On the Structure of Time in Computational Semantics of a Variable-Step Solver for Hybrid Behavior Analysis

Justyna Zander*, Pieter J. Mosterman, Grégoire Hamon**, Ben Denckla*****

* Harvard University, HHI, 14 Story St., Cambridge, MA 02138 USA (e-mail: justyna.zander@gmail.com)
** MathWorks, 3 Apple Hill Drive, Natick, MA 01760 USA (e-mail: pieter.mosterman@mathworks.com)
*** Independent Researcher (e-mail: bdenckla@alum.mit.edu)

Abstract: Hybrid dynamic systems combine continuous and discrete behavior. Often, computational approaches approximate behavior of an analytic solution, for example, numerical integration to approximate differential equation behavior. The accuracy and computational efficiency of the integration usually depend on the complexity of the method and its implicated approximation errors, especially when repeated over iterations. This work formally defines the computational semantics of a solver in a denotational sense so as to analyze discrete- and continuous-time behavior of time-based block diagram models. A stream-based approach is used to analyze the numerical integration implemented by the solver. The resulting solver applies the principle of nonmonotonic time and so consecutive values may be computed in a temporally nonmonotonic manner. This allows shifting the evaluation points backward and forward in time. Stratification recovers a partially ordered structure in time. Solver dynamics are thus made explicit and can be studied in concert with behavior of discontinuous models parts.

1. INTRODUCTION

Innovation in engineered systems such as airplanes, automobiles, and mobile phones is increasingly implemented in software (see, e.g., Broy *et al.*, 2007). As exceedingly sophisticated features are demanded, the size of software that implements the corresponding functionality is growing rapidly. As a consequence, software producibility has become prohibitively complex to a timely design and implementation of modern systems such as the F22 fighter airplane (U.S. Government Accountability Office, 2006) and cyber-physical systems in general. Therefore, new paradigms are imperative for *efficient and effective software production.*

Model-Based Design (Shenoy *et al.*, 2007) relies on computational representations of a system under design as first class deliverables. Sharing such deliverables between teams of engineers significantly enhances the potential for reuse of effort. For example, the implementation of a design specification can exploit automatic code generation to reduce the overall effort required. To this end, it is critical to precisely define the semantics of each of the computational representations. For example, the execution of automatically generated code should be unambiguously defined in the corresponding model of this software that is used for simulation. Where semantics definition is well understood for discrete-event and discrete-time formalisms, the semantics of a computational implementation of solvers for models developed with continuous-time formalisms has been less investigated. This work attempts to formally define the computational semantics of a solver for ordinary differential equations (ODE) in terms that apply to discrete-time and discrete-event formalisms to facilitate the analysis of hybrid dynamics for engineered systems.

The behavior of hybrid dynamic systems is typically defined on a domain that at least includes time as an independent variable. A successful approach to behavior generation for hybrid dynamic systems is the core set of continuous-time blocks in Simulink® (Simulink, 2010), extended with triggered and enabled subsystems.

The structure of time as a continuous domain covered by intervals was found insufficient in work studying *chattering* behavior (Iwasaki *et al.*, 1995). Here, a system was defined to evolve according to two different sets of differential equations, while discrete logic caused it to switch infinitely fast between the two. In order to facilitate analysis, time was defined in terms of hyperreals of *nonstandard analysis* to allow infinitesimal steps. More recent work revisits the use of nonstandard analysis to provide hybrid dynamic system semantics in a more general sense (Benveniste *et al.*, 2010).

In case of sophisticated switching logic, a hybrid system may move through a series of discrete states before behavior is governed by a set of differential equations again (so called *event iteration*). This motivated a structure of time as a series of abutting closed intervals as introduced by Guckenheimer and Johnson (1995). Since interval bounds overlap, the intervals were adorned with an index to establish a unique independent domain. As a consequence, behavior could be defined to only hold at a point in time, which proved to be a useful abstraction of physics (e.g., Mosterman, 2002). The use of a tuple consisting of time and an index has been revisited in work by Lee and Sangiovanni-Vincentelli (1996).

