METAMODEL BASED LANGUAGE AND COMPUTATION PLATFORM FOR ALGORITHMIC ANALYSIS OF HYBRID SYSTEMS

By

Abhishek Dubey

Thesis
Submitted to the Faculty of the
Graduate School of Vanderbilt University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

in

Electrical Engineering

August, 2005

Nashville, Tennessee

Approved:

Professor Takkuen John Koo

Professor Gabor Karsai

This work is dedicated to my family
without whom this would have been impossible.

ACKNOWLEDGEMENTS

I am grateful to all those who helped me complete this work, especially my advisor Dr. T. John Koo. He not only helped me by giving the right direction and vision to this research work, but also opened my eyes to the “bigger picture” of this entire area of research.

This work cannot be presented without outlining the efforts of my colleagues Hang Su and Xianbin Wu who worked with me on this research. We worked as a team in bringing this work to its current status. They helped me in developing some of the algorithms and computation of related results that have been presented in this thesis.

This thesis would not have been possible without the support of Dr. Thao Dang of Verimag and Dr. Ian Mitchell of University of British Columbia. They helped us by providing the necessary computation kernels and guided us in using them in our approach.

I am also grateful for the help provided by Qiaozhu Mei, Steve Nordstrom, and Stephen Rochelle during the early days of this work.

I am indebted to my colleagues Jie Chen, and Chip Clifton for a lot of insightful thoughts. Their help and suggestions saw me through many rough spots. Special thanks to Dr. James Davis and Dr. Greg Nordstrom for introducing me to the MIC approach. Also, I would like to thank Dr. Gabor Karsai for helping me review and improve this thesis.

Finally, thanks to all of my friends and family who supported me through out graduate school. Their presence made this journey worthwile.

This work is supported by NSF’s Faculty Early Career Development (CAREER) Program, Award No. 0448234, and Department of EECS, Vanderbilt University.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

LIST OF SYMBOLS

2^X	Power set of X.
Δt	Maximal time step for computing reachable sets using a specific computation method.
$\emptyset$	Null set.
$\mathbb{Z}$	Set of Integers.
ϕ	Flow associated with the continuous evolution of hybrid automaton.
$\mathbb{R}$	Set of real numbers.
$\mathbb{R}^n$	n-dimensional continuous real space.
$\wedge$	Logical And.
$cPost_{cT}$	Constrained bounded time continuous successors for a time range $[0, T]$.
$cPre_{cT}$	Constrained bounded time continuous predecessors for a time range $[0, T]$.
D_{q_i}	Domain associated with a discrete state $q_i \in Q$.
f_{q_i}	Vector field associated with a discrete state $q_i \in Q$.
$G_{q'q''}$	Guard associated with an edge between two discrete states $q', q'' \in Q$.
$I = [0, T]$	A continuous time range from 0 to T. $I \subseteq \mathbb{R}$.
$Post_d$	Discrete successor sets.
$Post_{cT}$	Bounded time continuous successors for a time range $[0, T]$.
Pre_d	Discrete predecessor sets.
Pre_{cT}	Bounded time continuous predecessors for a time range $[0, T]$.
Q	Set of discrete states of a hybrid automaton.
$R_{q'q''}$	Reset map associated with an edge between two discrete states $q', q'' \in Q$.
X	Continuous state space. $X \subseteq \mathbb{R}^n$.

CHAPTER I

INTRODUCTION

The information and digital revolution that has taken place in the last half of 20th century has changed the world around us. Integration of information processing with physical processes has enabled the use of computing devices for purposes of controlling modern systems ranging from toy robots to complex space crafts with minimal human intervention. Such systems with one or more dedicated computing devices are called embedded systems, while the processing of information integrated with physical processes in these systems is called embedded computing. Pervasiveness of embedded computing is evident by the fact that almost 95% of the microprocessors produced are utilized for embedded computing tasks [63].

Formalizing methodologies for development of embedded systems is one of the most formidable challenges faced by industry and academia today. Embedded systems are required to meet multiple design objectives, while satisfying the requirements for system performance and system reliability. Due to limitation of physical resources such as power sources, traditional approach of maintaining exaggerated safety-margins in order to meet the requirements is not acceptable in embedded systems design.

In order to address these challenges, a new system theory that provides a solid mathematical backbone for embedded system design is required. While traditional system theory focused on behaviors of homogeneous idealized systems, the new theory needs to address the inherent heterogeneity of embedded systems, which is present due to the coupling of physical processes with the information world.

Hybrid system theory [46] is being developed to provide a mathematical foundation for study of systems that contain both physical and computation processes. A typical example of hybrid system is an embedded software system, which interacts with physical environment using sensors and actuators. This new theory is a bridge between traditional control systems theory, which is inadequate for capturing computational aspects of embedded system, and classical computer science theory, which is inadequate for capturing the physical processes related to embedded systems.

Model-based design [61, 33, 65] provides a scalable methodology for system design and analysis based on hybrid system theory in order to integrate efforts in system specification, design, synthesis, validation, verification and design evolution. In this approach, mathematical foundation provided by hybrid system theory enables the encapsulation of related aspects of embedded systems as models. Design complexity is handled by supporting the manipulation and integration of models with manageable complexity during different design phases. Model transformation techniques are useful in refining the model for design evolution by either adding the implementation details for system generation, or abstracting away the implementation details for managing complexity. Moreover, model transformation techniques provide capability to infer analytical models from design models that can be used as input to tools that perform validation and verification.

In this thesis, we describe the use of model based approach for system design, validation and verification. Currently, we have developed a platform for formalizing the design and analysis of models for hybrid systems. In the future, we will expand the scope of this platform and develop a chain of integrated tool suites that can be used for system development from system design to system generation.

The rest of this chapter will proceed as follows: first, an introduction to model driven system development will be given; then, a review on hybrid systems will be provided; the last three sections will describe the scope of this thesis, which is to provide a platform and modeling language for designing and analyzing hybrid system models.

Model Driven System Development

In [56], the act of *modeling* has been described as, "representing a system formally in order to describe and analyze the working of some relevant portions of the concerned system". A model could be *mathematical*, in which case it can be viewed as assertions about properties of the system such as its functionality and physical appearance. Such mathematical models are often found in many disciplines such as control engineering [52] where the functionality of a plant is described using ordinary differential equations (ODE). A model can also be *descriptive* [35], in which case it defines a formal computation procedure to emulate certain behaviors of a system. Descriptive models should be machine readable in order to enable the use of various algorithms to ascertain the expected behavior of system in response to environmental stimuli. This is necessary because models of complex systems might not be tractable for a human being to analyze. A model can also be *generative*, in which case it possesses enough information to be able to automatically generate the actual system.

An approach to system design and analysis is to use model driven system development [33] for using models in all stages of system development from design and analysis to production. In this approach, models are used to formally encapsulate the necessary information of the system and perform analysis or even design changes before actual implementation. The advantage of using models is that (i) models allow system perspectives at varying level of details in order to provide a less complex view of the whole design; (ii) models enable formalization of correct methods, which when reused, guarantee that the system would meet the design specifications; (iii) models enable use of formal validation and verification techniques.

In model driven system development, models are used as input and output in all stages of system development from system specification to production. Multiple models are used to encapsulate a particular *design* of system. Such models are also known as design models. The task of *analysis* uses descriptive models which can be inferred from the design models in order to ascertain if a certain design meets the specification. Thereafter, the design of a system can be successively refined in order to assure that corresponding system would have the desired functionality. Model transformation rules ensure that analysis models can be translated to design models and vice-versa. After refining the design, if the models are machine readable, by providing correct transformation rules, models can be transformed into executable artifacts that can be deployed for production of actual system. One such model driven system development methodology is Model Integrated Computing (MIC) [65, 41].

MIC brings in key concept of domain modeling to the paradigm of model driven system development. It has been argued that no single modeling language can be used to describe all kinds of models[41]. Rather, systems should be modeled using modeling languages tailored to the needs of a particular domain. A key capability supported by MIC is the definition and implementation of *domain-specific modeling languages* (DSMLs). Crucial to the success of DSMLs is *metamodeling* and *auto-generation*. A *metamodel* defines the elements of a DSML, which is tailored to a particular domain. The modeling language which is used to construct metamodels is known as metamodeling language. Auto-generation involves automatically synthesizing useful artifacts from models, thereby relieving DSML users from the specifics of the artifacts themselves, including their format, syntax, or semantics.

The Generic Modeling Environment (GME) [45] is a design tool suite which provides an end-end solution for building and deploying MIC applications. It provides graphical modeling environment for specifying the metamodels and consequently domain specific models. Graphical modeling has been preferred over text based models since pictures provide intuitive management of information. Moreover, graphical modeling allows use of hierarchy to encapsulate information for abstraction. The metamodels in GME are specified using a metamodeling language called MetaGME. GME has built-in mechanism to implement tools for transforming metamodels to modeling language, analyzing models or using models to generate actual implementation of the system.

Object of Study: Hybrid Systems

Hybrid systems are heterogeneous systems that include continuous-time components interacting with discrete components. Such systems include embedded systems with software and hardware components executing in discrete time steps and physical environment executing in continuous time. A typical example of a hybrid system is the transmission system of an automotive. In the transmission systems, each gear shift marks a change in continuous dynamics of the automotive engine. The change in continuous dynamics affects the speed and torque ratio in the engine. Thus each gear position can be considered as a mode of operation in which different continuous dynamics of engine are applicable.

Modeling and study of continuous systems and discrete systems have been developed as two separate cultures, control engineering, and computer engineering. Hybrid systems lie on the boundary of these two cultures. Traditionally, ordinary differential equations (ODEs) models (see [57]) have been extensively used to study continuous dynamical systems since they can model rich non-linear phenomenon. On the other hand, discrete modeling structures such as Finite State Machine (FSM) [35] have been used to study the discrete systems. In order to analyze systems with tightly coupled discrete and continuous components, a model representation is required which cancombine the two paradigms.

Timed automaton [6] is a model which combines the discrete modeling structure of finite state machine with simple continuous dynamics. It allows the study the temporal properties of systems. Timed automaton has been proved to be very effective in modeling a large range of phenomena in diverse domains such as systems with real-time constraints [18], scheduling in manufacturing systems [1].

Hybrid automaton [36] is another model, which combines the discrete modeling structure of finite state machine with complicated continuous dynamics modeled as ODE. These models have been used by control engineers to model the environments they want to control along with the discrete components. Use of these models has been demonstrated in various applications such as coordinating robot systems [7], automobiles [12] and aircrafts [70].

Hybrid automaton and timed automaton are used to perform validation of a design of hybrid systems. *Simulation* is the most widespread validation technique used. A number of software tools provide simulation of hybrid systems, including Charon [5], HyVisual [17], Modelica [68], and Simulink/Stateflow[1]. However, one shortcoming of simulation methods is that they examine only one possible system behavior at a time. They cannot be used to check if all the possible system behaviors satisfy certain property.

[1] Simulink and Stateflow are registered trademarks of Mathworks corporation

An alternative form of more rigorous validation is *verification.* Formal verification refers to methods for determining whether or not some given properties (specifications) are true for a given model of dynamic system. Verification of discrete-state systems has gained industrial importance, especially in the area of hardware design and communication protocols [23]. Extending these techniques to continuous systems and thus hybrid systems has been a challenge. Apart from validation, hybrid automaton and timed automaton are also used for *synthesis* of useful design parameters for refining the design so that it meets certain functional requirements.

In general, there are two approaches to formal verification: *theorem proving* and *model checking.* In the former, a specification for a system model is either inferred or contradicted using a set of logical deductions, whereas in the latter approach, various states of the systems are explored to arrive at the set of states which satisfies the specification. The attraction of theorem proving approach is that it is not restricted to finite state systems. However, theorem proving usually requires human intervention. In contrast, algorithms exist and have been implemented for model checking techniques. In these techniques, concept of reachable set that formalizes all possible system behaviors for a given set of system states, is used. For certain class of hybrid systems [38], the state space can be reduced to an equivalent finite bisimulation, which can be explored for verifying certain properties. However, in other classes direct exploration of infinite state space is performed by using symbolic representation of continuous sets. In [39], the former approach has been categorized as reductionist, while the latter has been termed as symbolic.

Need for a New Design and Analysis Language for Hybrid Systems

In symbolic method based approach, only for certain classes of dynamical systems as discussed in [4], continuous state sets can be exactly represented. In most cases, computation methods have to be used for representing state sets. Boolean operations on continuous state sets depend heavily on the geometry chosen to represent the continuous state sets. Since, for most hybrid systems, computation of exact reachable set is undecidable, various methods [51, 10, 20, 73, 37, 14] have to be employed for computing approximation of reachable sets. Equipped with representation of continuous state sets, Boolean operations on state sets, and approximate reachable set, a number of algorithms can be developed for model checking the hybrid automaton.

Various computation tools have been developed for specific analysis needs and have specific computation capabilities. For a class of linear hybrid systems, Checkmate [20] is used for verification. It represents continuous state sets [64] as flow pipes, and approximates reachable sets by using numerical integration and polyhedral approximation. Checkmate then converts the hybrid automaton model into a polyhedral-invariant hybrid automaton for purposes of model checking. Another tool for linear hybrid systems is d/dt [10], which represents continuous state sets as union on convex polyhedron and then uses numerical integration methods for computing reachable sets. Unlike Checkmate, d/dt performs model checking directly on the hybrid automaton model by using reachable sets. For non-linear systems, Level Set method [53], applied in Level Set toolbox [51], might be used. This method represents continuous state sets using a higher dimensional implicit level set function. Level Set method computes the evolution of a continuous set governed by an ordinary differential equation as the interface evolution problem by solving the associated partial differential equation.

Currently, due to diversity in representation of continuous state sets and methods for computing reachable sets in the computation tools, use of algorithms for purpose of analyzing models of hybrid systems has been seriously constrained. However, we feel that algorithm design should be made at an abstract level which hides the possible continuous state sets representation and methods for computing the reachable sets. In fact, for designing analysis algorithm, only the mathematical definition of reachability operations and Boolean state set operation is required. Moreover, some of the tools lack the support data structures such as multidimensional lists needed for algorithmically exploring the hybrid state space.

For this purpose, we want to enable a computation platform which allows the design of analysis algorithms as models that can abstract away the implementation details. Use of models, when coupled with model transformation techniques, will enable adoption of various computation methods for set representation and computation of reachable sets. This would be necessary in order to implement the analysis algorithms. In order to adopt the computation methods we want to enrich the existing computation tools with support structures, and use them as *computation kernels* for implementing the algorithms designed at the abstract level.

Scope of The Thesis

In the Embedded Computing Software Laboratory (ECSL) at Vanderbilt University, we have developed a metamodel based modeling language, Hybrid System Analysis and Design Language (HADL) [28], using the MIC paradigm offered by GME. Use of metamodel allows formal specification of all the concepts and notions needed to design hybrid system models and analyze them, such as reachable set operations and Boolean set operations (such as union and intersection). HADL is designed based on the mathematical definitions of these operations and is designed to ensure that there exists a correspondence between the semantics of computation kernels and the semantics of HADL. Therefore, one can use the semantics of HADL to anchor the semantics of these kernels, which is referred to as semantic anchoring in [19]. Because of this feature, we can design analysis algorithms by using the mathematical semantics of these operations instead of considering the detailed implementation of continuous state sets and the corresponding reachable set operations. Furthermore, HADL enriches the functions of its computation kernels by providing constructs and operations more than these computation kernels, such as multidimensional list and its corresponding operations. Currently we have enriched the computation tools d/dt and Level Set toolbox and use them as computation kernels.

Figure 1, which is derived from the MIC multigraph architecture [66], depicts how MIC approach is applied to encapsulate HADL and automate the design and implementation process. This architecture has three model development stages, namely meta-model, domain specific models and the executable artifacts. The first level is the meta-programming interface, which is used to define the meta-model of HADL. This meta-model is based upon abstract entities found in the symbolic method based computation tools and is

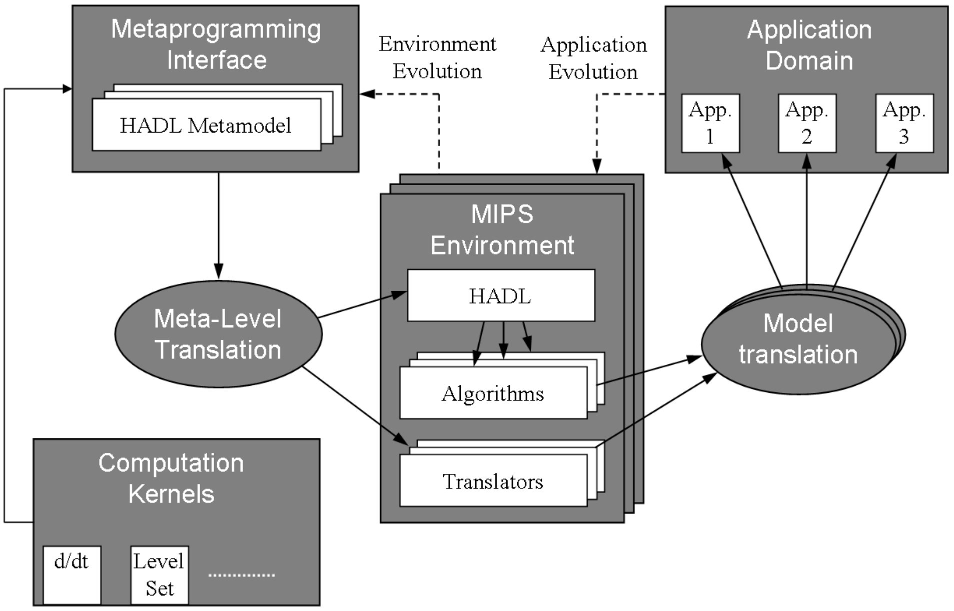


Figure 1: Creation of the ReachLab platform using MIC multigraph architecture.

later implemented as the domain specific modeling language, HADL, using the meta-translation facility provided by GME. Model-Integrated Program Synthesis (MIPS) environment [67] is the second level and provides tools to build and modify system models and the analysis algorithms using HADL. This level also supports construction of translators to automatically translate the algorithm models into implementation. The last level is the different applications (implementations) that can be generated by translators from these models. Environment evolution refers to modification of HADL meta-model to update features. The models of algorithms can also be refined to evolve the analysis application.

Using HADL, we have developed a computation platform called ReachLab for providing a graphical environment to analyze and design hybrid systems using models. This platform currently supports design of hybrid systems using a hybrid automaton model. In future, it can be expanded to other models as well. In this platform, (i) analysis algorithms are modeled in order to specify how the state space of the system model should be explored for performing analysis; (ii) concerns of design and implementation of analysis algorithms are separated by making use of a modeling language which provides abstraction from the implementation details; (iii) use of models automate the analysis process

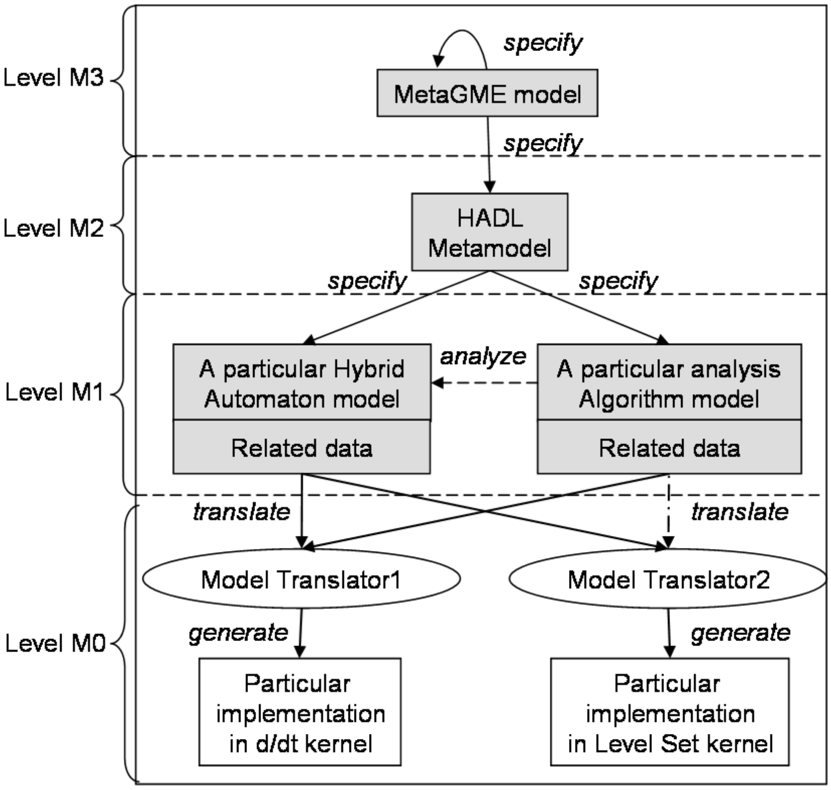


Figure 2: Various stages of use of models in ReachLab Architecture

by automatically generating implementation in order to compute results. In ReachLab, on one hand, the models of analysis algorithms are abstract and therefore the design of algorithms can be made independent of implementation details. On the other hand, translators are provided to automatically generate implementations from the models for computing analysis results based on computation kernels. Multiple computation kernels, which are based on specific computation tools such as d/dt and the Level Set toolbox, have been enriched to support numerous algorithms designed in ReachLab, in order to enable hybrid state space exploration.

ReachLab has three distinct levels of model development stage. Figure 2 illustrates these levels and different models that may exist. At the top level (M3) lies the MetaGME model which specifies the metamodeling language of HADL. In the next lower level (M2) lies the metamodel of HADL. Using this metamodel, an analysis algorithm is modeled which is used to analyze hybrid automaton based model of a system in the second level (M1). In the lowest level lies any particular implementation generated by using the models of analysis algorithm and system. This job is done using translators.

