A Computational Model of Time for Stiff Hybrid Systems Applied to Control Synthesis

Pieter J. Mosterman[a], Justyna Zander[b], Gregoire Hamon[a], Ben Denckla[c]

[a] *The MathWorks, Natick, MA 01760, USA*
(e-mail: pieter.mosterman@mathworks.com, gregoire.hamon@mathworks.com
fax: +1 508 647 7012).
[b] *Fraunhofer Institute FOKUS, 10589 Berlin, Germany*
HHI, Harvard University, Cambridge, MA 02138, USA
(e-mail: justyna.zander@gmail.com)
[c] *Denckla Consulting, Los Angeles, CA, USA*
(e-mail: bdenckla@alum.mit.edu)

Abstract

Computation has quickly become of paramount importance in the design of engineered systems, both to support their features as well as their design. Tool support for high-level modeling formalisms has endowed design specifications with executable semantics. Such specifications typically include not only discrete-time and discrete-event behavior, but also continuous-time behavior that is stiff from a numerical integration perspective. The resulting *stiff hybrid dynamic systems* necessitate variable-step solvers to simulate the continuous-time behavior as well as solver algorithms for the simulation of discrete-time and discrete-event behavior. The combined solvers rely on complex computer code which makes it difficult to directly solve design tasks with the executable specifications. To further leverage the executable specifications in design, this work aims to formalize the semantics of stiff hybrid dynamic systems at a declarative level by removing implementation detail and only retaining 'what' the computer code does and not 'how' it does it. A *stream-based approach* is adopted to formalize variable-step solver semantics and to establish a computational model of time that supports discrete-time and discrete-event behavior. The corresponding declarative formalization is amenable to computational methods and it is shown how *model checking* can automatically generate, or *synthesize*, a feedforward control strategy for a stiff hybrid dynamic system. Specifically, a stamper in a surface mount device is controlled to maintain a low acceleration of the stamped component for a prescribed minimum duration of time.

Keywords: Computational methods, Computer simulation, Computer-aided control system design, Embedded systems, Numerical simulation, Synchronous data flow, Systems design, Variable-structure systems, Verification, Zero crossings

1. Introduction

Over the past decades, the feature set of engineered systems has rapidly increased. To a large extent this increase has been enabled by merit of the flexible realization of functionality in embedded software. Simultaneously, Model-Based Design (e.g., Friedman and Ghidella (2006); Nicolescu and Mosterman (2009); Potter (2004)) has become essential to competitively, if not just successfully, engineer embedded systems (e.g., Jones (2005)). Computation can then be identified as the main driver that enables (i) the design of modern systems and (ii) an unparallelled feature differentiation. This trend has put forward a distinct need to formally capture the computations that underpin an executable model which is particularly challenging for the computational approximations of continuous-time behavior. This article attempts to formalize these computations, especially to support systems that are stiff from a numerical integration perspective and that further comprise discrete event behavior, resulting in stiff hybrid dynamic systems. The formalization intends to capture 'what' the effects of the necessary computations are without the necessity to account for specifically 'how' the computations are evaluated and implemented.

1.1. Model-Based Design for Engineered Systems

To understand this objective, consider engineered systems such as *cyber-physical systems* that include physics, computation, and networking aspects. The design of such systems is abstractly depicted in Fig. 1. In the middle row of the figure, a triangle that increases in size from left to right illustrates how the specification of an engineered system under design is increasingly refined and extended. The additional detail becoming available in each step is illustrated by an added layer to the specification.

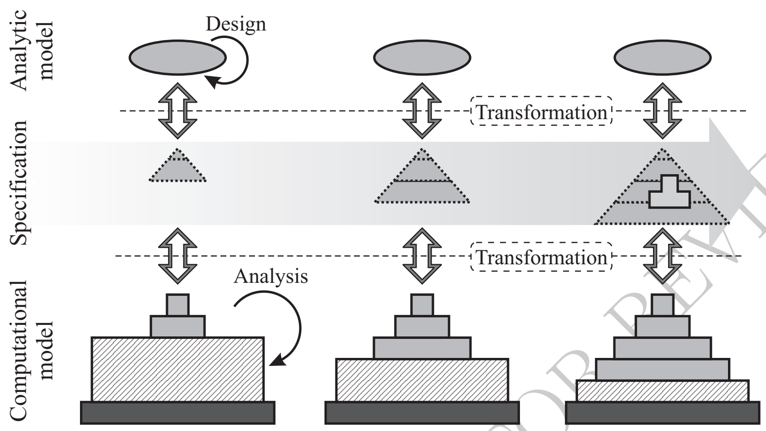


Figure 1: Engineered system design

Traditionally, such specifications have been paper documents. To perform design tasks, an analytic representation would be derived such as, for example, a particular control law architecture. This analytic model, illustrated in the top row in Fig. 1, may comprise a linear approximation of the *plant* to be controlled so as to facilitate linear control design methods (e.g., Åström and Wittenmark (1984)). The control structure and parameters so found to satisfy the specification are then integrated back into a now more detailed specification with an additional level of complexity.

The performance of the control on a more detailed plant model can then be studied by computational methods. Sophisticated simulation technology allows the study of very detailed plant effects based on a computational model, as indicated in the bottom row in Fig. 1. In domains such as Scientific Computing where large scale models are simulated in great detail such as described by Post and Votta (2005), a computational approximation is typically still obtained by software engineers producing low-level code, such as FORTRAN. In the domain of control system design, however, technical computing environments are available with a built-in code base that supports simulation of high-level (often graphical) modeling formalisms. This code base (typically C), illustrated in Fig. 1 by the hashed rectangles, is tailored to an implementation technology such as, for example, an Intel x86 microprocessor, illustrated by the solid dark rectangle at the bottom. Note that as the specification becomes more detailed, the distance to the underlying technology reduces and the code base required for simulation becomes smaller. The support for high-level modeling formalisms allows pertinent parts of the specification to be easily transformed into a computational counterpart for analysis based on computational simulation. The results of such analysis may then help modify the control parameters to account for more detailed phenomena such as nonlinear effects because these are typically supported by computational simulation. Consequently, models of higher fidelity are employed and so a more robust design is obtained (e.g., Looye et al. (1998)), which can then be integrated back into the more detailed specification.

The specification continues to be extended and refined based on such analytic and computational approximation till it is sufficiently detailed to be implemented. In Fig. 1, this implies that the base of the specification triangle has reached a certain implementation technology (e.g., a target embedded processor). Notice that the outline of the specification triangles in Fig. 1 has a dotted line style. This is to indicate that in general the specifications are inherently vague because they require *interpretation*. There may be well-defined parts, though, such as a computational procedure, and this is illustrated by the staircase element inside the right-most triangle. One cause for vagueness stems from the differential equation behavior that typically is extensive (comprising many state variables), has nonlinearities, and contains switching effects. As such, no closed-form solution exists, and only a computational approximation can provide a precise definition.

As computational approximations have become increasingly available and powerful, they have steadily gained in importance till they triggered an entire paradigm shift. Instead of being a tool for analysis of the prime deliverables in document form, the computational models have become the prime deliverables themselves. This is illustrated in Fig. 2 where the specification triangles in Fig. 1 have been replaced by their computational approximations. Because computational, these specifications are now immediately executable. Note that a system specification still contains many artifacts other than computational models. The illustration only serves to highlight how computational models and, indeed, computational semantics, have become part of the primary path in control system design.

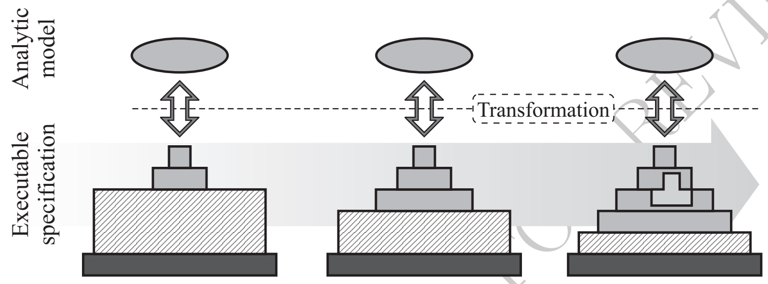


Figure 2: Model-Based Design of engineered systems

This status of computational models as first-class deliverables is the foundation of Model-Based Design and enables:

- An interconnected infrastructure for management of design artifacts with support for fine-grain active links. For example, requirements for production code can be linked to the specific code lines with automatic tracing between them.
- Generating behavior from specifications to provide early insight into design decisions, share unambiguous information between design teams, and automate tasks such as optimization and experimental design.
- Verification of domain specific constraints that are static as well as dynamic in nature. This includes statically checking not only static semantics (e.g., type checks) but also dynamic semantics (e.g., indexing an array out of its bounds).

