion of the case study, which was also tied to time 0.

In (96) and (97) we see clearly the mismatch between the guards of the two ASM rules, and how inadequate is the postcondition of *StopTrainInit* as regards establishing the precondition of *StopTrainFin*. So we add a new goal, Adhere[ComfortableTimelyStoppingDynamics] to the goal tree. This can mediate between the postconditions of the refined Achieve[StopTrainInit] goal and the preconditions of the refined Achieve[StopTrainFin] goal. The new goal tree is given in Fig. 4.

Seeking to improve the precision of our design, we consider what Comfortable and Timely might amount to in application terms. In line with the more informal discussion in earlier sections, we somewhat artificially decide that the train should have a linearly increasing deceleration while stopping, and that comfort and timeliness amount to choosing an appropriate stopping distance, bearing in mind the cruise velocity when stopping is initiated. (Regarding the validation side of these requirements, the task would be to ensure that an appropriate and consistent set of parameters is selected to ensure optimum performance and user satisfaction. This would involve things like passenger trials and other user centred activities. In reality, our choice of linearly increasing deceleration would be hard to reconcile with such desiderata, but we discussed the reasons for our choice of dynamics at length in Section 3, so for simplicity, we will assume that the design that we have is fit for purpose without further ado.)

Thus goal Adhere[ComfortableTimelyStoppingDynamics] can be refined to a conjunction of lower level goals. The first of them is:

Adhere[LinearDecelerationWhileStopping] (98)

This corresponds to the KAOS LTL formula:

$$\mathsf{stopping} = \mathsf{true} \rightarrow (\exists\, accel : \mathbb{R} \bullet \boxminus\, \mathsf{accelerationrate} = accel) \tag{99}$$

Note that the decision to characterize the dynamics using a constant accelerationrate means that we *could* have treated accelerationrate as a mode variable. However, in more sophisticated designs, it is unlikely that this aspect of the dynamics would truly be constant, so such a decision would have been unwise. The point is illustrated though, that the choice of which variables to nominate as mode variables amounts to more than merely determining whether their values will be piecewise constant.

When we decide on a specific value $-a$ for the accelerationrate variable, (99) is refined to:

$$\mathsf{stopping} = \mathsf{true} \rightarrow \boxminus\, \mathsf{accelerationrate} = -a \tag{100}$$

which is the LTL formula corresponding to the first of the following conjunction of five lowest level goals:

Adhere[LinearDecelerationAppropriate] (101)

AND

Reach[StoppingDistanceAppropriate] (102)

AND

Reach[StoppingTimeAppropriate] (103)

AND

Reach[TrainStopped] (104)

AND

$$\text{Adhere[StopTrainCon]} \tag{105}$$

In principle, the last four of these could be refinements of existentially stated goals like (98), but we short-circuit that to write simple definitional forms as follows. Thus the last four of (101)-(105) respectively correspond to KAOS LTL formulae:

$$\text{stopping} = \text{true} \rightarrow \triangleright\ \text{displacement} = D \tag{106}$$

$$\text{stopping} = \text{true} \rightarrow \triangleright\ \text{t} = T_{Stop} \tag{107}$$

$$\text{stopping} = \text{true} \rightarrow \triangleright\ \text{velocity} = 0 \tag{108}$$

$$\text{stopping} = \text{true} \wedge \begin{bmatrix} \text{displacement}(0) \\ \text{velocity}(0) \end{bmatrix} = \begin{bmatrix} 0 \\ V \end{bmatrix} \rightarrow \bigcirc\!\!\!\!\!\!\bigcirc \begin{bmatrix} \text{displacement}(t) \\ \text{velocity}(t) \end{bmatrix} = \begin{bmatrix} Vt - \frac{1}{3!}at^3 \\ \frac{1}{2}at^2 \end{bmatrix} \tag{109}$$

In the $\bigcirc\!\!\!\!\!\!\bigcirc$ term of (109) we mentioned just the after-state at time t of the transition relation of the differential equation governing the dynamics. This was for simplicity, since the before-state (at time 0) is already mentioned in the hypothesis. In more realistic applications, where the transition relation (which amounts to the solution of the differential equation) may not be explicitly available, we could replace the $\bigcirc\!\!\!\!\!\!\bigcirc$ term in (109) by something more implicit such as "TrRel($RDEs$)" where TrRel refers to the (not yet available) transition relation, and $RDEs$ would be the differential equation that needed to be solved.

We operationalize the Adhere[StopTrainCon] goal as follows:

$$\begin{array}{l} StopTrainCon(acceleration(t)) \stackrel{c}{=} \\ \textbf{if}\ \begin{bmatrix} displacement(t_{L(t)}) \\ velocity(t_{L(t)}) \end{bmatrix} = \begin{bmatrix} 0 \\ V \end{bmatrix}\ \textbf{and}\ stopping = \text{true} \\ \textbf{then} \\ \quad \textbf{choose}\ t, displacement'(t), velocity'(t) \\ \quad \textbf{with}\ t \in (t_{L(t)} \ldots t_{R(t)})\ \textbf{and} \\ \qquad \Bigg(\forall \tilde{t} \in (t_{L(t)} \ldots t)\ \textbf{a.e.}\ \begin{bmatrix} \mathcal{D}\, displacement(\tilde{t}) \\ \mathcal{D}\, velocity(\tilde{t}) \end{bmatrix} = \begin{bmatrix} velocity(\tilde{t}) \\ acceleration(\tilde{t}) \end{bmatrix}\ \textbf{and} \\ \qquad\ acceleration(\tilde{t}) = -a\tilde{t}\ \textbf{and} \\ \qquad\ \begin{bmatrix} \overrightarrow{displacement(t)} \\ \overrightarrow{velocity(t)} \end{bmatrix} = \begin{bmatrix} displacement'(t) \\ velocity'(t) \end{bmatrix} \Bigg) \\ \quad \textbf{do}\ \begin{bmatrix} displacement(t) \\ velocity(t) \end{bmatrix} := \begin{bmatrix} displacement' \\ velocity' \end{bmatrix} \end{array} \tag{110}$$

In (110), the input variable is *acceleration*, which is the acceleration value imposed from outside — it therefore constitutes the input control signal in control terminology (there being no output in this simple example).

In the **if** clause of *StopTrainCon*, we see mirrored the hypothesis of (109), while the **choose** clause names the values to be assigned, namely the after-values $displacement'(t)$ and $velocity'(t)$, and the time t at which these after-values are to hold. The **with** clause defines the constraints that must be satisfied and consists of four terms. The first constrains t to lie in the interval of interest $(t_{L(t)} \ldots t_{R(t)})$. This, as we mentioned above, is somewhat redundant (considering the semantics) but is retained for clarity. More generally, since we would be dealing with a syntactic description of the operation, and different occurrences of it at runtime would have different explicit values of $t_{L(t)}$ and $t_{R(t)}$, the notations $t_{L(t)}$ and $t_{R(t)}$ serve only to define a generic mechanism for referring to the initial and final time values of any runtime occurrence where needed in the syntactic description.[18]

The second states the differential equation to be solved in a suitable form. For clarity in a context that uses longer variable names, we use $\mathcal{D}$ to signify differentiation with respect to time. Since we are interested in the solution at time t, we focus on the subinterval $(t_{L(t)} \ldots t)$ ending in t. The equality between the matrices is the differential equation itself in vectorial form, which, to uniquely specify the variables at time t, must hold throughout the subinterval. The **a.e.** punctuation, replacing the usual bullet of the quantification, reminds us that since we allow discontinuities in the

[18] We use a left-open interval since we already know the values of the variables at its left limit —they are the initial values— we don't need to solve the differential equation to find those.

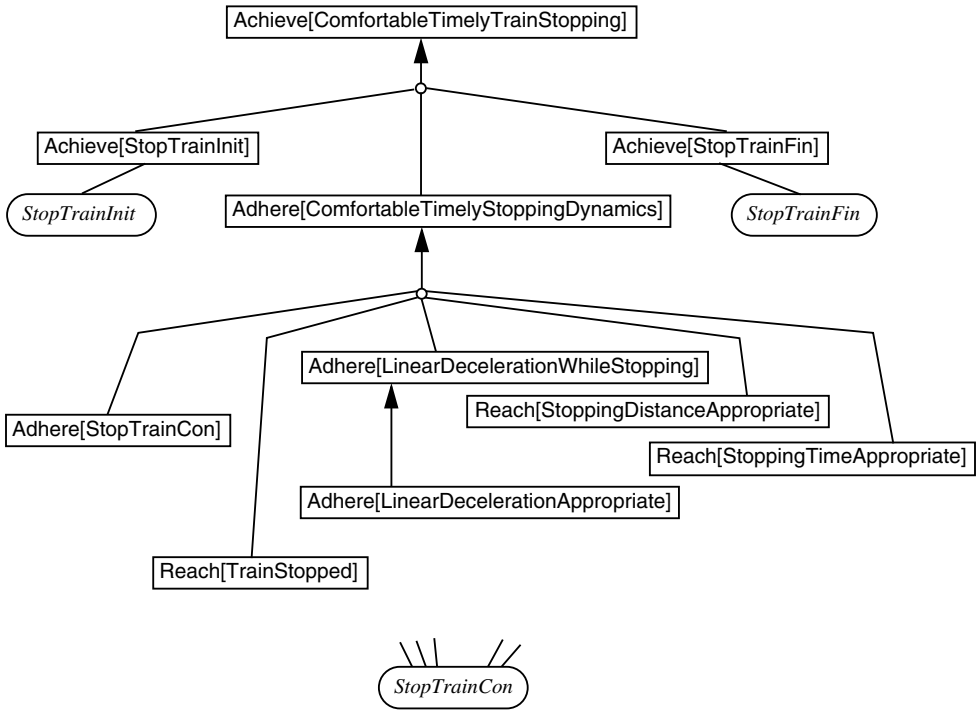

Fig. 5. A further refinement of Fig. 4 to include a detailed treatment of the stopping episode, including the lowest level operationalization to *StopTrainCon*.

right hand sides of our differential equations, their solutions, although still absolutely continuous, may be nonsmooth, i.e. their derivatives may not exist at every point. In other words the equalities in the differential equations may hold only **a**lmost **e**verywhere.[19] The third equates the *acceleration* variable to the linear function of time determined earlier, enforcing the property of the input signal decided *a priori*. The fourth, still under the quantification, states that the values attained by the variables at the right limit of the subinterval equal the values nominated in the **choose** clause. Finally, the **do** clause actually assigns the variables at the time t to the values specified in the **choose** clause.

The syntax of (110) is a bit unwieldy. In particular, connecting the values chosen, $displacement'$ and $velocity'$, at time t, to the solution of the differential equation is cumbersome, because the solution values themselves are only indirectly connected (via the non-trivial mathematics that establishes the existence and uniqueness of solutions) with the data in the differential equation (which is all about the derivatives of the solutions). In a practical syntax, much of this machinery may be taken for granted and the notation simplified. At minimum, the differential equation needs to be stated, as do any enabling criteria such as the initial values. We retained all the details here in order to emphasise that, as we have presented matters, in principle, there is hardly any distinction between transitions in the discrete and continuous worlds. In the discrete world there are isolated, unrelated, individual transitions. In the continuous world, there are families of transitions parameterized by time, and bound together by continuity and differentiability properties of various kinds. But at a suitable level of abstraction, they are all still just transitions, i.e. pairs consisting of a before-state and an after-state, each of these itself being just a valuation of the requisite family of variables.

We thus arrive at the complete operationalized model for the continuous stopping problem. In the initial state, $stopping = \mathsf{false}$, $accelerationrate = 0$, $acceleration = 0$, and $velocity = V$, the latter also being the derivative of the $displacement$ variable. Adapting the syntax for *StopTrainCon* as suggested above, the model becomes:

$$StopTrainInit \;=\; \ldots \text{ as in (96)} \tag{111}$$

$$StopTrainFin \;=\; \ldots \text{ as in (97)} \tag{112}$$

[19] A more thorough treatment of these technical details is relegated to the Appendix.