Thesis Outline

We provide an overview of hybrid automaton model in Chapter II. In the same chapter we would give an overview and classification of different computation tools developed for analysis of models of hybrid systems. We will introduce our domain specific modeling language in chapter III. In chapter IV, we will focus on the computation platform called ReachLab, which implements this language and provides various model translators using the facilities provided by GME. These translators are used to automatically generate model implementations for various computation kernels. In the same chapter we describe various enrichments that we made to d/dt and Level Set toolbox. Chapter V provides some case studies in order to illustrate the usefulness of ReachLab. We conclude with our comments for future works in chapter VI.

CHAPTER II

HYBRID SYSTEMS

Hybrid systems are dynamical systems that involve the interaction of different types of dynamics. The interaction of continuous and discrete dynamics have been of particular interest due to large number of engineering applications such as electrical circuits [59, 47], mechanical systems [54, 42], biological systems [3] and embedded computation [8, 58]. By their nature, these systems have both discrete and continuous state variables. While discrete states can change value only through discrete jumps, continuous states change values by "flowing" in continuous time according to a differential equation. A hybrid system has both of these dynamics.

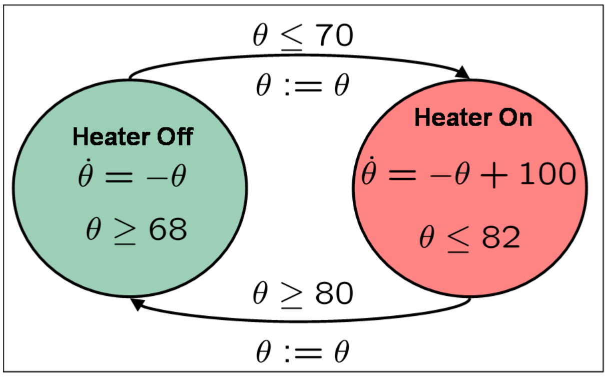


Figure 3: A model for thermostat

Consider a thermostat used to control the temperature of a room. Suppose it has to switch the heater off or on, if the current value of room temperature rises above 80 or falls below 70. Such a system can be studied by using a continuous state variable, $\theta \in X \subseteq \mathbb{R}$, in order to track the current value of room temperature. X here denotes the analysis set, which restricts the possible value of room temperature. For this example let us consider $X = [68, 82]$. Since the system will behave differently when the heater is on, as to when it is off, we can use two ordinary differential equations to model system behavior.

$$\dot{\theta} = \begin{cases} -\theta + 100 & if\ the\ heater\ is\ on \\ -\theta & if\ the\ heater\ is\ off \end{cases}$$

For such systems it is customary to depict them as a graph with each vertex denoting one possible mode of continuous evolution. The edges are associated with the conditions for switching between these two modes. Figure 3 gives a visual description of the thermostat in this form.

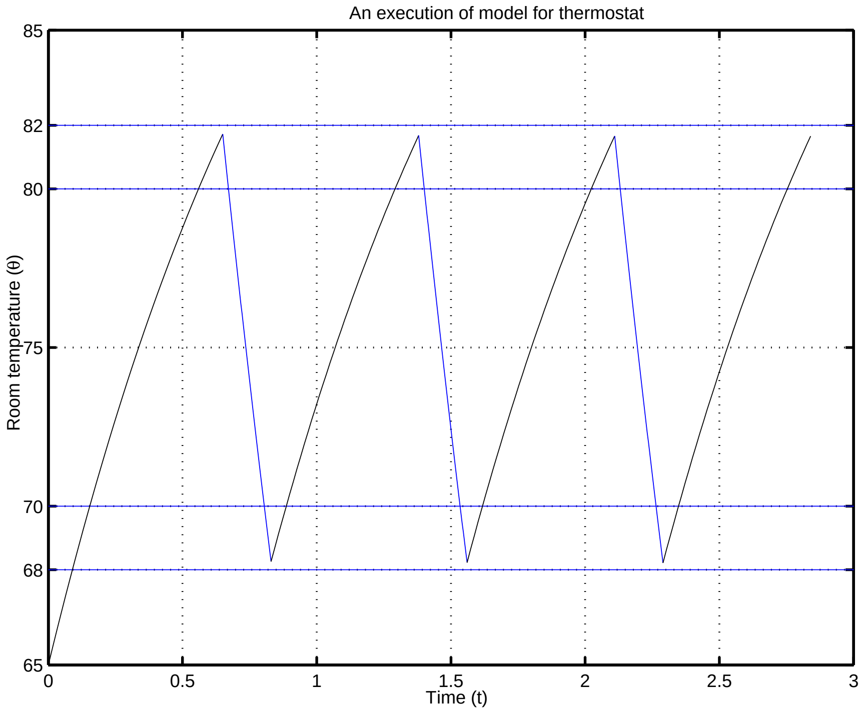


Figure 4: A possible trajectory of the thermostat

Note that this system has two discrete modes of operation and a continuous state variable. Thus this system can be considered as a hybrid system. To formally analyze these systems, models such as hybrid automaton [36, 48] and timed automaton [6] have been proposed. In this thesis, we are particularly interested in the hybrid automaton model of hybrid systems.

Hybrid Automaton

Hybrid automaton is a mathematical model used to formally encapsulate the behavior of hybrid systems with tightly coupled discrete and continuous components. In [48], the mathematical definition of a hybrid automaton has been given as follows:-

Definition 1 *Hybrid automaton is a collection* $H = (Q, X, f, Init, D, E, G, R)$ *where* $Q = \{q_1, \ldots, q_N\}$ *is a set of discrete states;* $X \subseteq \mathbb{R}^n$ *is the continuous state space;* $Init \subseteq Q \times X$ *is a set of possible initial states;* $f : Q \to (X \to \mathbb{R}^n)$ *assigns to every discrete state a time invariant Lipschitz continuous vector field on* $\mathbb{X}$*;* $D : Q \to 2^{\mathbb{X}}$ *assigns each* $q \in Q$ *a domain. D is sometimes also known as the invariant set;* $E \subseteq Q \times Q$ *is a collection of discrete transitions;* $G : E \to 2^{\mathbb{X}}$ *assigns each* $e = (q, q') \in E$ *a guard;* $R : E \times X \to 2^X$ *defines a reset relation.*

In this thesis, we shall use the notation f_{q_i} to indicate the vector field associated with a discrete state $q_i \in Q$. We will use the notation D_{q_i} to denote the domain of a discrete state $q_i \in Q$. We shall use the notation $G_{q'q''}$ to indicate the guard on the edge between discrete state $q', q'' \in Q$. We will use the notation $R(q', q'', x)$ or $R_{q'q''}(x)$ for the reset map between discrete states $q', q'' \in Q$ for a continuous state $x \in X$.

It is customary to draw a hybrid automaton as a directed graph as shown in Figure 3. Each vertex of the graph denotes a discrete state. The directed edges of the graph represent the edge of hybrid automaton. An edge is associated with the guard condition and reset map. Each vertex is associated with the corresponding vector field and domain.

Referring back to the thermostat example presented in the previous section, the hybrid automaton can be described as: $Q = \{q_1, q_2\}$; Continuous state: $\theta \in X \subseteq \mathbb{R}$, $X = [68, 82]$. $Init = \{q_1, q_2\} \times X$; $f_{q_1}(\theta) = -\theta + 100$, $f_{q_2}(\theta) = -\theta$; $D_{q_1} = \{\theta \in \mathbb{R} \mid \theta \leq 82\}$ and $D_{q_2} = \{\theta \in \mathbb{R} \mid \theta \geq 68\}$; $E = \{(q_1, q_2), (q_2, q_1)\}$; $G_{q_1q_2} = \{\theta \in \mathbb{R} \mid \theta \geq 80\}$ and $G_{q_2q_1} = \{\theta \in \mathbb{R} \mid \theta \leq 70\}$; The reset relation for both edges is an identity map.

The two discrete states represent the modes of operation "heater on" and "heater off". The analysis set for continuous state space is restricted to the region $\theta \leq 82 \wedge \theta \geq 68$. When the heater is on, heat is pumped into the room, and the dynamics of continuous state variable θ is given by $\dot{\theta} = -\theta + 100$. When the heater is off, heat dissipation happens at a rate of $\dot{\theta} = -\theta$. Considering that the room temperature can have any value between 68, and 82, and that the heater can be either off or on at that time, the set of all possible initial states is same as the complete state space of the system. There are two possible transitions for thermostat: from discrete state heater on to discrete state heater off (q_1, q_2), and from discrete state heater off to discrete state heater on (q_2, q_1). The guards represent the requirement that heater should be switched on when the temperature falls below 70 and should be switched off when the temperature rises above 80.

Execution of Hybrid Automaton

Hybrid automaton has two types of possible state evolutions: discrete and continuous. Before describing the discrete evolution we must understand the concept of continuous evolution.

In general, for the continuous evolution during a time interval $I = [0, T]$, it is required that the discrete state remains unchanged during the interval. Let us suppose hybrid automaton is in a discrete state q' during the whole time interval I. Then the evolution of hybrid automaton during that interval is given by the flow of continuous variables under the influence of vector field $f_{q'}$. Starting from $x_s \in X$ at $t = 0$, the continuous state would be specified by the solution of ordinary differential equation (ODE) $\dot{x}(t) = f_{q'}(x(t)),\ x(0) = x_s, t \geq 0$. We assume the vector field to satisfy Lipschitz condition [32] over the entire continuous space X. Then a solution in the sense of caratheodary means a continuous differential function of time $x(t)$ satisfying

$$x(t) = x_s + \int_0^t f(x(\tau))d\tau \tag{1}$$

The continuous state of the system at time $t' \in I$, starting from x_s at t=0 given by $x(t')$ is called the flow and is denoted by $\phi(t', x_s)$. It is assumed that this flow is well-defined in the domain associated with discrete state q'. The flow satisfies the following conditions:

$\forall x_s \in X_s \; \forall t, t' \geq 0$ such that

$$\begin{array}{rll} \text{a.} & \phi(0, x_s) = x_s & \text{init. condition} \\ \text{b.} & \dot{\phi}(t, x_s) = f(\phi(t, x_s)) & \text{differential eqn.} \\ \text{c.} & \phi(t', \phi(t, x_s)) = \phi(t' + t, x_s) & \text{semi} - \text{group} \\ \text{d.} & \phi^{-1}(-t, x_s) = \phi(t, x_s) & \text{inverse} \end{array} \tag{2}$$

For continuous evolution to be part of a legitimate execution the flow should always be inside the domain associated by discrete state q' i.e. Given a time interval I during which the hybrid automaton evolves due to continuous evolution starting from any continuous state $x_s \in X$, $(\forall t \in I)(\phi(t, x_s) \in D_{q'})$.

Discrete evolution of hybrid automaton happens from a discrete state $q' \in Q$ to another discrete state $q'' \in Q$ via an edge $(q', q'') \in E$ at any time t, if $\phi(t, x_s) \in G_{q'q''}$. After a discrete transition occurs, the continuous state of the system is reset according to the corresponding reset map $R(q', q'', \phi(t, x_s))$.

For more details on semantics and execution of hybrid automaton readers may refer to [48, 36].

Issues Related to Hybrid Systems

The effort in study of hybrid systems [36] can be categorized in the following sub areas:

- *Formal verification* of safety properties aims at certifying that for a given set of possible system states, the hybrid system would never exhibit an unsafe behavior.
- *Synthesis* of design parameters for designing controllers which would govern the operation of the hybrid system.

A key concept required in answering both of the above problems is reachable set. Reachability is a fundamental concept in the study of hybrid systems. In general, a state $(q^{'}, x^{'}) \in Q \times X$ of a hybrid automaton is said to be reachable if the hybrid automaton

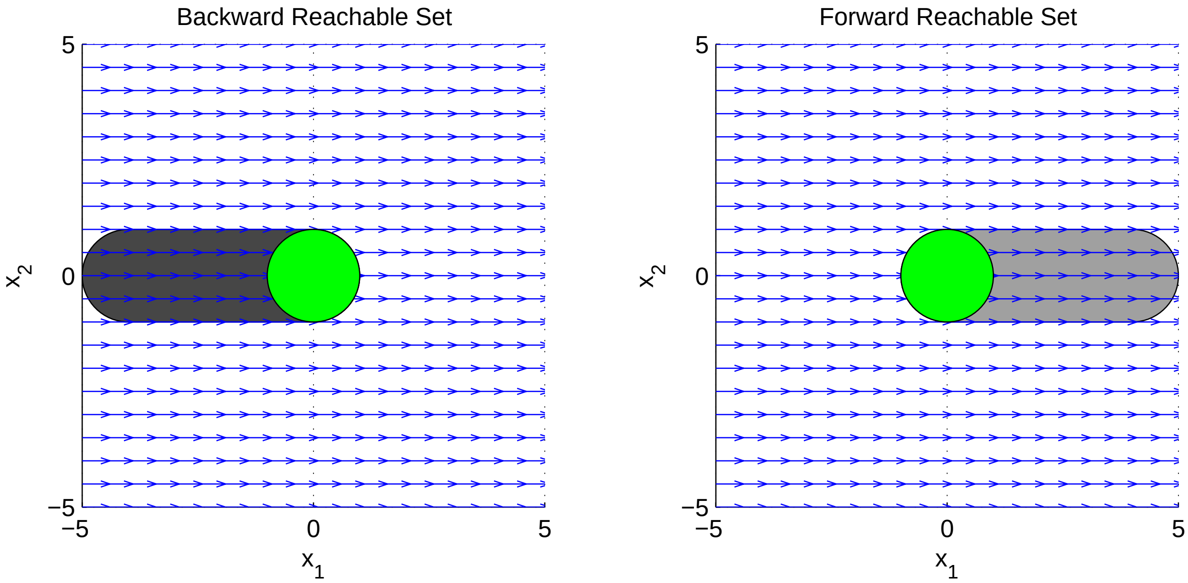


Figure 5: This figure shows backward and forward reachable set for a hybrid automaton while its discrete mode remains unchanged. The arrows show a constant vector field in the x direction.

can move along one of its possible executions and find its way to that particular state. A set of such states which can be reached by the hybrid automaton are called reachable sets. Using reachable sets, problem of verification can be specified as

Problem 1 (Verification) *Given an initial set of states for the hybrid automaton, would any finite execution of the hybrid automaton starting from one of the initial states ever reach somewhere inside the bad set.*

Figure 5 illustrates the notion of forward and backward reachable sets. The vector field for this example can be understood as a constant field of magnitude 1 in the x direction. The starting set is an ellipse on origin as shown in the figure. The light gray region shows the forward reachable set which can be reached in a time interval of 3 seconds. The dark gray region illustrates the backward reachable set for a time interval of 3 seconds.

In order to use algorithmic methods for model checking the hybrid automaton using concepts of reachable sets one requires various approximations for representing continuous state sets.

Representation of Continuous State Sets

Only for certain classes of continuous dynamical systems as discussed in [4, 44], continuous state sets can be exactly represented and manipulated and the algorithms can be performed at symbolic level. In most cases, computational methods have to be used for representing and manipulating continuous state sets. Due to finite precision in representation and computation, only approximate results can be obtained.

There are a number of approaches to represent state sets and perform computations for more expressive continuous dynamics, for example, polygonal projections [34], flowpipes [21, 64], ellipsoids [15, 43], griddy polyhedra [11], level sets [51].

In [34, 21, 64, 15, 43], an interface, which is the boundary of a state set, is tracked by interface elements and the evolution problem is formulated in a *Lagrangian* framework. A Lagrangian method provides a numerical scheme based on a parameterized description of the moving interface, and since the method follows a local representation of the interface rather than using a global one, it can suffer from numerical instability and topological limitations.

In [51, 11], the representations and operations of state sets are performed in an *Eulerian* framework, that is, one in which the underlying coordinate system remains fixed. This framework makes set operations and more advanced constructive solid geometry operations straightforward to apply.

Reachability Operations

In order to automate the analysis of hybrid automaton model using the reachable set let us try to define some basic reachability operations. Using these reachability operators we would be able to write several algorithms for performing analysis of hybrid automaton. We reviewed in previous sections that a hybrid automaton has two types of evolutions, discrete and continuous. Therefore, one can classify the predecessors and successors sets as either discrete or continuous.

Definition 2 (Continuous successor and predecessor set) *Continuous successor set, $Post_{cI}(q,P) : 2^X \rightarrow 2^X$, where $q \in Q$ and $P \subseteq X$ is a collection of states in X, and each state can be reached by a state in a P following the dynamics of discrete mode q in some time set specified by the interval $I \subset \mathbb{R}^+$. This can be expressed as:*

$$Post_{cI}(q,P) = \{(q,x')|\exists x' \in X \ \exists y \in P \ \exists t \in I \ s.t. \ y = \phi(t,x')\} \tag{3}$$

One can define continuous predecessor set, in a fashion similar to continuous successor set. Continuous Predecessor set $Pre_{cI}(q,P) : 2^X \rightarrow 2^X$ is given by

$$Pre_{cI}(q,P) = \{(q,x')|\exists x' \in X \ \exists y \in P \ \exists t \in I \ s.t. \ x' = \phi(t,y)\} \tag{4}$$

Usually the time interval I is bounded i.e. $I = [0,T]$. Then the continuous successor (predecessor) set is called *bounded continuous successor (predecessor) set.* For bounded continuous successor (predecessor) set, $Post_{c[0\ T]}$ ($Pre_{c[0\ T]}$), we would use the notation $Post_{cT}$ (Pre_{cT}) through out this thesis.

During continuous evolution in a time interval $I = [0,T]$, there may exist some state constraints that the continuous state variables have to satisfy. These, constraints specify the continuous set $\psi \subseteq X$, in which the continuous state of the system must operate for all time. In typical hybrid automaton this constraint set is same as the domain of discrete state active during that time interval.

Definition 3 (Constrained continuous successor and predecessor set) *Constrained continuous successor set is represented by $cPost_{cI} : (q,P) : 2^X \rightarrow 2^X$ where $q \in Q$ and $P \subseteq X$ is a collection of states in X. Given a constraint set $\psi \subseteq X$, constrained continuous successor set is given by the equation*

$$cPost_{cI}(q,P) = \{(q,x')|\exists x' \in X \ \exists y \in P \ \exists t \in I \ s.t. \ y = \phi(t,x') \wedge \forall \tau \in [0,t] \ \phi(\tau,x') \in \psi\} \tag{5}$$

One can define constrained continuous predecessor set, $cPre_{cI}(q, P) : 2^X \rightarrow 2^X$, in a similar fashion.

$$cPre_{cI}(q, P) = \{(q, x')|\exists x' \in X \; \exists y \in P \; \exists t \in I \; s.t. \; x' = \phi(t, y) \wedge \forall \tau \in [0, t] \; \phi(\tau, y) \in \psi\} \tag{6}$$

Usually the time interval I is bounded i.e. $I = [0, T]$, hence bounded time constrained successor (predecessor) can be denoted by $cPost_{cT}$ ($cPre_{cT}$).

Definition 4 (Discrete successor sets) *Discrete successor sets, $Post_d : Q \rightarrow 2^Q$, for a state $q_i \in Q$ is given by $Post_d(q_i) = \{q \in Q \mid \exists e \in E \; s.t. \; e = (q_i, q)\}$.*

Definition 5 (Discrete predecessor sets) *Discrete predecessor sets, $Pre_d : Q \rightarrow 2^Q$, for a state $q_i \in Q$ is given by $Pre_d(q_i) = \{q \in Q \mid \exists e \in E \; s.t. \; e = (q, q_i)\}$.*

Once we know the discrete successor set of a discrete state, one can use the intersection operation to check if the corresponding guard condition is satisfied. If the guard condition if satisfied then the execution of automaton would continue with continuous evolution with the initial continuous state for new discrete mode being computed by the reset map. We have implemented the algorithm for computing the reachable set using these basic reachability operator and we will present it later in a case study.

Computation of Reachable Sets

Equipped with these reachable operators one can ask questions like whether the hybrid automaton would ever have an execution which would lead it to some unsafe operating states. Backward reachability based operators can be used to rule out a set of initial conditions which might make the hybrid automaton operate in unsafe states. However, the most formidable problem in analysis of hybrid automaton models is the actual computation of these reachable sets. For most of the systems the hybrid state space in infinite, therefore potentially the true reachable set would also be infinite. This makes the problem difficult [4].

A formidable research initiative in this area has been dedicated for developing methods to get some reasonable approximate estimate of the reachable set. This has lead to a variety of algorithmic methods for verification [4, 13, 34, 11] and controller synthesis [72, 26, 9]. These different methods have been implemented in various computational tools such as UPPAAL [14], HyTech [37], or KRONOS [73], d/dt [10] , Level Set toolbox [51], Checkmate [20]. Most of these approaches can be classified into two groups, *reductionist* approach and *symbolic* approach [39]. The reductionist approach attempts to solve the problem by dividing the infinite state space into equivalent regions which are finite in nature and then perform analysis on those equivalent regions. The symbolic approach attempts to use various geometric approximations to represent continuous sets and then rely on exploring the whole state space by algorithmic methods.

Reductionist methods have been employed in tools such as UPPAL, HyTech and KRONOS. However, these tools are unsuitable for systems with complex continuous dynamics. On the other hand symbolic methods based tools such as Level Set tool box and d/dt have their own representations for continuous state sets and their own implementation of the reachability operations discussed in the previous section. Each of these tools has been defined specifically for certain analysis purposes and does not provide flexibility for designing generic algorithms that can be used for a number of analysis purposes. Moreover, they sometimes lack certain operations needed for writing certain analysis algorithms. For example, both Level Set toolbox and d/dt lacks support for discrete successor and predecessor operations as defined in previous section.

We chose the computation tools d/dt and Level Set toolbox to form computation kernels for our architecture because both of them use an *Eulerian* framework for operation on state sets. This framework makes set operations and more advanced constructive solid geometry operations straightforward to apply. Due to the use of grids, the analysis space is divided into finite number of divisions and hence the algorithms are guaranteed to terminate.

Next two sections will illustrate different implementations of d/dt and Level set toolbox of the same continuous successor operation.