- Automated completion of a partial design and generation of an implementation by means of transformations. For example, data types of variables in a design can be automatically determined while imperative source code for an implementation can be generated from a declarative design.

1.2. Formalizing Computational Approximations for Design

A contribution of the work presented here is a basis to further leverage computational models. In particular, with physics comprising an essential part of embedded systems, there is an interest in also defining the semantics of the corresponding modeling formalisms as denotational composition of functions on streams. To apply Computer Science methods such as *model checking* (e.g., Andersson et al. (2002)) for purposes of control system design, the computational approximation must be formalized. In the context of the work presented here, this then applies to the computational methods that are employed to compute a simulation of a continuous-time model, potentially with stiff numerical behavior and interspersed discrete events. Since physics is well modeled by differential equations (e.g., Breedveld (1984); Cellier et al. (1996)), either ordinary differential equations (ODEs) or differential and algebraic equations (DAEs), a unifying framework must encompass the computational semantics of discrete-time and continuous-time models. Moreover, discrete state changes are often part of the otherwise continuous-time models of physics, for example to capture mode changes in models of a component such as a valve or diode. Thus, the framework should further support defining semantics of discrete-event models.

The work presented here then aims to formulate the computational semantics of the continuous-time part of a model when it is specified by differential equations. It recognizes that the numerical integration algorithms employed for computational simulation are key in precisely capturing how a model executes. In order to honor the continuity requirements of differential equations (such as a continuous domain and time-derivative constraints), a multi-stage variable-step solver is studied. The specific challenge that is addressed in this paper is providing a declarative specification of a variable-step numerical integration algorithm such that it can be integrated with discrete-time and discrete-event behavior to ultimately enable the systematic formalization of the computational approximation. Such a formalization then unlocks the potential for using Computer Science methods such as model checking for synthesizing control of stiff hybrid dynamic systems.

The necessity and value of the contribution can be conceptually depicted as in Fig. 3. Following Fig. 2, the analysis based on simulation of a computational model includes a large software stack (Fig. 3(a)) that is necessary to make a model in a high-level formalism executable. This software stack is akin to the execution engine code of a product such as Simulink® (2008) and generally prohibitively complex for model checking. This problem is solved by introducing progressively higher levels of description such as an operational specification in Fig. 3(b) that captures the control flow as a transition system but that removes the implementation complexity of the code. At an even higher level, a denotational specification in Fig. 3(c) maintains only the function of the computations as a declarative system of equations but that removes the complexity of sequences of state changes as in the transition system. As a result, the complexity that the analysis must account for is reduced to the levels above the dashed line in Fig. 3(c). This paper illustrates how computational methods can then be applied for *design* to systems that would be prohibitively complex otherwise.

To derive the denotational specification, the work presented here builds on previous work by Denckla and Mosterman (2006). There, the computational semantics of discrete-time modeling formalisms such as time-based block diagrams in Simulink was defined as a composition of pure functions on streams. The strict functional approach allows capturing the meaning of a model in a denotational sense (i.e., *what* it does) as opposed to an operational sense (i.e., *how* it does something) (e.g., Nielson and Nielson (1992); Zhang and Xu (2004)). This decouples a specification from its implementation and in general provides a representation that is easier to understand and reason about, for one because with pure functions there is no internal state to account for, as eloquently presented by Backus (1978).

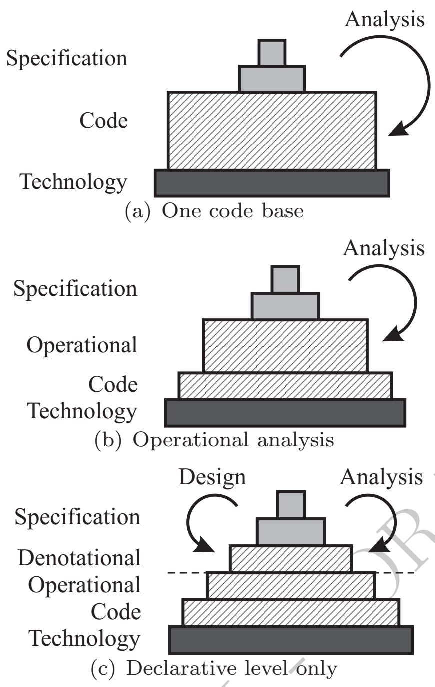


Figure 3: Reducing complexity.

In its application, the computational semantics are represented in a strict subset of Simulink blocks. This subset does not include the continuous-time integrator block so that the integration behavior that is otherwise approximated by the code of the Simulink execution engine now becomes explicit as a declarative block diagram. As per Fig. 3(c), this then allows computational methods of Simulink® Design Verifier™ (which is not applicable to models with continuous-time integration) to synthesize a control law. To illustrate, a sequence of feed-forward control actions, a *control force profile*, is synthesized for a surface mount device that stamps components onto a printed circuit board. Such a device is required to maintain an acceleration profile of the component for a minimal time to allow the stamped component to adhere to the board. Using the methodology described in this paper, Simulink® Design Verifier successfully produces a control force profile that achieves this requirement. This illustrates how the methodology enables industrial control engineering practitioners to exploit the models they have in a novel manner.

1.3. Structure of the Presentation

The presentation is structured so that Section 2 introduces the computational solution of differential equations by means of numerical integration. In particular, the approximation error that results from the numerical integration is briefly discussed. In Section 3, two numerical integration schemes are developed to constitute the basis of a computational semantics. Section 4 then presents a functional representation of a variable-step solver based on these integration schemes. In Section 5, a case study combines the variable-step solver with discrete-time and discrete-event model parts in order to illustrate how the resulting abstract notion allows for a new kind of control synthesis. The approach also unlocks the potential for a methodological process for solving control engineering problems that can be applied in parallel to traditional methods. Section 6 evaluates how the presented work advances previous achievements as well as its more general contribution. Section 7 concludes and outlines future work.

2. Computational Approximation From Numerical Integration

In the design of cyber-physical systems, such as high-integrity embedded control systems, a model of the physics that the information components interact with is indispensible (e.g., Åström and Wittenmark (1984); SC-167 (1992); High Confidence Software and Systems Coordinating Group (2009)). Given the macrophysical principles of *conservation of energy* and *continuity of power* (e.g., Paynter (1961); Falk and Ruppel (1976)), dynamic models of physical systems are often represented by differential equations, possibly supplemented with algebraic constraints (e.g., to formulate balance equations). These models can be designed based on first principles, such as the laws of physics. Parameters are measured or estimated and once a system of equations is arrived at, it is common to further tune the parameters to best fit the model to measured data. In the extreme case of empirical models, only the order of a model may be provided as an approximation of required detail.

This is illustrated in Fig. 4 where a physical system as shown on the left-hand side may be modeled based on first principles as a system of differential equations, as shown in the center. The *meaning* of such a physical system is indicated by measurements of pertinent variables. The relation between the physics and measurements is informally indicated as a denotation. Likewise, the meaning of the system of differential equations is given by the trajectories that it embodies. These trajectories are *validated* to match the corresponding measurements.