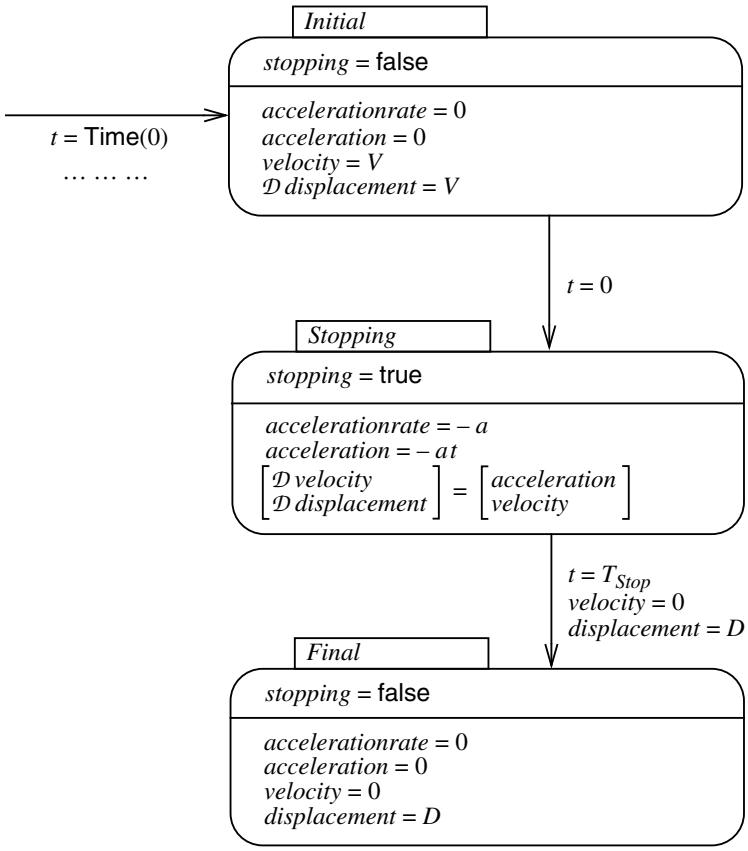

Fig. 6. A hybrid automaton reflecting the structure of the operationalized lowest level train stopping development in the continuous world.

$$StopTrainCon(acceleration(t)) \stackrel{c}{=} \tag{113}$$
$$\textbf{if } \begin{bmatrix} displacement(t_{L(t)}) \\ velocity(t_{L(t)}) \end{bmatrix} = \begin{bmatrix} 0 \\ V \end{bmatrix} \textbf{ and } stopping = \text{true}$$
$$\textbf{then with } acceleration(t) = -at$$
$$\textbf{do } \begin{bmatrix} displacement(t) \\ velocity(t) \end{bmatrix} := \textbf{solve} \left\{ \begin{bmatrix} \mathcal{D}\, displacement(t) \\ \mathcal{D}\, velocity(t) \end{bmatrix} = \begin{bmatrix} velocity(t) \\ acceleration(t) \end{bmatrix} \right\}$$

The expanded goal tree corresponding to (101)-(105) is shown in Fig. 5. Because goals (101)-(104) are consistent with the solution of the differential equation under the given initial conditions, *StopTrainCon* refines all of (101)-(105).

8.1. Operational Models and Hybrid Automata

When we have a system design refined down to a fully operational level, which is well formed in the sense of Section 7.3, i.e. each mode transition enables a pliant transition and each pliant transition enables a mode transition, then the system model amounts to a definition of a hybrid automaton in a sense comparable to the structure in [Tab09] or [Pla10b]. Our case study is of this form, and the corresponding hybrid automaton is illustrated in Fig. 6.

Using our terminology, in a hybrid automaton, the states correspond to tuples of values of the mode variables. In our case, $stopping = \text{false}$ determines automaton states *Initial* and *Final*, while $stopping = \text{true}$ determines automaton state *Stopping*. An automaton state is entered when the mode transition that causes the mode variables to take the required values is triggered. The required triggers and any accompanying discontinuous change in any pliant variable required by the mode transition decorate the arrow that causes the entry into the automaton state. For economy of

description, an automaton state contains the pliant transition following the mode transition that causes entry into it. In an automaton state, the mode variables and their values appear above the horizontal line, and the pliant variables, and either their actual values or the differential equations governing their dynamics, appear below the line.

Thus in Fig. 6, the automaton enters the *Initial* automaton state, prompted by the *Init* mode transition which sets up the initial values of all the variables. *Init* is not labelled explicitly, but is indicated by the right-directed arrow into *Initial*. The initial values themselves comprise $t = \mathsf{Time}(0)$ shown explicitly, and the remainder, indicated by the ellipsis. These remaining values refer to the values written out in detail in the body of the *Initial* state, which represents a pliant transition in which all variables (except time itself) hold their static values.[20]

Once time has progressed far enough, the trigger $t = 0$ causes the *Stopping* automaton state to be entered, coinciding with the mode transition that changes $stopping = \mathsf{false}$ to $stopping = \mathsf{true}$, and initiating the pliant transition described by the differential equations we discussed above. The triggers $t = T_{Stop}$, $velocity = 0$, $displacement = D$ (any one of which implies the other two by the consistency of the dynamics)[21] cause the exit from the *Stopping* automaton state into another completely static automaton state, *Final*, (which, in our formulation, persists indefinitely). We note that Fig. 6 is slightly incomplete since it does not specify (whether directly or indirectly) the duration of the *Initial* state. Since we have only discussed the velocity (expressed both via the *velocity* variable itself, and via the derivative of the *displacement* variable), the actual start point, corresponding to the initialisation via *Init* at $\mathsf{Time}(0)$, is an unspecified constant of integration. The only thing we know is that since $t = 0$ causes an exit from the *Initial* state, $\mathsf{Time}(0)$ must be negative.

9. The Discretized ASM Stopping Model

Having discussed the continuous ASM stopping model at some length, constructing the discretized version is straightforward. We utilize the same structures as before, but with a slightly different external control, and different timescales. For convenience, we subscript the discretized variables corresponding to earlier continuous variables with a 'D', as we did earlier. Given identical starting conditions as for the continuous model, we can write down the initial and final rules immediately:

$$
\begin{array}{l}
StopTrainInit_D = \\
\textbf{if } stopping_D = \mathsf{false} \textbf{ and } displacement_D = 0 \textbf{ and } velocity_D = V \textbf{ and } t = 0 \\
\textbf{then do} \\
\quad stopping_D := \mathsf{true}, \\
\quad accelerationrate_D := -a_D, \\
\quad acceleration_D := -a_D
\end{array}
\tag{114}
$$

$$
\begin{array}{l}
StopTrainFin_D = \\
\textbf{if } stopping_D = \mathsf{true} \textbf{ and } displacement_D = D_D \textbf{ and } velocity_D = 0 \\
\textbf{then do} \\
\quad stopping_D := \mathsf{false}, \\
\quad accelerationrate_D := 0, \\
\quad acceleration_D := 0
\end{array}
\tag{115}
$$

The analogue of *StopTrainCon* in the continuous stopping model is two rules. One, $StopTrainInc_D$, handles the increments in the discretized *acceleration* variable via mode transitions. The other, $StopTrainCon_D$, handles the continuous dynamics during the finegrained intervals of length T via pliant transitions. While the discretized dynamics is running, (the completion of) each of these rule enables the other.

$$
\begin{array}{l}
StopTrainInc_D = \\
\textbf{if } stopping_D = \mathsf{true} \textbf{ and } 0 < t \textbf{ and } t/T \in \mathbb{N} \textbf{ and } t < T_{Stop} \\
\textbf{then do} \\
\quad acceleration_D := acceleration_D + accelerationrate_D
\end{array}
\tag{116}
$$

[20] We did not discuss earlier the initial mode rule *Init*, or the pliant rule corresponding to the *Initial* automaton state, nor indeed the pliant rule corresponding to the *Final* automaton state, since they do not do anything technically interesting, though rules such as these are needed if we want the model to describe the constant velocity period before stopping, or the stationary period after stopping, in order that system traces conform to the structure of traces described in Section 7.3.

[21] The dynamical laws impose constraints on the variables, so that there are several ways of saying the same thing. Thus what appears in the automaton description need not be identical to what appears in the ASM rules. Nevertheless, the implications of both are the same.

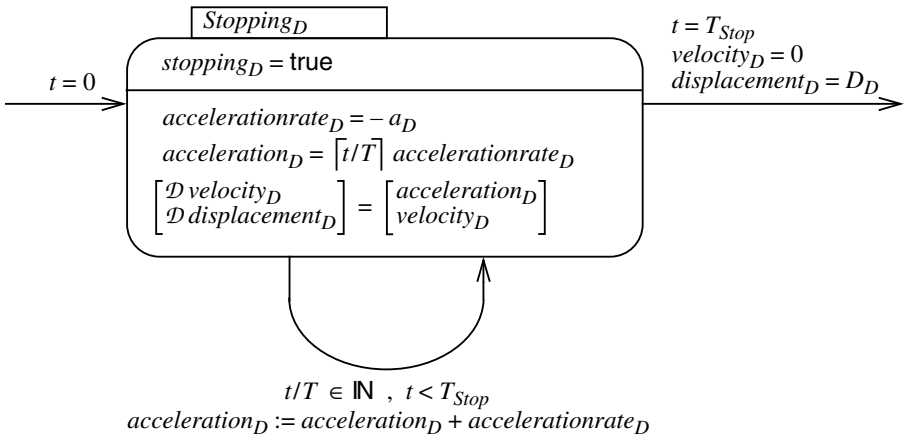

Fig. 7. A hybrid automaton reflecting the structure of the operationalized lowest level train stopping development in the continuous world.

$$\begin{array}{l} StopTrainCon_D(acceleration_D(t)) \stackrel{c}{=} \\ \textbf{if} \;\; stopping = \mathsf{true} \\ \textbf{then with} \;\; acceleration_D(t) = \lceil t/T \rceil \, accelerationrate_D \\ \quad \textbf{do} \;\; \begin{bmatrix} displacement_D(t) \\ velocity_D(t) \end{bmatrix} := \textbf{solve} \left\{ \begin{bmatrix} \mathcal{D}\, displacement_D(t) \\ \mathcal{D}\, velocity_D(t) \end{bmatrix} = \begin{bmatrix} velocity_D(t) \\ acceleration_D(t) \end{bmatrix} \right\} \end{array} \tag{117}$$

The dynamics of the above system can be visualised using the hybrid automaton in Fig. 7, which is analogous to the one in Fig. 6 for the smoothly evolving model. In Fig. 7, the $Initial_D$ and $Final_D$ states are more or less identical to their Fig. 6 counterparts, and have been suppressed. Thus the earlier *Stopping* state has changed into the $Stopping_D$ state and now has a self-loop. This accommodates the effect of the $StopTrainInc_D$ rule whose mode transitions cause discontinuous jumps in the value of $acceleration_D$.

The discussion so far completes the modeling of the system into the discretized domain. It has focused on ensuring suitable properties of the train as it stops since that is the key objective of the development. However, the properties that we have analyzed in such detail do not constitute an implemented system. The system that would actually be implemented consists of two subsytems: the *plant* — in our case the train, with its known physical properties, and the *controller* — a system whose *outputs* impose on the plant being controlled (i.e. the train) the control inputs required by the analysis done above. The next task in the practical development thus becomes one of *plant identification* and *controller synthesis*. Before doing that though, we need to cover ASM refinement and retrenchment.

10. ASM Refinement and Retrenchment

In this section we examine ASM refinement and retrenchment. We start by reviewing these notions for discrete ASM, and then extend it to include continuous behaviour, just as we did for KAOS.

The standard reference for the ASM method is [BS03], building on the earlier [Bör03]. A basic ASM rule denotes a guarded update applied to some subset of the system's variables, and we have already seen the structure in (73). Thus we have a rule name (followed optionally by a parenthesised list of inputs and outputs), after which we have the rule's guard in the **if** clause — at each state transition every rule is applied iff its guard is true (and all applicable rules must specify a consistent overall state transition). After the **if** clause comes the variable update itself in the **do** clause. A straightforward update is written as an assignment of the variables to values of expressions. A more complex update can be specified by nominating new values for the variables-to-be-assigned in the **choose** clause, and writing the properties they must satisfy in the **with** clause. Variables not updated by the rule may be updated concurrently by another rule whose guard is also true.

In the discrete world containing mode variables exclusively, individual transitions follow one another in the usual way: the after-state of one transition enables the rules that describe the next transition — if the enabled rules are inconsistent the execution aborts, and if there are no enabled rules the execution stops. Above, we extended this structure to include time-indexed families of transitions in the continuous world, so that mode transitions could interleave

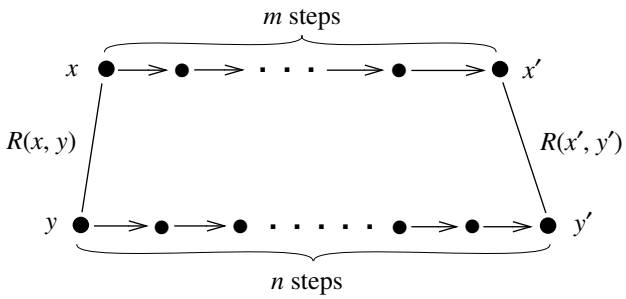

Fig. 8. An ASM (m,n) diagram, showing how m abstract steps, going from state x to state x' simulate n concrete steps, going from y to y'. The simulation is embodied in the retrieve relation R, which holds for the before-states of the series of steps $R(x,y)$, and is re-establshed for the after-states of the series $R(x',y')$.

with pliant transitions, in which the after-states of the time-indexed family of variable updates captures the continuous evolution. We now look at refinement and retrenchment in this enriched framework.

