Algorithms for Feature Based Formal Equivalence Checking between Hybrid Systems

Bruto da Costa Antonio Anastasio

Algorithms for Feature Based Formal Equivalence Checking between Hybrid Systems

Thesis submitted to
Indian Institute of Technology Kharagpur
for the award of the degree of

Master of Technology
in
Computer Science and Engineering

by

Bruto da Costa Antonio Anastasio
(Roll No: 12CS60R20)

Under the guidance of
Professor Pallab Dasgupta

Deptartment of Computer Science and Engineering
Indian Institute of Technology - Kharagpur
West Bengal - India, 721302
April 2014

Dedicated to my dearest parents.

CERTIFICATE

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
WB 721302, India.

This is to certify that the thesis entitled **Algorithms for Feature Based Formal Equivalence Checking between Hybrid Systems**, submitted by **Bruto da Costa Antonio Anastasio**, Roll Number: **12CS60R20**, in the *Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur, India*, for the award of the degree of **Master of Technology**, is a record of original research work carried out by him under my supervision and guidance. The thesis fulfills all the requirements as per the regulations of this institute. Neither this thesis nor any part of it has been submitted for any degree or academic award elsewhere.

Prof. Pallab Dasgupta
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur

DECLARATION

I, Bruto da Costa Antonio Anastasio, Roll No. 12CS60R20, registered as a student of the M.Tech program in the Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur, India (herein after referred to as the 'Institute') do hereby submit my project report, titled: Algorithms for Feature Based Formal Equivalence Checking between Hybrid Systems (herein after referred to as 'my thesis') in a printed as well as in an electronic version for holding in the library of record of the Institute.

I hereby declare that:

1. The electronic version of my thesis submitted herewith on CDROM is in PDF Format.

2. My thesis is my original work of which the copyright vests in me and my thesis does not infringe or violate the rights of anyone else.

3. The contents of the electronic version of my thesis submitted herewith are the same as that submitted as final hard copy of my thesis after my viva voce and adjudication of my thesis in April/May 2014.

4. I agree to abide by the terms and conditions of the Institute Policy and Intellectual Property (herein after Policy) currently in effect, as approved by the competent authority of the Institute.

5. I agree to allow the Institute to make available the abstract of my thesis in both hard copy (printed) and electronic form.

6. For the Institute's own, non commercial, academic use I grant to the Institute the non-exclusive license to make limited copies of my thesis in whole or in part and to loan such copies at the Institute's discretion to academic persons and bodies approved of from time to time by the Institute for non-commercial academic use. All usage under this clause will be governed by the relevant fair use provisions in the Policy and by the Indian Copyright Act in force at the time of submission of the thesis.

7. Furthermore,

    (a) I agree to allow the Institute to place such copies of the electronic version of my thesis on the private Intranet maintained by the Institute for its own academic community.

   (b) I agree to allow the Institute to publish such copies of the electronic version of my thesis on a public access website of the Internet should it so desire.

8. That in keeping with the said Policy of the Institute I agree to assign to the Institute (or its Designee/s) according to the following categories all rights in inventions, discoveries or rights of patent and/or similar property rights derived from my thesis where my thesis has been completed.

   (a) With use of Institute-supported resources as defined by the Policy and revisions thereof,

   (b) With support, in part or whole, from a sponsored project or program, vide clause 6(m) of the Policy. I further recognize that:

   (c) All rights in intellectual property described in my thesis where my work does not qualify under sub-clauses 8(a) and/or 8(b) remain with me.

9. The Institute will evaluate my thesis under clause 6(b1) of the Policy. If intellectual property described in my thesis qualifies under clause 6(b1) (ii) as Institute-owned intellectual property, the Institute will proceed for commercialization of the property under clause 6(b4) of the policy. I agree to maintain confidentiality as per clause 6(b4) of the Policy.

10. If the Institute does not wish to file a patent based on my thesis, and it is my opinion that my thesis describes patentable intellectual property to which I wish to restrict access, I agree to notify the Institute to that effect. In such a case no part of my thesis may be disclosed by the Institute to any person(s) without my written authorization for one year after the date of submission of the thesis or the period necessary for sealing the patent, whichever is earlier.