Introduction to d/dt

d/dt [27, 10] can perform verification or synthesis for a class of hybrid systems that have affine continuous dynamics of the form $f(x) = Ax + Bu, x \subseteq \mathbb{R}^n$, where $u \in \mathbb{R}^n$ is uncertain, bounded input ranging inside a convex polyhedron, A is a $n \times n$ matrix and B is a $n \times 1$ matrix . Unlike the conventional approaches which attempt to find exact solutions and are thus limited by undecidability of most non-trivial systems, d/dt attempts to compute the reachable set as an over- or under-approximation of the exact reachable set. Usually in verification analysis over-approximation is used and in synthesis applications under-approximation is used.

d/dt represents continuous sets as convex polyhedra and makes use of the premise that convex sets maintain convexity under affine transformations. Therefore, if the vector field is affine (which is always the case for systems which can be analyzed using d/dt) an initial convex set will have a forward reachable set which is also convex in nature. Note that this notion of using polyhedral representation for interface evolution is a Lagrangian approach. Since the initial set is represented as union of convex polyhedron in d/dt its reachable set also forms a union of convex polyhedron. In order to effectively manipulate these unions of convex polyhedron d/dt approximates them as orthogonal polyhedra [16] which are represented by using a grid structure over the analysis space. Thus the continuous sets in d/dt are represented and manipulated in an Eulerian framework.

In order to compute the reachable sets for a time range $[0, T]$, d/dt uses an iterative algorithm which computes the reachable set for a maximal time range $[0, \Delta t]$ $(\Delta t > 0)$ for $N = T/\Delta t$ steps. The procedure to compute reachable set for a maximal time range $[0, \Delta t]$ as illustrated by Figure 6 can be summarized as follows:-

1. Starting from the initial set F, compute the reachable set after the exact time Δt. Since the convex set maintains convexity, this reachable set would be given by the line segment shown by $Postc_{\Delta t}(F)$ in the figure.

2. In order to approximate the whole reachable region (set) in the time range $[0, \Delta t]$, d/dt forms a convex hull on the initial set F and the set $Postc_{\Delta t}(F)$.

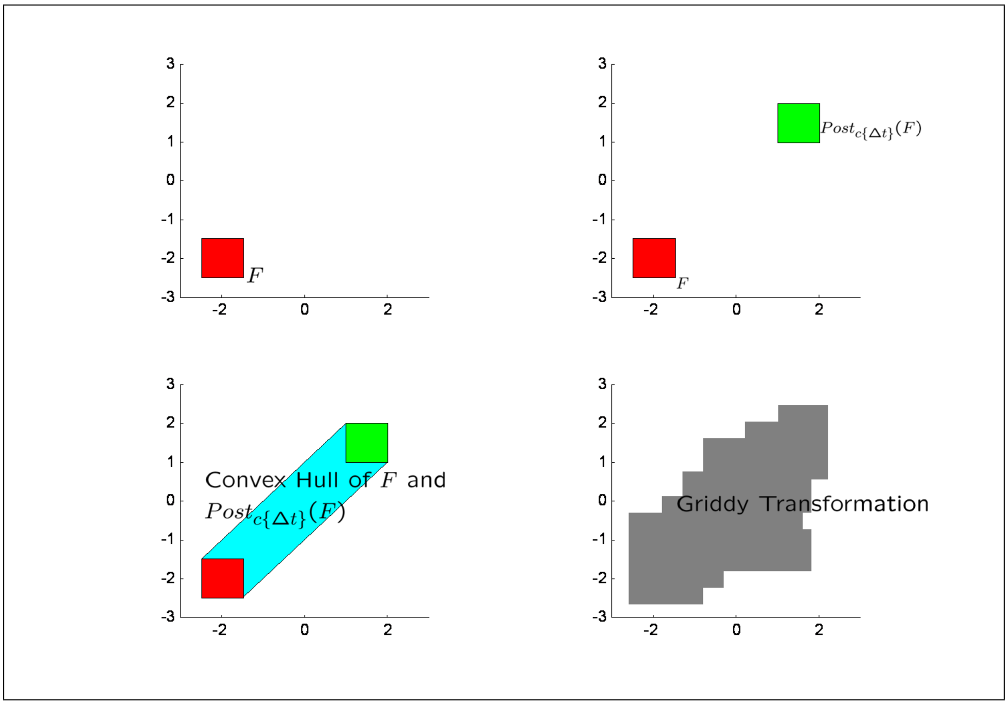


Figure 6: Reachable Set Computation in d/dt. From top left to right: first figure shows the initial continuous set F. Second figure shows computation of successor set starting from convex set F in a single time step Δt. The next figures show the convex hull operation and the bloat operation. Last figure shows the griddy transformation to convert the convex set into union of orthogonal polyhedron.

3. In order to ensure an over- or under- approximation a bloat operator is used. Consider a convex polyhedron C specified by the intersection of several half spaces. This can be understood as shifting each of the half spaces by some real number d. It has been shown in [27] that a positive (negative) real number d can be chosen such that the approximate reachable set is always a superset (subset) of the exact reachable set to provide over (under) approximation.

4. This whole procedure is repeated for $N = T/\Delta t$ steps. The reachable set of each step is stored as union of an orthogonal polyhedron by using a grid formed over the analysis space.

5. At the end of the procedure the union of orthogonal polyhedron is returned as an approximation of the reachable set for the time range $[0, T]$.

In principle, to compute bounded time constrained continuous successors subject to the constraints of a domain D can be performed by d/dt. However, currently it only implements an algorithm which gives an under-approximation of the constrained reachable set since it stops the algorithm iteration as soon as in any step the reachable set after the exact time Δt does not remain completely inside the domain D.

The computation methods implemented in d/dt are faster than those implemented in Level Set. Therefore, for linear systems d/dt is used. However, it is unable to handle separation and merging of continuous state sets. For such scenarios level set toolbox has to be used.

Introduction to Level Set Toolbox

The Level Set toolbox [51] is based on the level set methods developed by Oshley and Sethian [53, 60]. This toolbox can be used for both linear and nonlinear continuous dynamical systems.

It has distinct advantages because it can implicitly handle state constraints and can be applied to both linear and non-linear continuous dynamical systems. This method uses an Eulerian framework by dividing the whole analysis space into grids. Rather than an explicit representation in terms of edges or faces, in the level set methods the continuous set $P \subseteq X \subseteq \mathbb{R}^n$, is implicitly represented as an interface by a level set function $\Phi : \mathbb{R}^n \times \mathbb{R} \rightarrow \mathbb{R}$. At any time t, the current boundary of reachable set is given as the set $\{x \in \mathbb{R}^n | \ \Phi(x, t) = 0\}$.

The initial continuous set, say X_s provides the initial value condition for level set methods i.e.

$$(\forall\ x \in X_s)(\Phi(x, 0) = 0). \tag{7}$$

Consider a continuous state, as shown in Figure 7, which is evolving under the influence of a time invariant vector field $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$. At any time t the solution of equation $\Phi(x, t) = 0$ gives the boundary of reachable set at that instant.

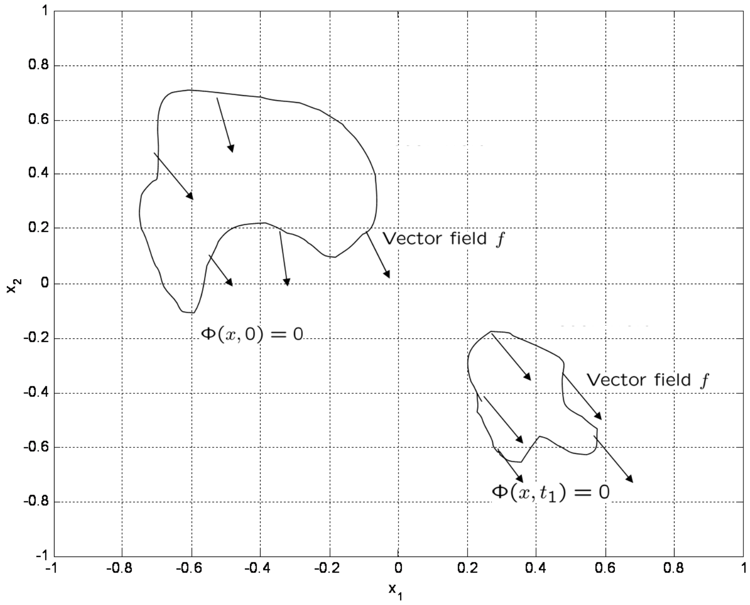


Figure 7: This figure illustrates the representation of continuous set using an implicit level function. At any time t, the solution of equation $\Phi(x,t) = 0$ provides the boundary of reachable set after exact time t.

Partially differentiating $\Phi(x,t) = 0$ with respect to t, we arrive at the following equation

$$\frac{\partial \Phi(x,t)}{\partial t} + \nabla \Phi(x,t) \cdot \frac{\partial x}{\partial t} = 0 \tag{8}$$

However, since $\frac{\partial x}{\partial t} = f(x)$ we can rewrite the above equation as

$$\frac{\partial \Phi(x,t)}{\partial t} + \nabla \Phi(x,t) \cdot f(x) = 0 \tag{9}$$

Equations 9, along with the initial value equation 7, are the level set equations given by Osher and Sethian. The term $\nabla\Phi(x,t) \cdot f(x)$, in equation 9 is the Hamiltonian, $H(x, \nabla\phi)$.

Level set toolbox provides numerical schemes for solving the partial differential equation (PDE) given in equation 9. The advantage of this methods is that it can implicitly handles constraints such that continuous state set should always remain in the interface described by the level set function $\psi(x)$, by solving the PDE subject to constraints $\Phi(x,t) \leq \psi(x)$.

Unlike d/dt, the Level Set toolbox does not provide options to compute under (over) approximate results. However, the viscosity solution of the PDE would reach the exact boundary of reachable set as the grid resolution is increased. Because of the Eulerian framework, union (intersection) operations between two continuous sets can be easily implemented in the Level Set toolbox by taking the *max* (*min*) value of the corresponding level set function at each grid point.

Since the reachable set is represented as a higher dimensional function, topological changes in the curve of continuous state sets such as separation and merging are handled implicitly.

However, the disadvantage of level set methods is the time taken for computation. Moreover, even though the method can be applied to systems with higher dimension, the computation requirements restrict analysis of system with higher dimensions.

Summary

In this chapter we reviewed some of the computational methods implemented in different analysis tools which are used for hybrid automaton based models. There are vast differences in the approximation methods of these tools. However, they all have the common background which is provided by the theory of hybrid automaton.

Currently, due to such diversity in computation tools and methods, the use of algorithms for the purpose of analyzing models of hybrid systems has been seriously constrained. Moreover, some of the tools lack the support data structures such as multidimensional lists needed for algorithmically exploring the hybrid state space. For example, both d/dt and Level Set toolbox do not provide a framework for expanding the discrete structure of a hybrid model using a tree.

An approach is to use the common semantics of execution and analysis of hybrid automaton to unify these tools. This has been the front goal of the interchange formats such as HSIF [62]. However, that initiative had its own problem. There was a partial interchangeability between tools such as Checkmate [20] , Charon [5], and HyVisual [46]. However, overall research community was convinced that HSIF cannot support each and every tool. Currently, other research groups such as the Hyper group[1] are working toward development of a standard package which will allow the model of hybrid automaton to be shared between various computational tools. However, these initiatives are not related to interchangeability of the analysis algorithms implemented in different tools and are more focused on the interchangeability of the hybrid system specification.

We want to use a model based approach and create a platform to abstract the different implementation details and allow design and analysis of hybrid automaton models with only the mathematical definition of operations in mind. In nutshell our goal can be summarized as (i) separate the concern of algorithm design for analysis of hybrid automaton model from any specific computation implementation; (ii) allow specification of analysis algorithms using models in order to automate the analysis process by automatically generating implementation and hence computation results using model translators. (iii) separate the specification of algorithm from the concerned hybrid automaton model so that the algorithm can be reused for other hybrid automaton models as well; (iv) enrich the computation tools so that they can be used as computation kernels in order to provide implementation to any generic analysis algorithm; (v) be able to construct a library of analysis algorithms which can be used at various times to perform related analysis purpose.

[1] http://chess.eecs.berkeley.edu/hyper/

CHAPTER III

HYBRID SYSTEM ANALYSIS AND DESIGN LANGUAGE

In order to provide a platform for enabling effective design and analysis of models of hybrid systems, Model Integrated Computing (MIC) [40, 41] approach is ideal. MIC is based on models and automatic generation of useful artifacts. It introduces modeling languages tailored to a particular domain to allow representation of relevant information for the systems as models which can later be used to predict the system behavior. In MIC, metamodels are used for formally specifying these domain specific modeling languages (DSML). In MIC, several tools and technologies for model transformation such as GReAT [2] and Builder Object Network [30] allow automatic generation of useful information for a different or same semantic domain. Consider the possibility of a domain specific modeling language which relates the concepts such as reachable set operations, Boolean set operations at a level which is impartial to any specific implementation strategy. Such language would provide an ideal setup for modeling analysis algorithms without the implementation concerns. By using model transformation technology one can automatically adopt any particular implementation for the models designed in that language.

A tool suite that supports this approach and provides a reusable framework for constructing metamodels and deploying DSMLs is the Generic Modeling Environment (GME) [45, 30]. Using the metamodeling environment provided in GME, we created a modeling language Hybrid System Analysis and Design Language. Use of metamodel allows formal specification of all the concepts and notions needed to design hybrid system models and analyze them, such as reachable set operations and Boolean set operations (such as union and intersection).

ReachLab is a platform which implements this language and provides the necessary translators. This platform is designed in such a way that the concerns of design and implementation of analysis algorithms are separated. On one hand, the models of analysis algorithms are abstract and therefore the design of algorithms can be made independent of implementation details. On the other hand, translators are provided to automatically generate implementations from the models for computing analysis results based on computation kernels. Multiple computation kernels, which are based on specific computation tools such as d/dt and the Level Set toolbox have been enriched to support numerous algorithms designed in ReachLab, in order to enable hybrid state space exploration.

Domain Specific Modeling Language (DSML)

Domain specific modeling languages have been formalized as a five tuple of concrete syntax (C), abstract syntax (A), semantic domain (S), semantic mapping (M_S) and syntactic mapping (M_C) [22].

$$L =< C, A, S, M_S, M_C >$$

Concrete syntax (C) defines the notation (textual or graphical) used to express the models. Abstract syntax (A) specifies all the syntactical elements of the language. It also includes the so called static semantics, which determines the integrity constraints to ensure the correctness of *sentences* of the language. Semantic domains (S) is defined by formalism which provides meaning to a correct sentence in the language. The mapping $M_S : A \rightarrow S$ relates every element of abstract syntax to a specific meaning in the semantic domain. Model translators are used for this semantic mapping. The mapping $M_C : A \rightarrow C$ assigns a notational construct to every elements of abstract syntax. The specification of abstract syntax as a meta-model requires meta-language to express concepts, associations and integrity constraints.

It can be noted that actually the meta-language is itself a DSML and is specifically used for constructing other languages. A very popular meta language for specifying DSML is provided by the GME. Its meta-language is based on Unified Modeling Language (UML) [25] and has been enriched with specific stereotypes and relations to allow the specification of abstract syntax of a DSML. In GME, the integrity constraints are specified using the

Table 1: Syntactical elements of HADL

Aspect	**Model of**	**Syntactical Elements**
Data	Data	Primitive data types: integer, float, Boolean; Data structure: multi-dimensional list.
System	Hybrid automaton	Discrete mode, associated with invariant; Discrete transition, associated with guard and reset; Continuous set and initial continuous set; Analysis set, as a specialization of continuous set; Computation parameters.

Programming	Control flow	Routine, hierarchical in nature;
		Looping: "*while*" loop;
		Branching: "*if-then-else*";

	Operators	Primitive data operations: $+, -, *$; Logical operations: equal, less than, and, or, not; Multi-dimensional list operations: new, delete, append, element; Reachable set operations: discrete successor and predecessor, (constraint) continuous successor and predecessor in a single step (in bounded time), reset, projection, visualization; Boolean set operations: intersection, union, complement.

Object Constraint Language (OCL) [31]. For more detail on GME and its meta-language reader can refer to [45].

Description of HADL

The objective of HADL is to separate the concerns of designing and implementing analysis algorithm for hybrid systems. Currently, the language only supports hybrid automaton based models. Any analysis algorithm implemented for a system model will always have three different components: data, control flow i.e. programming logic and the system model. HADL has three different aspects called system aspect, data aspect and programming aspect in order to separately specify the three different components of analysis algorithm. By separating these components HADL allows the reuse of algorithm for different systems.

Table 1 lists all the different syntactical elements which forms the abstract syntax of HADL. These provide the elements which can be used for modeling a system as hybrid automaton. The operators and control flow can be used to model algorithms for analyzing those hybrid automaton models. Notice the clear separation of data, system and programming components. These are the three aspects of HADL. These aspects have a weak coupling between them. For example, there are operations which perform the reachability analysis on the models in the system aspect. Furthermore, data defined in data aspect is used in system and programming aspect. To allow this interaction, HADL uses references [1] [45] to provide weak coupling between the elements from different aspects.

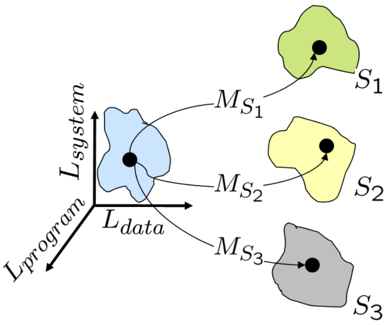


Figure 8: Each point in the space is the design of an algorithm along with the system model. M_{S_1}, M_{S_2}, M_{S_3} are semantic mappings to semantic domains S_1, S_2, S_3 of three different analysis tools. Movement along the $L_{program}$ axis signifies the collection of algorithms which can be used for the same system model. The direction of L_{system} represents the models which can be analyzed using the same algorithm.

The semantic domain of HADL is the mathematical definition of its constituents. It does not directly deal with the implementation details. However, its semantics allows the designer to take important design decisions by formally defining the expected result of the analysis algorithms. The actual computation of the analysis algorithm is provided by the computation tools which have their own semantics. By finding a correspondence between the elements of HADL and those in the computation kernel or providing the kernel with an implementation for HADL element , one can anchor the semantics of the kernel to that of HADL. This concept is similar to the concept of semantic anchoring introduced in [19].

Due to the separation of concerns, abstract syntax of HADL can be written as a three tuple $A =< L_{data}, L_{system}, L_{program} >$. Let S_i be the semantic domain of any computation

[1] References are parts that are similar in concept to pointers found in various programming languages.

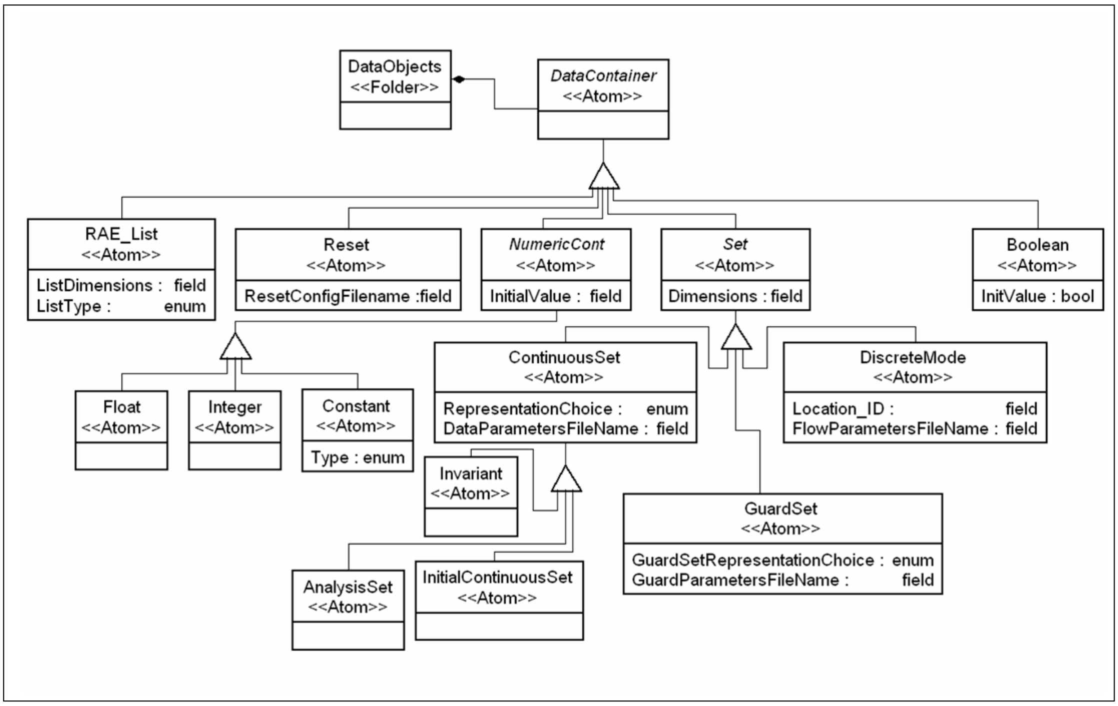


Figure 9: This figure shows the metamodel designed in MetaGME language for data aspect of HADL. Discrete mode, guard set, continuous set all are types of sets. This relation is represented via inheritance.

kernel to which the HADL semantics are anchored. Model translators can be used to provide the semantic mapping $M_{s_i} : L_{data} \times L_{system} \times L_{program} \rightarrow S_i$. This semantic mapping $M_{s_i} = M_{ss_i} \circ M_s$, where M_s is the mapping to the semantic domain of HADL and M_{ss_i} is the semantic mapping from semantics of HADL to that of the computation kernel. Hence, a translator is required for each computation kernel. Figure 8 shows the semantic mapping to different domains and illustrates the idea of reuse of an algorithm for different models.

In the next three sub sections we will describe the abstract syntax and semantics of the modeling components found in these three aspects.

Syntax and Semantics of Data Aspect

For any algorithm we need notions of simple data types such as integers, Boolean, and floats to enable the representation and manipulation of information. In analysis algorithms related to hybrid automaton we also need data structures which can represent tree-like structures. These tree-like structures are very useful in unfurling the hybrid state space in order to explore if the reachable set would ever reach certain bad state. Furthermore, there is a need to be able to specify the discrete modes, continuous sets, guard sets, invariants, analysis set and resets used to specify a hybrid automaton.