Refinement and retrenchment are both notions that relate two system models. Before plunging into the details, we outline a few notational conventions. The two system models are conventionally called the abstract system Abs and the concrete system Conc. In discussing them, we typically refer to states of the system as a whole (i.e. the tuple of values of all the system's variables) rather than to individual variables and their values. For Abs, states are referred to as x (with notational embellishments as needed), inputs are referred to as i and outputs are o. For Conc, states are conventionally y, inputs are j and outputs p. There is always a retrieve relation (also called an abstraction relation, or a gluing relation), written $R(x,y)$, that relates the set of Abs states to the set of Conc states once and for all (and usually in a problem-independent manner). Besides the retrieve relation, there are also other relations involving states, inputs and outputs depending on whether we are discussing refinement or retrenchment.

10.1. Discrete ASM Refinement

In general, to prove a discrete ASM refinement, one verifies so-called (m,n) diagrams, in which m abstract steps simulate n concrete ones (i.e. 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). The situation is illustrated in Fig. 8, in which we suppress input and output for clarity. Diagrams such as these, in which the retrieve relation R holds at the beginning and end of the (m,n) pair of m abstract and n concrete steps, are the building blocks using which, one shows that the retrieve relation R holds periodically during the course of corresponding abstract and concrete executions.

The general policy was made rigorous (and proved formally in KIV [Kar]) in the work of Schellhorn [Sch01, Sch05]. For the present paper, it will be sufficient to focus on the refinement proof obligations (POs) which are the embodiment of this policy. The first is the initialization PO:

$$\forall y' \bullet CInit(y') \;\Rightarrow\; (\exists x' \bullet AInit(x') \wedge R(x',y')) \tag{118}$$

In (118), it is demanded that for each concrete initial state y', there is an abstract initial state x' such that the retrieve or abstraction relation $R(x',y')$ holds.

The second PO is correctness, and is concerned with the verification of the (m,n) diagrams. For this, we have to have some way of deciding which (m,n) diagrams are sufficient for the application. This is very much an application-specific issue, outside the scope of this paper. For the purposes of the present description, we will assume that in some manner, we have determined the collection of (m,n) diagrams that we will need. Let *CFrags* be the set of fragments of concrete execution sequences that we have previously determined will permit a covering of all the concrete execution sequences of interest for the application — in other words *CFrags* contains all fragments needed so that any concrete execution sequence of interest can be built up as an abutting sequence of occurrences of elements from *CFrags*.[22] 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.

[22] In this connection, we observe that since *any* concrete execution sequence is just the concatenation of its individual steps, letting *CFrags* contain all individual concrete steps of interest is always going to be sufficient, leading to a collection of $(m,1)$ diagrams to base the refinement on.

We refer to a collection of ASM rules that is enabled to cause some state transition as an **operation**. If some $y::ys::y'$ in $CFrags$ is caused by a sequence of concrete operations $COps$, consuming inputs js and generating outputs ps, then we write $COps(y::ys::y', js, ps)$ to denote this fact. A similar setup holds for the abstract system. We write $AOps(x::xs::x', is, os)$ to mean that there is a fragment $x::xs::x'$ in the set of fragments of abstract execution sequences of interest $AFrags$, which is caused by a sequence of abstract operations $AOps$, while consuming inputs is and generating outputs os.

Given $AOps$ and $COps$, is and js, let $In_{AOps,COps}(is, js)$ denote a suitable input relation depending on all of these data. Likewise let $Out_{AOps,COps}(os, ps)$ be an output relation depending on the same $AOps$ and $COps$, and on os and ps. Then the correctness PO reads:

$$\begin{array}{l}\forall x, is, y, ys, y', js, ps \bullet y::ys::y' \in CFrags \wedge \\ \quad R(x,y) \wedge In_{AOps,COps}(is, js) \wedge COps(y::ys::y', js, ps) \;\Rightarrow \\ \quad\quad (\exists xs, x', os \bullet x::xs::x' \in AFrags \wedge AOps(x::xs::x', is, os) \wedge \\ \quad\quad\quad R(x', y') \wedge Out_{AOps,COps}(os, ps))\end{array} \tag{119}$$

In (119), it is demanded that when there is a concrete execution fragment in $CFrags$ of the form $COps(y::ys::y', js, ps)$, such that the retrieve and input relations $R(x,y) \wedge In_{AOps,COps}(is, js)$ hold between concrete and abstract before-states and inputs, then a suitable abstract execution fragment of the form $AOps(x::xs::x', is, os)$ can be found in $AFrags$ to re-establish the retrieve and output relations $R(x', y') \wedge Out_{AOps,COps}(os, ps)$. In other words, given the "L" shaped part of Fig. 8, we are assured that can find values for the remaining "T" shaped part, in order to complete the (m,n) diagram.

If we are sure that every Conc execution of interest can be broken up into a sequence of fragments from $CFrags$, such that for each fragment we can find an (m,n) diagram such that (119) holds, and such that the corresponding Abs fragments compose into an Abs execution, then Conc is a **correct refinement** of Abs. If *in addition*, using the same data interpreted in the converse manner (i.e. reversing the roles of Abs and Conc data), we can show that Abs is a correct refinement of Conc, then Conc is a **complete refinement** of Abs. In a complete refinement, *all* Abs executions of interest correspond to Conc executions in the manner described — if a refinement is not complete, only some of them might do so.

Beyond what we have stated, the ASM refinement policy also demands that non-termination be preserved from concrete to abstract, but we will not need that in this paper. We now turn to retrenchment.

10.2. Discrete ASM Retrenchment

Definitive accounts for retrenchment are [BPJS07, BJP08]; latest developments are found in [Ret]. See also [Ban] for formulations of retrenchment adapted to several specific model based refinement formalisms including ASM. For retrenchment, we have a similar setup as for refinement: two systems Abs and Conc, two sets of execution fragments $AFrags$ and $CFrags$ assumed adequate for our purposes, a retrieve relation R, and further relations to be described in a moment. Like refinement, retrenchment is also characterized by proof obligations: an initialization PO identical to (118), and a "correctness" PO which weakens (119) by inserting three new relations into (119) at various places. In the hypothesis of (119) we replace In by the *within* relation $W_{AOps.COps}(is, js, x, y)$. This completely generalizes the conditions on the abstract and concrete before-states and inputs that we demand of the the operation sequences to be related by the retrenchment. In the conclusion of (119) we replace Out by the *output* relation $O_{AOps.COps}(x, x', is, os, y, y', js, ps)$. This similarly allows an arbitrary strengthening of what is asserted in the conclusion when the retrieve relation is re-established. Crucially though, in the conclusion we also allow a disjunctive option when the retrieve relation cannot be re-established. This is expressed using the *concedes* relation $C_{AOps.COps}(x, x', is, os, y, y', js, ps)$. This latter relation gives room for many kinds of "exceptional" behaviour to fall inside the remit of a retrenchment, behaviours that would be impossible to cope with using refinement alone. The result is:

$$\begin{array}{l}\forall x, is, y, ys, y', js, ps \bullet y::ys::y' \in CFrags \wedge \\ \quad R(x,y) \wedge W_{AOps,COps}(is, js, x, y) \wedge COps(y::ys::y', js, ps) \;\Rightarrow \\ \quad\quad (\exists xs, x', os \bullet x::xs::x' \in AFrags \wedge AOps(x::xs::x', is, os) \wedge \\ \quad\quad\quad ((R(x', y') \wedge O_{AOps,COps}(x, x', is, os, y, y', js, ps)) \vee \\ \quad\quad\quad\quad C_{AOps,COps}(x, x', is, os, y, y', js, ps)))\end{array} \tag{120}$$

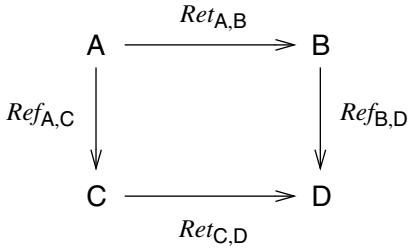

Fig. 9. The *Tower Pattern* basic square, with refinements vertical, retrenchments horizontal.

To ensure that retrenchment only deals with well defined transitions, and to ensure smooth retrenchment/refinement interworking, we also insist that $R \wedge W_{Op}$ always falls in the domain of the requisite operations:

$$\forall x, y, is, js \bullet R(x,y) \wedge W_{AOps,COps}(is, js, x, y) \;\Rightarrow\; \mathrm{dom}\, AOps(x, is) \wedge \mathrm{dom}\, COps(y, js) \tag{121}$$

Refinement is a quite exacting notion to demand of a relationship between two systems. When used as the sole means of progressing an abstract system design towards implementation, it enables strong guarantees to be given about the properties of the abstraction that are preserved in the implementation. Sometimes though, it is too demanding in the face of real-world considerations. For this reason retrenchment —which is considerably more liberal as regards what systems may be related using it— was invented. To get the best of the two worlds, we need to use refinement when it can be carried through, and to use retrenchment sparingly where it cannot. And to have a robust development methodology, we need the two techniques to be compatible.

The smooth interworking between refinements and retrenchments is guaranteed by the *Tower Pattern*, which we already mentioned earlier in the context of our overall design approach. The basic construction for this is shown in Fig. 9 (and underpins the overall construction shown in Fig. 1). There, A, B, C and D are entire systems, between which, refinement or retrenchment relationships may hold. We show refinements as vertical arrows and retrenchments as horizontal arrows, and the two paths round the square from A to D (given by composing the refinement $Ref_{\mathsf{A,C}}$ with the retrenchment $Ret_{\mathsf{C,D}}$ on the one hand, and on the other, by composing the retrenchment $Ret_{\mathsf{A,B}}$ with the refinement $Ref_{\mathsf{B,D}}$) are compatible, in the sense that they each define a portion of a (potentially larger) retrenchment from A to D.

10.3. Continuous ASM Refinement and Retrenchment

Given the formulation of refinement and retrenchment for discrete systems, a straightforward generalisation to continuous systems can be engineered more or less immediately. The easiest thing to do is to re-interpret the execution fragments $AOps(x :: xs :: x', is, os)$ and $COps(y :: ys :: y', js, ps)$ in the right way.

In the context of the mode transitions and pliant transitions of the continuous world, the abstract $x :: xs :: x'$ and the concrete $y :: ys :: y'$ of the correctness proof obligation should each consist of an alternating sequence of mode transitions and pliant transitions (starting with either kind and ending with either kind). By taking the appropriate limit at either end if needed, all together, such an abstract $x :: xs :: x'$ yields a function from some closed interval $[t_{\mathrm{a}} \ldots t_{\mathrm{a}'}]$ of $\mathcal{T}$ to abstract state values. A similar construction applies to the abstract inputs and outputs, *is*, *os*: both also yield functions from $[t_{\mathrm{a}} \ldots t_{\mathrm{a}'}]$ to input and output values. We do the same for the concrete world. The fragment $y :: ys :: y'$ yields a function from some (possibly different) closed interval $[t_{\mathrm{c}} \ldots t_{\mathrm{c}'}]$ of $\mathcal{T}$ to concrete state values; similarly for concrete inputs and outputs *js*, *ps*.

With this understanding of the notation, we can give an interpretation of the proof obligations (119) and (120) in a natural way. 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. Predicates such as $O_{AOps,COps}(x, x', is, os, y, y', js, ps)$ refer to these earliest and latest state values, and to the whole of the input and output functions. In this way, (119) and (120) continue to define ASM refinement and retrenchment in this enlarged context.

Two further things are worthy of note. The first is that this formulation of refinement and retrenchment makes no stipulation about the relative progress of time in the abstract and concrete models; i.e. we demand no constraint regarding how t_{a}, t_{c}, $t_{\mathrm{a}'}$, $t_{\mathrm{c}'}$ might be related. This aspect is left unrestricted, giving designers the maximum flexibility, and must emerge from the application context. The second is that there is no formal policing of the relationship between abstract and concrete state values during the course of pliant transitions (that happen to occur within $x :: xs :: x'$

and $y::ys::y'$). If this is a concern (something that again must be judged against the application context), then there are two potential remedies. In the first, the durations of contributing pliant transitions can be reduced, as can the values of m and n, until the granularity of the observations available via the usual (m,n) diagrams is fine enough. In the second, given a notion of correspondence between the progress of time in $[t_a \ldots t_{a'}]$ and $[t_c \ldots t_{c'}]$, the endpoints x' and y' of the two fragments $x::xs::x'$ and $y::ys::y'$ can be regarded as time dependent variables, yielding a time indexed family of correctness proof obligations, so that the state value at every intermediate moment (in the abstract and concrete fragments) is required to correspond to its counterpart. This, in effect repackages what would, almost invariably, need to be done anyway in discharging the more simply formulated end-to-end proof obligations — in the case of our case study, the information that this would produce emerges quite naturally during the course of the $\mathcal{L}^\infty$ analysis. (This is a widely applicable observation, which strongly promotes $\mathcal{L}^\infty$ techniques in general in comparable contexts.) Since we do not need this richer level of formal detail in this paper (and it somewhat flies in the face of the spirit of ASM in the discrete world), we will not pursue these possibilities further here.