Bruto da Costa Antonio Anastasio
(Name of the Student)

Prof. Pallab Dasgupta
(Name of Supervisor)

Acknowledgements

I am most grateful to the Almighty for blessing me with the opportunity to be part of this great institute, and for the wonderful people that have made this journey memorable.

My experience as an M.Tech student at IIT-Kharagpur has been nothing short of amazing. I have had the opportunity to challenge myself in many areas, and I am blessed to have had a wonderful support structure to guide me through these two years.

First and foremost, I wish to thank my adviser Prof. Pallab Dasgupta. Words cannot express how much he has influenced my work and my life during these years. He has been my inspiration, a true teacher, who first introduced me to the world of formal methods, and ignited my curiosity in the area. He has been involved in my work, every step of the way, steadily steering me when needed, always lifting my spirits and encouraging me to work harder, and be better than I ever thought I could. I thank him for his patience, enthusiasm, motivation, and immense knowledge. A quick visit to his office is sufficient to lighten my mood and lift my spirits. In the past year, he has taken the role being my project guide, my councilor and my confidante. I could not have imagined having a better guide during my M.Tech.

I acknowledge the support of Semiconductor Research Corporation (SRC) through the research grant, 2012-TJ-2267. I also thank Dr. Rubin Parekhji and Mr. Lakshmanan Balasubramanian of Texas Instruments, India, for their comments and encouragement.

I want to express my gratitude to all the members of the Formal Verification group that have supported me. Special thanks to Antara Ain, and Aritra Hazra for their guidance and support.

Dozens of people have helped teach me during my M.Tech and I am most grateful to them all. I also wish to thank the faculty and staff of the Department of Computer Science, for their help and support; and for promoting a stimulating and welcoming academic and social environment. My sincere thanks to Prof. Partha Pratim Das, our class faculty adviser for his overall guidance during the M.Tech program.

I also acknowledge the Directorate of Higher Education - Goa, for awarding me with the Goa Scholars scholarship, enabling me to pursue the M.Tech course at IIT-Kharagpur.

I am greatly indebted to all my colleagues and friends at IIT-Kharagpur who have supported me through this journey. Special gratitude go to Santhosh, Jit, Nirvik, Nita and Remya, for their kindness, friendship, support and for adding spice to my life at IIT.

Home is where my heart is, and I would not have had the strength to make it through these two years, had it not been for the support from my wonderful friends

from home. It is you all who have helped mould me into who I am today. The difficulties of being in a new place, thousands of miles away from home, dissipated via extended phone calls, skype calls and google hangouts. Some who knew would say that I carried home along with me. My gratitude goes especially to Monalizza Lobo, Geena Sandhu, Shweta Nadkarni, Melody Coelho, Elton Carvalho, Roshwin Carvalho, Godwin Clovis da Costa, Elroy Pereira, Stephanie Barreto and Clarice Pereira. Snehalesh Mahale, I would not be in IIT, had you not encouraged and mentored me. I will always be most grateful for having you all in my life.

The role played by teachers never ends, and as such I would like to thank Prof. Maruska Mascarenhas (Goa Engineering College) who always believed in my academic abilities and for always having words of encouragement for me. Sometimes students need a little nudging to head back into academia. Thank you for seeing potential where I hadn't, and for helping me see far beyond.

Finally, I wouldn't have had the strength to push through the varied challenges I've faced without the unwavering support of my parents and brothers. You have always had faith in me, encouraging me to be myself and do what I love.