Yet still a further extension to such a structure of time was proposed by Nishida and Doshita (1987) to allow reasoning about causal effects in physical systems. In particular, it was found that in order to *understand* how a behavior emerges, it was important to be able to *consider* a model as moving into a certain state, even though it may never actually assume this state. The state that enabled reasoning was called *mythical*. This notion was later formalized and related to *parameter abstractions* (as opposed to *time scale abstractions*) in models of physical systems (Mosterman, 2002).

A comprehensive framework based on all these studies resulted in a formal ontology of behaviors of hybrid dynamic systems by Mosterman and Biswas (2000). However, although the studies have introduced very versatile and powerful structures in the domain on which behavior are defined, they all attempt to address the analytic semantics. As such, the effects introduced by providing a computational behavior generation algorithm are often relegated to mere 'approximations' and there are few attempts to precisely define the effects. This holds true in particular for the effects of applying a specific numerical solver to generate a behavior for a differential equation. Consequently, it is often disregarded how applying a numerical solver interacts with other parts of the system such as discrete-event behaviors.

To illustrate, consider a ball that is released from a certain height to bounce off a floor that is modeled as a system with stiffness and viscous friction. The first two piecewise behaviors are the initial phase where the ball is falling down under the influence of gravity and the consecutive phase where the floor exerts an opposing force to reverse the ball velocity. The two piecewise behaviors are modeled by two sets of differential equations where the system switches from one to the other upon contact with the floor.

The differential equations can be solved by a *numerical solver* that utilizes a numerical integration scheme over discrete intervals of time. The duration of each interval may be adapted based on active dynamics so integration can be sufficiently accurate, given a predefined tolerance. Typically, when the numerical solver makes a step, the error over that step is estimated and if the error exceeds the tolerance, the size of interval is reduced. Further reduction may follow till the tolerance is satisfied and the end point of the corresponding interval is *accepted* as the next step in time.

Now, a semantic choice can be made whether the discontinuity because of the ground forces that become active should be effected while the interval size is still being reduced, or whether it should only then become active once the next step in time is accepted. Figure 1 shows the difference in computed behavior and the analytic solution derived with the Symbolic Math Toolbox™ (Symbolic Math Toolbox, 2010). The comparison shows the importance of a precise definition of the numerical solver semantics and its interaction with other model elements in a hybrid dynamic system so as to reliably produce correct (as determined by interaction semantics choice) and consistent (once the semantics are chosen) results.

This work aims to find a structure of time that allows such a precise definition of the *computational* semantics of a hybrid dynamic system. In previous work, Denckla and Mosterman (2006, 2008) have studied the formal definition of semantics of temporal behavior. However, it has proven difficult to present a comprehensive framework that applies to both continuous-time and discrete-time semantics (e.g., Hardebolle, 2007). As such, there is a lack of a flexible, comprehensive computational concept that applies to the high-level languages with both types of semantics so as to analyze their integration and interaction.

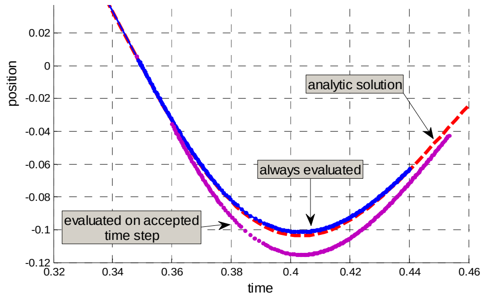

Fig. 1. Velocity reversal of a bouncing ball with different interaction semantics between the solver and discontinuities

In contrasts to a more versatile imperative application programming interface (e.g., the Ptolemy project [Lee and Zheng, 2007] or the S-Function specification [Simulink S-Functions, 2007]), analysis of a computational implementation is often best supported by a denotational approach. Previous work by Halbwachs *et al.* (1991, 1999) has studied the computational semantics of *synchronous languages* using the synchronous model of computation (as reviewed by Benveniste *et al.*, 2003). Caspi (2006) has studied computational semantic based on synchronous data-flow. In this approach functional design elements are considered to *react* instantaneously (i.e., their evaluations do not consume time), and are activated simultaneously and in *synchrony* with an underlying discrete-time clock.