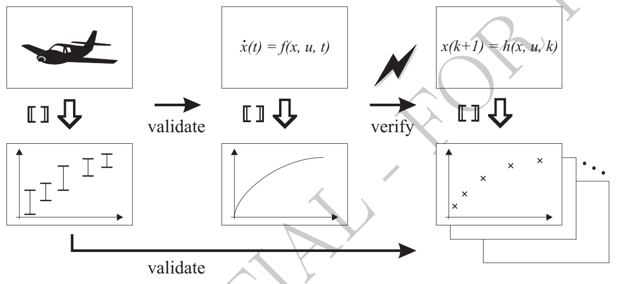


Figure 4: The validated computational representation deviates from its verified representation

When large systems are composed or in general when no analytic solution to the differential equations exists, the parameters are fit based on computational simulation of the differential equations. This is indicated by the computational model on the right-hand side in Fig. 4. The structure of this model and initial parameter values are typically *verified* to correspond to the model that is based on first principles. Since the parameter estimation is based on validating the computational model against the original measurements, as indicated by the bottom arrow, this model may significantly change in order to obtain a best parameter fit. The computational model then starts to lose its correspondence with the model that is based on first principles till at some point the computational model becomes the single most important artifact used for further analysis, synthesis, and design effort.

As a testimony to the absence of an innate and unique separation between solver and model, the computational model then incorporates the computational characteristics of the numerical solver that generates the simulation that is used to fit the parameters. So, parts of a model may be moved to the solver and vice versa (e.g., Schiela and Olsson (2000)).

Though the mathematical derivation of solver equations provides an estimate of the error bounds, and as such it can be claimed that the solver semantics are defined at the mathematical level, these error bounds are only local. Guckenheimer (2002) elaborates that the global error of numerical integration is unknown for most practical cases. For example, consider an ideal inductor/capacitor oscillator

$$\begin{cases} p = -\dot{q} \\ q = \dot{p} \end{cases} \tag{1}$$

where p may represent flux, q may represent charge, and where the mass and capacitance parameters are chosen to be 1. The initial flux is chosen $p_0 = 1$ and the initial charge $q_0 = 0$. This system of equations can be solved in Simulink with a variable-step Dormand-Prince integration method (`ode45`) that computes fourth and fifth order Runge-Kutta solutions and adapts the step size based on the difference. The solution exhibits a decrease in energy over time, as shown by the solid line in Fig. 5.

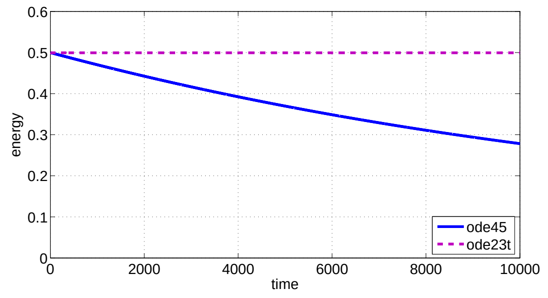


Figure 5: Numerical simulation of total energy in an ideal oscillator

Numerical dissipation is typical for integration schemes, where the mathematical theory is mostly concerned with the accuracy of the integration algorithm as the discretization tends to 0. Dynamic systems theory in turn, is more concerned with asymptotic behavior but finds it difficult to determine the long time error in general. Qualitative analyses exploit structure in the underlying problem and may be the best approach. For example, Sanz-Serna (1992) provides an overview of how symplectic integrators preserve the properties of Hamiltonian systems such as the ideal oscillator in (1). Fig. 5 illustrates this by the dotted line, which is a solution generated by numerical integration with the modified trapezoidal scheme in Simulink (`ode23t`) that according to Shampine et al. (1999) eliminates numerical damping.

Shampine et al. (1999) continue to state that numerically damped behavior at infinity may, in fact, be desirable. Indeed, symplectic integrators have limited applicability in general, which brings about a situation where mathematical semantics of the long time error behavior of general-purpose solvers are still not well developed. This is even more pronounced for solvers with variable step and sophisticated error control (e.g., Bujakiewicz (1994); Petzold (1982)). Heuristics to improve performance for classes of systems further exacerbate the matter. Moreover, to be able to efficiently handle continous-time behavior interspersed with discrete changes, these solvers interact with algorithms to accurately detect and locate when discrete events occur. The delicate interplay between these different algorithms with different error bounds and convergence characteristics makes it difficult to establish a comprehensive mathematical analysis. This holds especially true in the face of potentially infinite sensitivity because of discrete state changes. In order to precisely define the semantics of a model that relies on a given solver, though, an accurate definition of the solver behavior is imperative. A computational semantics then provides a representation for analysis at a useful level without attempting to solve the dynamic systems problems.

Not any less important a ground for developing a computational semantics is the intent to establish a framework that allows integrating continuous-time semantics with discrete-time and discrete-event semantics of various formalisms. In this regard, a computational representation of a solver may facilitate capturing the semantics of combined formalisms, which enables a sound approach to the study of the intricacies that emerge from such combinations. Given that discrete-time behavior is well defined by functions that operate on streams (e.g., Caspi and Pouzet (1997); Reekie (1994)), unification can be achieved by choosing stream-based functions to capture the computational semantics of continuous-time behavior as well. This unified framework then enables a comprehensive mathematical consideration of continuous-time, discrete-time, and discrete-event semantics as a composition of pure functions.

3. A Computational Model of Continuous Time

To obtain a computational model of time, a fixed-step Euler and trapezoidal integration scheme are first reviewed. Next, a variable-step approach is outlined.

3.1. Fixed-Step Integration

Numerical integration is performed in a number of stages at which the dynamic system is evaluated. A single-stage and a multi-stage approach follow after introducing some preliminaries.

3.1.1. Preliminaries

Disregarding the forcing function for now, a continuous-time function can often be represented as an ODE

$$\frac{dx}{dt} = \dot{x} = f(x); \quad f : R^n \to R^n \tag{2}$$

where x is the state vector. The vector field (2) that this defines is assumed to be Lipschitz continuous and to satisfy the usual conditions for existence and uniqueness of a solution. In a neighborhood of $R^n \times 0$, this field then has a unique flow $\Phi : R^n \times R \to R^n$ such that $\Phi(x, 0) = x$ and $\dot{\Phi}(x, t) = f(\Phi(x, t))$. Since there rarely are explicit formulae for Φ in terms of f, iterative numerical integration algorithms are the norm for obtaining solutions over time based on discrete approximations.

3.1.2. Single-Stage Integration

To compute the time, t, map for each iteration, a Forward Euler integration scheme approximates the state that results from making a time step of magnitude h as

$$x((k+1)h) = x(kh) + h\dot{x}(kh) = x(kh) + hf(x(kh)) \tag{3}$$

with k the natural numbers representing the iteration step.

To investigate the numerical accuracy of this method, the Taylor series of f can be expanded around kh to determine the value at $(k+1)h$ (note that $(k+1)h - kh = h$)

$$x((k+1)h) = x(kh) + \frac{\dot{x}(kh)}{1!}h + \frac{\ddot{x}(kh)}{2!}h^2 + \dots \tag{4}$$

This agrees to the second degree with the Forward Euler approximation in (3). The estimate of the error per time step (the local error) then becomes $\frac{\ddot{x}(kh)}{2}h^2$ or $O(h^2)$.

The long time error can be investigated by first defining a discrete map of step h, $E_h(x) = x + h \cdot f(x)$. To integrate to a point in time, t_e, this map can be iteratively applied $E_h^{k+1}(x) = E_h(E_h^k(x))$ with $E_h^0(x) = x$. Now, by reducing the step size $h \to 0$ and taking $l \to \infty$ steps such that $l \cdot h = t_e$, the iterative solution $E_h^l(x) \to \Phi(x, t_e)$.

The compounded error bound at t_e becomes $l\frac{\ddot{x}(kh)}{2}h^2$. Though the error can be made arbitrarily small by reducing h, in turn l becomes arbitrarily large. A large l has two complications: (i) an arbitrarily large error from floating point computations is introduced and (ii) the computation becomes arbitrarily slow because each of the l evaluations requires a finite amount of computation.

3.1.3. Multi-Stage Integration

To mitigate the error and performance problems, at least to an extent, a multi-stage solver can be applied. For example, a trapezoidal integration scheme employs the average of the gradient at the beginning and end point of the integration step as

$$x((k+1)h) = x(kh) + \frac{h}{2}\left(\dot{x}((k+1)h) + \dot{x}(kh)\right). \tag{5}$$

This can be rewritten to

$$x((k+1)h) = x(kh) + h\dot{x}(kh) + \frac{h^2}{2}\left(\frac{\dot{x}((k+1)h) - \dot{x}(kh)}{h}\right) \tag{6}$$

and with a finite difference approximation $\ddot{x}(kh) = \dfrac{\dot{x}((k+1)h) - \dot{x}(kh)}{h} + O(h)$ this matches the Taylor series up to the third degree. The error term then becomes of order $O(h^3)$. As a result, the error converges an order of magnitude quicker to 0, thereby reducing the size of l to get to t_e with the same error bound, while only requiring a linear increase in the number of computations.