11. Plant Identification and Controller Synthesis

Returning to our case study, we now face the plant identification and controller synthesis tasks. While for a large complex system, the job requires some care, so that the new system remains equivalent to the old one, in a simple system like ours it is relatively obvious what we need to do: we need to partition the variables of the integrated treatment into plant and controller variables, and to reorganize the ASM rules so that the rules for each subsystem update only the variables it "owns", taking care of any renaming or other lexical issues that arise.

All of this harks back to the KAOS idea of agents and their responsibilities, mentioned briefly in Section 7. As in the KAOS world, we view it as a bad idea to worry about this aspect too early in the design process, in case it leads to a premature commitment to design choices that later prove to be suboptimal. The partitioning process that we need to engage in is also reminiscent of structuring mechanisms found in other formal techniques, such as event decomposition in Event-B [Abr10]. For the specific case of ASM though, the process is quite simple. If we cleanly divide a rule into two subrules, each enabled as before but updating only its own variables, ASM semantics will ensure that both subrules execute simultaneously when the original rule can, leading to behaviour equivalent to the original system (under benign circumstances). If one or other of the subrules trivializes, we are at liberty to discard it without materially affecting the executions of the resulting ruleset. This therefore, is the approach we will use.

We start with the train subsystem. It will have state variables $displacement_{train}$ and $velocity_{train}$ (these directly correspond to the variables used in the informal control design of Section 3). It will also have an input variable $acceleration_{train}$ through which it gets the value of the control variable from the controller. If we examine the ASM rules for the discretized model, remove reads of variables not accessible to the train, remove updates to variables not owned by the train, and do what renaming is necessary, we end up with the following model of the train subsystem.

$$\left\{\begin{array}{l} StopTrainInit_{train} = \\ \textbf{if } displacement_{train} = 0 \textbf{ and } velocity_{train} = V \textbf{ and } t = 0 \\ \textbf{then do} \\ \quad \text{skip} \end{array}\right\} \tag{122}$$

$$\left\{\begin{array}{l} StopTrainFin_{train} = \\ \textbf{if } displacement_{train} = D_D \textbf{ and } velocity_{train} = 0 \\ \textbf{then do} \\ \quad \text{skip} \end{array}\right\} \tag{123}$$

$$\left\{\begin{array}{l} StopTrainInc_{train} = \\ \textbf{if } 0 < t \textbf{ and } t/T \in \mathbb{N} \textbf{ and } t < T_{Stop} \\ \textbf{then do} \\ \quad \text{skip} \end{array}\right\} \tag{124}$$

$$\begin{array}{l} StopTrainCon_{train}(acceleration_{train}(t)) \stackrel{c}{=} \\ \quad \textbf{do } \begin{bmatrix} displacement_{train}(t) \\ velocity_{train}(t) \end{bmatrix} := \textbf{solve}\left\{ \begin{bmatrix} \mathcal{D}\, displacement_{train}(t) \\ \mathcal{D}\, velocity_{train}(t) \end{bmatrix} = \begin{bmatrix} velocity_{train}(t) \\ acceleration_{train}(t) \end{bmatrix} \right\} \end{array} \tag{125}$$

In the above model, it is clear that the first three rules do not amount to very much, since they just skip. They do that because in the discretized model, their counterparts only updated variables not owned by the train. (It is also clear that their guards are true iff the guards of the corresponding discretized rules are true, bearing in mind all the constraints

that hold in the rest of the model). Since they just skip, we could discard them, a fact indicated by putting them in braces. (Note however, that if we insist that the rules of each subsystem yield a well-formed system in its own right, discarding such trivial rules breaks the well-formedness conditions.) Looking at the fourth rule, we see that much of it has been trivialized. It is now the responsibility of the controller subsystem, defined below, to ensure the correct dynamics, by sending the correct acceleration values to the train throughout the dynamics. Inspection of the rules of both subsystems makes clear that the train does in fact receive acceleration values that guarantee the same behaviour as in the earlier model.

Turning to the controller, its state variables are $stopping_{ctrlr}$ (which switches stopping on and off, as usual), $accelerationrate_{ctrlr}$ (whose value paces the increments in acceleration), and $acceleration_{ctrlr}$ (which holds the current value of the output acceleration). Also there is an output variable $output_{ctrlr}$ for communicating the required acceleration value to the train. The initial values of $stopping_{ctrlr}$, $accelerationrate_{ctrlr}$ and $acceleration_{ctrlr}$ are false, 0 and 0 respectively. Here are the expected rules:

$$StopTrainInit_{ctrlr}(output_{ctrlr}) = \tag{126}$$
if $stopping_{ctrlr} = \mathsf{false}$ **and** $t = 0$
then do
$stopping_{ctrlr} := \mathsf{true}$,
$accelerationrate_{ctrlr} := -a_D$,
$acceleration_{ctrlr} := -a_D$,
$output_{ctrlr} := -a_D$

$$StopTrainFin_{ctrlr}(output_{ctrlr}) = \tag{127}$$
if $stopping_{ctrlr} = \mathsf{true}$ **and** $t = T_{Stop}$
then do
$stopping_{ctrlr} := \mathsf{false}$,
$accelerationrate_{ctrlr} := 0$,
$acceleration_{ctrlr} := 0$,
$output_{ctrlr} := 0$

$$StopTrainInc_{ctrlr}(output_{ctrlr}) = \tag{128}$$
if $stopping_{ctrlr} = \mathsf{true}$ **and** $0 < t$ **and** $t/T \in \mathbb{N}$ **and** $t < T_{Stop}$
then do
$acceleration_{ctrlr} := acceleration_{ctrlr} + accelerationrate_{ctrlr}$,
$output_{ctrlr} := acceleration_{ctrlr} + accelerationrate_{ctrlr}$

$$\{StopTrainCon_{ctrlr} \stackrel{c}{=} \textbf{do } \mathsf{skip}\} \tag{129}$$

In the controller rules, we see the converse of what we saw in the train rules. The guards of the first three rules now refer to the $stopping_{ctrlr}$ variable, and, being a renaming of the respective guards in the earlier model, are also true iff the earlier guards are true. They also manipulate the $stopping_{ctrlr}$ variable and the variables connected with acceleration in a non-trivial way, rather than just skipping, as in the train subsystem.

The fourth rule appears in braces because it is just a continuous version of skip, or, to put it in the terminology of Section 7.3, it defines a notional pliant transition. Just as for the skipping rules in the train subsystem, its presence is not needed to define what is now a purely discrete transition system, suitable for digital implementation, with its successive transitions fired by an external clock. So, as for the rules in braces in the train subsystem, we can discard it, modulo concerns about well-formedness.

11.1. Recomposing Plant and Controller

We now briefly consider the problem of recomposing two subsystems derived as just discussed, and the equivalence of the recomposition to the original system. Ideally of course, it should be provable that the recomposed system is a complete refinement of the original model.[23] Suppose then, that we have two systems, A and B, and an input i_A of A is coupled to (i.e. identified with) an output o_B of B, and an input i_B of B is coupled to an output o_A of A.[24] We create a single system C, intended to encompass the interacting behaviour of both, as follows. We identify the variables of A

[23] A more extensive and more detailed treatment of this decomposition and recomposition problem, together with a discussion of complete refinement, appears elsewhere; see [BZSW12a].
[24] Our case study is simpler since the coupling goes only one way.

and B that are common (i.e. global variables), and ensure C contains a single copy of each of them (for instance this would apply to time, regarded as a read-only variable). For the remaining variables, regarded as referring to distinct concepts, C contains their union, built with suitable renaming in case there are clashes. The same principles apply to the input and output variables of A and B which are *not* coupled to counterparts in the other system.

Regarding the coupled I/O variables, we replace i_A and o_B with a new state variable, x_B say, in the composed system C, and replace i_B and o_A with a new state variable x_A in C — we adjust the signatures of all rules using i_A, o_B, i_B, o_A, so that these variables no longer appear in the I/O signature. For rules that assign to o_B and o_A, we just replace the assignments by assignments to x_B and x_A. For rules that read the values of i_A and i_B we replace those reads by reads of $\overrightarrow{x_B}$ and $\overrightarrow{x_A}$. This convention has no impact for pliant transitions — they just read the continuous values immediately as they are written. For mode transitions, our stipulating that it is the left limits $\overrightarrow{x_B}$ and $\overrightarrow{x_A}$ that are read, is consistent with our earlier conventions in (80).

The convention that mode transitions and pliant transitions interleave, prevents the well-foundedness and fixed-point problems that can potentially arise when inputs of one system are coupled instantaneously to the outputs of another in a cyclic manner (cf. the situation for Statecharts [Esh09]). In our setup, the outputs of mode transitions are consumed by the inputs of the succeeding pliant transitions, while for pliant transitions, consistency reduces to the consistency of a larger set of differential equations.

Above, we decomposed our discretized model into a train model and a controller model. We can now apply the above recomposition technique, and argue informally that the result is equivalent to the original discretized model. To start with, for each of the D rules, $Init$, Fin, Inc, Con, there is only one (undiscarded) subsystem subrule — the first three of them are in the controller subsystem, and the fourth is in the train subsystem. This simplifies the reconstruction, since there is no actual recomposition to do.

Consider the $Init$ rules. The controller subrule (126) has both an $acceleration_{ctrlr}$ variable and an $output_{ctrlr}$ output variable.[25] The $output_{ctrlr}$ variable will be identified with the train's $acceleration_{train}$ input variable, while the $acceleration_{ctrlr}$ variable remains internal to the controller. But all the controller subrules manipulate these two controller variables identically, so that the presence of the extra internal variable does not materially alter the system dynamics. Besides this, the D rule (114) has a stronger guard than (126). But one can attribute the extra terms, $velocity_D = V$ and $displacement_D = 0$, to the initial state and to external environmental assumptions, so in this context, the two $Init$ rules are equivalent. We can argue in a similar manner for the other two controller subrules.

Regarding the only train subrule (125), we can argue that the stronger guard of the corresponding D rule (117), $stopping = \mathrm{true}$, is true iff the $acceleration_{train}$ input variable of the train subrule is non-zero. Analogously, the non-trivial **with** clause of (117) merely records the discrete values of the $acceleration_{train}$ input variable that the train subrule receives from the controller subsystem. So again, we can conclude that the two rules generate equivalent effects. Summarizing, it is thus reasonable to claim that given the appropriate formal context, the recomposed system would be a complete refinement of the original discretized model.

Thus, the decomposition and recomposition for the discretized system can be handled quite straightforwardly. We could undertake the same approach to the original continuous system. There, while the majority of the work would be very similar, the analogue of $StopTrainCon_{ctrlr}$, i.e. rule (129), would no longer be a notional pliant transition, since the continuous controller subsystem would have to feed the continuous train subsystem with a continuously varying control input.

12. Formalizing the Continuous to Discrete Modeling Change

Having both the continuous and the discretized ASM models of our case study, the next step along the formal route is to formalize the change from continuous modeling to discretized modeling, which we have already discussed informally in Section 5.

The first port of call in formalizing some development step is to try to use refinement if it will work. In our case study, we are immediately confronted with an impediment to this approach because the discrepancy between the continuous state and the discretized state increases monotonically as time progresses. This is the content of (44) when understood as a function of T_{Stop}. Under the circumstances, a bound on the discrepancy between the states that is expressed in a retrieve relation R, and that holds at the beginning of a pliant transition, is not guaranteed to still hold at the end of that pliant transition as the states continue to drift apart from each other. So, in general, the

[25] Having both an internal state variable to retain the current value of a state component, and an output variable to communicate its value to the outside world, is a conventional design principle.

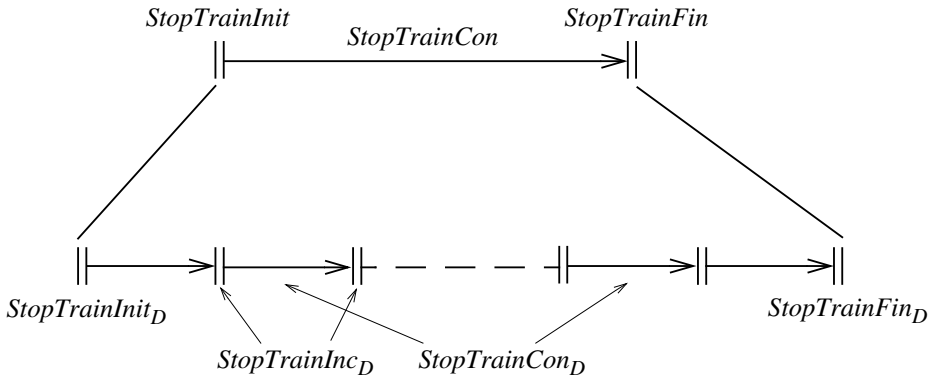

Fig. 10. A $(3, 2N+1)$ diagram of the discretization development consisting of *StopTrainInit*, *StopTrainCon*, and *StopTrainFin* at the abstract level, and of $StopTrainInit_D$, followed by $N-1$ repetitions of "$StopTrainCon_D$ followed by $StopTrainInc_D$", a final occurrence of $StopTrainCon_D$, and then $StopTrainFin_D$ at the concrete level.

exigencies of formal refinement are too exacting to be able to accommodate the kind of relationships that can hold between continuous and discretized systems. Retrenchment though, has been purposely designed to be more forgiving as regards all manner of system properties, so we will use it instead of refinement to formalize our earlier results.