As such, these synchronous languages have been strictly limited to functionality with a periodic base rate of a discrete-time nature, which has impeded its utility in the study of hybrid systems. As long as the continuous-time dynamics can be discretized by a fixed step and single stage (often forward Euler) numerical integration, it can be described by a synchronous language. The continuous part of a hybrid system, however, may evidence stiff dynamic behavior that requires different numerical integration methods for efficient computation, for example, during simulation.

Previous work by Denckla and Mosterman (2008) has shown that more complex variable-step integration methods for generating continuous-time behavior can be included by hierarchical decomposition. This, however, requires strict modularization so as to isolate the variable-step size of the computation. Furthermore, the numerical integration was implemented using a state-based approach which is less amenable to analyses of behavior than a stream-based approach with true stateless functions. As other work by Lee and Zheng (2005) also employs a stateful approach, it forgoes the advantages of stateless functions such as *referential transparency* (e.g., Backus, 1978).

This paper attempts to support the variable-step nature of differential equation solvers in combination with discrete-time behavior. Further, this work provides stream-based semantics where block diagrams are functions whose input and output are streams and a stream consists of a potentially infinite sequence of values (Denckla and Mosterman, 2008) with continuous-time behavior that is generated by variable-step integration. To this end, the synchronous languages framework is adapted to support a variable time step. This is related to the *tagged signal model of computation* introduced by Lee and Sangiovanni-Vincentelli (1996) that allows tags to be an annotation to signal values. In this paper, the associated tags represent an evaluation index, not time. The evaluation index is then projected into a separate dimension that captures the temporal aspect. By that, the total order of time is not required. Instead, the notion of an untimed partially ordered model of computation is employed.

The benefit of this approach is that only the evaluation index increases monotonically and time can now be regarded as a *nonmonotonically* increasing quantity, which allows analysis of variable-step solvers by explicitly recognizing time steps that are discarded by the solver but that are essential to its computational semantics.

The remainder of this paper is structured as follows. Section 2 presents the details of related work. In Section 3, the proposed approach in the context of block diagrams is described. Section 4 introduces the realization of the stream-based execution framework, which supports the design of a discrete- and continuous-time model. Conclusions complete the paper.

2. BACKGROUND

Previous work (Caspi, 2006; Denckla and Mosterman, 2006, 2008) has extensively concentrated on defining the computational semantics of discrete-time systems. These efforts employ state-based and stream-based approaches. In general, the latter are preferred for analysis whereas the former are more efficient for behavior generation.

Continuous-time systems, much less the subject of study, can be made to fit the discrete-time framework by means of time discretization. Complication may arise in case of a multistage numerical integration scheme (Denckla and Mosterman, 2006), though. For example, consider the ODE

$$\dot{x}(t) = \frac{dx}{dt} = f(x,t) \tag{1}$$

where x is the state of the differential equation, t represents time, and f is a well-behaved function. A forward Euler integration scheme computes the state according to

$$x(t_{k+1}) = x(t_k) + h_k f(x(t_k), t_k) \tag{2}$$

where t_k represents a time point, and h_k is the step size between two consecutive time points (t_k and t_{k+1}). This allows a straightforward combination with discrete-time system elements although in strict production/consumption based computation, this may already cause difficulty when $x(t_k)$ is consumed twice for a single production.

A more complex integration scheme such as a multistage algorithm may be even less amenable to combination with discrete-time elements. Consider a second order Runge-Kutta numerical integration with a fixed-integration step, h, (3–5)

$$\begin{aligned} a_1 &= h \cdot f(x(t_k), t_k) \\ a_2 &= h \cdot f(t_k + \frac{h}{2}, x(t_k) + \frac{a_1}{2}) \\ x(t_{k+1}) &= x(t_k) + a_2 \end{aligned} \tag{3–5}$$

which consists of two stages in the evaluation as f is evaluated at t_k as well as at the midpoint between t_k and t_{k+1}. This illustrates that now the gradient function, f, must be evaluated at an undefined point in the discrete-time interval. In previous work, Denckla and Mosterman (2006, 2008) argued that an explicit rate transition is helpful for the modeler of the hybrid system to understand the complexity and potential semantic implication.