$\tau h \alpha \eta \kappa\ \gamma o \upsilon$

Bruto da Costa Antonio Anastasio
Roll No. 12CS60R20
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur

List of Abbreviations

AMS	-	Analog and Mixed Signal
LTL	-	Linear Temporal Logic
PTL	-	Propositional Temporal Logic
MTL	-	Metric Temporal Logic
MITL	-	Metric Interval Temporal Logic
TPTL	-	Timed Propositional Temporal Logic
STL	-	Signal Temporal Logic
PORV	-	Predicate Over Real Variables
SVA	-	SystemVerilog Assertions
PSL	-	Property Specification Language
LDO	-	Linear Drop-Out Regulator

List of Symbols

$\wedge, \&$	–	Logical AND
$\vee, \|\|$	–	Logical OR
$\neg$	–	Logical NOT (Negation)
$\times$	–	Cartesian Product
$\mathbb{R}$	–	Real set
$\mathcal{G}$ or $\Box$	–	Global operator of LTL.
$\mathcal{F}$ or $\Diamond$	–	Future operator of LTL.
$\mathcal{U}$	–	Until operator of LTL.
$\Rightarrow$	–	Implication operator used in AMS-LTL
$[\#\#\ t]$	–	an exact delay of t time units used in AMS-LTL and SVA
$@^{+}(X)$	–	At the positive edge of the signal X.
$@^{-}(X)$	–	At the negative edge of the signal X.
$\in$	–	Belongs to
$\subset$	–	Subset of
$\forall$	–	for all
$\exists$	–	there exists
$=$	–	is equal to
$\neq$	–	does not equal

Abstract

Equivalence checking for finite state systems is well studied and many algorithms exist that can determine whether or not two finite state systems are equivalent. Analysing infinite state systems is very complex. Infinite state systems can be modeled as Hybrid Systems, that have discrete and continuous behaviours that describe the hybrid state. Many methods for checking equivalence between two Finite State Systems exist. However, for infinite state systems, this traditional crude definition of equivalence is too constrained. A more effective equivalence definition is one that is best expressed in terms of features or behavioural signatures of the infinite state systems. Behavioural aspects used for feature comparison can be expressed as a linear combination of state variables. For a given Hybrid Automaton, the range of values taken on by a specific feature aspect can be obtained by observing the state space of the Hybrid System reachable from an initial state. However, we cannot directly apply the traditional reachability analysis to determine feature based equivalence. This report formalizes the notion of feature based reachability analysis of Hybrid Systems and describes feature driven modifications required to prepare the given Hybrid Automata for feature based equivalence checking.

Keywords: *Integrated Circuits, Verification, Mathematical logic, Formal Languages, Hybrid Automata, Reachability Analysis.*

Contents

List of Figures

List of Tables

Chapter 1
Introduction

"There is nothing more difficult to take in hand,
more perilous to conduct, or more uncertain in its success,
than to take the lead in the introduction of a new order of things."
-Niccolo Machiavelli

Dynamical Systems can be broadly classified based on the evolution of the system state over time. The state of the system can be viewed as consisting of the valuation of a finite number of variables. Some system variables may change discretely whereas others may evolve continuously as time progresses. With this knowledge we can proceed to classify systems as follows:

1. Continuous: If the state takes values in the Euclidean space $\mathbb{R}^n$ for some $n \geq 1$. We will use $x \in \mathbb{R}^n$ to denote the state of a continuous dynamical system.

2. Discrete: If the state takes values in a countable or finite set $\{q_1, q_2, ...\}$. We use q to denote the state of a discrete system.

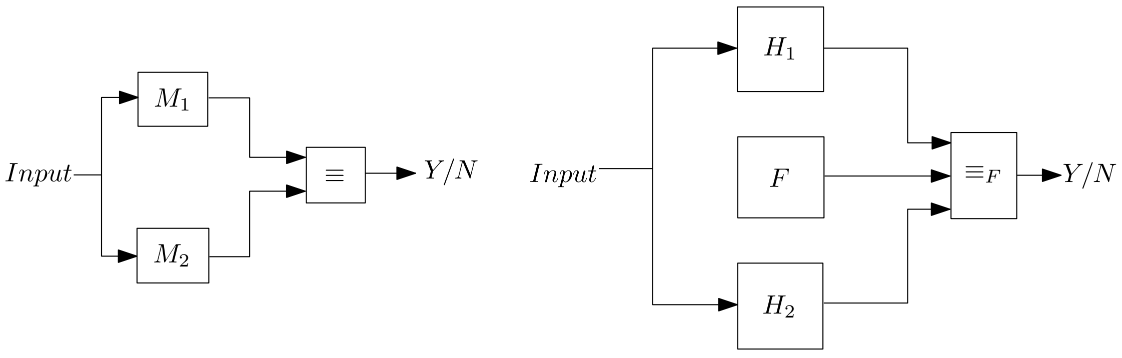
3. Hybrid: If one part of the system state takes values in $\mathbb{R}^n$ while another part takes values in a finite set.