It turns out that retrenchment can be used to formalize the relationship between our continuous and discretized systems in many ways. We employ what we regard as the most generic approach in this section, and we discuss the pros and cons of alternatives in Section 14.1 below.

Regarding our case study, in the continuous world (this being the abstract system), there is essentially only one episode of interest, the stopping episode from $t = 0$ to $t = T_{Stop}$. Correspondingly, in the discretized world (this being the concrete system), the episode of interest should therefore also be the stopping episode from $t = 0$ to $t = T_{Stop}$. We will concentrate on these.[26]

At this point there is a minor tactical choice to be made. We have available two formalizations, in principle. Firstly the original continuous and discretized systems; secondly the restructured train and controller subsystems, built explicitly in the discretized case, and having a similar structure (though not explicitly constructed above) in the continuous case. Although the actual system architecture conforms more closely to the decomposed model structure, it is a little easier to deal with the original system models, since all the variables are present together and their interactions are easier to discern. We just have to remember that although it is the controller subsystem that will be built in software, and whose correctness is at issue (according to the criteria that we decide are in force), it is the train subsystem whose properties we have to worry about (since it is the train's properties that the correctness of the system as a whole depends on). Once we have the discretized train and controller subsystems, we can easily identify which variables in the original discretized model correspond to the train subsystem, and so, which variables of the original systems need to figure in the retrenchment.

We therefore focus on building a single (m,n) retrenchment diagram as follows — this will be formulated in terms of the original continuous and discretized systems. The abstract level will consist of three ASM steps: *StopTrainInit*, *StopTrainCon*, and *StopTrainFin* — two mode transitions interleaved with a pliant transition. The concrete level will consist of $2N+1$ ASM steps: $StopTrainInit_D$, followed by $N-1$ repetitions of "$StopTrainCon_D$ followed by $StopTrainInc_D$", next a final occurrence of $StopTrainCon_D$, and lastly $StopTrainFin_D$. This is illustrated in Fig. 10, in which mode transitions are depicted as pairs of short vertical lines, while pliant transitions are depicted as long arrows. Since we only consider one (m,n) diagram (the $(3, 2N+1)$ diagram we have just described), in the (subscripts of the) retrenchment data below, we do not include the names of the abstract and concrete operations involved.

[26] N.B. In order to focus on the technically interesting parts of the case study, we will neglect the initial states (which are identical up to names of variables), and also the final states (which are identical up to names of variables, within an $O(1/N)$ discrepancy in the displacements).

12.1. Turning Rigorous Bounds into Retrenchment Data

In section 5 we discussed the relationship between continuous and discretized models on the basis of rigorous results presented in the Appendix. Our job now is to show how such results may be used to yield provable retrenchment POs for suitable (m,n) diagrams in a formal setting.

As we saw above, a retrenchment between the operation sequences in an (m,n) diagram needs four pieces of data: a retrieve relation between the state spaces, a within relation for the before-states and inputs, an output relation for the after-states and outputs (and before-states and inputs too if these need to be mentioned), and a concedes relation for the after-states and outputs (and before-states and inputs too if they are needed). We now give these for the abstract and concrete operation sequences just discussed.

Regarding the retrieve relation R, since in applications such as ours, the states in the two models can diverge over time, it is not convenient, neither is it particularly productive, to try to relate the state spaces in the retrenchment correctness PO using a non-trivial retrieve relation. Consequently, we set R to true:

$$R(\langle displacement, velocity\rangle, \langle displacement_D, velocity_D\rangle) \equiv \mathsf{true} \tag{130}$$

Given the vacuous role of the retrieve relation, the job of keeping the before-states suitably matched, as well as relating the control inputs, is taken on by the within relation W:

$$\begin{aligned} &W(acceleration(t), acceleration_D(t),\\ &\quad \langle displacement(0), velocity(0)\rangle, \langle displacement_D(0), velocity_D(0)\rangle) \equiv\\ &\qquad displacement(0) = displacement_D(0) \wedge velocity(0) = velocity_D(0) \wedge\\ &\qquad ||acceleration - acceleration_D||_2 \leq a_D T\sqrt{T_{Stop}} \end{aligned} \tag{131}$$

Note that while W relates just the continuous and discrete before-states (at $t = 0$), it also relates the continuous and discrete control inputs during the whole of the duration of the dynamics.

The output relation O is responsible for saying what has been achieved at the end of the period of interest. In our case, on the basis of the rather heavy calculations that came earlier, we can use O to say that the after-states diverge by no more than the bound derived in (43).

$$\begin{aligned} &O(\langle displacement(T_{Stop}), velocity(T_{Stop})\rangle, \langle displacement_D(T_{Stop}), velocity_D(T_{Stop})\rangle) \equiv\\ &\qquad |displacement(T_{Stop}) - displacement_D(T_{Stop})| \leq e^{T_{Stop}} a_D T T_{Stop} \wedge\\ &\qquad |velocity(T_{Stop}) - velocity_D(T_{Stop})| \leq e^{T_{Stop}} a_D T T_{Stop} \end{aligned} \tag{132}$$

Note that in (132) we had no need to mention outputs (since there are none in these models), nor did we need before-states or inputs.

Since our system is so simple, O already captures all that we need to say, and the kind of exceptional behaviour that needs to be taken into account in more realistic engineering situations, which feature less analytic solvability and less certainty, is not present. Accordingly we can set the concedes relation C to false:

$$C(\langle displacement(T_{Stop}), velocity(T_{Stop})\rangle, \langle displacement_D(T_{Stop}), velocity_D(T_{Stop})\rangle) \equiv \mathsf{false} \tag{133}$$

The above gives the retrenchment data for the $(3, 2N+1)$ diagram that we discussed above, and depicted in Fig. 10. Referring to the figure, the sloped vertical lines connecting the upper (abstract) and lower (concrete) levels represent R, trivial in this case. The remaining relations of the retrenchment data concern various parts of the diagram. For instance, the within relation W relates the inputs received throughout the dynamics relevant to the diagram. On the other hand, the output relation O just concerns the after-states, even if it could have concerned other variables if the abstract and concrete models had produced a more compicated dynamics. The trivial concedes relation C again does not play any significant role due to the simplicity of the dynamics.

12.2. Corroboration Revisited

With the retrenchment data in (130)-(133), the proof obligation (120) becomes provable on the basis of the rather heavy calculations presented earlier in the paper and in the Appendix. In outline it would go as follows.

We already know that if we have the continuous and discretized dynamics obeying the restrictions mentioned in the within relation, then on the basis of the results in the Appendix, the $\mathcal{L}^\infty$ norm of the difference between continuous and discretized states will not exceed the value used in the output relation at the completion of the dynamical evolution.

Applying this to the specific example of our case study, the explicitly discussed continuous and discretized dynamics of the train stopping problem satisfied the relevant restrictions. That means that at the completion of the dynamics, the conclusions mentioned in the output relation would also be satisfied. In fact we know more, because having calculated both models exactly, we were able to know what the relevant $\mathcal{L}^\infty$ norm was exactly, and this proved to be well within the bounds claimed by the generic results in the output relation. This was the import of (44) in Section 5.2.

13. The Road to Implementation: A Simple ASM Model for Braking

At this point, we have reached the downward arrow on the right hand side of the system development framework in Fig. 1. Taking it as read, that the appropriateness of the retrenchment just derived guarantees the adequacy of the controller submodel (based on the equivalence of the original model and recomposed submodels, expressed in the complete refinement from the original to the recomposed models indicated earlier), it is sufficient to proceed formally by refining the controller submodel.

Note that in general, when there is feedback from the plant back to the controller, any refinement of the controller submodel must be a *complete refinement*. The reason for this is that an arbitrary concrete refinement of an abstract model only implements a subset of the abstract model's behaviours. However, when there is feedback from the plant, the only position we can take when the interaction between plant and controller has been shown to be correct at an "abstract" level of abstraction, is that any "concrete" refinement of the controller must be able to cope with *any* feedback behaviour from the plant. The only reliable way to ensure this (in the absence of application-specific stronger information), is to demand that the concrete controller can emulate *any* of the abstract controller's behaviours. In other words, we must have a complete refinement. (This argument gains even more force if we extend consideration to cases where the plant is subject to noise, leading to unpredictable disturbances in the controller's inputs that the controller must deal with in a manner that still enforces the overall system goals.)

In our case study though, there is no feedback from plant to controller —the overall system is structured so that the controller just tells the train what to do— so an arbitrary refinement will suffice. We also observed earlier that the controller merely outputs values which are held constant for a period, so that treating the controller as a purely discrete transition system is entirely appropriate. So, since refinement in the purely discrete world is well understood by now, we make do with a simple refinement towards implementation, for the sake of illustration.

We will assume that the slowing down of the train is accomplished via a traditional disc braking system, in which the plates of the disc brake have pressure applied to them to force the plates together. Since one plate is fixed to the train chassis and the other is fixed to the rotating axle, the contact between them generates friction which in turn generates a torque on the axle that slows down the train.

The deceleration that this process generates is proportional to the torque created. Thus to increase the torque in a linear fashion, it is sufficient to increase the frictional force in a linear fashion. For simplicity, we will assume that the frictional force is proportional to the pressure applied to the disc, so to increase the torque linearly, it is sufficient to increase the pressure linearly. Thus the (negative) stepwise increments $-a_D$ in the acceleration $acceleration_{ctrlr}$ of the controller submodel, get refined to (positive) stepwise increments stp_{brk} to the applied pressure $press_{brk}$ (or, more accurately, to the output signal that communicates this value to the disk pressure actuator). Also in the following model, we use the *Inc* rule (which increments the the internal state variable $press_{brk}$) to precompute the value ready for the *next* invocation, so that the rule can set $output_{brk}$ to be the before-value of $press_{brk}$, saving a little computation. In this manner we refine the controller submodel to the braking model as follows, completing our formal development.

$StopTrainInit_{brk}(output_{brk}) =$ (134)
if $stopping_{brk} = \mathsf{false}$ **and** $t = 0$
then do
$stopping_{brk}$:= true,
$pressrate_{brk}$:= stp_{brk},
$press_{brk}$:= $2stp_{brk}$,
$output_{brk}$:= stp_{brk}

$$\begin{array}{l} StopTrainFin_{brk}(output_{brk}) \ = \\ \textbf{if} \ \ stopping_{brk} = \mathsf{true} \ \ \textbf{and} \ \ t = T_{Stop} \\ \textbf{then do} \\ \quad stopping_{brk} \ := \ \mathsf{false}, \\ \quad pressrate_{brk} \ := \ 0, \\ \quad press_{brk} \ := \ 0, \\ \quad output_{brk} \ := \ 0 \end{array} \tag{135}$$

$$\begin{array}{l} StopTrainInc_{brk}(output_{brk}) \ = \\ \textbf{if} \ \ stopping_{brk} = \mathsf{true} \ \ \textbf{and} \ \ 0 < t \ \ \textbf{and} \ \ t/T \in \mathbb{N} \ \ \textbf{and} \ \ t < T_{Stop} \\ \textbf{then do} \\ \quad press_{brk} \ := \ press_{brk} + stp_{brk}, \\ \quad output_{brk} \ := \ press_{brk} \end{array} \tag{136}$$

It is clear that the controller model and the braking model are related by a $(1,1)$ refinement, in which the output relation *Out* (which is the same for all three rules *Init*, *Fin*, *Inc*, so is undifferentiated) is:

$$Out(output_{ctrlr}, output_{brk}) \ \equiv \ output_{brk} = -L_{c,b}\, output_{ctrlr} \tag{137}$$

(where $L_{c,b}$ is the absolute value of the proportionality constant between acceleration and pressure), and where the retrieve relation $R_{ctrlr,brk}$ is:

$$\begin{array}{l} R_{ctrlr,brk}(acceleration_{ctrlr}, press_{brk}) \ \equiv \\ \qquad stopping_{brk} = stopping_{ctrlr} \ \wedge \\ \qquad pressrate_{brk} = -L_{c,b}\, accelerationrate_{ctrlr} \ \wedge \\ \qquad\qquad press_{brk} = \begin{cases} stp_{brk} - L_{c,b}\, acceleration_{ctrlr} & \text{if } acceleration_{ctrlr} \neq 0 \\ 0 & \text{otherwise} \end{cases} \end{array} \tag{138}$$

We notice, as usually happens once we are firmly on one side or the other of the discretization boundary, that we can now make good use of a static retrieve relation such as $R_{ctrlr,brk}$. Looking back, we see that although there is a perfectly sensible static relation between the state spaces of the continuous and discretized models (given by an identity in our case), it cannot be used as a retrieve relation because of the position the retrieve relation occupies in the correctness proof obligation (of either refinement or retrenchment).