Though the second order Runge-Kutta numerical integration scheme is multistage, it is still of a fixed-step nature. In other words, the step size h_k is uniform and independent of k. In contrast, this work provides a semantic framework to analyze a variable-step approach implemented by a *solver*. The illustrative solver employs two numerical integration schemes and adapts the step size based on an error bound evaluation of the states computed by the two integration schemes. Such solvers are essential to address industrial simulation needs as is exemplified by the family of different solvers supported for Simulink® models.

This work adopts the *synchronous assumption* of languages such as LUSTRE (Halbwachs *et al.*, 1991). It then defines a declarative model of computation inspired by *lambda calculus* (i.e., a formal system, where the notion of computable function is defined by this system [see Caspi, 2006; Nielsen and Nielson, 1988]) to establish the semantics of block diagrams with discrete-time and continuous-time behavior. This practice abstracts from the imperative notion that is necessitated by an executable implementation and thus again, allows reasoning about the semantics without consideration of the implementation and runtime details.

3. STREAM-BASED SOLVER

The integrated and comprehensive description and analysis of discrete-time and continuous-time behavior yields a unified semantic of a variable-step discretized ODE in a stream-based execution framework. This enables analysis of hybrid system behavior as discussed by Mosterman *et al.* (2009). This section describes the meaning of a stream-based approach and introduces the principles of the applied method of computation. The next section presents the realization of a *discretized ODE solver (diODE)* as a Simulink® model.

3.1 A Stream-based Approach

The semantics of an ODE in mathematical terms is not explicitly considered but left implicit in the numerical integration scheme. Instead, only the computational semantics of an ODE based on the selected solver are studied. The resulting *diODEs* allow a constructive unified stream-based computational representation. In other words, the meaning of execution will be defined from the perspective of computer science, not as a mathematical representation and will be given in a discretized manner.

The benefits of a stream-based approach are the following:
(1) A set of pure functions can be analyzed in terms of function composition, in contrast to complicating state behavior of a state-based execution. For example, the state-based approach cannot easily describe multi-rate systems, whereas the stream-based model can (cf. Caspi, 2006).
(2) A declarative general computational representation avoids the additional complexity of an implementation choice.

Capturing the semantics in a computational sense allows a very precise formulation of the execution of a system with differential equation behavior (Mosterman *et al.*, 2009). This precision is especially important when continuous-time and discrete-event behaviors interact, given the infinite sensitivity to perturbations of the latter. Moreover, because the error approximation of the numerical mathematics is only local, global error accumulates and the long term behavior becomes poorly defined as studied by Guckenheimer (2002).

3.2 Time versus Evaluation

To formalize a framework, a model of computation that is related to the *tagged signal mode*l (Lee and Sangiovanni-Vincentelli, 1996) is used, where a tag representing an evaluation is associated with each computed value. A time stamp in a separate dimension is associated with each evaluation. *Time (t)* and *evaluation (e)* should be synchronized. The evaluation points can be interpreted as a sequence of *indexed stamps* that are classified and marked as *accepted* and *rejected*. Accepted are those, for which the computed value is based on the currently selected step size; rejected are those for which the computation must be repeated applying a smaller step size. Then, states are elicited and the properly computed evaluation intervals are used

In a sense, this work adopts an *untimed model of computation* such as in work by Lee and Sangiovanni-Vincentelli, (1996) 0where the tags are abstract objects that may only bear a partial ordering relationship. In the framework presented here, *time* then becomes *a variable*. It is assumed that time increases in a stratified manner. In other words, time can be described in a nonmonotonic manner. Note that while in a more general hybrid system framework time may have to be held constant across computation, for purposes of defining the semantics of *diODEs*, this is not strictly necessary. Further, the *sequence of time stamps* is not considered, but a *sequence of evaluations* is. These evaluations are ordered but without a distance measure. Each of the evaluations has a time association, which enables the solver to compute the *values* of the analyzed signal at arbitrary points within a temporal stratum (and thus potentially moving backwards in time).