Figure 9 shows the metamodel of HADL's data aspect. Along with the basic primitive data type, it also provides multidimensional lists called *RAE_List*. These lists are used for constructing algorithms which might need to maintain a list of objects. This class diagram has been built using the inheritance and association concepts and it represents the different data types found in the analysis tools. For example, a set can be of different types. It can be *Discrete mode*, used to model any discrete state ($q \in Q$) of the hybrid automaton, or it can be *Continuous set*, used to represent regions of continuous state space. An *Analysis set* is used for representing the continuous state space ($X \subseteq \mathbb{R}^n$) of the hybrid automaton. *Guard set* is used to represent the guard conditions which are either specified as half spaces (inequalities) or regions of continuous state space. *Invariants* are used to represent the domain associated with a discrete mode. These constructs are very generic in nature and can have different implementations based on chosen computation kernel. *Reset* is used to represent the reset map associated with an edge of hybrid automaton. These types can be summarized as

- *Numeric Data*: Integer, Float, Boolean, and Constant
- *Sets*: Discrete mode, Continuous set, Guard set, Analysis Set, and Invariant
- *Reset*
- *Lists*: Multidimensional list of either numeric or set data types

Continuous sets, Guard, Analysis set and Invariant all have a filename attribute. These points to an external file which can describe the continuous state space by using inequalities or specifying the vertices of a polyhedra. These files are provided in order to accommodate different representation of continuous state space in different computation kernels. Similarly, a vector field can be associated with a discrete mode using an external file. Each discrete mode also has an attribute which provides a distinct location id for identification purposes.

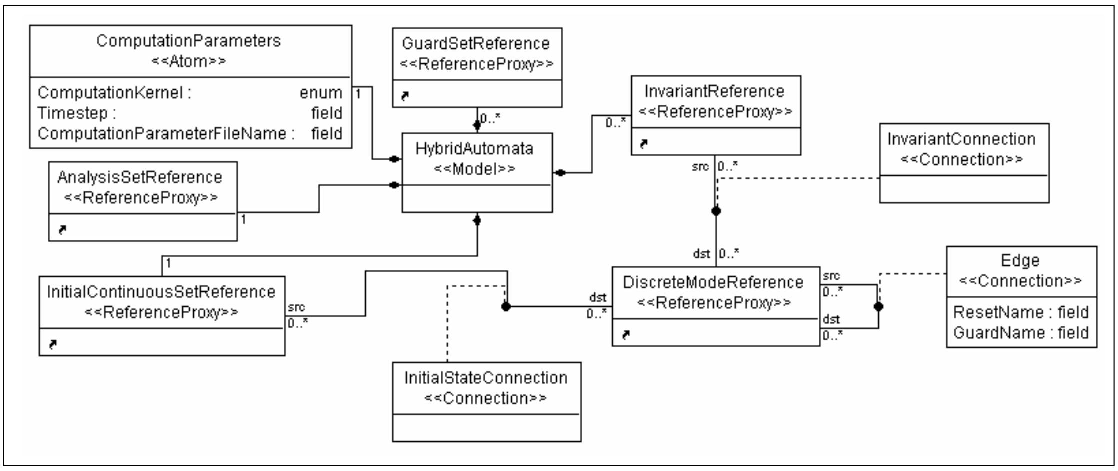


Figure 10: This figure shows the metamodel for system aspect of HADL.

Data variables provided in HADL and used in analysis algorithms are strong-typed i.e. there is no rule specified for automatic casting of data. Currently, only global scoping is supported. However, in the future, it will allow local scoping as well.

Syntax and Semantics of System Aspect

Figure 10 is the metamodel of system aspect. It is used to define a hybrid automaton model using the discrete mode and continuous sets defined in data aspect. Currently, only one hybrid automaton can be specified in the system aspect. However, due to the nature of design, it can be easily scaled, while maintaining backward compatibility, to allow a network of automata.

We defined the hybrid automaton in Chapter II. To specify the hybrid automaton we need the collection H=(Q, X, f, $Init$, D, E, G, R). All the necessary recipe for the hybrid automaton is defined in the data aspect by specifying the discrete modes Q, guards G, Initial continuous set $Init$, Resets R. They are referred using references in this aspect. The continuous space X is described by using the analysis set specified in the system aspect. This also sets the dimension of continuous state of the system. Edges between the discrete modes are represented by using the Edge connection. Invariants are associated with each discrete mode to specify domain D. Vector field f is specified as an ordinary differential equation written in an external text file whose name is provided as an attribute to the discrete mode. Computation parameters are also specified as external file in order to specify computation kernel related settings. These settings can be either the grid size used to form a mesh over the analysis space and the maximal time step Δt which is used by some d/dt in order to compute $Post_{cT}$.

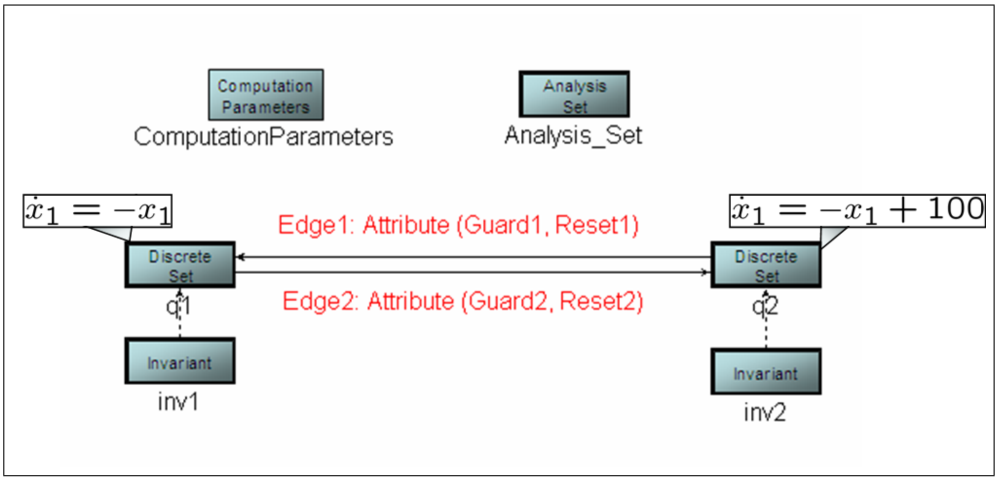


Figure 11: This figure shows an example model constructed in the system aspect. This hybrid automaton has two discrete states.

Figure 11 illustrates the construction of a hybrid automaton of a thermostat (see Figure 3). This is the same example which was presented in previous chapter. The continuous state space is modeled as the analysis set. Each discrete mode is associated with a vector field, which is specified as an external file. The parameter x_1 is the continuous state variable. Parameters of the invariants are also described using similar external files.

Syntax and Semantics of Programming Aspect

Programming aspect is used to specify models of algorithms for analyzing hybrid automaton modeled in system aspect. This aspect is composed of two types of entities: Operators which provide atomic entities used for manipulating data and performing set operations; Control flow components which are used to provide the algorithmic logic.

Control Flow Components

Figure 12 shows the metamodel of HADL's programming constructs. All control flow components are referred as control flow entities in HADL. This is shown by the abstract

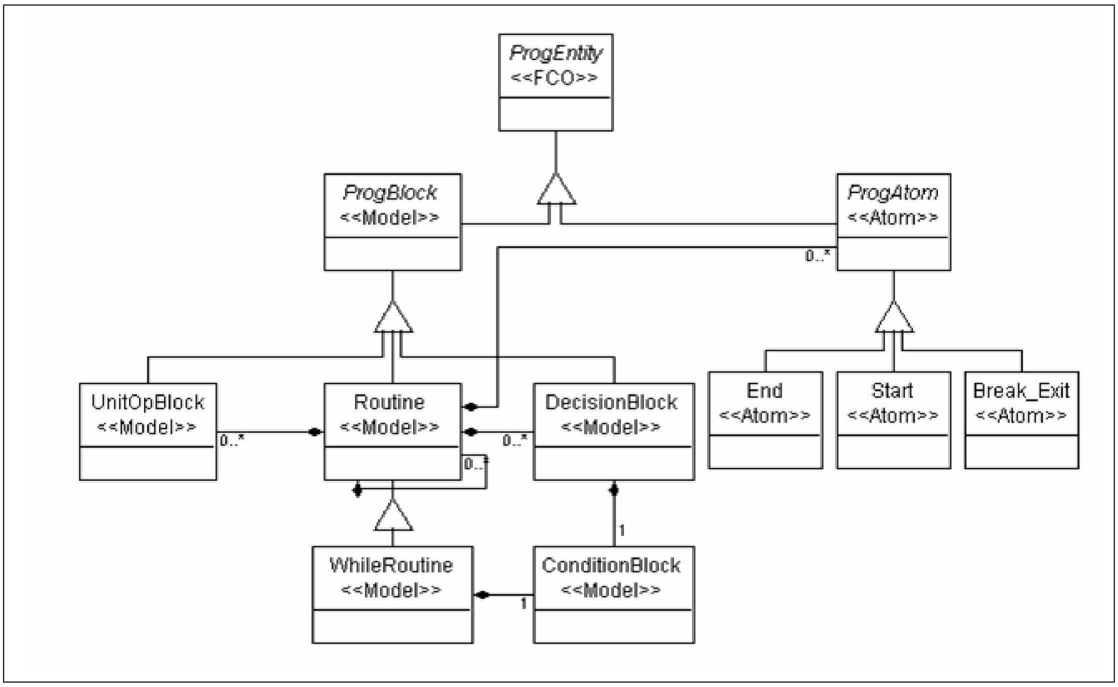


Figure 12: This figure shows the metamodel of HADL's programming constructs. These are very general algorithmic concepts and can be mapped to every target semantic domain. Routines are used to provide the hierarchical composition of algorithms.

base class. CFEntity (short form for control flow entity). This is done to preserve common properties such as connection association between each control flow element. As shown in Figure 13 CFEntity2CFEntityConn are connections between two CFEntity. The direction of the connection represents flow of control through the algorithm. In some aspects this idea is similar to that of a flow chart.

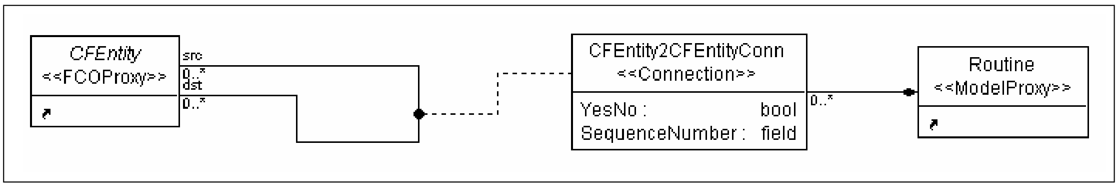


Figure 13: UML diagram of various types of connections possible between programming blocks. These connections are used to represent the control flow through the algorithm.

Since algorithms can be hierarchical in nature, HADL uses hierarchical *Routines* to encapsulate them. Each routine starts from a *Start* and end in a *end.* Statements inside a function are modeled by *Unit operations*, which can contain data references and operators. As the name suggests, unit operation blocks can perform only a single operation at a time.

The interaction between the data and operators is modeled by using a data flow approach. The operator and data connections shown in Figure 16 are used for this purpose. Loops in an algorithm are modeled as *WhileRoutine*, which are a specialization of *Routine* . To model the branching logic of a program, *DecisionBlock* are used. Both *DecisionBlock* and *WhileRoutine* contain a *ConditionBlock*. *ConditionBlocks* are used to model Boolean expressions. They can contain operations which result in a Boolean output. Execution of *WhileRoutine* is terminated only when the value of the Boolean expression inside its *ConditionBlock* is false. Its execution can also be interrupted by using a *BreakExit* block. These constructs can be used to model many sophisticated programming logics. Figure 14 illustrates a simple design of control flow of an algorithm.

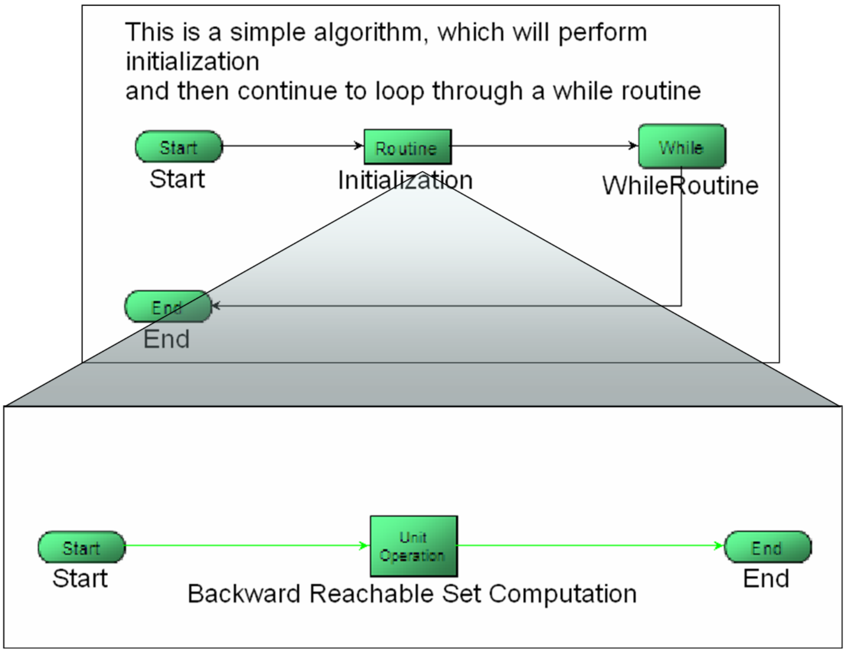

This is a simple algorithm, which will perform initialization
and then continue to loop through a while routine
Start
Start
Routine
Initialization
While
WhileRoutine
End
End
Start
Start
Unit Operation
Backward Reachable Set Computation
End
End

Figure 14: A simple algorithm designed using control flow components of HADL.

Operators

Figure 15 shows the class diagram of these operators. The operators are divided into four main sub-classes. These are *Numeric Operators*, *Boolean Operators*, *Set Operators*, and *Data Structure Operators*. Numeric operators are used to manipulate the numeric data containers. Boolean operations are used on Boolean data in logical expressions.

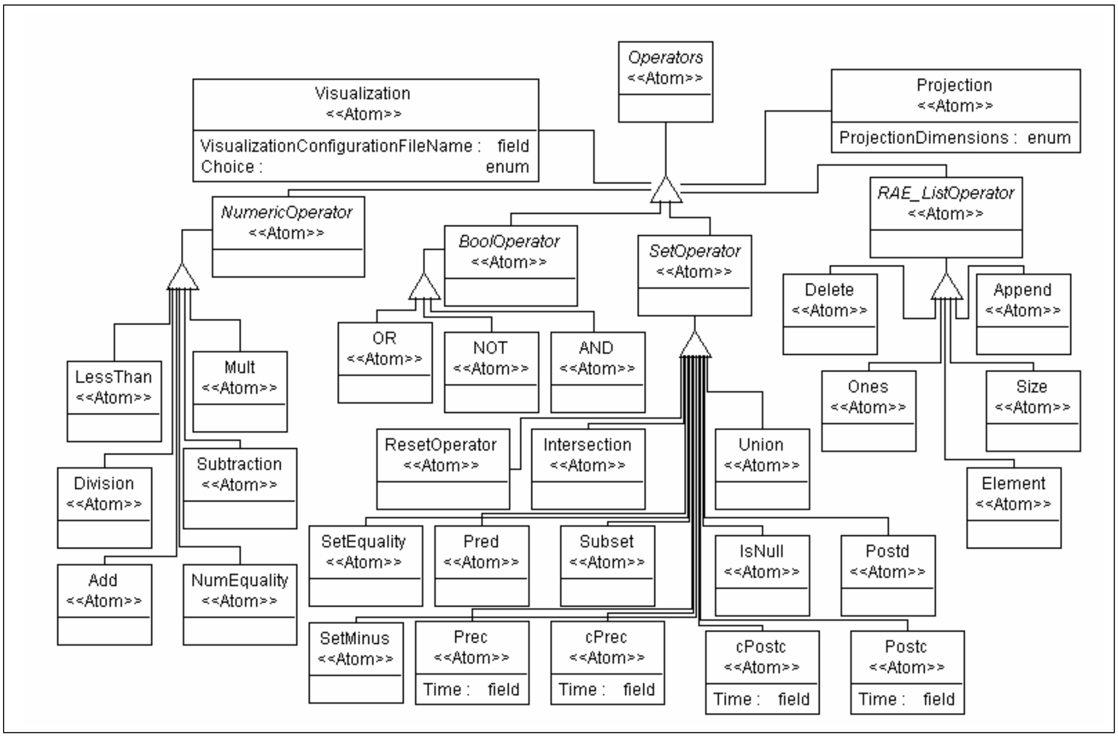


Figure 15: This figure shows the metamodel of operators provided in HADL for constructing algorithms to enable the symbolic methods based analysis for hybrid automaton based models of embedded software systems.

Furthermore, set membership predicates such as subset are provided as Boolean operations on state sets. The set manipulation operation such as *union* and *intersection* have the usual semantics. The *subset* operator returns true if a given set is subset of other set. *SetEquality* operator is used to check if two given sets either discrete or continuous are exactly equal. *SetMinus* operator is used to compute the complement of a set with respect to another set.

Previous chapter described bounded time continuous successors (predecessor) sets, $Post_{cT}$ (Pre_{cT}), bounded time constrained continuous successors (predecessor) sets, $cPost_{cT}$ ($cPre_{cT}$) and discrete successor (predecessor) sets, $Post_d$ (Pre_d). In HADL one can describe a maximal time step Δt for computing the reachable sets using a specific computation method, which could be used for performing exact or approximate computation. For some computation methods, if a required time step is larger than Δt, the solution quality at each step could deteriorate. For such methods the reachable set for time interval $[0, T]$ would use $N = T/\Delta T$ successive iterations in order to compute the complete reachable set.

Apart from the reachability operators, two important set operations are reset operations and projection operations. Reset operation is performed to compute the continuous state after a transition has been taken. Projection operation is usually used for viewing higher dimensional continuous sets by projecting on two a two or three dimensional space.

Definition 6 (Reset Operator) *Given a continuous set $P \subseteq \mathbb{X}$, a discrete transition $e \in E$; the associated reset function maps P onto P', where $P' = \{p' \in \mathbb{X} \mid \exists p \in P \; s.t. \; p' \in R(e, p)\}$.*

Definition 7 (Projection Operator) *Consider a set $P \subseteq \mathbb{R}^n$, where $n \in \mathbb{Z}^+$, an element $p = [x_1, \ldots, x_i, \ldots, x_n]^T \in P$, and an index array $I = [\lambda_1 \ldots \lambda_k]$, where $k \in \mathbb{Z}^+, k < n$, and $\lambda_i \neq \lambda_j$ if $i \neq j$. Also $\lambda_i \in \mathbb{Z}^+ \wedge \lambda_i < n$. Then projection of P using the index array I is given by $P' = \{y | y \in \mathbb{R}^k \wedge \exists p \in P \; s.t. \; y = proj(p, I)\}$ where $proj(p, I) = [x_{\lambda_1} \ldots x_{\lambda_k}]^T \in \mathbb{R}^k$*

The multidimensional list provided by HADL can be manipulated by using these operators. We will explain the semantics of these operators with the help of an example multidimensional list of $n \in \mathbb{Z}^+$ dimension. We will call this list as NDimList. This list operates in a fashion similar to multidimensional arrays of many programming languages. The only difference being that it does not have to be uniform across each dimension. For example it can be a list of two dimensions with 10 elements in the first row and 20 elements in the second row. To refer to any element inside this list we use a finite sequence of integers as index. Let use define an index sequence I to be a finite sequence of positive integers given as $I =< \lambda_j >_{j=1}^{j=k}$, such that $k \leq n$, and $\lambda_j \subseteq \mathbb{Z}^+$. Using this index we can define the data structure operators.

Definition 8 (Delete) *Given a list, NDimList of n dimensions and an index array $I =< \lambda_j >_{j=1}^{j=n}$ with the operation Delete(NDimList, I) deletes the λ_nth element of the list by referring to the first n-1 dimensions using the indices $\lambda_1 \ldots \lambda_n$. If no index is specified this operator deletes the contents of the whole list. This index referral scheme is the same as used to refer to elements in a matrix.*

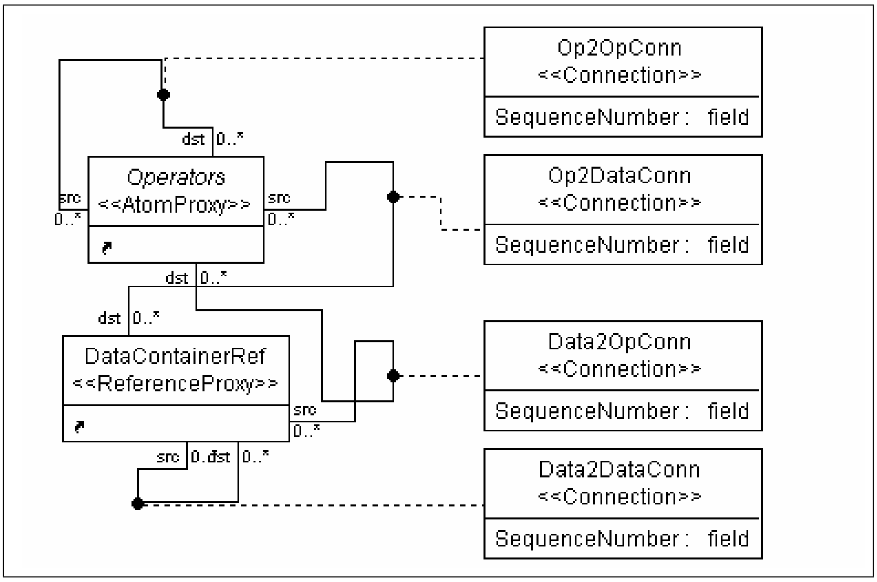


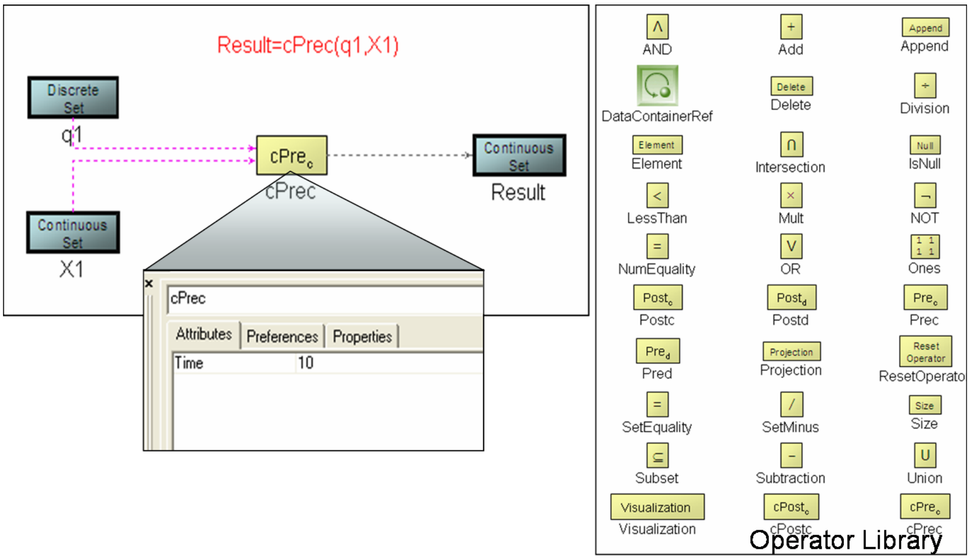
Figure 16: This figure shows various types of connections that are possible between operators and data containers. Each connection signifies an association between its source and destination. For example a connection going into an operator signifies the incoming input, while the connection going out of an operator signifies the processed information going out of the operator.