14. Associated Issues

In this section we discuss a number of issues which, while not on the critical path of the development framework of this paper, nevertheless impinge on the topics investigated earlier, if we broaden the perspective beyond the very simple case study done here.

14.1. Alternative Retrenchments

We mentioned earlier that there was more than one way to use retrenchment to convey the relationship between the continuous and discretized versions of the dynamics of our case study, and by implication, similar remarks apply more widely. We look at this more closely now.

In the context of designing an (m,n) diagram that is to encompass the entire dynamics of a system using retrenchment, we always have the choice of choosing some relation on states as our retrieve relation R, provided it is adequate for describing the discrepancy between continuous and discretized states at the beginning of the dynamics. Then, even if there is an increase in the discrepancy as the dynamics progresses, we can use the concession of the retrenchment to handle any overspill. While this works mathematically, we argue that it is not the best way to proceed, for a number of reasons.

Firstly, we may want to obtain finer formal judgements about the dynamics than can be achieved in a single end-to-end (m,n) diagram. In such a case, the various (m,n) diagrams that make up the relationship between a continuous and a discretized execution, typically need to be applicable at many places in the two executions. Therefore, a discrepancy

between the states that is expressed using R, and that is adequate to describe the before-states situation at one point in the two executions, may be breached by the after-states, and thus need not be adequate for the before-states at another point in the two executions, blocking the applicability of the (m,n) diagram. Secondly, such a strategy entails using the concession of the retrenchment to handle what is essentially a "routine" and "correct" behaviour of the two systems. Methodologically, it is preferable to retain the concessions for behaviours that are genuinely exceptional in some sense.

The strategy being discussed works a lot better when the dynamics of the control problem is stable, a case that arises extremely commonly. Stability implies that every system execution converges to a single point in the state space, and consequently that any two executions eventually tend to get closer and closer to each other as they both converge to the same point. In such a case, choosing an R that is globally adequate (in terms of expressing a discrepancy between continuous and discretized dynamics that is never breached), can yield a satisfactory account of the dynamics in terms of a pure refinement (since a concession will never be needed), but on the down side, it may be tricky to prove that some *specific* value for the maximum discrepancy (to be captured in R) is adequate, even if it is known on the basis of general principles that *some* such value must exist. In summary then, we prefer the strategy used earlier, which trivialized R and treated state discrepancies via W and O, because it is the most generic, and is the simplest to apply.

14.2. Other Discretization Issues

In this paper we have focused in detail on a very simple case study, taking it through from high level goals down to a representative controller implementation, all in a thoroughly rigorous manner. In this section, without intending to be comprehensive, we indicate a much broader range of application areas in which the techniques that we have used may be deployed.

To start with, there are many other kinds of discretized "hold" that are considered in the control literature, requiring different data for their specification. For example a "first order hold" approximates the desired continuous control signal with a series of linear pieces. Such linear pieces require two numbers to specify them for each short interval: the initial value and the constant slope. Of course the success of implementing such a higher order hold using digital hardware (which can only output some constant values) depends on having physical apparatus for which linear behaviour can be elicited by inputting suitable constant values. These higher order holds can also be treated by the techniques we have used in this paper.

In all that we have done, we have assumed that (genuine) real numbers of unlimited precision are always available. But it is well known that real digital hardware is only capable of limited precision. The discrepancy between an ideal real number (whether belonging to a continuous or to an "ideally" discretized system) and a finite precision counterpart, is of the same character as the discrepancies we have been dealing with, and can be tackled using the same techniques.

Besides what has been mentioned, all the discretization approaches that we are discussing can themselves lead to alterations of the nature of the control problem being addressed, for instance by introducing destabilization in some parts of the dynamics. This can also be addressed using our techniques.

In a celebrated recent address, Kalman [Kal05] said "Get the physics right. The rest is mathematics." (See also [Wil07].) This is a very relevant maxim these days, when domain specific modelling languages are being widely designed and implemented for many application areas, many of them being strongly connected to physics and engineering. The point being made in the quotation is that in many cases, for purposes of succinctness and/or computational efficiency, such languages embed specific assumptions and approximations. The integration of such languages, frequently made in the multi-engineering applications of today, bring these hidden assumptions and approximations into conflict with each other, to the detriment of the application being developed. The same point is even more pertinent in the cyber physical systems that are increasingly being developed today. (See [Szt11].) Again, the rigorous approach that we have taken in this paper, can bring the relevant issues to light, and can help resolve them.

An interesting point relates to the tension between exact and approximate modeling. In this paper we started out by modeling in an exact way, and only when we came to consider discretization, did the necessity of gauging the quality of the approximation we were engaging in, raise its head. An alternative approach would have been to specify the continuous dynamics at the outset, not as an exact system trajectory, but only within an allowed margin of error. In a sense, this is closely related to the approach discussed in Section 2, in which the relevant approximation is cast at the highest level of abstraction. In essence, the $\Box$ properties that such a specification expressed, would describe a "tube" in the dynamics, outside of which the system trajectory was not allowed to stray. Starting from such a perspective, the discretization programme could potentially have been performed using a kind of refinement, in which the retrieve relation carries the degree of approximation being tolerated. This is essentially the same idea discussed in Section

14.1, in which finding an actual bound for the approximation may become non-trivial. In our judgement, this approach brings with it a premature mixing of concerns — right at the outset, one has to contend with the complexities of the approximation needed, even before one has decided what the dynamics ought, ideally, to do. In our judgement, it is much preferable to do early design in the simplest possible framework, and to contend with the unavoidable approximations needed, later, at the discretization stage. Indeed, the desirability of avoiding such premature mixing of concerns, and the ability to postpone their consideration to a more appropriate point in the development, has been an overriding *leitmotif* of the retrenchment approach from the outset.

14.3. Reasoning Issues

In this paper we have used a wide variety of mathematical techniques which have been brought to bear on the problem we tackled in an integrated way. We make some further comments on this aspect of the work now.

In the body of our case study, we made use of "plain vanilla" $\mathcal{L}^\infty$ analysis. In other words, given that the state vector to be reasoned about had two components, displacement and velocity, in gauging the quality of the approximation in our $\mathcal{L}^\infty$ estimates, one unit of displacement was regarded as of equal weight to one unit of velocity, measured in the default units that resulted from regarding these quantities as simple real numbers. In genuine physical applications, two things disturb this simple picture. Firstly, the relevant physical quantities each have their own physical units, and are normally regarded as having their own "dimensionality". For instance, displacement would have dimensionality of length, denoted "L", while velocity would have dimensionality of length divided by time, denoted "LT^{-1}". Secondly, and following on from this dimensionality, in measuring the distance between two states, it may not be appropriate to regard a unit of one physical quantity, e.g. displacement, of dimensionality L, as comparable to a unit of another quantity, e.g. velocity of dimensionality LT^{-1}. We can address both issues by using *weighted* $\mathcal{L}^\infty$ analysis, in which the different components of the state are compared only after each is multiplied with a suitable weight. The weight chosen for each component of the state would take into account two factors: (i) the influence of the physical units used, and their dimensionality (so that an $\mathcal{L}^\infty$ norm would calculate a comparison between quantities having the same dimensions), and (ii) the relative importance of the different components relative to each other in determining the quality of the approximation between two states. Such a dimensional recasting of $\mathcal{L}^\infty$ analysis would entail making minor adjustments to relevant "plain vanilla" results to be found in e.g. [HJ85, HJ91] and elsewhere.

In generalizing the the original discrete KAOS and ASM formalisms to continuous phenomena in real time, we were partly inspired by the real-time ASM work of [CS08,SV08]. Those authors also use half open half closed intervals of the reals, although all their variables are piecewise constant throughout. In these works, state variables (such as our displacement and velocity) consist of a series of left-open right-closed intervals, while the inputs they react to are left-closed right-open intervals. This allows there to be a single value of the real time at which the old state values coexist with the new input values as the state reacts to the new inputs, an elegant modeling trick. In our work it proved more convenient to use left-closed right-open intervals throughout, and to have mode transitions to handle discontinuities.

The use of both mode transitions and pliant transitions was intended, as far as possible, to enable the separation of reasoning about continuously varying quantities from reasoning about discontinuously varying ones. While there is no particular reason to worry about this from a theoretical point of view, when we contemplate creating tools to support a framework such as the one being developed in this paper, the interaction between the requirements that mechanized reasoning imposes in various subdomains of mathematics becomes an important issue. Having both mode and pliant transitions in the formalism (at the very least) facilitates the strategy of separating the reasoners that deal with the discrete aspects and mode transitions, from the reasoners that deal with the continuous aspects and pliant transitions, enabling the different kinds of algorithm in the two fields to play to their strengths. Still, imagining that a total separation of discrete and continuous reasoning is achievable is illusory. We saw this already in our case study when discussing plant identification and controller synthesis. There, to prove that certain subsystem rules were enabled only when the appropriate counterpart rules of the original system were enabled, required reasoning about the interaction between the subsystems — this brings the two kinds of reasoning into direct contact.

The contrast between reasoning in the discrete and continuous domains is interesting. While discrete reasoners all have a similar flavour —they all deal with relatively unstructured objects, and have no choice but to manipulate them in the fairly constrained ways that the axiom system being used allows— in the continuous domain, the objects being dealt with (e.g. real numbers and functions of reals, etc.) have a rich set of built-in properties, based on the complex way they are constructed axiomatically. By focusing on different areas within this rich family of properties, different reasoning systems have radically different inference capabilities. This can dramatically affect their ability to deal with the kind of calculations needed in our case study and similar applications. We briefly mention a few approaches that typify the variety available.

The KeYmaera approach [Pla10b, LPN11], focused on (quantified) differential dynamic logic [Pla10b, Pla10c, Pla10a] and on a tractable class of differential equations which have behaviours that are smooth throughout (precluding problems needing a non-smooth treatment), integrates a theorem prover with the well known Mathematica [Mat] system (and with other reasoners too). This combines the discrete reasoning of the theorem prover with the extensive applied mathematics capabilities of Mathematica. On the face of it, this sounds like an ideal combination, until we realise that the reasoning in tools like Mathematica can be somewhat pragmatically based. In [HT98] the authors give a thorough discussion of why the combination of the HOL theorem prover and the Maple [Map] computer algebra system can yield unsound results — similar remarks could apply to KeYmaera in principle. A lot depends on which parts of the reasoning capabilities of the tool were being exploited, and how (though Appendix D of [Pla10b] gives a lot of reassurance that care has been taken over this point). The upshot of this is that such combinations run the risk of yielding proofs that were not as rigorous as those possible theoretically using the techniques of this paper, of which a hint was given in Section 12.2.

Other reasoning systems intended for continuous time, such as interval logic in general [Cau], or duration calculus in particular [ZHR91, OD08]), are typically targetted at durations of discrete properties, and are usually focused on issues connected with model checking and decidability for classes of behaviours that can yield such properties. From a by now very large literature we can point to e.g. [AH93, AH94, HK97, HMP92, Oua02]. (Decidability *per se* is obviously impossible for such expressive systems, which explains why formal systems with curtailed expressivity are popular in the literature.)

A more fundamentalist approach, based on a ground-up formal reconstruction of the required mathematics, is possible, although it is a big job to build up enough of the necessary machinery from scratch. Some indicative work along the required lines includes the support for both applied mathematics and its more rigorous counterparts in PVS [COR$^+$95, PVS], or the ground-up development of topology in PVS [Les07]. One can also mention the HOL-Light suite, and its complex analysis library as work that is headed in the right direction [Har07, Har96, GM93]. However, it is fair to say that a suite of formally developed mathematics that could properly support all stages of the work done in this paper is still lacking. Given the undecidability of any system rich enough for the job, use of such a system would certainly involve a good deal of interactive proof.

15. Conclusion

The various sections of this paper have all contributed to one overall goal: to demonstrate that it is perfectly feasible to do a development, starting from initially imprecise requirements, proceeding via continuous modeling in the physical world, making the transition to a discretized model, and following through to a digital implementation, all within an integrated formal development methodology.