3.3 The Nonmonotonic Notion of Time

Introducing the principle of *nonmonotonic time* used in the numerical solver, the following assumptions hold. Time in its general form refers to *logical time* which is the time corresponding to an evaluation. The logical time at an evaluation where the solver satisfies its tolerance criterion (i.e., an *accepted* integration point) is called *simulation time*. Simulation time does not change at evaluations where the solver fails to meet its tolerance criterion. At those evaluations, simulation time equates the logical time of the previous accepted integration point.

The notion of simulation time computed in this manner can be related to the actual time it takes to perform the computations, the so-called *physical time*. The physical passage of time during system execution increases monotonically. Logical time in the computations, however, may change nonmonotonically. Finally, if the system executes in *real time*, the advance of simulation time is paced so it corresponds to the passage of physical time.

So, the logical time is an abstract concept introduced to precisely explain how a solver handles the accepted and rejected integration points, how it interprets the process when providing the solution, and how this process relates to the simulation time, physical time, and real time.

To illustrate, in Fig. 2 the evaluation points, e_i, are ordered and monotonically increasing. Projected onto the time axis, these evaluations are producing tuples of evaluation and time (e.g., $<e_1,t_1>$, $<e_2,t_4>$, $<e_3,t_2>$), where the notion of nonmonotonic time allows for shifting the evaluation points backward, forward, or keeping them constant in time depending on the computational needs. This is exploited by variable step solvers in general (e.g., Petzold, 1982).

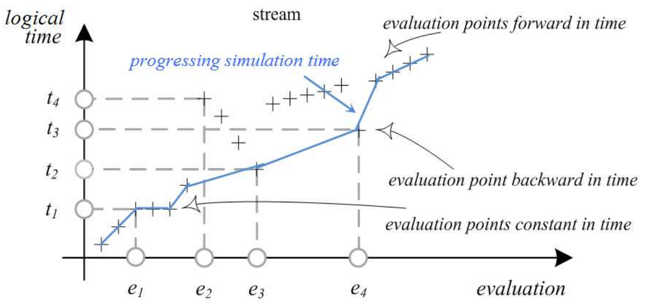
logical time
stream
evaluation points forward in time
progressing simulation time
t_4
t_3
t_2
t_1
evaluation point backward in time
evaluation points constant in time
e_1
e_2
e_3
e_4
evaluation

Fig. 2. The notion of evaluation with respect to time

The evaluation dimension can be partitioned into a set of strata. Each evaluation point for which the current step size of the solver is reset to its initial value is accepted by the solver and is called a stratum boundary point. This situation is illustrated in Fig. 2 by e_3 while the points between e_2 and e_3 are rejected evaluations in the same stratum.

The temporal dimension can be partitioned into a set of strata as well. Within each stratum, time is monotonically decreasing with an increasing evaluation index. Each stratum has a lower bound and upper bound on this temporal variation. An example of temporal stratification is depicted in Fig. 2 where the lower bound of the first stratum is t_1 while the upper bound is t_2. Though in a more general framework, time in the succeeding strata could be equal or less, here time in a succeeding stratum is always larger than the time associated with the final evaluation of the preceding strata. This lower bound on time in each stratum results from the error evaluation and step size control as performed by numerical solvers being local to each time step only.

Figure 2 also visualizes the progressing *simulation time* by connecting all the accepted integration points. *Simulation time* increases monotonically as opposed to *the logical time.* Now, the definition of *simulation time* is quantified and embedded in an analysis framework that includes evaluation points, relation to the logical time, and strata. The analysis of the simulation results is not affected by the notion of time applied for the solver, though it is explicitly related to this notion and enhanced by its clear semantics. The strata can be exploited for a further study of the correctness and consistency of the solver as such. Anomalies or certain properties (e.g., frequency of strata, monotonicity of time, duration of certain behavior) can be easier identified and quantified. In consequence, the solver dynamics becomes clear and better understood and can help study the behavior discontinuities of the model itself.