Definition 9 (Element) *Given a list, NDimList of n dimensions and an index array* $I =< \lambda_j >_{j=1}^{j=k}$*, the operation Element(NDimList, I) gives a list of* $n - k - 1$ *dimensions present at the* λ_k *position specified by the* $\lambda_1 \ldots \lambda_k$ *indices in the first k dimensions.*

Definition 10 (Size) *Given a list, NDimList of n dimensions and an index array given by* $I =< \lambda_j >_{j=1}^{j=k}$*, the operation Size(NDimList, I) gives the number of elements of list of* $n - k - 1$ *dimensions present at the* λ_k *position specified by the* $\lambda_1 \ldots \lambda_k$ *indices in the first k dimensions.*

Definition 11 (Append) *Given a list, NDimList of n dimensions and another list MDimList of m dimensions s.t.* $m < n$*, the append operation takes in an index array given by* $I =< \lambda_j >_{j=1}^{j=n-m-1}$ *and appends the MDimList to NDimList at the* λ_{n-m-1} *position specified by the* $\lambda_1 \ldots \lambda_k$ *indices in the first* $n - m - 1$ *dimensions.*

Figure 16 shows the various connections that can be used to associate operators. Each operator can either be connected to other operator or to a data element. This connection can either be outgoing or incoming (Op2OpConn, Op2DataConn, or Data2OpConn). Depending upon its direction it represents input or output information. If there is more



cPrec

Time	10

Figure 17: HADL's operators can be used to form directed acyclic graphs. This example is a model constructed for continuous backward reachable set computation from initial continuous state $X1$ and discrete mode $q1$. The time for which the backward reachable set should be computed is specified in the properties window.

than one input to an operator, sequence numbers are used to prioritize between different inputs to the operator. Direct assignment between data variables is represented by a direct connection between two data containers.

Figure 17 shows the use of operators in HADL. This figure illustrates bounded time constrained backward reachability set computation. The time range for the constrained predecessor operation is specified in the properties window. Operator library shows all the operators that can be used for making models in HADL.

Individually these three different aspects represent the three different concerns in analysis algorithm design. By choosing the elements from the language judiciously several different compositions of data, program and system can be designed. OCL constraints are used to ensure the correctness of this composition. Furthermore, because of the separation of concerns, the algorithms and system models can be reused and a generic library of algorithms can be created for future use.

Summary

In this chapter we described the metamodel of HADL, which allows the formal specification of the abstract syntax. We further described the semantics of these concepts though mathematics. Any analysis algorithm modeled in programming aspect in order to analyze hybrid automaton modeled in system aspect is implementation independent. However, to actually compute the analysis results we would have to use the semantic domain of a particular computation kernel.

The separation of analysis algorithm into the three aspects helps to reuse an algorithm for another system model. Therefore, useful algorithms can be modeled in HADL and used as a generic library.

In order to automatically generate implementation from the models designed in HADL, the models have to be traversed and transformed to refer to the corresponding entities in the computation kernels. This process is completed using model translators equipped in ReachLab.

Next chapter will describe the computation platform, ReachLab, which implements HADL and provides model translators. In the same chapter we will also review the algorithms used for transforming the models of algorithms in order to automatically generate implementation. We would also describe some of the enrichments carried out for d/dt and Level Set in order for them to be used as computation kernels in ReachLab.

CHAPTER IV

COMPUTATION PLATFORM: REACHLAB

In this chapter we will describe the computation platform which has been developed by utilizing the modeling language HADL. The architecture of ReachLab, as shown in Figure 18, is designed to separate the concerns of algorithm design from implementation details. The model integrated program synthesis (MIPS) environment facilitated by GME, provides support to build graphical algorithm and system models in ReachLab. Different graphical model entities and components are connected according to the rules specified by HADL meta-model. Therefore, models can be designed in ReachLab graphically according to HADL specification.

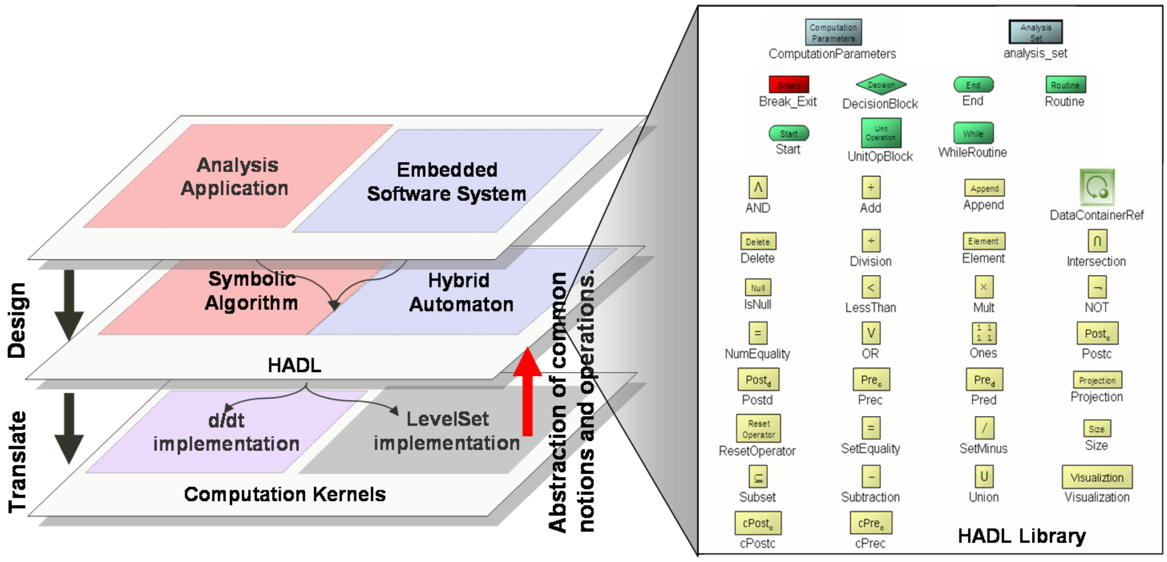


Figure 18: The bottom layer of ReachLab architecture is comprised of different computation kernels. The middle layer is the implementation of HADL containing the library shown toward right. The top layer contains the analysis application and the embedded software system. Design is a process of modeling the system and the analysis algorithm in ReachLab. Translation allows mapping from middle to bottom layer.

Besides model design, the other key process is the use of translators to automatically translate the models into executable artifacts. This translation process requires mapping of the abstract entities into concrete implementations for the target domain of a computation kernel. In [41], the translation process has been summarized as a graph transformation:

- *Creation of "input graph"* : The models with different interconnected components are implicitly represented by a graph structure.
- *Model traversal and Semantic mapping* : The translation process requires creation of a "target graph" (data structure for the executable artifact) from an "input graph". This requires the translator to visit various objects in the "input graph", recognize their patterns and calculate attributes of output objects in the "target graph" using semantic mapping.
- *Printing the product* : In this step, the translator serializes the "target graph" to generate executable artifacts pertaining to the related domain.

In ReachLab, the traversal process uses the data structures provided by GME to store the "input graph" along with necessary information. These data structures are very generic and remain the same for different translators. However, the data structures used to store the "target graph" vary due to implementation differences among different computation kernels. Once the process of traversal is completed semantic mapping ensures that the implementation is generated from the models.

In order to make the semantic mapping to the computation kernels possible we need to find a correspondence between the entities defined in HADL and those provided by the computation kernels. Currently, we have implemented this mapping to two computation kernels d/dt kernel and Level Set kernel. However, some of the features such as multidimensional lists and support for discrete predecessor and successor sets were absent from both d/dt and Level Set toolbox. Therefore, we had to enrich these tools and write our own support functions for these routines.

Computation Kernels

Both d/dt and Level Set toolbox implement basic forward and backward reachability operators using different approximations. They also implement basic set manipulation operations on state sets. These include HADL operators: intersection, union, subset, set minus, and set equality. However, what these tools lacked were supporting structures for defining and organizing the hybrid automaton as a graph. This graph structure is very important to support HADL operations such as discrete predecessor and successor set operations. They also lacked an organized way to implement multidimensional list data structures for handling more complex reachability based algorithms. With the inputs from the original authors of these tools we were able to augment these tools with these additional facilities to form our computation kernels. In general, other analysis tools which might have to be supported by ReachLab in future might require this additional work as well. In this section, we will try to describe some of these additional support structures in both d/dt and Level Set.

In d/dt, a discrete mode can be specified as an instance of a class[1]. To create a hybrid automaton we used a graph like structure with nodes and edges. Each edge was associated with properties such as the guard set and possible reset. We have called this graph structure "Hybrid table" and it allows the specification of relations between two different discrete modes as edges of the graph. The class definition for this structure is provided in Table 2. This structure also provided operations such as discrete predecessor and successor.

We will illustrate the use of this structure by considering a hybrid automaton with two discrete modes $q1$ and $q2$. We will also assume that there are two possible edges, one from $q1$ to $q2$, and the other in the reverse direction. Suppose two guards $Guard1$ and $Guard2$ are associated with the two edges and the resets are $Reset1$ and $Reset2$. Then to create the automaton using this structure we will have to take the following steps:

[1] A set, collection, group, or configuration containing members regarded as having certain attributes or traits in common

Table 2: Class definition for the Hybrid table in d/dt

```
struct nodeinfo
{
LOCATION discrete_state;
GRIDDY guard;
RESET reset;
};
//Hybrid Table class is based on the singleton pattern
class hybrid_table
{
typedef std::vector<nodeinfo> Pre_Post_List;
typedef std::vector<LOCATION> List_Of_Next_State;
typedef std::vector<GRIDDY> List_Of_Guards;
typedef std::vector<RESET> List_Of_Resets;
typedef std::map<int,Pre_Post_List> Adjacency_Map;
typedef std::map<int,LOCATION> DiscreteLocations;

public:
static hybrid_table* instance ();
˜hybrid_table ();
int getNumberOfDiscreteLocations ();
bool addRelation(LOCATION from , LOCATION to, GRIDDY guard,RESET reset);

List_Of_Next_State pre_d (LOCATION current);
List_Of_Next_State post_d (LOCATION current);
List_Of_Next_State pre_d (LOCATION current,List_Of_Guards& lg);
List_Of_Next_State post_d (LOCATION current,List_Of_Guards& lg);
List_Of_Next_State pre_d (LOCATION current,List_Of_Guards& lg,
List_Of_Resets& lr);
List_Of_Next_State post_d (LOCATION current,List_Of_Guards& lg,
List_Of_Resets& lr);

private:
hybrid_table ();
static hybrid_table* pinstance_;
bool initialized_;
DiscreteLocations location_map;
Adjacency_Map pre_map;
Adjacency_Map post_map;
};
```

1. Instantiate the discrete modes, and the two guards, and the two reset. Note that the instantiation of these objects is done using the definitions provided by d/dt. Resets in d/dt have to be affine transformation ($Y = AX + B$) , where $X \subset \mathbb{R}^n$ is the continuous set which is reset to the continuous set $Y \subset \mathbb{R}^n$, A is a matrix of size $n \times n$, B is a column matrix of size $n \times 1$. For resets we have defined a new class structure called Reset.

2. Instantiate the hybrid table and add the two edges to the table by using the function

   ```
   hybrid_table::instance->addRelation(q1,q2,Guard1,Reset1);
   hybrid_table::instance->addRelation(q2,q1,Guard2,Reset2);
   ```

3. With this hybrid table set in the memory we can use its *pred* and *postd* functions to find the discrete predecessor and successor sets. For example, upon calling the pre_d($q1$) we will get $q2$ as output. These functions have been overloaded to also return the list of corresponding guards and resets.

For the Level Set toolbox, which has been implemented in Matlab, we had to make a similar structure. However, we will not describe that structure here because it is similar to the structure made for d/dt.

In order to support the multidimensional list in d/dt we used the standard template library vector container class provided in C++. A vector is a sequence that supports random access to elements, constant time insertion and removal of elements at the end, and linear time insertion and removal of elements at the beginning or in the middle. The number of elements in a vector may vary dynamically; memory management is automatic. By dynamically creating a vector of vectors and so on we can create the notion of a multi-dimension list. For Level Set toolbox we created a linked list using pointers[2].

[2] Though pointers are not implicit to Matlab, they are available as additional libraries from their website.

Model Traversal

As described in the introduction to this chapter, the translation process involves model traversal, understanding the patterns and relations of the entities in the models and then applying the semantic mapping to these patterns. For this purpose we use the data structures provided by GME to capture the model as an input graph. These data structures are very generic and do not change between the tools. Upon application of semantic mapping, these input graphs get converted to target graphs which have different data structures depending upon the chosen computation kernel. We will describe this whole process in two steps of model traversal and semantic mapping.

Translators need to perform the traversal of all three aspects in order to understand the patterns and collect all useful information. This traversal process is based on graph search techniques such as depth first search [24]. The complete process can be broken down into four sub-tasks reviewed below.

Traversal of Data Aspect: All the data are defined in one single data folder as a list. Translator traverses this list in a linear fashion to collect all useful information about the data elements.

Traversal of System Aspect: The hybrid automaton model specified in the system aspect can be understood as a graph, in which the discrete modes are vertices and the discrete transitions of hybrid automaton are the edges. The translators traverse this graph by using depth first search starting from the initial discrete state to collect all useful information.

Traversal of Control Flow of Algorithms: The traversal of programming aspect is more complex. Every algorithm has a root *routine* which is the entry point to the algorithm. *Routines* can be hierarchical and may contain other sub-routines as shown in Figure 19.

The control flow inside each *routine* routes from a *“start”* to an *“end”* . However, there might be other exit routes from a *routine* through *“break-exit”*, which is used in the same way as the break in many programming languages. For example, the constraint

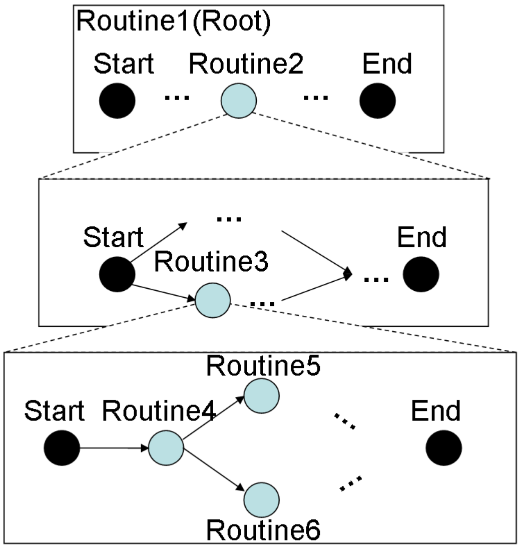


Figure 19: Routines in HADL are hierarchical. Each routine contains a start and an end node. The components of a routine can be visualized as a directed acyclic graph. Overall, the complete hierarchy leads to a hierarchical graph;

continuous successor set operation in bounded time T, denoted as $cPost_{cT}$, can be implemented by iterating $T/\Delta t$ times by using the bounded time constrained post operator with time range set to $[0, \Delta t]$, $cPost_{c\Delta t}$. As explained in an earlier chapter this is sometimes necessary because in some implementation methods the approximation quality of reachable set would deteriorate if a larger time step is used. Therefore, the routine to implement $cPost_{cT}$ can use the routine of $cPost_{c\Delta t}$ as its sub-routine. The language also provides a specialization of *routine* called *while routine* for implementation of looping constructs such as *do-while* which is traversed in the same manner as a *routine.*

The control flow inside a routine is sequential, however it can have multiple branches due to *decision blocks*. Cycles in the control flow are disallowed to demote the use of sudden jumps such as “goto”. Therefore, the control flow inside each routine is a directed acyclic graph (DAG) [24] with its directed edges depicting the route of control flow and each node depicting a block of algorithm. Since routines can contain other routines, the overall control flow of the complete algorithm is a hierarchical DAG.

The translators traverse the graph structure of algorithms in a depth-first search manner to extract information. In each routine, the traversal starts from *“start”* block

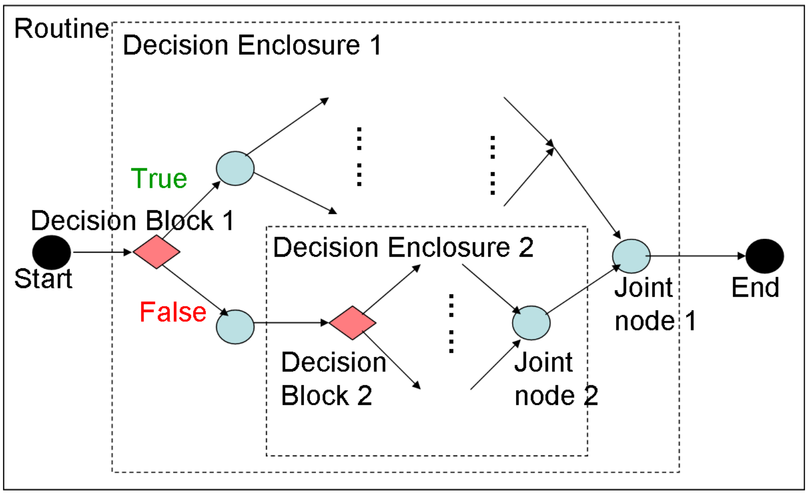


Figure 20: The decision enclosure is sub-graph starting from a decision block and ending at its corresponding joint node. A decision enclosure branches at the decision block and merges back at the joint-node.

and follows the directed edges. If any of the traversed entity is hierarchical, translators will traverse its subcomponents in a depth-first manner. *Decision blocks* are used inside routines to design a logical branching in the control flow sequence. For each of these blocks, the branching starts from itself, and finally merges at a *joint-node*. The sub-graph enclosed by the *decision block* and the joint-node in the DAG is called a *decision-enclosure*. This is illustrated by Figure 20. The traversal algorithm has to recognize the *"if true"* and *"if false"* part of each *decision block* so that they can be mapped to the corresponding decision logic in the implementation. This requires knowledge of its *decision-enclosure*. Table 3 gives an algorithm based on breadth first search technique for determining *decision-enclosure* of each *decision block*. This algorithm has a complexity of $O(n^2)$, where n is the number of blocks in the DAG.

The key of this algorithm is to find the joint-node, and since a joint-node is where all branches from the *decision block* merge, by using breath-first search and keeping all branching paths from the *decision block*, the first block that belongs to every recorded branching path is the joint-node.

Traversal of Operators: Operators are used for data manipulation. as shown in Figure 21, Every assignment expression forms a tree structure, with the left-hand-side data variable as the root of the tree. All data variables on the right-hand-size of the

Table 3: Decision-Enclosure Algorithm

```
Input:
    $DecisionBlock$ = the starting node of the enclosure
Initialization:
    $InitPath := DecisionBlock$
    $Paths := \{InitPath\}$
Start:
While $true$ do
    For each $path$ in $Paths$ do
        $Fringe$ := the tail of $path$
        $Succ$ := successor nodes of $Fringe$
        If $Succ \neq \phi$ then
            Add $Succ[0]$ to the fringe of $path$
            $Succ := Succ - Succ[0]$
            For each $s$ in $Succ$ do
                $path' := path$
                Add $s$ to the fringe of $path'$
                Add $path'$ to $Paths$
            End For
            If $\exists s \in Succ\ s.t.\ \forall p \in Paths, s \in p$ then
                Return $s$ as the joint-node
            End If
        End If
    End For
End While
```

expression correspond to the leaves of this tree, and operators on the right-hand-side correspond to the internal nodes of the tree. The expression can be formed by post-order traversal [24].