The drawback is the use of $\dot{x}((k+1)h)$ on the right-hand side, which because of $k+1$ leads to an implicit integration scheme. This can be solved by computing a Forward Euler approximation $\dot{x}((k+1)h) = f(x(kh) + h\dot{x}(kh))$ to obtain an explicit scheme again.

It is important to note that the first-order and higher-order derivatives impose continuity constraints on the continuous-time behavior. Mixing discretized continuous-time with discrete-time behavior may invalidate the mathematical assumptions and corresponding error bounds.

3.2. Variable-Step Solver

Because of the inverse relation between the step size h and the rate of change in $f(x)$, the step size h can be varied over time as $f(x)$ changes, without compromising the local error bound. For x where $f(x)$ changes with a relatively high rate with respect to t, the system is said to be *stiff*. The solver can thus selectively choose small steps in stiff intervals, while larger steps can be taken elsewhere to improve efficiency.

Integration schemes with an adaptive step size are typically referred to as *variable-step solvers*. An estimate of the error term may be responsible for the change in step size during integration. In some approaches, the error term is approximated by evaluating the difference between the change of x as computed by two different numerical integration algorithms (e.g., the Dormand-Prince method). If this difference exceeds a given threshold, the step size chosen is reduced and the evaluation performed anew.

4. A Functional Variable-Step Solver

Previous work by Denckla and Mosterman (2008) developed a combined stream-based and state-based approach to defining the computational semantics of block diagrams. A functional semantics (i.e., *without* explicit state), BdFun, and a systems semantics (i.e., *with* explicit state), BdSys, were specified in the general *lambda calculus* (e.g., Peyton-Jones (1987)) framework of computation. The functional language Haskell (e.g., Jones (2003)) was chosen for the implementation which allows defining a stream as a potentially infinite (because of *lazy evaluation*) list of values. Embedding state-based semantics into a stream-based semantics was achieved by hierarchical decomposition. This, in turn, supported the implementation of a variable-step solver as a system with explicit state so as to let the variable-step solver manipulate the state freely.

The work presented here aims at eliminating the strict decomposition boundary around the variable-step solver by providing it as a stream-based representation. As such, reasoning about the continuous-time aspects embedded in a discrete-time model becomes completely transparent.

A variable-step solver is then represented as a pure function (i.e., without side effects), g, on an input stream, u, returning an output stream, y, as in

$$y = g(u). \tag{7}$$

To implement an explicit variable-step solver based on the Forward Euler and trapezoidal integration schemes of Section 3, a two-stage evaluation is required. The first stage computes the Euler approximation and the second stage employs this approximation to compute the average gradient over the integration step for the trapezoidal approximation. A functional implementation of the solver cannot rely on internal state to reinstate the values at the beginning of the integration step. Instead, the computed change in state is subtracted if the step size must be reduced. This results in the Euler integration scheme

$$y_{euler}(e) = \begin{cases} \sum_{i=1}^{e} u(i)h(i) - u(i-2)h(i-2)p(i) & \textit{if odd(e)} \\ y_{euler}(e-1) & \textit{otherwise} \end{cases} \tag{8}$$

with e the evaluations as natural numbers larger than 0 and where p is 1 if the step size h must be reduced and 0 otherwise. The undefined initial values ($u(-1)$, $u(0)$, $h(-1)$, and $h(0)$) are taken to be 0. The function *odd* returns *true* for odd values of e and *false* otherwise. Note that the Euler integration is computed every other evaluation to allow the two-stage nature of the trapezoidal scheme.

The trapezoidal approximation adds the contribution at the beginning and end of the integration step based on the same integration step size. If necessary, the state is reinstated at the beginning of the integration step by subtracting the aggregate contribution computed for the previous step. This results in the following integration scheme

$$y_{trap}(e) = \sum_{i=1}^{e} \frac{(u(i-1)+u(i))h(i-1)}{2} - \frac{(u(i-3)+u(i-2))h(i-3)}{2}p(i-1) \tag{9}$$

where undefined initial values can be taken to be 0.

The contributions of the Euler and of the trapezoidal approximations over the integration step are then compared based on the difference in the contribution to each of the states

$$d(e) = (u(e-3)+u(e-2))\frac{h(e-3)}{2} - u(e-2)h(e-2) \tag{10}$$

If the maximum of each of the absolute differences, $|d(e)|$, is less than a predefined tolerance, *tol*, the step is 'accepted' and time moves forward. Otherwise, the time step is reduced. The acceptance test is implemented by the variable p as

$$p(e) = \begin{cases} 0 & \textit{if max}(|d(e)|) < \textit{tol} \\ 1 & \textit{otherwise} \end{cases} . \tag{11}$$

The step size is adapted based on bisection, starting from the maximum step size, h_{max}, as prescribed by the user

$$h(e) = h_{max}(1 - p(e)) + \frac{h(e-1)}{2} p(e). \tag{12}$$

The solver output, y, alternates between the Euler approximation (to enable the trapezoidal scheme) and the trapezoidal approximation, where the trapezoidal approximation is considered to be more accurate

$$\begin{aligned} y(2e+1) &= y_{euler}(2e+1) \\ y(2e+2) &= y_{trap}(2e+2) \end{aligned} \tag{13}$$

These equations are implemented in Simulink, using the *Memory* block as a 'pre' operator (a function $y(e) = g_{pre}(u(e))$ that produces $y(e) = u(e-1)$). The evaluations are performed iterating on a discrete evaluation step with nominal value, 1.

5. Control Synthesis for a Stiff Hybrid Dynamic System

The behavior of the solver developed in Section 4 is studied based on a surface mount device (SMD) that stamps components onto a printed circuit board (PCB). For an overview of the various SMD configurations, see Ayob (2005). The system includes stiff behavior when the stamper is in contact with the board and discrete-event behavior when contact with the board is made. Moreover, upon contact, discrete-time control attempts to keep acceleration low for a specific duration so that the component can attach itself to the board.

5.1. The System Under Control

SMDs are used to manufacture PCBs by rapidly stamping components such as integrated circuits (ICs), resistors, and capacitors onto the board. The machines operate at high speeds with the time of one stamping cycle in the order of tens of milliseconds. The stamper of the SMD studied here is depicted in Fig. 6. At the bottom it shows an IC held against the stamper by an underpressure created by a vacuum nozzle. A pipette connected tot he nozzle can move the held component up and down. By controlling the force with which the pipette moves a component, the component is brought to the PCB as quickly as possible. Upon contact, the force exerted by the stamper must be such that the acceleration of the component is sufficiently low over a prescribed duration of time to enable the component to attach to the board.

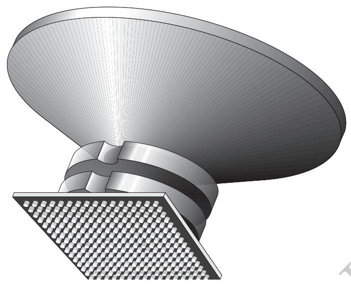


Figure 6: Surface mount device

5.1.1. Modes of Continuous-Time Behavior

A model of the stamping action is given in Fig. 7. A rigid body with mass m represents the component that is moved to a printed circuit board with a control force $F_{control}$. The position x indicates the location of the component. A contact switch, $Sw_{contact}$, activates the contact behavior between the component and the board. The contact behavior is modeled as a stiff spring, C, and damper, R, that combine to exert a reaction force, F_{board}.