3.4 Simulink® Realization of diODE

The implementation of *diODE* was realized using a subset of Simulink blocks that do not have a temporal aspect other than the *Memory* block. Though this does not exploit the sophisticated mechanism built into Simulink to handle time, it does make the temporal semantics explicit as a declarative specification. Moreover, it still takes advantage of the facilities in Simulink to transform a declarative constraint based formulation into an operational form. To this end, a discrete solver is employed with a normalized step size of 1.

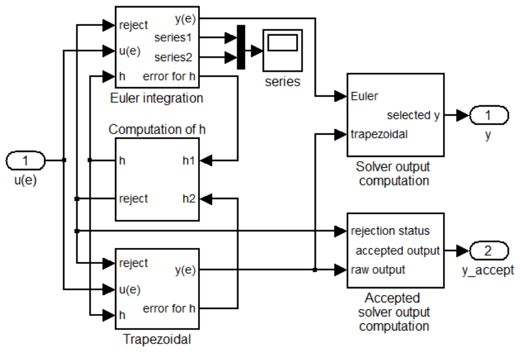

Fig. 3. Simulink® realization of diODE

The realization of *diODE* then employs two numerical integration schemes Euler and Trapezoidal (represented by the *Euler* and *Trapezoidal* blocks in Fig. 3) that are applied independently and their error estimates are compared to determine the error for the step size *h*. Based on this comparison, the step size for current integration is computed (represented by the *Computation of h* block). This process is elaborated in previous work (Mosterman *et al.*, 2009).

In particular, Trapezoidal is implemented as a trapezoidal numerical integration algorithm that averages the gradient of the start state and of the end state to compute a more accurate approximation of the end state. The forcing function allows one evaluation delay to obtain the gradient at the final state. In a more general implementation described by Mosterman *et al.* (2009), however, the gradient at the final state may be obtained by first employing the forward Euler computation to obtain an estimate of the end state and then averaging the gradient at this end state with the gradient at the start state

$$\begin{aligned} \hat{x}(t_{k+1}) &= x(t_k) + h_k f(x(t_k), t_k) \\ x(t_{k+1}) &= x(t_k) + h_k \frac{f(x(t_k), t_k) + f(\hat{x}(t_{k+1}), t_{k+1})}{2} \end{aligned} \qquad (6–7)$$

4. AN APPLICATION EXAMPLE

An example illustrates the computational analysis framework as implemented by a Simulink block diagram. Integrating a constant gradient (i.e., a ramp) with respect to time is studied. While in simulation time the ramp input to the solver may be monotonically increasing, the solver computations may require logical time at some evaluations to decrease in order to satisfy the accuracy tolerance requirement. When the solver moves time backward, the ramp input must produce previous values and so a nonmonotonic implementation of the ramp is required.

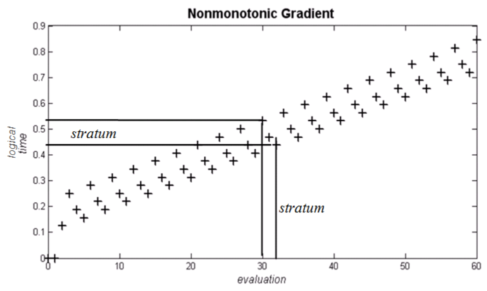

Fig. 4. Illustration of the nonmonotonic gradient

Figure 4 depicts a scenario where the *diODE* method computes the time-integrated values of the ramp over 60 evaluations. This shows the evaluation and temporal strata during which time may decrease. The accepted computations are the lower bounds of each stratum and those correspond to the monotonically increasing output of a traditional solver.

Note that the first two evaluations have a time stamp of 0 seconds. This is because the trapezoidal integration requires two evaluations to obtain the average gradient if implemented without the use of an estimate. Also note that the 60 evaluations cover a time interval of a little over 0.7 seconds of simulation time and about 0.85 seconds of logical time.

The bouncing ball example of Section 1 can now be revisited in this formal framework. Figure 5 zooms in on the point of ball contact with the floor. It shows the individual evaluations as made by the numerical solver including the ones that are not accepted (the lines between points are included for readability only). If the discontinuity in the ground force is only evaluated at accepted time steps, the numerical solver finds the discontinuity that occurs when the ball reaches the floor after a full step with the initial step size.