To make the process concrete, we introduced a small continuous control problem. We first developed the crucial models informally, to set the scene. Most impotantly, this entailed investigating the relationship between continuous control and discretized control in a rigorous manner, the more intricate technical details of which we relegated to the Appendix. The informal discussion made for a good panoramic perspective on the issues that we contended with. In particular we could describe clearly the engineering benefits of a quantitative comparison between continuous and discretized control models.

Having thus overviewed the development route and the main technical issues involved, we were then able to tackle the formalized development. We started with a simple version of the requirements goals, in discrete event KAOS. To make these more precise in the context of an application that needed continuous variables, we developed an extension of KAOS that was able to capture aspects of the continuous behaviour in the required way. The operationalization of the lowest level KAOS requirements entailed also developing a continuous variable extension of ASM, which was the formalism we used to move our development from the requirements engineering world to the model based refinement world. Then came what was the main novel contribution of the paper, a formalization of the rigorous treatment of the continuous to discretized modeling transformation via an appropriately constructed retrenchment between the continuous and discretized worlds, based on the analysis done previously. From there we were able to continue the development in the purely discrete domain towards implementation.

As noted earlier, for continuous control problems, formally supported development normally starts already in the discrete domain, so the ability to connect this with the continuous world in a reasoned way, is a significant extension of the potential of formal techniques to underpin developments of such systems. Equally importantly, in making essential use of retrenchment to forge the connection between continuous modeling and discretized modeling, this work gives a fresh confirmation of the utility of the concept as a worthwhile adjunct to model based refinement in enlarging the range of problem scenarios that can be comprehensively addressed using model based refinement techniques.

Of course the path we followed in this paper is by no means unique. One could imagine many alternative techniques for the various stages we went through, that one could use in place of the specific approaches that were used here. Equally, we made use of only one very specific result from mathematical control theory to underpin the transition from continuous to discretized modeling. Not only was this a result that turned out to be suboptimal from an engineering viewpoint (in that it provided an exponentially increasing estimate for a quantity that was in fact increasing polynomially), but it was also a result that would not necessarily be particularly applicable in other contexts, relying, as it did, on the fact that the two behaviours that were being compared, started from the same state. Its crucial strong point though, was that it derived an $\mathcal{L}^\infty$ estimate for the difference between the two evolving system states that held throughout the interval under investigation. Not only did this provide an absolute quantitative bound for the difference between the after-states of the interval that could be relied on during subsequent discrete development, but also, in being an $\mathcal{L}^\infty$ bound, it provided a guarantee about the state difference in the interior of the interval, that the subsequent discrete development, which was incapable of taking interior behaviour into account, could then safely disregard, content that the overall system goals were not endangered by this lack of attention.

In more realistic engineering situations, the problem will be less amenable to analytic solution, and the feedback gained from exactly solvable case studies will be invaluable for giving an idea of how good any generically derived bound is likely to be. Thus it is important to investigate a useful number of exactly solvable representative scenarios to build up the required experience. Once a suitable collection of widely applicable and useful results of this kind have been established, the way is open for the incorporation of these into appropriate formal development tools, about which we commented in the previous section.

References

[Abr96] J.-R. Abrial. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, 1996.

[Abr10] J.-R. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.

[ACHH93] R. Alur, C. Courcoubetis, T. Henzinger, and P-H. Ho. Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems. In *Proc. Workshop on Theory of Hybrid Systems*, volume 736 of *LNCS*, pages 209–229. Springer, 1993.

[AD94] R. Alur and D. Dill. A Theory of Timed Automata. *Theor. Comp. Sci.*, 126:183–235, 1994.

[AH93] R. Alur and T. Henzinger. Real-Time Logics: Complexity and Expressiveness. *Inf. and Comp.*, 104:35–77, 1993.

[AH94] R. Alur and T. Henzinger. A Really Temporal Logic. *J. A.C.M.*, 41:181–204, 1994.

[Ahm06] N. Ahmed. *Dynamic Systems and Control With Applications*. World Scientific, 2006.

[AM06] P. Antsaklis and A. Michel. *Linear Systems*. Birkhauser, 2006.

[Ban] R. Banach. Model Based Refinement and the Design of Retrenchments. Available from [Ret].

[Ban10] R. Banach. A Deidealisation Semantics for KAOS. In Lencastre, M. (RE track), editor, *Proc. ACM SAC-10 (RE track)*, pages 267–274. ACM, 2010.

[Bar75] S. Barnett. *Introduction to Mathematical Control Theory*. Oxford University Press, 1975.

[BH95] J. Bowen and M. Hinchey. Seven More Myths of Formal Methods. *IEEE Software*, 12:34–41, 1995.

[BH99a] J. Bowen and M. Hinchey. *High-Integrity System Specification and Design*. Springer, 1999.

[BH99b] J. Bowen and M. Hinchey. *Industrial-Strength Formal Methods in Practice*. Springer, 1999.

[BJ] R. Banach and C. Jeske. Retrenchment and Refinement Interworking: the Tower Theorems. Submitted. Available from [Ret].

[BJP08] R. Banach, C. Jeske, and M. Poppleton. Composition Mechanisms for Retrenchment. *J. Log. Alg. Prog.*, 75:209–229, 2008.

[Bör03] E. Börger. The ASM Refinement Method. *FACJ*, 15:237–257, 2003.

[BPJS07] R. Banach, M. Poppleton, C. Jeske, and S. Stepney. Engineering and Theoretical Underpinnings of Retrenchment. *Sci. Comp. Prog.*, 67:301–329, 2007.

[Bro10] J. Broenink. Embedded Control Software Design with Formal Methods and Engineering Models. *BCS-FACS Evening Seminar, Sept. 2010*, 2010.

[BS03] E. Börger and R.F. Stärk. *Abstract State Machines. A Method for High Level System Design and Analysis*. Springer, 2003.

[But] Butler, M. Private communication.

[But96] R. Butler. NASA Technical Memorandum 110255. An Introduction to Requirements Capture Using PVS: Specification of a Simple Autopilot. Technical report, 1996.

[BZSW] R. Banach, H. Zhu, W. Su, and X. Wu. "Moded Operation, Continuous ASM, and an Approach to Pacemaker Sensing. Submitted.

[BZSW12a] R. Banach, H. Zhu, W. Su, and X. Wu. ASM and Controller Synthesis. In Derrick and Fitzgerald and Gnesi and Khurshid and Leuschel and Reeves and Riccobene, editor, *Proc. ABZ-12*, volume 7316, pages 51–64. Springer, LNCS, 2012.

[BZSW12b] R. Banach, H. Zhu, W. Su, and X. Wu. Continuous ASM, and a Pacemaker Sensing Fragment. In Derrick and Fitzgerald and Gnesi and Khurshid and Leuschel and Reeves and Riccobene, editor, *Proc. ABZ-12*, volume 7316, pages 65–78. Springer, LNCS, 2012.

[Cau] Cau, A., and Moszkowski, B. and Zedan, H. Interval Temporal Logic. http://www.cse.dmu.ac.uk/~cau/papers/itlhomepage.pdf.

[Chi06] C. Chicone. *Ordinary Differential Equations with Applications*. Springer, 2nd edition, 2006.

[Cla87] F. Clarke. *Optimization and Nonsmooth Analysis*. Society for Industrial Mathematics, 1987.

[CLSW97] F. Clarke, Y. Ledyaev, R. Stern, and P. Wolenski. *Nonsmooth Analysis and Control Theory*. Springer, 1997.

[COR+95] J. Crow, S. Owre, J. Rushby, N. Shankar, and M. Srivas. A Tutorial Introduction to PVS. In Larrondo-Petrie France, Gerhart, editor, *WIFT'95: Workshop on Industrial-Strength Formal Specification Techniques*. IEEE Computer Society Press, 1995.

[CS08] J. Cohen and A. Slissenko. Implementation of Timed Abstract State Machines with Instantaneous Actions by Machines with Delays. Technical Report TR-LACL-2008-2, LACL, University of Paris-12, 2008.

[DB01] J. Derrick and E. Boiten. *Refinement in Z and Object-Z: Foundations and Advanced Applications*. Springer-Verlag UK, 2001.

[DB10] R. Dorf and R. Bishop. *Modern Control Systems*. Pearson, 2010.

[DH95] J. D'Azzo and C. Houpis. *Linear Control System Analysis and Design: Conventional and Modern*. McGraw Hill, 1995.

[DHR05] L. Doyen, T. Henzinger, and J.-F. Raskin. Automatic Rectangular Refinement of Affine Hybrid Systems. In *Proc. FORMATS-05*, volume 3829 of *LNCS*, pages 144–161. Springer, 2005.

[DIRR09] F. Dotti, A. Iliasov, L. Ribeiro, and A. Romanovsky. Modal Systems: Specification, Refinement and Realisation. In *Proc. ICFEM-09*, volume 5885 of *LNCS*. Springer, 2009.

[DM05] L. Debnath and P. Mikusinski. *Introduction to Hilbert Spaces with Applications*. Springer, 2005.

[dRE98] W P de Roever and K Engelhardt. *Data Refinement: Model-Oriented Proof Methods and their Comparison*. Cambridge University Press, 1998.

[DTB97] K. Dutton, S. Thompson, and B. Barraclough. *The Art of Control Engineering*. Addison Wesley, 1997.

[Esh09] R. Eshuis. Reconciling Statechart Semantics. *Sci. Comp. Prog.*, 74:65–99, 2009.

[FPW96] G. Franklin, J. Powell, and M. Workman. *Digital Control Systems*. Prentice Hall, 1996.

[FV09] M. Fadali and A. Visioli. *Digital Control Engineering: Analysis and Design*. Academic Press, 2009.

[GM93] M. Gordon and T. Melham. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*. Cambridge University Press, 1993.

[Hal90] J. Hall. Seven Myths of Formal Methods. *IEEE Software*, 5:11–19, 1990.

[Hal07] J. Hall. Realising the Benefits of Formal Methods. *JUCS*, 13:669–678, 2007.

[Har96] J. Harrison. HOL Light: A Tutorial Introduction. In *Formal Methods in Computer Aided Design*, volume 1166 of *LNCS*. Springer, 1996.

[Har07] J. Harrison. Formalisong Basic Complex Analysis. In *From Insight to Proof: Festschrift in Honour of Andrzej Trybulec*, volume 10 of *Studies in Logic, Grammar and Rhetoric*, pages 151–165. University of Białystok, 2007.

[He94] J. He. From CSP to Hybrid Systems. In Roscoe, editor, *A Classical Mind, Essays in Honour of C.A.R. Hoare*, pages 171–189. Prentice-Hall, 1994.

[Hei07] C. Heitmeyer. Formal Methods for Specifying, Validating, and Verifying Requirements. *JUCS*, 13:607–618, 2007.

[Hen96] T. Henzinger. The Theory of Hybrid Automata. In *Proc. IEEE LICS-96*, pages 278–292. IEEE, 1996. Also http://mtc.epfl.ch/~tah/Publications/the_theory_of_hybrid_automata.pdf.

[HJ85] R. Horn and C. Johnson. *Matrix Analysis*. Cambridge University Press, 1985.

[HJ91] R. Horn and C. Johnson. *Topics in Matrix Analysis*. Cambridge University Press, 1991.

[HK97] T. Henzinger and O. Kupferman. From Quantity to Quality. In *Proc. HART-97*, volume 1201 of *LNCS*, pages 48–62. Springer, 1997.

[HMP92] T. Henzinger, Z. Manna, and A. Pnueli. What Good are Digital Clocks? In *Proc. ICALP-92*, volume 623 of *LNCS*, pages 545–558. Springer, 1992.

[HT98] J. Harrison and L. Théry. A Skeptic's Approach to Combining HOL and Maple. *J. Aut. Reas.*, 21:279–294, 1998.

[IEE] IEEE Standard 1474. IEEE Standard for Communications-Based Train Control (CBTC) Performance and Functional Requirements: IEEE Std 1474.1-2004; IEEE Standard for User Interface Requirements in Communications-Based Train Control (CBTC) Systems: IEEE Std 1474.2-2003; IEEE Recommended Practice for Communications-Based Train Control (CBTC) System Design and Functional Allocations: IEEE Std 1474.3-2008.

[Jes05] C. Jeske. *Algebraic Integration of Retrenchment and Refinement*. PhD thesis, University of Manchester, 2005.

[Kal05] R. Kalman. Opening lecture, IFAC World Congress, Prague, Czech Republic, July 4, 2005.

[Kar] Karlsruhe Interactive Verifier. http://www.informatik.uni-augsburg.de/lehrstuehle/swt/se/kiv/.

[Koy92] R. Koymans. *Specifying Message Passing and Time-Critical Systems with Temporal Logic*. LNCS Volume 651, Springer, 1992.

[Kuo92] B. Kuo. *Digital Control Systems*. Oxford University Press, 1992.

[Lan93] S. Lang. *Real and Functional Analysis*. Springer, 1993.

[Les07] D. Lester. Topology in PVS: Continuous Mathematics with Applications. In Rushby and Shankar, editors, *Proc. AFM-07*. ACM, 2007.