The operators have different semantic meanings depending on the input data types. And since HADL is "strong-typed", the data types of the tree leaves, which are predefined,

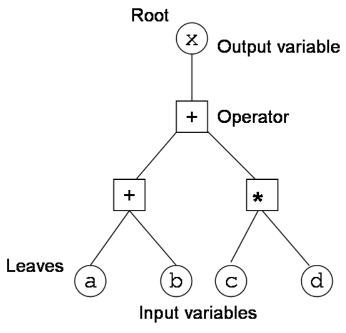
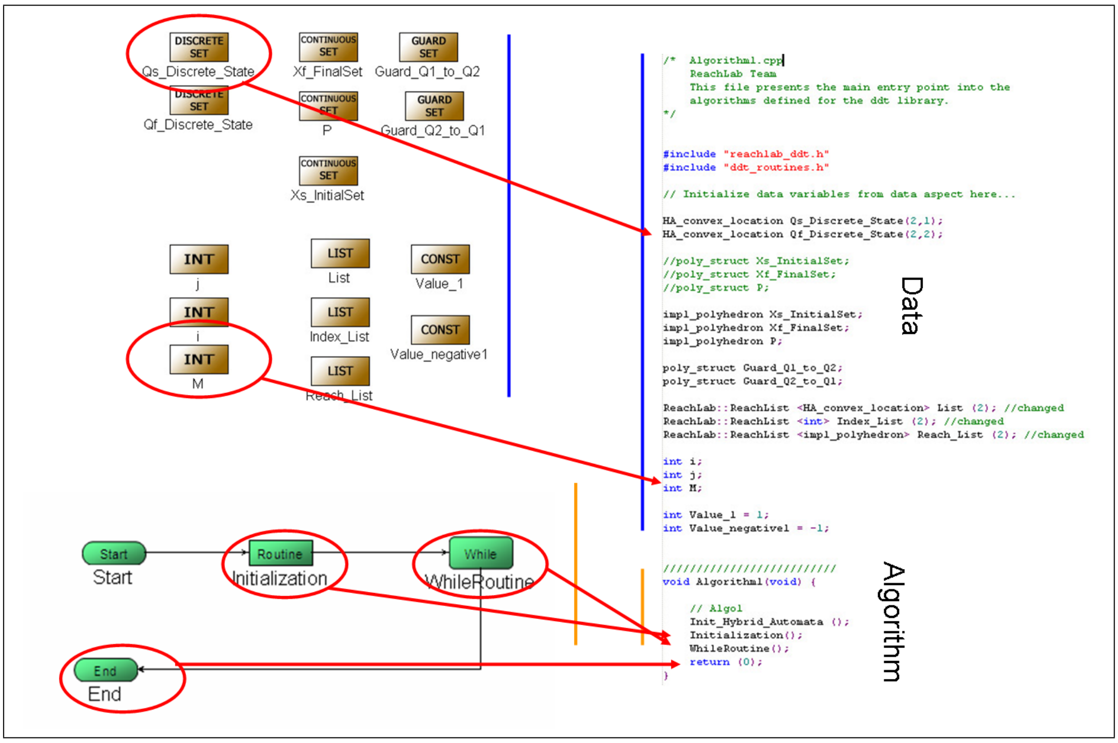


Figure 21: Example of an operation tree. This operation tree represents $x = (a+b)+(c*d)$

will finally determine the input data type of the operator connected to the root data. Therefore, it is important to propagate the data type information from leaves to the root in a post-order manner [24]. For example, for the tree shown in Figure 21 the traversal sequence would be $a, b, +, c, d, *, +, x$. Then we can map this operation tree to its semantic which would be $x = (a + b) + (c * d)$.



```
/*  Algorithm1.cpp
    ReachLab Team
    This file presents the main entry point into the
    algorithms defined for the ddt library.
*/


#include "reachlab_ddt.h"
#include "ddt_routines.h"

// Initialize data variables from data aspect here...

HA_convex_location Qs_Discrete_State(2,1);
HA_convex_location Qf_Discrete_State(2,2);

//poly_struct Xs_InitialSet;
//poly_struct Xf_FinalSet;
//poly_struct P;

impl_polyhedron Xs_InitialSet;
impl_polyhedron Xf_FinalSet;
impl_polyhedron P;

poly_struct Guard_Q1_to_Q2;
poly_struct Guard_Q2_to_Q1;

ReachLab::ReachList <HA_convex_location> List (2); //changed
ReachLab::ReachList <int> Index_List (2); //changed
ReachLab::ReachList <impl_polyhedron> Reach_List (2); //changed

int i;
int j;
int M;

int Value_1 = 1;
int Value_negative1 = -1;


//////////////////////////
void Algorithm1(void) {

    // Algo1
    Init_Hybrid_Automata ();
    Initialization();
    WhileRoutine();
    return (0);
}
```

Figure 22: Example showing mapping from a simple algorithm to the C++ code for d/dt. Note that the data items are mapped to a corresponding data declaration list in the codes. Each routine is mapped to a C++ function and is used as a function call.

Semantic Mapping

Since the semantics of a computation kernel are anchored to the semantics of HADL, we can find a corresponding implementation for HADL constructs in the computation kernel. These constructs include sequential programming features, Boolean operations on state sets, as well as the reachable set operations. However, in some cases, the operations, such as data structure manipulation operations, are not directly supported by the computation kernel and have to be specifically added to the computation kernel as new functions. The process of associating the HADL constructs to its implementation in computation kernel is akin to providing a meaning to them and is therefore referred to as semantic mapping. For example, if we want to generate implementation for d/dt the control flow inside a routine would be mapped to the sequential flow of logical commands inside a C++ program. This is illustrate in the Figure 22 We would use *"if-else-end"* to implement decision blocks and *"while-end"* to implement the looping constructs. The Boolean operations on state sets and reachability operations can be mapped to their corresponding implementation.

We will illustrate some of the aspects of the semantic mapping process by using the example of Level Set kernel. Level Set kernel has been implemented as Matlab functions. It supports all the basic data types in HADL except the multi-dimensional list structure, which we have to specifically implement along with the relevant operations in Matlab. The hybrid system specific data types such as *discrete mode* and *continuous set* are mapped to Matlab *struct* and *mesh* on analysis space, respectively. This *mesh* is an internal structure used by Level Set kernel. The control flow inside a routine is mapped to the sequential flow of logical commands inside a function. We use *"if-else-end"* statement in Matlab to implement branching and *"while-end"* statement in Matlab to implement looping. Boolean operations on state sets and reachable set operations are mapped to their corresponding implementation in Level Set kernel. However, for some of the operations defined in HADL, there are no straight-forward mappings, therefore we have to write specialized functions for them by using operations provided by the kernel.

One of the most important aspects of semantic mapping is to find a correspondence for reachability operators in both d/dt and Level Set toolbox. For the discrete reachability operators we use the hybrid table described earlier in the chapter. By using this table, one can create a hybrid automaton in both these tools and then use the discrete reachability operators. However, the continuous reachability operators provided in these computation kernels had to be repackaged into separate functions to provide a suitable

Table 4: Continuous Successor Set in Bounded Time (*Postc*) Algorithm for d/dt

```
Input:
    $q$, $P$, $T$, $\Delta t$
Initialize:
    $N = T/\Delta t, count = 0, Temp = P$
Start:
    While $count < N$ do
        If $Post_{\Delta t}(q, Temp) \subseteq Temp$ Then
            Return $Temp$
        End If
        $Temp = Post_{\Delta t}(q, Temp)$
        $count + +$
    End While
    Return $Temp$
End
```

Table 5: Continuous Successor Set in Bounded Time (*Postc*) Algorithm for Level Set toolbox

```
Input:
    $q$, $P$, $T$, $\Delta t$
Initialize:
    $N = T/\Delta t, count = 0, Result = P, Temp = P$
Start:
    While $count < N$ do
        $Temp = Postc_{\Delta t}(q, Temp)$
        If $Temp = Result$ Then
            Return $Result$
        End If
        $Result = Temp$
        $count + +$
    End While
    Return $Result$
End
```

Table 6: Constrained Continuous Successor Set in Bounded Time (*cPostc*) Algorithm for d/dt

```
Input:
    $q$, $P$, $T$, $\Delta t$
    Constraint set $X_\psi$; Domain of discrete mode $q$:$D(q)$
Initialize:
    $N = T/\Delta t, count = 0, Temp = P$
Start:
    While $count < N$ do
        If $Post_{\Delta t}(q, Temp) \subseteq Temp$ Then
            Return $Temp$
        End If
        If $\neg(Post_{\Delta t}(q, Temp) \subseteq D(q) \cap X_\psi)$ Then
            Return $Temp$
        End If
        $Temp = Post_{\Delta t}(q, Temp)$
        $count++$
    End While
    Return $Temp$
End
```

```
Input:
    $q$, $P$, $T$, $\Delta t$
    Constraint set $X_\psi$;Domain of discrete mode $q$:$D(q)$
Initialize:
    $N = T/\Delta t, count = 0, Result = P, Temp = P$
    While $count < N$ do
        $Temp = Postc_{\Delta t}(q, Temp)$
        $Temp = max(Temp, X_\psi \cap D(q))$
        If $Temp = Result$ Then
            Return $Result$
        End If
        $Result = Temp$
        $count++$
    End While
    Return $Result$
End
```

Table 7: Constrained Continuous Successor Set in Bounded Time (*cPostc*) Algorithm for Level Set toolbox

semantic mapping. We will try to illustrate this idea by using the continuous forward reachability operators.

Both Level set and d/dt provide basic implementation of Continuous Successor Sets for a maximal time range $[0, \Delta t]$. If this range is too large it deteriorates the approximation for reachable set. Therefore in order to approximate continuous successor sets for a larger time range $[0, T]$ we have to use the algorithms described in tables 4 - 7. Note that all these operators require the information about the discrete mode q, continuous set P, bounded time T, and the single time step Δt.

Summary

In this chapter we presented a computation platform called ReachLab for enabling automatic analysis of embedded software systems modeled as hybrid automata. It implements the meta-model based language HADL whose abstract entities allow users to model their algorithms and the system in an implementation independent manner. These models are then translated to implementations for different computation kernels. Currently, ReachLab supports d/dt and Level Set toolbox as its computation kernels.

We can summarize the advantages of using ReachLab to design analysis algorithms instead of ad hoc implementation based on the chosen computation kernel as follows:

1. Since ReachLab implements HADL whose semantics are based on the mathematics of hybrid system theory, it can be used to implement algorithms without demanding the intimate knowledge of approximations and intricacies of different computation methods.

2. Because of use of translators, a single ReachLab model can be very rapidly transformed into multiple implementations for different computation kernels.

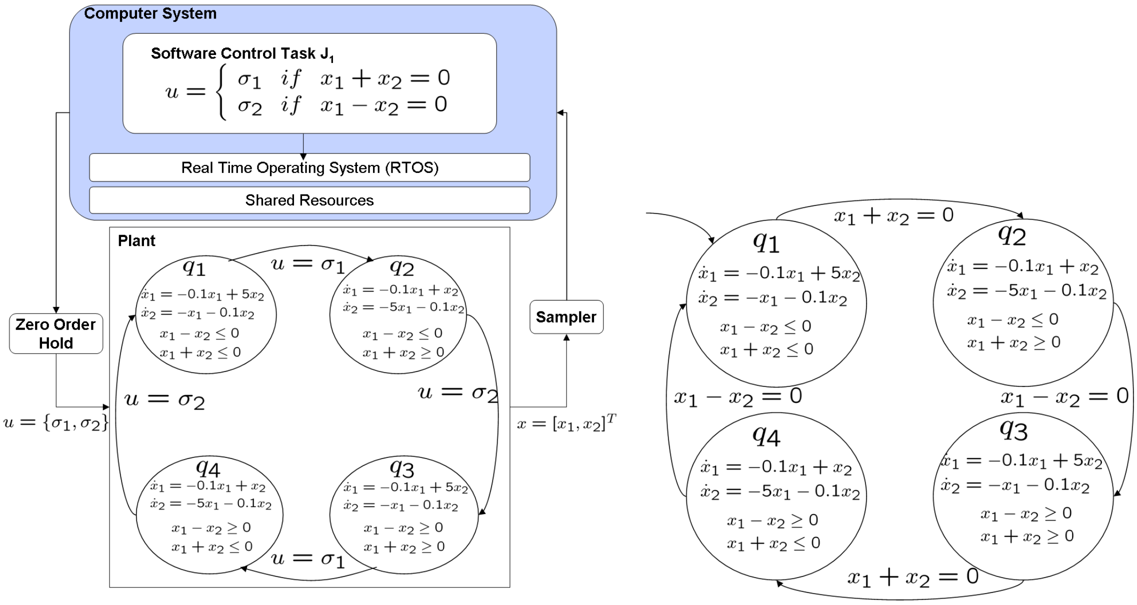
3. Because the semantics of this computation kernels are anchored to the semantics of HADL one can be compare the results obtained from multiple implementations of the same algorithm. This is important because in some situations the approximations of one computation kernel can be better than those of the others.

However, there are certain difficulties in use of ReachLab with other computation tools since they are usually not available in the form of a library which can be updated to support all the entities of HADL. In cases of such computation tools only a subset of HADL's entities can be used in modeling.

CHAPTER V

WORKING WITH REACHLAB

In this chapter, we will illustrate the design and implementation process for analysis algorithms in ReachLab by designing a forward reachability analysis algorithm for the embedded software system shown in Figure 23(a). We will then showcase some other results obtained by designing algorithms in order to synthesize initial states for continuous-time dynamical systems[1] so that some temporal properties, such as safety and liveness properties, are satisfied. The initial sets produced by the algorithms are related to some classical concepts for continuous dynamical systems, such as domains of attraction.


Computer System
Software Control Task J_1
$u = \begin{cases} \sigma_1 & if \quad x_1 + x_2 = 0 \\ \sigma_2 & if \quad x_1 - x_2 = 0 \end{cases}$
Real Time Operating System (RTOS)
Shared Resources
Plant
Zero Order Hold
Sampler
$u = \{\sigma_1, \sigma_2\}$
$x = [x_1, x_2]^T$
q_1
$\dot{x}_1 = -0.1x_1 + 5x_2$
$\dot{x}_2 = -x_1 - 0.1x_2$
$x_1 - x_2 \leq 0$
$x_1 + x_2 \leq 0$
q_2
$\dot{x}_1 = -0.1x_1 + x_2$
$\dot{x}_2 = -5x_1 - 0.1x_2$
$x_1 - x_2 \leq 0$
$x_1 + x_2 \geq 0$
q_3
$\dot{x}_1 = -0.1x_1 + 5x_2$
$\dot{x}_2 = -x_1 - 0.1x_2$
$x_1 - x_2 \geq 0$
$x_1 + x_2 \geq 0$
q_4
$\dot{x}_1 = -0.1x_1 + x_2$
$\dot{x}_2 = -5x_1 - 0.1x_2$
$x_1 - x_2 \geq 0$
$x_1 + x_2 \leq 0$
$u = \sigma_1$
$u = \sigma_2$
$u = \sigma_2$
$u = \sigma_1$
$x_1 + x_2 = 0$
$x_1 - x_2 = 0$
$x_1 - x_2 = 0$
$x_1 + x_2 = 0$

(a) (b)

Figure 23: (a) This figures shows an example of a simple embedded software system. The plant on the bottom has four running modes with different continuous dynamics, which are controlled by the software control task J_1. Plant and the software control task interact through the sampler and zero-order hold; (b) By considering the direct interaction between the control task and the plant, we can model the system as a hybrid automaton.

[1] Continuous-time dynamical systems can be modeled as a hybrid automaton with one discrete state and no edges

Figure 23(a) shows a plant with one continuous state $x = [x_1, x_2]^T \in \mathbb{R}^2$, which is controlled by a software control task. Depending upon the current state of the plant, it determines input $u \in \{\sigma_1, \sigma_2\}$. By considering the direct interaction between the control task and the plant, we can model the system as a hybrid automaton as shown in Figure 23(b). Moreover, multiple tasks which share common resource with the control task, the scheduler and the interface elements such as sampler and the zero order hold can be modeled by a more complex hybrid automaton. The corresponding hybrid automaton model can be implemented in the system aspect of ReachLab as shown in Figure 24

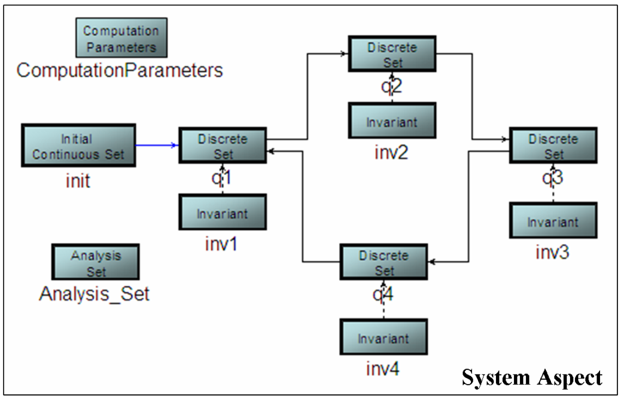


Figure 24: Hybrid automaton model shown in Figure 23(b) in the system aspect of ReachLab.

It has been shown in [55] that this system is stable in the sense of Lyapunov[52]. Starting from anywhere in the continuous state space, the continuous state of the automaton moves toward the origin in a flower-like trajectory. For this system, we are interested in computing forward reachable set using symbolic methods based algorithms, in order to verify that starting from certain initial state, whether or not the system can eventually enter some desired set.

Table 8 gives the specification of a generic forward reachability algorithm for hybrid automaton. It uses the concepts of both discrete and continuous successor set and finds the reachable set starting from a given initial set. This algorithm unfolds the hybrid automaton into a tree like structure and explores it by using breadth first search. Termination of this algorithm is guaranteed because of the limit M on the depth of this

Table 8: Algorithm for computing forward reachable set

```
Input:
    $H_A$, $Q_S, X_S$, $Q_F, X_F$, $X_B$, where
    $Q_S$ is list of initial discrete modes, $X_S$ is list of initial continuous sets, $Q_F$ is
    list of final discrete modes, $X_F$ is list of final continuous sets, and $X_B$ is bad set.
Constant:
    $T$ as time limit for $cPost_{cT}$, $M$ as search depth limit
Initialization:
    $Reach = X_S, List = \{\}, Successors = \{\}, R = \phi, Queue = Q_S$
    $Depth = 1, i = 0, j = 0$
Start:
    While ¬Empty($Queue$) do
        $List$ = PopAll $Queue$
        For $i = 1$ : Size($List$) do
            $R = cPost_{cT}(List(i), Reach(i))$
            $Successors = Postd(List(i))$
            For $j = 1$ : (Size($Successors$) do
                If $R \cap Guard_{List(i),Successors(j)} \neq \emptyset$
                    Then Push $Successors(j) \rightarrow Queue$
                    Append $R \cap Guard_{List(i),Successors(j)} \rightarrow Reach$
                End If
            End For
        End For
        $Depth = Depth + 1$
        If $Depth > M$ Then
            Stop
        End If
        Pop first Size($List$) elements of $Reach$
    End While
```

tree. The data structure *Reach* is used to store the reachable set. It can be noted that this specification does not delve into the actual implementation method of reachable set operations. However, the process of semantic mapping will relate those operations to a specific implementation method based on the concerned computation kernel. This algorithm can be used to verify if the system would ever execute into some desired state. In order to perform verification, the algorithm systematically explores the hybrid state space and check if the forward reachable set overlaps with the desired set. The main concern with these types of algorithms is the termination of computation. But if we perform the computation in an *Eulerian* framework (one in which the underlying coordinate system is fixed) within a bounded continuous state space, the algorithms will terminate due to the fact that the partition of state space has finite number of representative elements.

Design Steps

To analyze the safety property of the hybrid automaton model in Figure 23(b) by using the forward reachability algorithm, we need to design its hybrid automaton model in the system aspect and design the algorithm in the programming aspect. The data used in both of the system models and the algorithm models are defined in the data aspect. The entire process can be summarized into three steps:

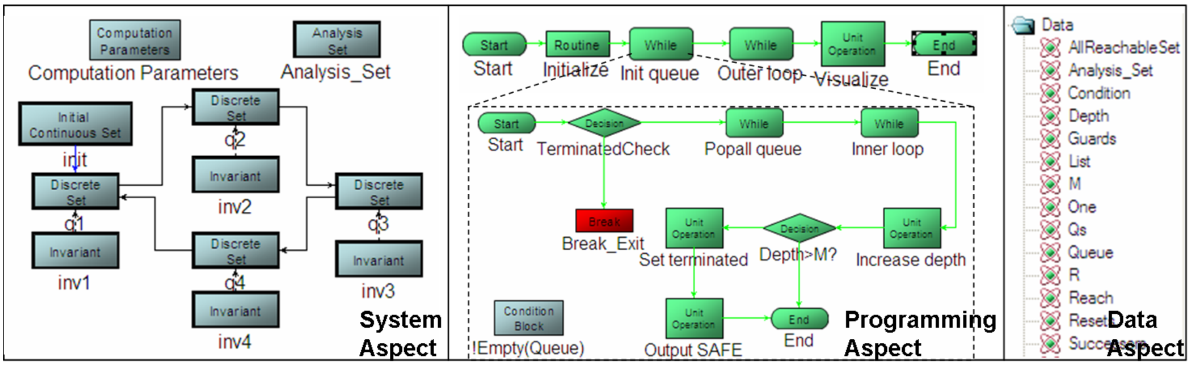


Figure 25: Hybrid automaton model for the corresponding plant in the system aspect, forward reachability analysis algorithm model in the programming aspect, and data used in the data aspect of ReachLab.

1. **Obtaining system model and algorithm specification**:

   Figure 23(b) and Table 8 provide the hybrid automaton and analysis algorithm specifications for this example.

2. **Design phase**:

   - Design of the system model: A hybrid automaton is drawn in the system aspect with discrete transitions connecting discrete modes, as in Figure 24.

- Design of the analysis algorithm: The analysis algorithm, which is hierarchical in nature, is modeled in the programming aspect by using ReachLab library elements. Figure 25 also gives part of the algorithm model for the algorithm given in Table 8, and the data required by both the hybrid automaton and the algorithm model.

- Specification of computation parameters: Input parameters to the algorithm and the computation kernels have to be specified before translation, such as the bounded time (T) for $cPost_{cT}$ operator, the analysis region, and how the analysis region is partitioned into grids.

3. **Implementation phase**:

   Translators are used to convert the designed models into implementation for a certain computation kernel. For this example, we translate the system and algorithm model into the d/dt implementation. Figure 26 shows the computation result. This result can be used to examine system behaviors, such as approaching the origin while evolving. It can also be used to verify system stability properties by testing intersection between the reachable set and the desired set.