Alternatively, if the discontinuities are always evaluated, the numerical solver finds the discontinuity as it is reducing the interval of the step size to improve the accuracy with which the differential equations are approximated. The effect is that the discontinuity is immediately (so sooner than before) accounted for, a phenomenon documented by Cellier (1979) in previous work but now formulated in a declarative framework. Consequently, this framework can be analyzed for correctness and consistency (e.g., to provide a reference semantics) whilst providing unifying quantifiable results.

In the example from Fig. 5 the immediate detection of discontinuity in force acting on the ball (cf. 'always evaluated scenario') results in interaction of solver dynamics within one stratum. This interaction can now be quantified and assessed more precisely than in the case where an implicit rate transition is inserted to only allow discontinuities on the boundary of strata (cf. 'evaluated on accepted time step').

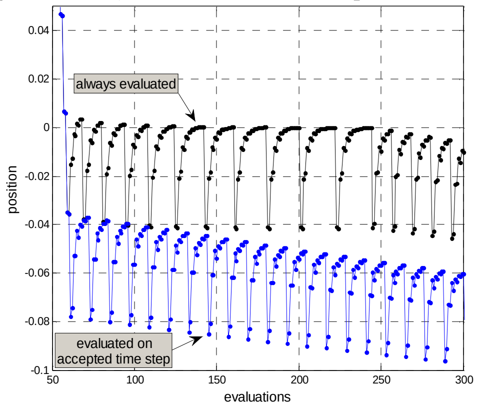

Fig. 5. Computations upon impact for different semantics

5. CONCLUSIONS

The introduced separation of evaluation from time enables reasoning on computation, where time becomes abstract. The presented work attempts to formally define the computational semantics of a solver for time-based block diagrams in a unifying framework. The ultimate target is to prepare the background for analysis of hybrid behavior of embedded systems that emerges by applying a *variable-step* differential equation solver. The presented work mainly focuses on continuous-time behavior. The formalization aims to facilitate a stream-based approach to analyze the considered numerical integration method. The resulting solver applies the principle of *nonmonotonic time*. That is, a new evaluation of model values is *explicitly* computed in a temporally nonmonotonic manner, which allows for shifting the evaluation points backward and forward, or holding them constant in time and for analysis of the interaction with other modeling semantics.

REFERENCES

Backus, J. (1978). "Can Programming Be Liberated from the von Neumann Style?," in *Communications of the ACM*, Vol. 21, Nr. 8, pp.: 613-641.

Benveniste, A., Caillaud, B., Pouzet, M. (2010). "The Fundamentals of Hybrid Systems Modelers," in *Proc. of the 49th IEEE International Conference on Decision and Control*, Atlanta, GA.

Benveniste, A., Caspi, P., Edwards, S.A., Halbwachs, N., Le Guernic, P., de Simone, R. (2003). "The Synchronous Languages Twelve Years Later," *Proc. of the IEEE*, Vol. 91, Iss. 1, pp.: 64–83.

Broy, M., Krüger, I.H., Pretschner, A., Salzmann, C. (2007). "Engineering Automotive Software," in *Proc. of the IEEE,* Vol. 95, no. 2, pp.: 356–373.

Caspi, P. (2006). "Some Issues in Model-Based Development for Embedded Control Systems," in *From Model-Driven Design to Resource Management for Distributed Embedded Systems*. Volume 225/2006, ISSN: 1571-5736. Springer Boston.

Cellier, F.E. (1979). *Combined Continuous/Discrete System Simulation by Use of Digital Computers: Techniques and Tools*, Ph.D. dissertation, Swiss Federal Institute of Technology, ETH Zürich, Switzerland.

Denckla, B., Mosterman, P.J. (2006). "Block Diagrams as a Syntactic Extension to Haskell," in *Proc. of the Workshop on Multi-Paradigm Modeling: Concepts and Tools*, October 3, Genova, Italy.

Denckla, B., Mosterman, P.J. (2008). "Stream- and State-Based Semantics of Hierarchy in Block Diagrams," in *17th IFAC World Congress*, pp. 7955-7960, July 6-11, Seoul, Korea.