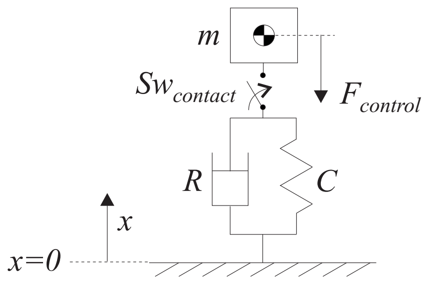


Figure 7: Model of the surface mount device action

The behavior of the overall system can be represented as an explicit ordinary differential equation on the position of the component, x, and the velocity of the component, v,

$$\begin{aligned} \dot{x}(t) &= v(t) \\ \dot{v}(t) &= \frac{F_{control}(t)+F_{board}(t)}{m} \end{aligned} \tag{14}$$

with the forces as a forcing term. The differential equations can be discretized by the solver of Section 4 based on the following mapping of variables

$$u = \begin{bmatrix} 1 \\ \dot{x} \\ \dot{v} \end{bmatrix},\ y = \begin{bmatrix} t \\ x \\ v \end{bmatrix},\ x_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{15}$$

where the first state variable, t, is included in order to obtain time as a function of evaluations e, $t(e)$. This allows a unifying framework where all dynamics such as continuous time, discrete time, and discrete event are represented as functions of e.

5.1.2. The Switching Semantics

The stamping action switches from a mode where there is no reaction force on the component to contact behavior that produces a reaction force based on the stiff spring/damper system. The spring/damper system models the behavior of contact between the component and the board and is activated when the component reaches the board level. From a first principles perspective, such physics are well modeled by simultaneous constraints (e.g., Otter et al. (2000)) and simultaneous inequalities capture the activation. In this case, the activation is evaluated simultaneously with the differential equations and the inequalities hold on the continuous time domain as

$$F_{board}(t) = \begin{cases} -\left(R \cdot v(t) + \frac{x(t)}{C}\right) & \textit{if } x(t) < 0 \\ 0 & \textit{otherwise} \end{cases} . \tag{16}$$

But what does it precisely mean that the activation is evaluated 'simultaneously'? Such interaction semantics must be, and now can be, precisely formulated. For example, the solver developed in Section 4 consists of two evaluation stages: one for the Euler and one for the trapezoidal integration, the odd and even evaluations, respectively. When the time at which the inequalities are evaluated is expressed in *evaluations* it becomes possible to precisely state when the truth value changes.

The importance of this precise formulation is documented in previous work by Zander et al. (2011) which showed that distinctly different behavior may emerge, depending on whether the truth values of the inequalities change for: (i) all evaluations, (ii) only the odd evaluations, or (iii) only on accepted integration steps (i.e., where time starts increasing again). The underlying reason for this is that the variable step causes time to 'zoom in' on the point in time at which the inequalities change their truth value (and the force acting on the component changes discontinuously). If this change occurs between an odd and even evaluation, the solver zooms in before accepting the time step. Otherwise, the time step may first be completed before the change in truth value occurs and the reaction force is not accounted for until the component is well below the level of contact.

In the proceedings, the interaction semantics of the inequalities and the discretized differential equations is such that the truth value of the inequalities only changes on odd evaluations. This leads to the following precise definition of the discontinuity:

$$\begin{aligned} F_{d,board}(2e+1) &= F_{board}(t(2e+1)) \\ F_{d,board}(2e+2) &= F_{board}(t(2e+1)) \end{aligned} \tag{17}$$

with $F_{d,board}(0) = 0$.

Note that from a *modeling* perspective, generally a more accurate model is obtained by, in addition to position, expressing the switching condition of the reaction force by the board in terms of velocity and force. A more detailed treatise is provided in work by Mosterman and Biswas (1996); Pfeiffer and Glocker (1996).

5.1.3. Effecting the Control Force

Similar to the reaction force by the board, the evaluations at which the control force are effected must be made precise. The digital control nature corresponds to values that change at certain points in time and are constant otherwise. As such, there is no interaction with the computations that are necessary to generate the behavior for the plant model. Consequently, the control force only changes at the points in time where numerical accuracy is satisfied, that is, at accepted integration steps. These integration steps move time forward and so the force remains constant as long as $t(e) \le t(e-1)$

$$F_{d,control}(e) = \begin{cases} F_{d,control}(e-1) & \textit{if } t(e) \le t(e-1) \\ F_{control}(t(e)) & \textit{otherwise} \end{cases} . \tag{18}$$

5.2. Generating a Force Control Profile

With the mapping in Section 5.1, the computational model is transformed into a discrete-event representation that includes the differential equations by incorporating the variable-step solver behavior. Because now issues such as the discontinuities in behavior and mode switches are treated in a unified discrete manner, model checking techniques can be applied to find the force control profile, even though stiff numerical behavior may be present. To directly synthesize a control force profile based on the computational model, first the requirement that the control must satisfy is formulated as a property for the model checker to prove true or false.

For the SMD, this property requires the control force $F_{control}$ to be such that the acceleration upon contact is less than a maximum value, a_{max}, for a minimal duration, dt_{min}. The acceleration part of the requirement is formulated as

$$dt(e) = \begin{cases} t(e) - t(e-1) + dt(e-1) & \textit{if } |\dot{v}(e)| < a_{max} \\ 0 & \textit{otherwise} \end{cases} \tag{19}$$

which adds the current time step, $t(e) - t(e-1)$, to the duration, $dt(e)$, if the acceleration, $\dot{v}(e)$, is less than the maximum value. Here the initial values $t(e-1)$ and $dt(e-1)$ are taken to be 0. The duration part of the requirement is formulated in the negative as

$$dt(e) \ngtr dt_{min} \tag{20}$$

Model checking then attempts to find a control force profile over time such that the duration property is violated. If model checking evidences such a violation, it produces the force profile that causes the violation, the so-called *counterexample*. This counterexample, then, is precisely a control profile that satisfies the requirement.

The formulated property is combined with the discrete event model of Section 5.1 and formulated in a strict subset of Simulink. This subset contains only the 'Memory' (one evaluation delay) block and a 'Latch' as (sample and hold), in terminology of Sander (2003), *sequential* blocks. All other blocks are purely *combinational* and either mathematical or logic operations. This strict subset introduces minimal semantic complexity and is amenable to model checking by Simulink® Design Verifier. In case continuous-time integration with one of the built-in variable-step solvers of Simulink were employed, Simulink® Design Verifier would not be applicable.

The top-level model is shown in Fig. 8.[1] At the top-left is the *variable-step solver* subsystem with initial step size *hinit* of 0.005 and tolerance *tol* of 0.0025. It further takes in the inital values for the continuous state as constant *xinit*, which is a vector [00.010]. The variables that this continuous state consists of is input as u and takes the vector with elements constant 1, velocity, v, and acceleration, a (cf. Eq. (15)). This vector is integrated and output as y, which is a vector consisting of elements time, t, position, x, and velocity, v. The *uodd* input to *yodd* output combination implements the rate transition for the inequalities to compute $F_{d,board}$ from F_{board}. The *uaccept* input to *yaccept* output combination implements the rate transition for the discrete-time control to compute $F_{d,control}$ from $F_{control}$. The output *accept* produces a Boolean that captures whether the current integration step is accepted or not.

[1]This model can be accessed at the MATLAB® Central web site.