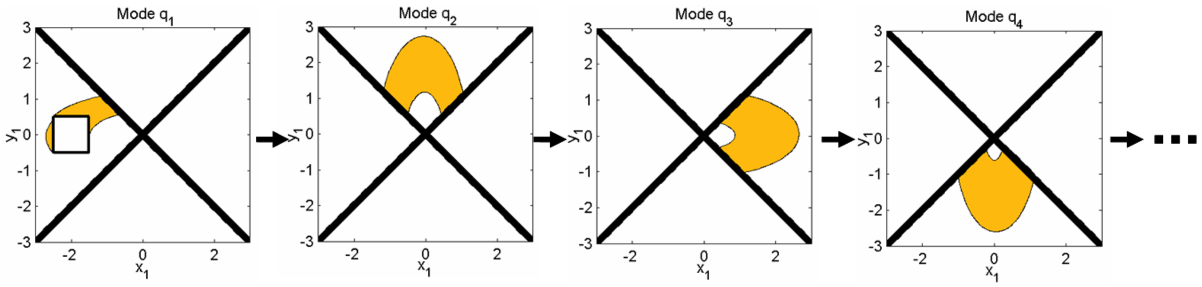


Figure 26: This figure shows the reachable set computed by using d/dt kernel. The white box is the initial set, [-2.5,-1.5]x[-0.5, 0.5]. Each subfigure denotes the reachable set in the corresponding discrete mode. Eventually, the reachable set will reach the origin. The analysis region is $[-3, 3] \times [-3, 3]$, the size of a representative elements in each dimension is 0.001, and the bounded time T is 5 seconds. Time taken for execution: 180 minutes on Pentium IV 2.59 GHz machine with 2 GB RAM.

Example: Analyzing Continuous Dynamical Systems

Using a similar design and implementation strategy we have designed synthesis algorithms synthesizing initial states for continuous dynamical systems so that some temporal properties, such as safety and liveness properties, are satisfied.

We will consider a continuous dynamical system, $\Sigma_c = (X, X_s, f)$, the state space $X \subseteq \mathbb{R}^n$, the state vector $x \in X$, the initial set $X_s \subseteq X$ and the vector field $f : X \to \mathbb{R}^n$. Such a system can be modeled as a hybrid automaton $H = (Q, X, Init, f, D, E, G, R)$ with $Q = \{q_i\}$, a single discrete mode. $Init = (q_i, X_s)$. D is same as the analysis space X. E, G, R are null sets.

Consider two kinds of temporal properties defined over the trajectory space of Σ_c: $\Box F$ and it's dual $\Diamond F$ with $F \subseteq X$ as defined in [69]. Given a set $F \subseteq X$ and a trajectory generated by Σ_c, $\Box F$ and $\Diamond F$ give $True$ or $False$ whether the trajectory *always* stays inside F or *eventually* reaches F, respectively. Sometimes, $\Box F$ is referred to *safety* property while $\Diamond F$ is referred to *liveness* property. They are formally defined as:

$$\Box F = \begin{cases} \text{True} & \text{if } \forall t \geq 0 \;\; x(t) \in F \\ \text{False} & \text{otherwise} \end{cases} \tag{10}$$

$$\Diamond F = \begin{cases} \text{True} & \text{if } \exists t \geq 0 \;\; x(t) \in F \\ \text{False} & \text{otherwise.} \end{cases} \tag{11}$$

$\Box F$ can be derived by using $\Box F = \neg \Diamond F^c$ where $F^c = X \setminus F$. These temporal properties are used in specifying verification and synthesis problem for transition systems, which are generalizations of discrete systems, continuous systems as well as hybrid systems.

In a synthesis problem for Σ_c, we are interested in finding a collection of states, denoted F_s, such that $\forall x_s \in X_s$ a temporary property Ω is $True$. The collection of states that satisfies the temporary property is called the *winning states*[50, 49, 69]. In the following, synthesis problems for Σ_c with respect to $\Box F$ and $\Diamond F$ are presented.

For these problems, we first derive a $\Diamond$-algorithm for Σ_c. We can also derive a $\Box$-algorithm for Σ_c by utilizing the $\Diamond$-algorithm, since we know that the $\Box$ and $\Diamond$ properties are related by $\Box F = \neg \Diamond F^c$ where $F^c = X \setminus F$. The algorithms for solving these problems are given by Tables 10 and 9 respectively.

Table 9: Algorithm : $\Diamond$-Algorithm for Σ_c

```
**Input**: $\Sigma_c = (X, X_s, f)$, $D$, $F$, $T$
**Output**: $F_s$
**Start**
    $P = F$
    $R_0 = \emptyset$
    **Repeat** $k = 0, 1, 2, \cdots$
        $R_{k+1} = P \cup cPre_{cT}(R_k)$
    **Until** $R_{k+1} \subseteq R_k$
    $F_s = R_{k+1}$
**End**
```

Table 10: Algorithm: $\Box$-Algorithm for Σ_c

```
**Input**: $\Sigma_c = (X, X_s, f)$, $D$, $F$, $T$
**Output**: $F_s$
**Start**
    $P = X \setminus F$
    $R_0 = \emptyset$
    **Repeat** $k = 0, 1, 2, \cdots$
        $R_{k+1} = P \cup cPre_{cT}(R_k)$
    **Until** $R_{k+1} \subseteq R_k$
    $F_s = X \setminus R_{k+1}$
**End**
```

Figure 27 shows the design of $\Diamond$-algorithm in ReachLab. This algorithm can be saved as a template for future to be reused with a different system. By composing this algorithm as a subroutine and giving it as input $P = X \setminus F$ as shown in algorithm described in Table 10 we can design the $\Box$-Algorithm. The graphical design of this algorithm is shown in Figure 28. In the next section we will show some computational results of these algorithms by using some continuous dynamical systems with rich behavior. These systems were modeled in ReachLab's system aspect.

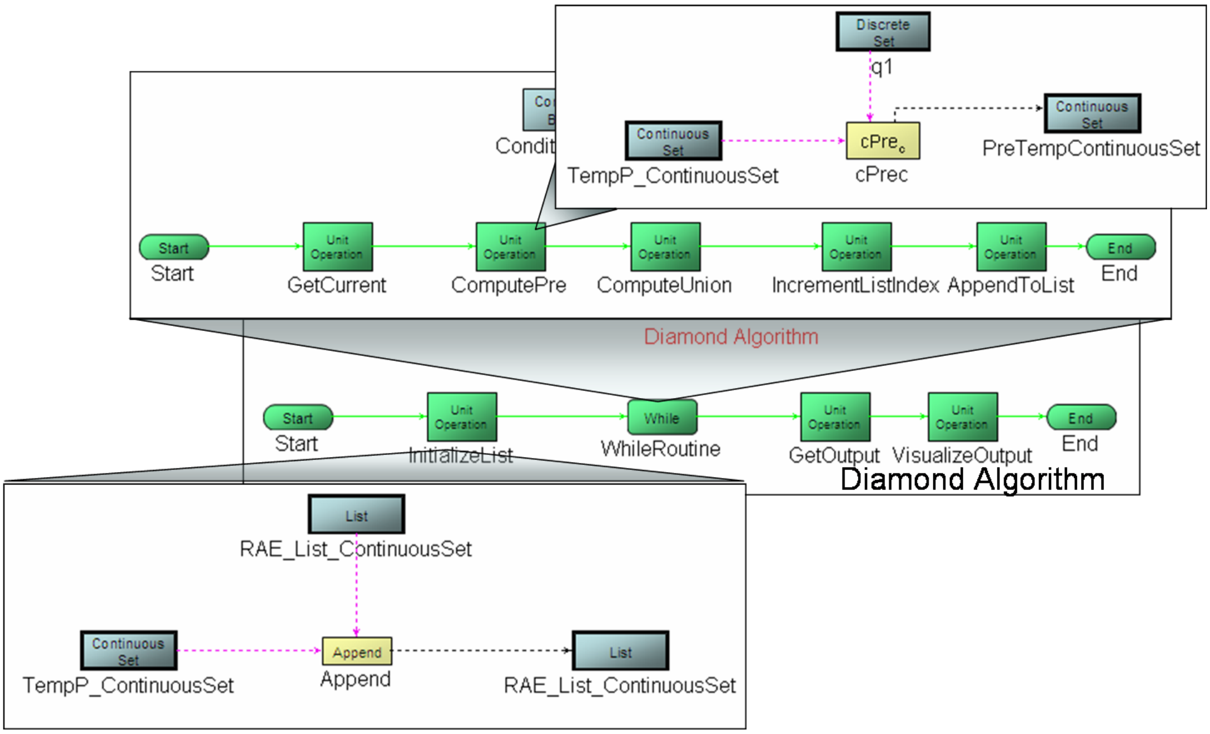
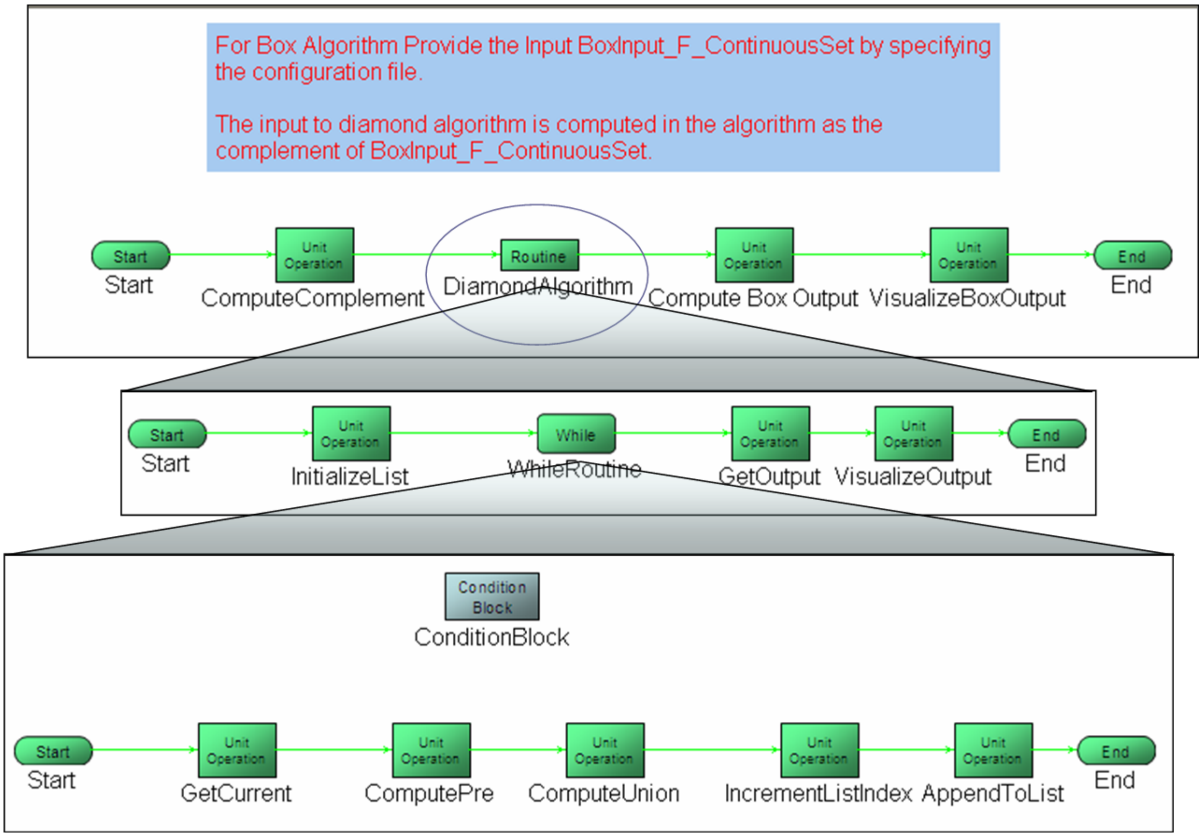


Figure 27: The ◇-algorithm designed in ReachLab. By changing the dynamics associated with the model of system in system aspect we can use this algorithm for different continuous systems.


For Box Algorithm Provide the Input BoxInput_F_ContinuousSet by specifying the configuration file.

The input to diamond algorithm is computed in the algorithm as the complement of BoxInput_F_ContinuousSet.

Figure 28: The □-algorithm designed in ReachLab.

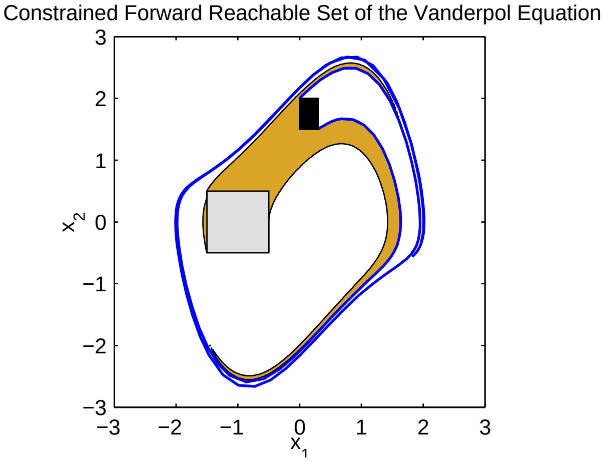


Figure 29: Constrained bounded-time Forward reachable set of the Vanderpol Equation, computed by the Level Set toolbox. The light gray box is the initial set F, the dark region is the forward reachable set, and the black box is the complement of the constraint set D.

Figure 29 shows the first example which uses the Vanderpol Equation [57] to demonstrate how $cPost_{cT}$ is computed with the Level Set toolbox. The constraint $D = X \setminus S$, where S is indicated as the black box in the figure that blocks the evolution of the reachable set. The reachable set is split into two parts but remains connected. The system dynamics are defined by the ODE $\dot{x}_1 = x_2$, and $\dot{x}_2 = x_2 \cdot (1 - x_1^2) - x_1$. The analysis set chosen was $[-3, 3] \times [-3, 3]$. The initial set $F = [-1.5, -0.5] \times [-0.5, 0.5]$. We chose a grid size of 0.025 in either dimension to create the mesh as required by level set toolbox. In order to compute the constrained continuous bounded time successor set used in this algorithm we chose a time range of $[0, 6.0]$ with a time step of $[0, 0.01]$. This computation took nearly 200 minutes on a Pentium IV 2.59GHz Windows machine with 512 MB memory.

The second example, shows the application of $\Diamond$-algorithm to a linear system in $\mathbb{R}^2$ with a stable focus. This example was first described in [57]. The system dynamics are defined by the ODE $\dot{x}_1 = -x_1 - 1.9x_2$, and $\dot{x}_2 = 1.9x_1 - x_2$. Figure 30 and 31 shows the results for this system using d/dt and Level Set toolbox. In both cases the algorithm terminates after the reachable set grows outside the analysis set. In both, figures the darker region gives the subset of the continuous state space from which it is possible to reach somewhere inside the continuous set F, which contains the equilibrium point.

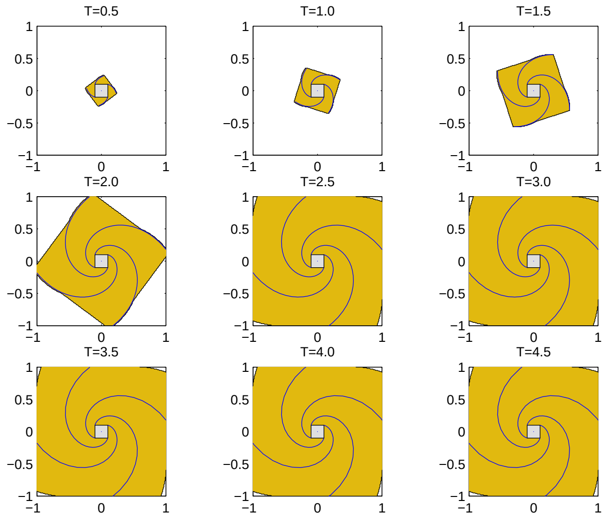

T=0.5 T=1.0 T=1.5 T=2.0 T=2.5 T=3.0 T=3.5 T=4.0 T=4.5

Figure 30: ◇-algorithm applied to a linear system with a stable focus, computed by d/dt. For each figure in the series, the light gray box in the center is F, and the dark region is the set that can be reached from F within time T. The solid curves are the trajectories from the four corners of F.

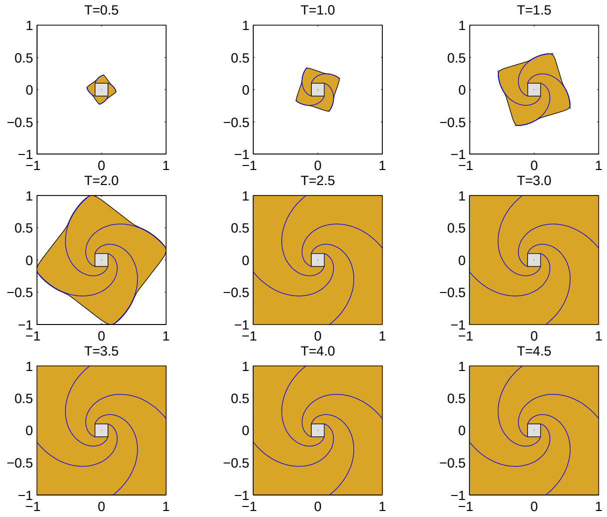
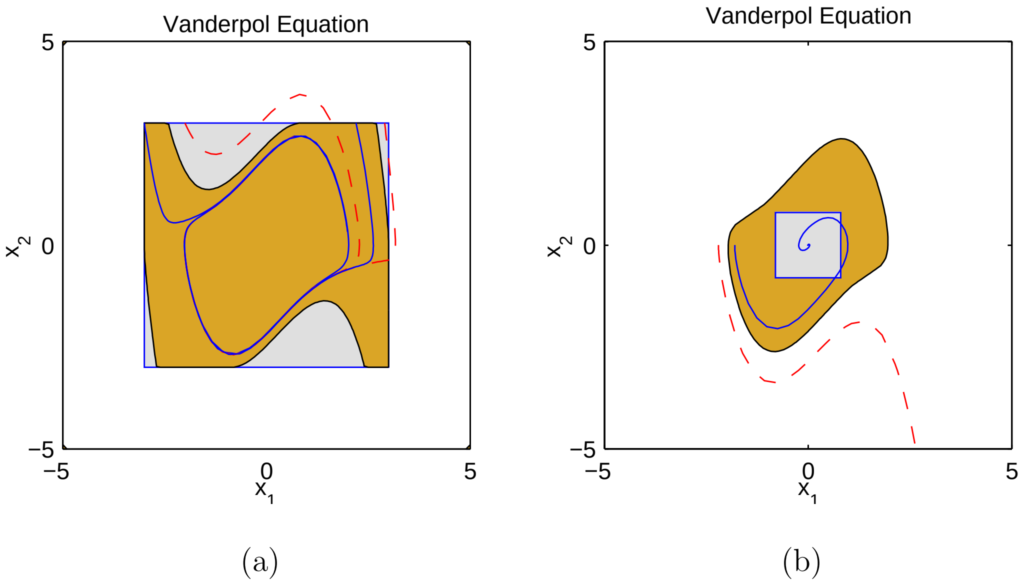

T=0.5 T=1.0 T=1.5 T=2.0 T=2.5 T=3.0 T=3.5 T=4.0 T=4.5

Figure 31: ◇-algorithm applied to a linear system with a stable focus, computed by Level Set toolbox. For each figure in the series, the light gray box in the center is F, and the dark region is the set that can be reached from F within time T. The solid curves are the trajectories from the four corners of F.

Both computations used the same parameters. The analysis set chosen was $[-1,1] \times [-1,1]$. The initial set $F = [-0.1, 0.1] \times [-0.1, 0.1]$. We chose a grid size of 0.01 in either dimension. In order to compute the constrained continuous bounded time successor set used in this algorithm we chose a time range of $[0, 4.5]$ with a time step of $[0, 0.05]$. This computation took nearly 9 minutes on a Pentium IV 2.59GHz Windows machine with 512 MB memory for Level Set toolbox. The same computation took less than a minute with d/dt kernel. This example illustrates the differences in implementation between the two kernels. However, since the algorithm used was designed in HADL we were able to compare the two results.



(a)

(b)

Figure 32: (a)$\Box$-algorithm applied to the Vanderpol Equation, computed by the Level Set toolbox. The light gray box is F, and the dark region is true for $\Box F$. The solid trajectory of the equation is always staying in F, and the dashed trajectory is not always inside; (b)$\Diamond$-algorithm applied to the ODE, by negating the vector field of the Vanderpol Equation. The light gray box is F, and the dark region is true for $\Diamond F$, which gives an estimate of the shape of the limit cycle.

The last example, shows the application of $\Box$-algorithm to the Vanderpol equation [57]. The dynamics of Vanderpol equation are same as the one in first example. This system has the property that it has a limit cycle around the origin. Therefore if the initial set lies inside the limit cycle the system will forever remain in a state which is inside the limit cycle. The computation result is shown in Figure 32(a). The sets F, F_s are indicated by the light gray region and the dark region, respectively. Trajectories starting from any point inside the dark regions will always stay inside, which shows the safety property of both systems are satisfied. These examples show how the positively invariant sets can be approximated. The analysis set chosen was $[-5,5] \times [-5,5]$. The set $F = [-3,3] \times [-3,3]$. We chose a grid size of 0.1 in either dimension to create the mesh as required by level set toolbox. In order to compute the constrained continuous bounded time successor set required for this algorithm we chose a time range of $[0,0.2]$ with a time step of $[0,0.01]$. It tools we took nearly 70 minutes on a Pentium IV 2.59GHz Windows machine with 512 MB memory.