[Let01] Letier, E. *Reasoning about Agents in Goal-Oriented Requirements Engineering*. PhD thesis, Dépt. Ingénierie Informatique, Université Catholique de Louvain, 2001.

[LPN11] S. Loos, A. Platzer, and L. Nistor. Adaptive Cruise Control: Hybrid, Distributed, and Now Formally Verified. In Butler and Schulte, editors, *Proc. FM-11*, volume 6664 of *LNCS*, pages 42–56. Springer, 2011. See also: Technical Report CMU-CS-11-107 Carnegie Mellon University (2011).

[Mac09] B. MacCluer. *Elementary Functional Analysis*. Springer, 2009.

[Map] Maple. http://www.maplesoft.com.

[Mat] Mathematica. http://www.wolfram.com.

[Mey] Meynadier, J.-M. Private communication.

[MGL08] A. Matoussi, F. Gervais, and R. Laleau. A First Attempt to Express KAOS Refinement Patterns with Event-B. In Börger, Butler, Bowen, Boca, editor, *Proc. ABZ-08*, volume 5238, page 338. Springer, LNCS, 2008.

[OD08] E.-R. Olderog and H. Dierks. *Real-Time Systems: Formal Specification and Automatic Verification*. Cambridge University Press, 2008.

[Oga08] K. Ogata. *Modern Control Engineering*. Pearson, 2008.

[Oua02] J. Ouaknine. Digitsation and Full Abstraction for Dense-Time Model Checking. In *Proc. TACAS-02*, LNCS, pages 37–51. Springer, 2002.

[Par96] P. Paraskevopoulos. *Digital Control Systems*. Prentice Hall, 1996.

[Pla10a] A. Platzer. Differential-Algebraic Dynamic Logic for Differential-Algebraic Programs. *J. Log. Comp.*, 20:309–352, 2010.

[Pla10b] A. Platzer. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, 2010.

[Pla10c] A. Platzer. Quantified Differential Dynamic Logic for Distributed Hybrid Systems. In Dawar, Veith, editor, *Proc. CSL-10*, volume 6247, pages 469–483. Springer, LNCS, 2010.
[PST96] B. Potter, J. Sinclair, and D. Till. *An Introduction to Formal Specification and Z*. Prentice Hall, 2nd edition, 1996.
[PVS] PVS Homepage. http://pvs.csl.sri.com.
[Pyt99] R. Pytlak. *Numerical Methods for Optimal Control Problems with State Constraints*, volume 1707 of *Lecture Notes in Mathematics*. Springer, 1999.
[RC04] J. Real and A. Crespo. Mode Change Protocols for Real-Time Systems: A Survey and a New Proposal. *Real-Time Syst.*, 26:161–197, 2004.
[Ret] Retrenchment Homepage. http://www.cs.man.ac.uk/retrenchment.
[RY00] B. Rynne and M. Youngson. *Linear Functional Analysis*. Springer, 2000.
[SAZH11] W. Su, J.-R. Abrial, H. Zhu, and R. Huang. From Requirements to Development: Methodology and Example. In *Proc. ICFEM-11*, volume 6991 of *LNCS*, pages 437–455. Springer, 2011.
[Sch01] G. Schellhorn. Verification of ASM Refinements Using Generalized Forward Simulation. *JUCS*, 7:952–979, 2001.
[Sch05] G. Schellhorn. ASM Refinement and Generalizations of Forward Simulation in Data Refinement: A Comparison. *Theor. Comp. Sci.*, 336:403–435, 2005.
[Son98] E. Sontag. *Mathematical Control Theory*. Springer, 1998.
[SS98] E. Sekerinski and K. Sere. *Program Development by Refinement: Case Studies Using the B-Method*. Springer, 1998.
[Sta02] T. Stauner. Discrete-Time Refinement of Hybrid Automata. In *Proc. HSCC-02*, volume 2289 of *LNCS*, pages 144–161. Springer, 2002.
[SV08] A. Slissenko and P. Vasilyev. Simulation of Timed Abstract State Machines with Predicate Logic model Checking. *JUCS*, 14:1984–2006, 2008.
[SYW+11] W. Su, F. Yang, X. Wu, J. Gou, and H. Zhu. Formal Approaches to Mode Conversion and Positioning for Vehicle Systems. In *Proc. 3rd IEEE International Workshop on Security Aspects of Process and Services Engineering (COMPSAC Workshops)*, pages 416–421. IEEE, 2011.
[Szt11] J. Sztipanovits. Model Integration and Cyber Physical Systems: A Semantics Perspective. In Butler and Schulte, editors, *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.
[Tab09] P. Tabuada. *Verification and Control of Hybrid Systems: A Symbolic Approach*. Springer, 2009.
[vL09] A. van Lamsweerde. *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley, 2009.
[Wal98] W. Walter. *Ordinary Differential Equations*. Springer, 1998.
[WD96] J. Woodcock and J. Davies. *Using Z, Specification, Refinement and Proof*. Prentice Hall, 1996.
[Wil07] J. Willems. Open Dynamical Systems: Their Aims and their Origins. Ruberti Lecture, Rome, 2007. http://homes.esat.kuleuven.be/~jwillems/Lectures/2007/Rubertilecture.pdf.
[ZHR91] C. Zhou, T. Hoare, and A. Ravn. A Calculus of Durations. *Inf. Proc. Lett.*, 40:269–276, 1991.

A. Some Functional Analysis

In this Appendix, we cover the crucial results from the rigorous theory of differential equations that enable us to formally connect the continuous and discrete control problems via an appropriate retrenchment in a highly generic way. The essential elements that we use are to be found in [Pyt99], building on foundations laid in [Cla87], and later in [CLSW97]. We assume known some elementary facts from functional analysis, such as can be found in any introductory text on the subject, e.g. [Lan93, RY00, Mac09, DM05]. Two mathematical control theory texts which review the needed material in a suitable way are [Ahm06, Son98].

In the sequel, $||x||$ is the $\mathcal{L}^\infty$ norm of the (finite dimensional vector valued) measurable function[27] x, defined on the measure space $\mathcal{T}$, which is a finite period of time represented as a finite interval of $\mathbb{R}$ with Lebesgue measure, in our case the interval $[0 \dots T_{Stop}]$. Other $\mathcal{L}^p$ norms are written $||x||_p$. Partial derivatives with respect to state variables are written as subscripts, thus ϕ_x is the partial derivative of ϕ with respect to x. Time derivatives are written with an overdot, thus $\dot{\varphi}$ is the time derivative of φ.

In [Pyt99], much use is made of a "basic existence/uniqueness theorem" derived from [Cla87], and which, adapted for our purposes, runs as follows.[28]

Theorem A.1. Let $\varphi : \mathcal{T} \times \mathbb{R}^n \to \mathbb{R}^n$ be such that $\varphi(\cdot, x)$ is measurable and $\varphi(t, \cdot)$ is continuously differentiable. Assume that there exists a constant $k_\varphi < \infty$ such that $||\varphi_x(t,x)|| < k_\varphi$ for all $(t,x) \in \mathcal{T} \times \mathbb{R}^n$. Then, if there is an absolutely continuous function z such that $\int_{\mathcal{T}} ||\dot{z} - \varphi(t, z(t))|| < \infty$, then there is a unique absolutely continuous function $\tilde{z}$ such that:

$$\dot{\tilde{z}}(t) \;=\; \varphi(t, \tilde{z}(t)) \quad \text{a.e. on } \mathcal{T} \tag{139}$$

[27] In this Appendix, we no longer write the state vector in bold.

[28] In [Pyt99] the author restricts to a unit time interval — we have reinstated a generic time interval. This gives rise to the T_{Stop} in (142) which is absent from the corresponding formula of [Pyt99]. Also, the continuous differentiability assumed in Theorem A.1 is more than is required to prove it. A Lipschitz condition on the state and measurability in time would be sufficient, but we retain the assumptions given for conformity with [Pyt99].

$$\tilde{z}(0) \;=\; z(0) \tag{140}$$

and

$$||\tilde{z}-z|| \;\leq\; e^{K_\varphi}\int_{\mathcal{T}}||\dot{z}(t)-\varphi(t,z(t))|| \tag{141}$$

where

$$K_\varphi \;=\; \int_{\mathcal{T}} k_\varphi \;=\; k_\varphi\, T_{Stop} \tag{142}$$

In [Pyt99], the author applies this to continuous control problems, which, in the state space picture, can be posed as the following initial value problem.

$$\dot{x}(t) \;=\; f(t,x(t),u(t)) \quad \text{a.e. on } \mathcal{T} \tag{143}$$

$$x(0) \;=\; x_0 \tag{144}$$

In (143), $f : \mathcal{T}\times\mathbb{R}^n\times\mathbb{R}^m \to \mathbb{R}^n$ is the control function, which specifies how, at a given time t, with a given current state $x(t)$ (of dimension n), and external control input $u(t)$ (of dimension m), the time derivative of $x(t)$ is determined.

The control inputs are defined via a set Ω of control input values. The control functions $u(t)$ are thus:

$$\mathcal{U} \;=\; \{u \,|\, u \in \mathcal{L}^1[\mathcal{T}] \wedge u(t) \in \Omega \text{ a.e. on } \mathcal{T}\} \tag{145}$$

We assume the following about control functions:

H1 Ω is a compact set of control input values; $f(\cdot,x,u)$, $f(t,\cdot,u)$ and $f(t,x,\cdot)$ are differentiable, with the x and u derivatives satisfying uniformly bounded local Lipschitz bounds $k_x(t) \leq K_x$ and $k_u(t) \leq K_u$ respectively.

The result that is crucial for us is contained in Prop. 1.1 of [Pyt99]. We just need part (ii) of it. In [Pyt99] the proof of this part is omitted (due to its triviality from that author's perspective). We supply the missing details.

Proposition A.2. Assume **H1**. For each $u \in \mathcal{U}$, (143)-(144) has a unique solution x^u in the class of absolutely continuous vector valued functions on $\mathcal{T}$. Furthermore, there exists a constant *K2* such that if u and u_D are both in $\mathcal{U}$, then:

$$||x^u - x^{u_D}|| \;\leq\; K2\,||u-u_D||_2 \tag{146}$$

Proof: Pick two fixed control input functions $u(t)$ and $u_D(t)$ in $\mathcal{U}$. We apply Thm. A.1 with $z(t) = x^{u_D}(t)$ and $\varphi(t,x(t)) = f(t,x(t),u(t))$. Let K_f be the K_φ for this choice of φ. We thus regard $x^{u_D}(t)$, the exact solution to the $f(t,x(t),u_D(t))$ control problem, as an approximate solution to the $f(t,x(t),u(t))$ control problem. This gives us that:

$$\begin{aligned}
||x^u - x^{u_D}|| \;&\leq\; e^{K_f}\int_{\mathcal{T}}||\dot{x}^{u_D}(t) - f(t,x^{u_D}(t),u(t))|| \\
&=\; e^{K_f}\int_{\mathcal{T}}||\dot{x}^{u_D}(t) - f(t,x^{u_D}(t),u_D(t)) + f(t,x^{u_D}(t),u_D(t)) - f(t,x^{u_D}(t),u(t))|| \\
&=\; e^{K_f}\int_{\mathcal{T}}||f(t,x^{u_D}(t),u_D(t)) - f(t,x^{u_D}(t),u(t))||
\end{aligned} \tag{147}$$

We dropped the two terms $\dot{x}(t)^{u_D} - f(t,x^{u_D}(t),u_D(t))$ in (147) since $x^{u_D}(t)$ is a solution to the $f(t,x(t),u_D(t))$ control problem. Now we use the Lipschitz properties of f to get

$$|f(t,x^{u_D}(t),u_D(t)) - f(t,x^{u_D}(t),u(t))| \;\leq\; k_u(t)\,|u(t)-u_D(t)| \tag{148}$$

Inserting this in (147) we get

$$||x^u - x^{u_D}|| \;\leq\; e^{K_f}\int_{\mathcal{T}} k_u(t)\,|u(t)-u_D(t)| \tag{149}$$

Being an integral of the product of two real measurable functions over a finite interval, the integral (149) is now an inner product in $\mathcal{L}^2[\mathcal{T}]$, so we can apply the Cauchy-Schwartz inequality to get

$$||x^u - x^{u_D}|| \;\leq\; e^{K_f}||k_u||_2||u-u_D||_2 \tag{150}$$

where

$$||k_u||_2 \;=\; \sqrt{\int_{\mathcal{T}}|k_u(t)|^2} \tag{151}$$

Thus for the *K2* claimed in (146) it is sufficient to set

$$K2 = e^{K_f} ||k_u||_2 \tag{152}$$

□

Equation (152), offering a specific bound for the proportionality constant between $||x^u - x^{u_D}||$ and $||u - u_D||_2$, was our objective for this Appendix.