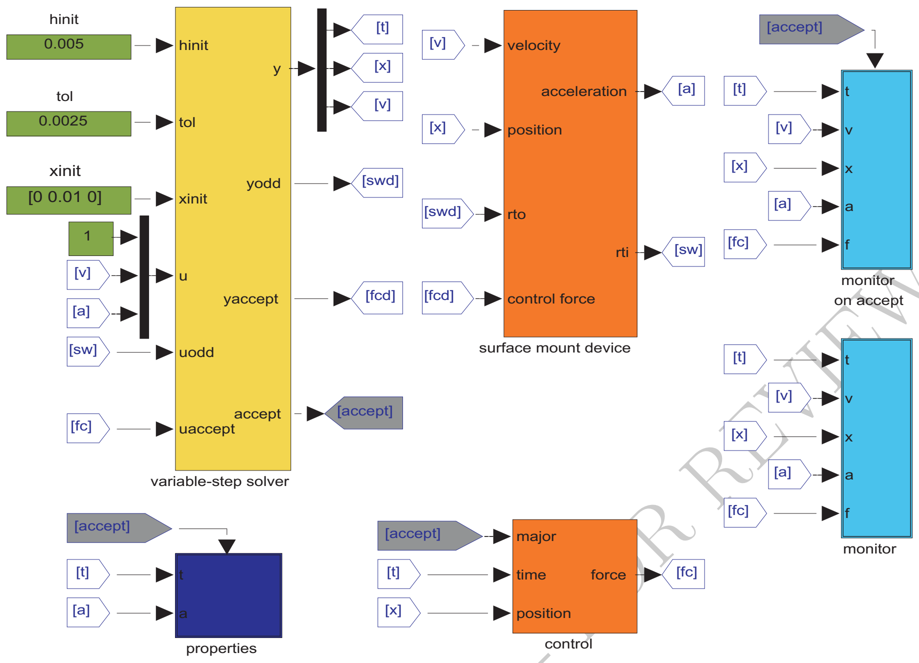


Figure 8: Top level of the Simulink® model

The *surface mount device* subsystem models the physics of Fig. 7 with mass $m = 2$, viscous friction $R = 5$, and spring constant $C = 400$. The *properties* block models the property formulated to generate the counterexample. The *control* subsystem generates the control force profile. The *monitor* subsystem consists of scopes to monitor the signal values during simulation and to write the computed values to a workspace variable. The *monitor on accept* subsystem is similar but only executed when an integration step is accepted.

The formulation as a Simulink model allows using the model checking capabilities of Simulink® Design Verifier. To this end, the *properties* subsystem formulates the property that the acceleration of the component after contact cannot be kept below 0.75 for 6.5 ms. Simulink® Design Verifier was then asked to prove this property false and produce the falsifying control force profile. To reduce the search space for the model checker, the control force was restricted to be one of -4.5, -5.25, and -6 based on expert knowledge. Furthermore, the stamping trajectory was decomposed into two phases: (i) the move of the component to the board and (ii) the hold of the component at low acceleration. The model checker was applied only to the second phase, which was initialized with the final values of the first phase. Because the functional approach in Section 4 renders all state in the system explicit (including solver state), the composition of the two phases was seamless and correct by construction.

Figure 9 shows the control force profile that was automatically generated by Simulink® Design Verifier in 1352 seconds on a 3GHz central processing unit with 12GB of RAM. Since the control force is only effected at accepted integration steps, which occur at evaluations 0, 6, 12, 18, and 24, only the force values at those evaluations must be included in the eventual control force profile. To be generally applicable, the force profile was translated to a form that can be automatically generated independent of initial conditions. The finite state machine shown in Fig. 10 generates the control profile by changing the output after each of the required time intervals, where 0.035 is the initial time of impact. Note that the conditions for a transition between states are time based as opposed to evaluation based. As a result, the control is independent of the number of evaluations that are required between accepted integration steps.

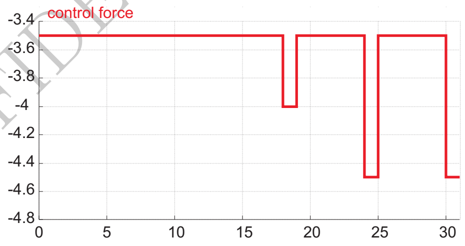


Figure 9: Automatically generated force profile

Overall results of the control so generated are presented in Fig. 11. In Fig. 11(a) the velocity of the component is shown as it is moved toward the board and then being pressed against it with a low force. The control force profile that was generated based on the generated counterexample is shown in Fig. 11(b).

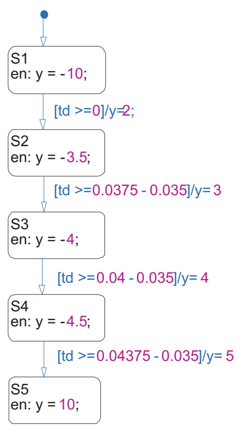
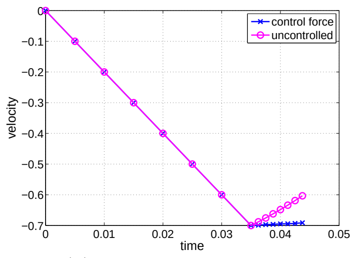


Figure 10: Control profile generator

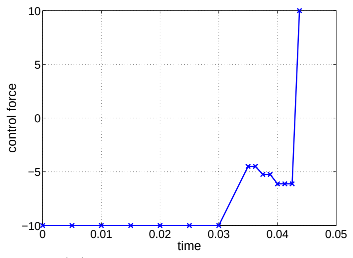
5.3. Validating the Control Force Profile

To validate whether the control force profile leads to an acceleration profile for the component that indeed satisfies the requirement (in other words, that it is indeed a counterexample), the acceleration during a simulation of a stamp action is depicted in blue (‘x’ markers) in Fig. 11(c). As expected, the acceleration remains close to 0 for the required duration. Next, the component is released from the stamper and a large positive control force retracts the stamper. To compare, the acceleration of a simulation where the control force remains constant is depicted in magenta (‘o’ markers). In this case, the acceleration is distinctly higher and in violation of the requirement.

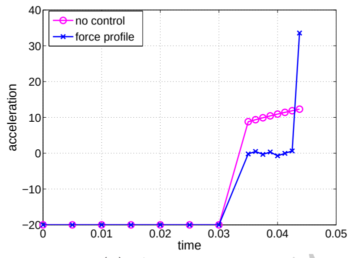
A more detailed study is presented in Fig. 12 by plotting the computations as a function of the evaluations. This provides insight into how the numerical solver from Section 4 behaves in order to achieve the required tolerance in accuracy of the computed control behavior. Figure 12(b) shows the position of the stamper with the component attached as they accelerate toward the board. At evaluation 13, the position first falls below 0, indicating that the component makes contact with the board. Because upon contact the reaction force of the board becomes active, there is a discontinuous change in force and the corresponding acceleration is shown in Fig. 12(d). On the following evaluation,



(a) Component velocity



(b) Force control profile

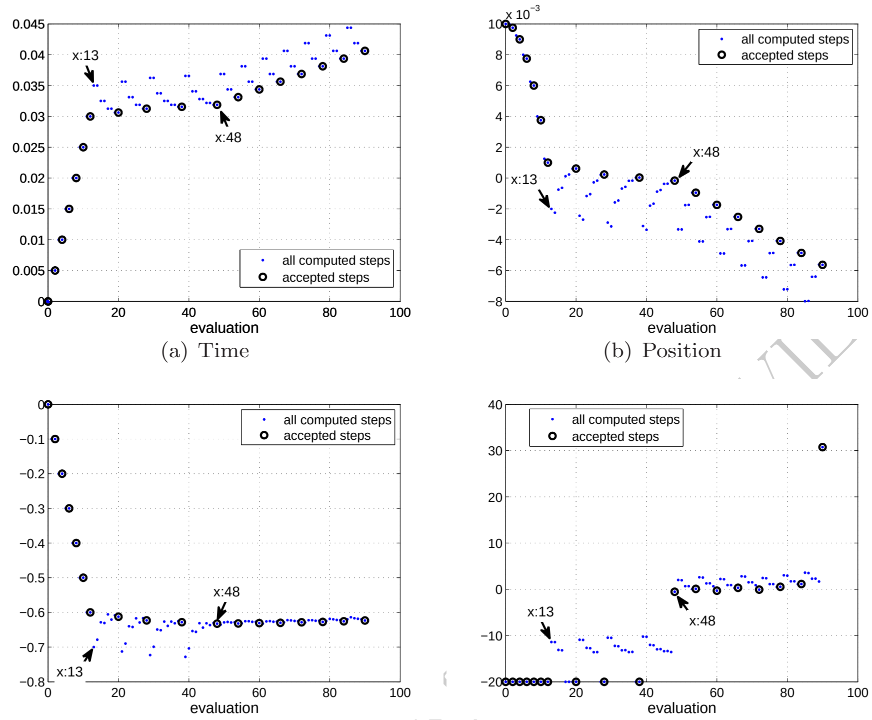


(c) Acceleration

Figure 11: Model checking based control

the trapezoidal method accounts for the change in acceleration and computes a velocity that is less negative, shown in Fig. 12(c). This change in velocity is such that the Euler approximation at evaluation 13 and the trapezoidal approximation at evaluation 14 deviate by 0.0215, which is more than the allowed tolerance of 0.0025. In response to this, the solver attempts to find a velocity that satisfies the tolerance at an earlier point in time and so time is moved back as shown in Fig. 12(a). Time continues to receed till the velocities computed at evaluations 19 and 20 are within tolerance.