Guckenheimer, J. (2002). "Numerical analysis of dynamical systems," in *Handbook of Dynamical Systems*, B. Fiedler (ed.), vol. 2, pp.: 345–390. Elsevier, Amsterdam, Netherlands.

Guckenheimer, J., Johnson, S. (1995). "Planar hybrid systems," *Hybrid Systems II*, Lecture Notes in Computer Science, 999, pp.: 202-225.

Halbwachs, N., Caspi, P., Raymond, P., Pilaud, D. (1991). "The synchronous data-flow programming language LUSTRE," in *Proc. of the IEEE*, Vol. 79, No. 9, pp.: 1305–1320.

Halbwachs, N., Raymond, P. (1999). "Validation of synchronous reactive systems: from formal verification to automatic testing," in *Asian Computing Science Conference (ASIAN'99)*, Phuket, Thailand. LNCS 1742, Springer Verlag.

Hardebolle, C., Boulanger, F., Marcadet, D., Vidal-Naquet, G. (2007). "A Generic Execution Framework for Models of Computation," in *Proc. of MOMPES'07*, pp.: 45–54.

Iwasaki Y., Farquhar A., Saraswat V., Bobrow D., Gupta, V. (1995) "Modeling Time in Hybrid Systems: How Fast Is 'Instantaneous'?," in *Proceedings of the Ninth International Workshop on Qualitative Reasoning*, pp.: 1773-1780.

Lee, E.A., Sangiovanni-Vincentelli, A. (1996). *The Tagged Signal Model: A Preliminary Version of a Denotational Framework for Comparing Models of Computation*, Memorandum UCB/ERL M96/33, ERL, University of California, Berkeley, CA 94720.

Lee, E. A., Zheng, H. (2005). "Operational semantics of hybrid systems," in Proc. of HSCC'05, Volume LNCS 3414, pp.: 25-35.

Lee, E. A., Zheng, H. (2007). "Leveraging Synchronous Language Principles for Heterogeneous Modeling and Design of Embedded Systems," in *Proc. of EMSOFT'07*, Salzburg, Austria, pp.: 114–123.

Mosterman, P.J. (2002). "HYBRSIM—A Modeling and Simulation Environment for Hybrid Bond Graphs," in *Journal of Systems and Control Engineering*, Vol. 216, Part I, pp.: 35–46.

Mosterman, P.J., Zander, J., Hamon, G., Denckla, B. (2009). "Towards Computational Hybrid System Semantics for Time-Based Block Diagrams," in *Proc. of ADHS'09*, A. Giua, C. Mahulea, M. Silva, J. Zaytoon (eds.), pp.: 376–385, plenary paper, Zaragoza, Spain.

Mosterman, P.J. and Biswas, G. (2000). "A Comprehensive Methodology for Building Hybrid Models of Physical Systems," *Journal of Artificial Intelligence*, 121, pp.: 171-209.

Nishida, T., Doshita, S. (1987). "Reasoning about discontinuous change," in *Proceedings of National Conference on Artificial Intelligence (AAAI-87),* pp.: 643-648.

Petzold, L.R. (1982). *A description of DASSL: A differential/algebraic system solver*, Technical Report SAND82-8637, Sandia National Laboratories, Livermore, CA.

Simulink® 7 User's Guide, MathWorks®, Natick, MA, March, 2010.

Simulink® 7, Writing S-Functions, MathWorks®, Natick, MA, 2007.

Symbolic Math Toolbox™, MathWorks®, Natick, MA, March, 2010.

Nielsen, H.R., Nielson, F. (1988). "Automatic binding time analysis for a typed λ calculus," *Science of Computer Programming,* 10:139-176.

Shenoy, R., McKay, B., Mosterman, P.J. (2007). "On Simulation of Simulink Models for Model-Based Design," in *Handbook of Dynamic System Modeling*, Paul A. Fishwick (ed.), Chapter 37, CRC Press.

United States Government Accountability Office, June 20, 2006, GAO-06-455R F-22A Tactical Aircraft, Washington, DC 20548.