We can also use these algorithms to estimate the shape of the limit cycle. Take the Vanderpol Equation as an example, it can be shown that there exist an attractive limit cycle and an equilibrium point inside the limit cycle. As shown in Figure 32(b), by negating the vector field of the Vanderpol Equation, the limit cycle becomes repulsive instead of attractive. Then, by having the set F surrounding the equilibrium point we can use our $\Diamond$-algorithm to estimate the shape of the limit cycle, as the dark region shown in the figure. The dynamics in this example is defined by the ODE $\dot{x}_1 = -x_2$, and $\dot{x}_2 = -(x_2 \cdot (1 - x_1^2) - x_1)$. The analysis set chosen was $[-5,5] \times [-5,5]$. The set $F = [-1,1] \times [-1,1]$. We chose a grid size of 0.1 in either dimension to create the mesh as required by level set toolbox. In order to compute the constrained continuous bounded time successor set required for this algorithm we chose a time range of $[0,0.1]$ with a time step of $[0,0.01]$. It tools we took nearly 60 minutes on a Pentium IV 2.59GHz Windows machine with 512 MB memory.

Summary

In this chapter we used ReachLab to design algorithms for analyzing linear as well as non linear time-invariant continuous dynamical systems. We also designed algorithm for computation forward reachable set of a hybrid automaton. Due to use of ReachLab in their design we were able to

1. Reuse the algorithms to quickly and efficiently design other algorithms.

2. Reuse the algorithms for analyzing different systems by modifying the system aspect. If we had to design these algorithms in each of the computation kernels separately without using ReachLab, it would have been a very cumbersome task.

3. Also, the design of these algorithms is independent of implementation. Therefore, one can store them and apply them to a number of systems easily.

We used d/dt for some examples while we used Level set toolbox for computation in other examples. In general one can choose either of them for their computation, provided that the system has linear continuous dynamics. However, we usually choose d/dt if we want faster computation for linear systems and use Level Set toolbox if we need to analyze non-linear systems.

CHAPTER VI

CONCLUSION

Model based approach for design and analysis of systems has made great advances in the last decade. Model driven system development methodologies like model integrated computing (MIC) allow integration and manipulation of models with manageable complexity in various aspects of system design. For embedded systems, model based design approach provides a scalable methodology for system design and analysis based on hybrid system theory in order to integrate efforts in system specification, design, synthesis, validation, verification and design evolution.

Hybrid systems, with tightly coupled discrete and continuous components, are analyzed using mathematical model formulations like hybrid automaton. Various algorithmic methods for formal verification and synthesis using hybrid automaton have been developed. These methods have been characterized as reductionist and symbolic methods. The main advantage of reductionist methods is the guarantee of termination. However, for a large class of hybrid systems, reductionist methods are unsuitable. Therefore, symbolic methods have to be used for a large class of hybrid systems.

This thesis reviewed various methods for continuous state set representation and approximations for computation of reachable sets that has been implemented in various symbolic method based computation tools. Each of these tools has been designed for specific analysis purposes and can be used for only a class of hybrid systems. Analysis algorithms developed using these tools are tightly coupled with the implementation details.

MIC approach has been used in order to enable the design of algorithms for analysis of hybrid automaton model such that the algorithms do not depend on any specific computational methods. However, we enable adoption of any specific computation methods for actual computation of results using the analysis algorithms. We also separated the algorithm logic from the concerned hybrid automaton model so that the algorithms can be reused for other hybrid automaton model as well.

Our approach is enabled by a computation platform called ReachLab, which provides a domain specific modeling language called Hybrid System Analysis and Design Language (HADL). This language uses a metamodel to relate abstractions of notions such as continuous state sets, operators for reachability computation, and Boolean operations on state sets. Implementation independent semantics of HADL is provided through the mathematical definitions of these abstractions. HADL provides multiple aspects to separate the concerns of analysis algorithms into programming logic, system models, and related data.

In ReachLab, on one hand, the models of analysis algorithms are abstract and therefore the design of algorithms can be made independent of implementation details. On the other hand, translators are provided to automatically generate implementations from the models for computing analysis results based on computation kernels. We enriched the capabilities of two computation tool d/dt and Level Set toolbox in order to use them as computation kernels.

Using this platform, we have developed many algorithms such as the synthesis algorithms for finding sets of initial states for the continuous dynamical systems so that some temporal properties, such as safety and liveness properties, are satisfied. We have also designed verification algorithms which can be used to check if the hybrid automaton will ever reach certain unsafe region. This platform has also been used to design synthesis algorithms for finding the execution sequence and switching surfaces for a DC-DC converter.

Future Works

While working with the two computation kernels in our architecture, we realized that even though the computation methods implemented in these tools are oblivious to higher dimensions, the constraints posed by memory requirements and computing power restrict their usage to systems with only three or maximum four dimensions. Considering that any practical system consists of several dimensions, this is a serious limitation. However, because of the use of ReachLab we can still keep on developing algorithms for analyzing these systems. In the future, we want to work at a more effective computation kernel which will be used along with ReachLab for effective analysis of system with higher dimensions.

Though currently, we only support a single hybrid automaton model, we would consider allowing networked hybrid automaton so that more sophisticated systems can be designed and analyzed. Use of shared memory for the communication between different hybrid automata can be adopted as well. Currently, clocks and continuous variables are modeled as continuous state variables in HADL which lead to an increase in continuous space dimension, resulting in a greater computation cost. In the future, clocks with deterministic rates will be supported as special variables decoupled from the continuous state space. This will lower the number of dimensions of the continuous state space required for analysis, so that existing tools can be used to handle analysis algorithms on more complex systems.

We have already started forming our library of algorithms which can be used for analyzing different hybrid automaton models in the future. We would want to augment this library with even more algorithms, especially for automatically checking temporal properties specified in temporal logic formulae such as in CTL and CTL* [29] for a hybrid automaton.

Our vision is to form an integrated suite of tools which would be used in every step of model based design of embedded systems. Currently, ReachLab enables the validation, verification, and synthesis for hybrid system models. We want to integrate ReachLab with Ptolemy II [71], which is a model based design tool for modeling, simulation and design of real-time embedded systems. By using metamodel based languages and model transformation tools like GReAT [2], we would be able to transform the Ptolemy II design models into models which can be used in ReachLab for either verification or controller synthesis. Moreover, we would want to augment this tool suite with code generation capabilities for generating executable artifacts for final systems.

This thesis is only a part of our complete vision; however, this work has given great insight into research which we need to do to achieve our vision of an integrated tool suite for model driven system development.

BIBLIOGRAPHY

[1] Yasmina Abdeddaim. *Modélisation et résolution de problèmes d'ordonnancement à l'aide d'automates temporisés*. PhD thesis, Institut National Polytechnique de Grenoble, November 2003.

[2] Aditya Agrawal, Gabor Karsai, Zsolt Kalmar, Sandeep Neema, Feng Shi, and Attila Vizhanyo. The design of a simple language for graph transformations. *Journal in Software and System Modeling*, 2005. Under Review.

[3] Rajeev Alur, Calin Belta, Vijay Kumar, Max Mintz, George J. Pappas, Harvey Rubin, and Jonathan Schug. Biocomputation: modeling and analyzing biomolecular networks. *Computing in Science and Engineering*, 4(1):20–31, January/February 2002.

[4] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A. Henzinger, Pei-Hsin Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995.

[5] Rajeev Alur, Thao Dang, Joel M. Esposito, Rafael B. Fierro, Yerang Hur, Franjo Ivancic, Vijay Kumar, Insup Lee, Pradyumna Mishra, George J. Pappas, and Oleg Sokolsky. Hierarchical hybrid modeling of embedded systems. In *EMSOFT '01: Proceedings of the First International Workshop on Embedded Software*, pages 14–31, London, UK, 2001. Springer-Verlag.

[6] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[7] Rajeev Alur, Joel M. Esposito, M. Kim, Vijay Kumar, and Insup Lee. Formal modeling and analysis of hybrid systems: A case study in multi-robot coordination. In *World Congress on Formal Methods*, pages 212–232, 1999.

[8] Rajeev Alur and Thomas A. Henzinger. Real-time system = discrete system + clock variables. *STTT*, 1(1-2):86–109, 1997.

[9] Eugene Asarin, Olivier Bournez, Thao Dang, Oded Maler, and Amir Pnueli. Effective synthesis of switching controllers for linear systems. *Proceedings of the IEEE*, 88(7):1011–1025, July 2000.

[10] Eugene Asarin, Thao Dang, and Oded Maler. The d/dt tool for verification of hybrid systems. In *Computer Aided Verification ,LNCS 2404*, pages 365–370. Springer-Verlag, July 2002.

[11] Eugene Asarin, Thao Dang, Oded Maler, and Olivier Bournez. Approximate reachability analysis of piecewise-linear dynamical systems. In Nancy A. Lynch and Bruce H. Krogh, editors, *Hybrid Systems: Computation and Control*, volume 1790 of *Lecture Notes in Computer Science*, pages 20–31, Pittsburgh, PA, USA, April 2000. Springer Verlag.

[12] Andrea Balluchi, Luca Benvenuti, Maria Domenica Di Benedetto, Claudio Pinello, and Alberto Luigi Sangiovanni-Vincentelli. Automotive engine control and hybrid systems: Challenges and opportunities. *Proceedings of the IEEE*, 88(7):888–912, July 2000.

[13] Alberto Bemporad and Manfred Morari. Verification of hybrid systems via mathematical programming. In Frits W. Vaandrager and Jan H. van Schuppen, editors, *Hybrid Systems: Computation and Control*, volume 1569 of *Lecture Notes in Computer Science*, pages 31–45, Berg en Dal, The Netherlands, March 1999. Springer Verlag.

[14] Johan Bengtsson, Kim Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Uppaal-a tool suite for automatic verification of real-time systems. In *Proceedings of the DIMACS/SYCON workshop on Hybrid systems III : verification and control*, pages 232–243, Secaucus, NJ, USA, 1996. Springer-Verlag New York, Inc.

[15] Oleg Botchkarev and Stavros Tripakis. Verification of hybrid systems with linear differential inclusions using ellipsoidal approximations. In Nancy A. Lynch and Bruce H. Krogh, editors, *Hybrid Systems: Computation and Control*, volume 1790 of *Lecture Notes in Computer Science*, pages 73–88, Pittsburgh, PA, USA, April 2000. Springer Verlag.

[16] Olivier Bournez, Oded Maler, and Amir Pnueli. Orthogonal polyhedra: Representation and computation. In Frits W. Vaandrager and Jan H. van Schuppen, editors, *Hybrid Systems: Computation and Control*, volume 1569 of *Lecture Notes in Computer Science*, pages 46–60, Berg en Dal, The Netherlands, March 1999. Springer Verlag.

[17] Christopher Brooks, Adam Cataldo, Edward A Lee, Jie Liu, Xiaojun Liu, Stephen Neuendorffer, and Haiyang Zheng. Hyvisual: A hybrid system visual modeler. Technical report, University of California, Berkeley, Berkeley, CA 94720, 2005.

[18] Giorgio C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers, Norwell, MA, USA, 2005.

[19] Kai Chen, Janos Sztipanovits, and Sandeep Neema. Toward a semantic anchoring infrastructure for domain-specific modeling languages. Fifth International Conference on Embedded Software (EMSOFT05), Jersey City, New Jersey, September 2005. (Accepted for publication).

[20] Alongkrit Chutinan and Bruce H. Krogh. Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations. In Frits W. Vaandrager and Jan H. van Schuppen, editors, *Hybrid Systems: Computation and Control*, volume 1569 of *Lecture Notes in Computer Science*, pages 76–90, Berg en Dal, The Netherlands, March 1999. Springer-Verlag.

[21] Alongkrit Chutinan and Bruce H. Krogh. Verification of infinite-state dynamic systems using approximate quotient transition systems. *IEEE Transactions on Automatic Control*, 46(9):1401–1410, September 2001.

[22] Tony Clark, Andy Evans, Stuart Kent, and Paul Sammut. The MMF approach to engineering object-oriented design languages. In *Workshop on Language Descriptions, Tools and Applications.LDTA*, Genova, Italy, 2001. Available via http://www.puml.org.

[23] Edmund M Clarke and Robert P. Kurshan. Computer-aided verification. *IEEE Spectrum*, 33(6):61–67, June 1996.

[24] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT PRESS, Cambridge, MA, USA, second edition, 2001.

[25] Rational Software Corporation. *UML Semantics ver 1.1*, 1997.

[26] José E. R. Cury, Bruce H. Krogh, and Toshihiko Niinomi. Synthesis of supervisory controllers for hybrid systems based on approximating automata. *IEEE Transactions on Automatic Control*, 43(4):564–568, April 1998.

[27] Thao Dang. *Verification Et Synthesis Des Systemes Hybrides*. PhD thesis, Institut National Polytechnique De Grenoble, October 2000.

[28] Abhishek Dubey, Xianbin Wu, Hang Su, and T. John Koo. Computation platform for automatic analysis of embedded software systems using model based approach. In *Automated Technology for Verification and Analysis*, October. Accepted for publication.

[29] E. Allen Emerson. Temporal and modal logic. *Handbook of theoretical computer science (vol. B): formal models and semantics*, pages 995–1072, 1990.

[30] Akos Ledeczi et. al. *GME 2000 Users Manual (v2.0)*. Institute For Software Integrated Systems, Nashville, TN, USA, 2000. Available from http://www.isis.vanderbilt.edu.

[31] Rational Software Corporation et a.l. *Object Constraint Language Specification ver 1.1*, September 1997.

[32] Edward D. Gaughan. *Introduction to Analysis*. Brooks Cole, Pacific Grove, CA, USA, fifth edition, December 1997.

[33] Jack Greenfield, Keith Short, Steve Cook, and Stuart Kent. *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*. John Wiley & Sons, New York, 2004.

[34] Mark R. Greenstreet and Ian Mitchell. Reachability analysis using polygonal projections. In Frits W. Vaandrager and Jan H. van Schuppen, editors, *Hybrid Systems: Computation and Control*, volume 1569 of *Lecture Notes in Computer Science*, pages 103–116, Berg en Dal, The Netherlands, March 1999. Springer Verlag.

[35] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.

[36] Thomas A. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS '96)*, pages 278–292, New Brunswick, New Jersey, 1996.

[37] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. HYTECH: A model checker for hybrid systems. *International Journal on Software Tools for Technology Transfer*, 1(1–2):110–122, 1997.

[38] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? In *STOC '95: Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pages 373–382, New York, NY, USA, 1995. ACM Press.

[39] Thomas A. Henzinger and Rupak Majumdar. A classification of symbolic transition systems. In H. Reichel and S. Tison, editors, *Proceedings of the 17th International Conference on Theoretical Aspects of Computer Science (STACS 2000)*, Lectures Notes in Computer Science, pages 13–34. Springer Verlag, February 2000.

[40] Gabor Karsai, Aditya Agrawal, and Akos Ledeczi. A metamodel-driven MDA process and its tools. Workshop in Software Model Engineering (WISME), San Francisco, USA, 2003.

[41] Gabor Karsai, Janos Sztipanovits, Akos Ledeczi, and Ted Bapty. Model-integrated development of embedded software. *Proceedings of the IEEE*, 91(1):145– 164, January 2003.

[42] Xenofon D. Koutsoukos and Panos J. Antsaklis. A hybrid feedback regulator approach to control an automotive suspension system. In Nancy A. Lynch and Bruce H. Krogh, editors, *Hybrid Systems: Computation and Control*, volume 1790 of *Lecture Notes in Computer Science*, pages 188–201, Pittsburgh, PA, USA, April 2000. Springer Verlag.

[43] Alexander B. Kurzhanski and Pravin Varaiya. Ellipsoidal techniques for reachability analysis. In Nancy A. Lynch and Bruce H. Krogh, editors, *Hybrid Systems: Computation and Control*, volume 1790 of *Lecture Notes in Computer Science*, pages 202–214, Pittsburgh, PA, USA, April 2000. Springer Verlag.

[44] Gerardo Lafferriere, George J. Pappas, and Sergio Yovine. Symbolic reachability computation for families of linear vector fields. *Journal of Symbolic Computation*, 32(3):231–253, September 2001.

[45] Akos Ledeczi, Miklos Maroti, Arpad Bakay, Gabor Karsai, Jason Garrett, Charles Thomason, Greg Nordstrom, Jonathan Sprinkle, and Peter Volgyesi. Generic modeling environment. In *WISP*, volume IEEE International Workshop on Intelligent Signal Processing, Budapest, Hungary, May 2001.

[46] Edward A. Lee and Haiyang Zheng. Operational semantics of hybrid systems. In Manfred Morari and Lothar Thiele, editors, *Hybrid Systems: Computation and Control*, volume 3414 of *Lecture Notes in Computer Science*, pages 25–53, Zurich, Switzerland, March 2005. Springer Verlag.

[47] Scott Little, David Walter, Nicholas Seegmiller, Chris J. Myers, and Tomohiro Yoneda. Verification of analog and mixed-signal circuits using timed hybrid petri nets. In Farn Wang, editor, *Automated Technology for Verification and Analysis*, volume 3299 of *Lecture Notes in Computer Science*, pages 426–440, Taipei, Taiwan, October/November 2004. Springer Verlag.

[48] John Lygeros. Lecture notes on hybrid systems. Technical report, University of Patras, Rio, Patras, GR-26500, Greece, 2004.

[49] Oded Maler. Control from computer science. *Annual Reviews in Control*, 26(1):175–297, 2002.

[50] Oded Maler, Amir Pnueli, and Joseph Sifakis. On the synthesis of discrete controllers for timed systems. In Ernst W. Mayr and Claude Puech, editors, *STACS*, volume 900 of *Lecture Notes in Computer Science*, pages 229–242. Springer Verlag, March 1995.

[51] Ian M. Mitchell and Jeremy A. Templeton. A toolbox of Hamilton-Jacobi solvers for analysis of nondeterministic continuous and hybrid systems. In Manfred Morari and Lothar Thiele, editors, *Hybrid Systems: Computation and Control*, volume 3414 of *Lecture Notes in Computer Science*, pages 480–494, Zurich, Switzerland, 2005. Springer Verlag.

[52] Katsuhiko Ogata. *Modern control engineering (3rd ed.).* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1997.

[53] Stanley Osher and Ronald Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*, volume 153 of *Applied Mathematical Sciences,*. Springer, 2003.

[54] Stefan Pettersson and Bengt Lennartson. Stability of hybrid systems using LMIs - a gear-box application. In Nancy A. Lynch and Bruce H. Krogh, editors, *Hybrid Systems: Computation and Control*, volume 1790 of *Lecture Notes in Computer Science*, pages 381–395, Pittsburgh, PA, USA, April 2000. Springer Verlag.

[55] Anders Rantzer and Mikael Johansson. Piecewise linear quadratic optimal control. *IEEE Transactions on Automatic Control*, 45(4):629–637, April 2000.

[56] Jeff Rothenberg. The nature of modeling. *Artificial intelligence, simulation & modeling*, pages 75–92, 1989.

[57] Shankar Sastry. *Nonlinear Systems: Analysis, Stability and Control.* Springer Verlag, New York, USA, 1999.

[58] Shankar Sastry, Janos Sztipanovits, R. Bajcsy, and H. Gill. Scanning the issue - special issue on modeling and design of embedded software. *Proceedings of the IEEE*, 91(1):3–10, 2003.

[59] Matthew Senesky, Gabriel Eirea, and T John Koo. Hybrid modeling and control of power electronics. In Oded Maler and Amir Pnueli, editors, *Hybrid Systems: Computation and Control*, volume 2623 of *Lecture Notes in Computer Science*, pages 450–465, Prague, Czech Republic, April 2003. Springer Verlag.

[60] James A Sethian. *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry. Fluid Mechanics, Computer Vision, and Materials Science*, volume 3 of *Cambridge Monographs on Applied and Computational Mathematics.* Cambridge University Press, October 1999.

[61] Marco Sgroi, Luciano Lavagno, and Alberto Sangiovanni Vincentelli. Formal models for embedded system design. *IEEE Design and Test of Computers*, 17(2):14–27, June 2000.

[62] Jonathan Sprinkle, Gabor Karsai, and Andras Lang. Hybrid systems interchange format v.4.1.8. Technical report, Institute of Integrated Software Systems, Vanderbilt University, January 2004.

[63] National Research Council Staff. *Embedded Everywhere: A Research Agenda for Networked Systems of Embedded Computers.* National Academy Press, Washington, DC, USA, 2001.

[64] Olaf Stursberg and Bruce H. Krogh. Efficient representation and computation of reachable sets for hybrid systems. In Oded Maler and Amir Pnueli, editors, *Hybrid Systems: Computation and Control*, volume 2623 of *Lecture Notes in Computer Science*, pages 482–497, Prague, Czech Republic, April 2003. Springer Verlag.

[65] Janos Sztipanovits and Gabor Karsai. Model-integrated computing. *Computer*, 30(4):110–111, 1997.

[66] Janos Sztipanovits, Gabor Karsai, Csaba Biegl, Ted Bapty, Akos Ledeczi, and Donald J Malloy. Multigraph: an architecture for model-integrated computing. In *Proceedings of the 1st International Conference on Engineering of Complex Computer Systems*, pages 361–368, November 1995.

[67] Janos Sztipanovits, Gabor Karsai, and Hubertus Franke. Model-integrated program synthesis environment. In *ECBS '96: Proceedings of the IEEE Symposium and Workshop on Engineering of Computer Based Systems*, pages 348–355, Washington, DC, USA, March 1996. IEEE Computer Society.

[68] Michael Tiller. *Introduction to Physical Modeling with Modelica.* Kluwer Academic Publishers, Norwell, MA, USA, 2001.

[69] Claire Tomlin, John Lygeros, and Shankar Sastry. A game theoretic approach to controller design for hybrid systems. *Proceedings of the IEEE*, 88(7):949–970, July 2000.

[70] Claire Tomlin, George J. Pappas, and Shankar Sastry. Conflict resolution for air traffic management: a study in multiagent hybrid systems. *IEEE Transactions on Automatic Control*, 43(4):509–521, April 1998.

[71] University of California at Berkeley, http://ptolemy.eecs.berkeley.edu/ptolemyII/.

[72] Howard Wong-Toi. The synthesis of controllers for linear hybrid automata. In *IEEE Conference of Decision and Control*, volume 5, pages 4607–4612, San Diego, CA, USA, December 1997.

[73] Sergio Yovine. Kronos: A verification tool for real-time systems. *International Journal on Software Tools for Technology Transfer*, 126:110–122, 1997.