The computed values at evaluation 20 are then accepted (indicated by the black circles in Fig. 12) and the maximum step (the *initial step* parameter, 0.005) forward in time is made. Again, the tolerance is initially exceeded and time moved back as before. Because of the discontinuity upon impact (the point where the position in Fig. 12(b) is 0) the solver makes increasingly smaller time steps around this point. Ultimately, at evaluation 48, the accepted position falls below 0, and a larger step size becomes acceptable because the behavior is continuous again.



(a) Time (b) Position

(c) Velocity (d) Acceleration

Figure 12: Values at all evaluations

At this evaluation 48, the control force profile is activated to attain and maintain an acceleration that is within the specified bounds (between -0.75 and 0.75). While computing the behavior, the initial step continues to be too large to satisfy the prescribed tolerance, and time continues to recede till the difference between the Euler and trapezoidal approximations is acceptable. As can be seen in Fig. 12(d), the computed accelerations at evaluations where the solver tolerance is exceeded may be beyond the specified bounds. However, the computed accelerations at accepted evaluations (indicated by circles in Fig. 12(d)) are indeed within bounds. This was also illustrated in Fig. 11(c).

The computations at all of the evaluations are necessary for the solver to properly compute the overall behavior for the SMD. As illustrated, this behavior includes numerically stiff as well as discontinuous behavior. The explicit formulation of these computations has made such hybrid dynamic system behavior amenable to model checking, which was illustrated by automatically synthesizing a control profile. It is important to note that the model checking approach presented here does not produce the optimal solution. In other words, the counterexample from which the control profile is derived simply satisfies the stated requirement, which does not include an optimization objective.

6. Evaluation

This section briefly discusses particularly pertinent previous work that is advanced as well as general results. Less directly related work has been referenced throughout, which should have provided a good starting point for further investigation.

6.1. Advancing Previous Work

The presented approach supports continuous-time differential equation models that rely on pure functions on streams. In previous work on *synchronous languages* reported by Benveniste et al. (2003), it was shown how functions on streams can capture discrete-time and discrete-event behavior. Combined with the continuous-time support, a unifying computational structure results. More recent work by Benveniste et al. (2010) employs nonstandard analysis to precisely define the semantics of hybrid dynamic systems. Nonstandard analysis is shown to facilitate an elegant formalization of the ideal continuous-time semantics. However, it does not attempt to formalize the *computational approximation*, which has become a first-class deliverable in the design of engineered systems (see Fig. 2).

An important distinction over synchronous languages is that there is no implicit underlying discrete clock in the work presented here. Instead, a sequence of evaluations is defined which is related to the *tagged signal model* introduced by Lee and Sangiovanni-Vincentelli (1996). The total order on the model of time to describe physics, however, would have been too restrictive to capture the computational semantics of the variable-step solver in Section 4. Instead, time can increase and decrease with increasing evaluations in order to capture variable-step solver semantics. So, in this unifying computational framework, not only is time possibly constant, as was documented in previous work Mosterman (2002, 2007), it may recede as well.

In general, decreasing time is well established as 'roll back' in the *time warp* discrete-event simulation algorithm by Jefferson (1985). Though the algorithm has been utilized for rigid body simulation by Mirtich (2000), focus was on efficient simulation rather than formulating a model of time for a unifying semantics.

6.2. Hybrid Dynamic System Semantics Definition

In a general hybrid dynamic systems sense, the presented work holds value along three axes. First, though the use of numerical integration is wide spread, the long time behavior is relatively poorly understood. This lack of understanding has not been much of a practical impediment given that the parameters of continuous-time models are typically fit against measurements by using computational simulation. As a result, the solver characteristics become incorporated into the model and this renders computational *consistency* paramount, which necessitates an explicit computational semantics and corresponding model of time.

Second, as documented by Jackson et al. (2009), the definition of a formalism such as a domain-specific one requires a precise specification of the semantics. This is important, for example, in order to be able to develop compilers and model transformations in general, but also to understand the expressiveness of a formalism. In addition, the semantics specification may provide a reference implementation that serves as an executable specification for more efficient execution engines. With a well-defined semantics, a model becomes truly defined by the semantics of the formalism, as opposed to the particulars of this underlying, often very sophisticated, execution engine. A precise and explicit semantics is especially important for hybrid dynamic systems with infinite sensitivity Nikoukhah (2007).

Finally, integrating different formalisms requires the study of their interaction semantics. The presented unifying framework introduces a common denominator that facilitates the systematic study of such interaction and makes subtle complications explicit. For example, as documented by Denckla and Mosterman (2006), the multi-rate character of a multi-stage numerical integration algorithm may present the need for a rate transition that can be easily overlooked otherwise.

It should be noted that the presented work does not address pathological behavior as identified by Mosterman et al. (1998b) to include *chattering*, *Zeno* behavior, and lack of *divergence of time*. Such behavior typically emerges as a result of additional solver algorithms such as root-finding, event iteration, and sliding mode simulation while more sophisticated solvers apply principles of *invariance of state* and *temporal evolution of state* Mosterman et al. (1998a). Future work intends to define such solver algorithms in a declarative sense to enable precise analysis of the computational characteristics.

7. Conclusions

Continuous-time behavior as represented by differential equations often does not have a closed form solution for the trajectories that the equations represent. While iterative application of numerical integration algorithms allows obtaining such trajectories, the behavior of the approximation error over repeated iterations is, in general, not well understood. To overcome this lack of a precise definition, a computational model of time has been presented based on the semantics of a variable-step solver. Often, the approximation error of a computational solution to differential equations is eliminated by using numerical integration to calibrate models against experiments. As such, a precise definition of the computations performed by the variable-step solver provides the foundation for a representation that can be reasoned about and formally operated on.

It was argued that capturing the definition in a declarative form discards the implementation complexity of solver software modules that are typically employed by industry strength simulation products. As a result, such definition allows better *understanding* the specifics of the solver behavior. The improved understanding manifests in support for computational methods such as *model checking* that apply well at higher levels of abstraction. This contrasts with the use of solvers at an implementation level which disallows model checking of continuous-time behavior when variable-step integration is required.

In the presented work, a variable-step solver was formalized in a declarative sense as pure functions of streams. The formalization fits the framework of other types of semantics such as discrete-time and discrete-event. It was illustrated how the comprehensive framework allows understanding interaction behavior in a unifying framework. Model checking could then exploit the formalization to synthesize discrete-time feedforward control for a plant model that comprises stiff continuous-time behavior interspersed by discontinuities.

Future work intends to focus on identifying structure in the nonmonotonic time as exploited for defining the solver semantics. This structure is expected to restrict possible behavior of stiff hybrid dynamic systems and support further analysis methods. Furthermore, as future work implements additional solver algorithms, pathological behavior such as *chattering*, *Zeno* behavior, and lack of *divergence of time* becomes precisely defined and analyzable in a computational sense. Finally, the mapping of integers to a floating point representation is a subject of potential further study. The numerical effects can lead to dramatically different behavior, and a scheme to attempt to eliminate such sensitivity is crucial when continuous-time and discrete-event models are combined.

8. Acknowledgment

The authors wish to thank Hans Vangheluwe for his continued involvement. In particular, Fig. 4 was developed as a collaborative effort.

™ Simulink Design Verifier is a trademark of The MathWorks, Inc.
See www.mathworks.com/trademarks for a list of additional trademarks.

Andersson, G., Bjesse, P., Cook, B., Hanna, Z., Aug. 2002. A proof engine approach to solving combinational design automation problems. In: Proceedings of the 39th annual Design Automation Conference. pp. 725–730.

Åström, K. J., Wittenmark, B., 1984. Computer Controlled Systems: Theory and Design. Prentice-Hall, Englewood Cliffs, New Jersey.

Ayob, M., June 2005. Optimisation of surface mount device placement machine in printed circuit board assembly. Ph.D. thesis, University of Nottingham, Nottingham, UK.

Backus, J., 1978. Can programming be liberated from the von neumann style? a functional style and its algebra of programs. Communications of the ACM 21 (8), 613–641.

Benveniste, A., Caillaud, B., Pouzet, M., Dec. 2010. The fundamentals of hybrid systems modelers. In: Proceedings of the 49th IEEE Conference on Decision and Control. Atlanta, GA.

Benveniste, A., Caspi, P., Edwards, S. A., Halbwachs, N., Guernic, P. L., de Simone, R., 2003. The synchronous languages twelve years later. Proceedings of the IEEE 91 (1), 64–83.

Breedveld, P. C., 1984. Physical systems theory in terms of bond graphs. PhD dissertation, University of Twente, Enschede, Netherlands.

Bujakiewicz, P., 1994. Maximum weighted matching for high index differential algebraic equations. PhD dissertation, TU Delft, Delft, Netherlands, iSBN 90-9007240-3.

Caspi, P., Pouzet, M., October 1997. A Co-iterative Characterization of Synchronous Stream Functions. Tech. Rep. 07, VERIMAG.

Cellier, F., Elmqvist, H., Otter, M., 1996. Modelling from physical principles. In: Levine, W. (Ed.), The Control Handbook. CRC Press, Boca Raton, FL, pp. 99–107.

Denckla, B., Mosterman, P. J., Oct. 2006. Block diagrams as a syntactic extension to haskell. In: Proceedings of the Workshop on Multi-Paradigm Modeling: Concepts and Tools. Genoa, Italy, pp. 67–79.

Denckla, B., Mosterman, P. J., Jul. 2008. Stream- and state-based semantics of hierarchy in block diagrams. In: Proceedings of the 17th IFAC World Congress. Seoul, Korea, pp. 7955–7960.

Falk, G., Ruppel, W., 1976. Energie und Entropie: Eine Einführung in die Thermodynamik. Springer-Verlag, Berlin, Heidelberg, New York, iSBN 3-540-07814-2.

Friedman, J., Ghidella, J., Apr. 2006. Using model-based design for automotive systems engineering – requirements analysis of the power window example. In: Proceedings of the SAE 2006 World Congress & Exhibition. Detroit, MI, pp. CD–ROM: 2006–01–1217.

Guckenheimer, J., 2002. Numerical analysis of dynamical systems. In: Fiedler, B. (Ed.), Handbook of Dynamical Systems. Vol. 2. Elsevier, Amsterdam, Netherlands, pp. 345–390, iSBN 978-0-444-50168-4.

High Confidence Software and Systems Coordinating Group, feb 2009. High-confidence medical devices: Cyber-physical systems for the 21st century health care. Tech. rep., Networking and Information Technology Research and Development Program.

Jackson, E., Thibodeaux, R., Porter, J., Sztipanovits, J., 2009. Semantics of domain specific modeling languages. In: Nicolescu, G., Mosterman, P. J. (Eds.), Model-Based Design for Embedded Systems. CRC Press, Boca Raton, FL, iSBN 978-1-420-06784-2.

Jefferson, D. R., 1985. Virtual time. ACM Trans. Program. Lang. Syst. 7 (3), 404–425.

Jones, H., 2005. Return on investment in Simulink® for electronic system design. Tech. rep., International Business Strategies, www.mathworks.com/roi.

Jones, S. P., Apr. 2003. Haskell 98 Language and Libraries. Cambridge University Press, Cambridge, UK, iSBN-10: 0521826144.

Lee, E. A., Sangiovanni-Vincentelli, A., jun 1996. The tagged signal model—a preliminary version of a denotational framework for comparing models of computation. Tech. Rep. UCB/ERL M96/33, University of California, Berkeley, California.

Looye, G., Varga, A., Bennani, S., Grübel, G., Sep. 1998. Robustness analysis applied to autopilot design, part 1: μ-analysis of design entries to a robust flight control benchmark. In: Proceedings of the 21st International Congress of Aeronautical Sciences. Melbourne, Australia, paper ID: ICAS-98-1.6.1.

Mirtich, B., 2000. Timewarp rigid body simulation. In: Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (ACM SIGGRAPH). New York, NY, pp. 193–200.

Mosterman, P. J., 2002. HyBrSim—a modeling and simulation environment for hybrid bond graphs. Journal of Systems and Control Engineering 216 (1), 35–46.

Mosterman, P. J., Sep. 2007. On the normal component of centralized frictionless collision sequences. ASME Journal of Applied Mechanics 74 (5), 908–915.

Mosterman, P. J., Biswas, G., Nov. 1996. Verification of dynamic physical system models. In: Proceedings of the ASME Congress '96. Atlanta, GA, pp. 707–714.

Mosterman, P. J., Biswas, G., Sztipanovits, J., 1998a. A hybrid modeling and verification paradigm for embedded control systems. Control Engineering Practice (6), 511–521.

Mosterman, P. J., Zhao, F., Biswas, G., Jul. 1998b. An ontology for transitions in physical dynamic systems. In: AAAI98. pp. 219–224.

Nicolescu, G., Mosterman, P. J. (Eds.), 2009. Model-Based Design for Embedded Systems. Computational Analysis, Synthesis, and Design of Dynamic Systems. CRC Press, Boca Raton, FL, iSBN: 9781420067842.

Nielson, H. R., Nielson, F., 1992. Semantics with Applications: A Formal Introduction. Wiley Professional Computing, Hoboken, NJ, iSBN 0 471 92980 8.

Nikoukhah, R., jul 2007. Hybrid dynamics in modelica: Should all events be considered synchronous. In: Proceedings of the 1st International Workshop on Equation-Based Object-Oriented Languages and Tools (EOOLT 2007). Berlin, Germany, pp. 37–48.

Otter, M., Remelhe, M. A. P., Engell, S., Mosterman, P. J., 2000. Hybrid models of physical systems and discrete controllers. at - Automatisierungstechnik.

Paynter, H. M., 1961. Analysis and Design of Engineering Systems. The M.I.T. Press, Cambridge, Massachusetts.

Petzold, L. R., 1982. A description of DASSL: A differential/algebraic system solver. Tech. Rep. SAND82-8637, Sandia National Laboratories, Livermore, CA.

Peyton-Jones, S., 1987. The Implementation of Functional Programming Languages. Prentice Hall, Englewood Cliffs, NJ.

Pfeiffer, F., Glocker, C., 1996. Multibody dynamics with unilateral contacts. John Wiley & Sons, Inc., New York.

Post, D. E., Votta, L. G., Jan. 2005. Computational science demands a new paradigm. Physics Today 58 (8), 35–41.

Potter, B., 2004. Use of the mathworks tool suite to develop do-178b certified code. In: ERAU / FAA Software Tools Forum. Daytona Beach, Florida.

Reekie, H. J., May 1994. Modelling Asynchronous Streams in Haskell. Tech. Rep. 94.3, Key Centre for Advanced Computing Sciences, University of Technology, Sydney.

Sander, I., April 2003. System modeling and design refinement in ForSyDe. Ph.D. thesis, Royal Institute of Technology, Stockholm, Sweden.

Sanz-Serna, J. M., 1992. Symplectic integrators for hamiltonian problems: an overview. Acta Numerica 1, 243–286.

SC-167, R., dec 1992. Do178b, software considerations in airborne systems and equipment certification.

Schiela, A., Olsson, H., Oct. 2000. Mixed-mode integration for real-time simulation. In: Modelica 2000. Lund, Sweden, pp. 69–75.

Shampine, L. F., Reichelt, M. W., Kierzenka, J. A., 1999. Solving index-1 daes in matlab and simulink. SIAM Rev. 41 (3), 538–552.

Simulink®, Sep. 2008. Using Simulink®. The MathWorks™, Natick, MA.

Zander, J., Mosterman, P. J., Hamon, G., Denckla, B., Sep. 2011. On the structure of time in computational semantics of a variable-step solver for hybrid behavior analysis. In: Proceedings of the 18th IFAC World Congress. Milan, Italy, in review.

Zhang, Y., Xu, B., 2004. A survey of semantic description frameworks for programming languages. ACM SIGPLAN Notices 39 (3), 14–30.

List of Figures