We are interested in the third classification of systems, i.e. Hybrid Systems. The behaviour of Hybrid Systems can be captured by Hybrid Automata [4]. The topic reachability analysis and model checking between Hybrid Automata has been well studied [4][9][13]. However, no algorithms are known, that meaningfully describe the equivalence relationship between Hybrid Automata.

1.1. Motivation

Many systems can be modelled as Hybrid Automata. Many classes of systems exist for which the traditional definition of state equivalence does not apply. Influenced by other work being done by the Formal Verification group at IIT-Kharagpur, this research looks specifically at the class of AMS circuits. AMS circuits can be modelled as Hybrid Automata [4][40] by capturing the discrete modes of the circuit(modes of the digital controller) as the locations of the hybrid automaton, and by using control variables and differential equations to capture the circuit's analog behaviour. The class of analog and mixed-signal(AMS) circuits is one class of Hybrid Automata wherein the definition of equivalence must evolve to consider behavioural signatures that capture the designer's definition of equivalence.

With respect to AMS circuits, in determining the equivalence, it is important to view the notion of equivalence with respect to domain specific features of the circuit. Complex AMS circuits have multiple modes of operation and the definition of equivalence often relates to specific ways in which the circuit is expected to behave in different operating modes [17]. For example, for a Linear Dropout regulator (LDO) the *rise time*, which is the time required for the output of the LDO to rise from 10% to 90% of its rated voltage, is a feature of the start-up mode of operation for an LDO. The rise time is one of the several features that is typically used in the definition of equivalence between LDOs, that is, it may be a requirement, that the rise times for two equivalent LDOs be equal, within a tolerance of 0.1μs.



(a) Traditional Equivalence (b) Feature Based Equivalence

Figure 1.1: Traditional v/s Feature Based Equivalence

As shown in Figure 1.1, the traditional definition of equivalence talks about two machines M_1 and M_2 being equivalent, if for the same inputs they produce the same outputs. This definition of equivalence is appropriate for systems having a countable number of states. However for infinite state systems, this notion becomes too restrictive and is considered outdated. Infinite state systems are Hybrid Systems and can be modelled as Hybrid Automata. Equivalence between two Hybrid Automata must be defined in the context of the behaviours portrayed by the two automata. These behaviours are termed as features, that essentially describe patterns observed in runs of the Hybrid Systems.

This work is thus motivated to define the equivalence between two Hybrid

Systems on the basis of specific features, where the features are based on the system designers understanding of equivalence between the two systems.

Given behaviours of interest to the designer, captured formally using features, there is a need to develop a method to have these features drive the process of equivalence checking through standard reachability analysis.

1.2. Problem Description and Objective

We first intuitively understand the notion of feature based equivalence. A *feature based* equivalence definition has two important components:

1. *The specification of the feature*, which in turn has two main parts:
   - A specific functionality(behaviour) over which the feature is evaluated.
   - A expression for evaluating the feature over the specified behaviour.
2. *The specification of the quivalence predicate*, which is a comparative function that relates the valuations of the feature over designs being compared.

Having formalized the notion of feature, the general problem this research addresses is defined below:

> Given two Hybrid Automata defined as $H_1 = (Q_1, X_1, f_1, Init_1, Dom_1, E_1, G_1, R_1)$ and $H_2 = (Q_2, X_2, f_2, Init_2, Dom_2, E_2, G_2, R_2)$, and a formal equivalence definition constrained by feature F, the goal is to compute the set of feature values for H_1 and H_2, using them to determine if the two Hybrid Automata H_1 and H_2 are equivalent with respect to the feature F. The aim is to answer the question,

$$\boxed{\text{Is } H_1 \equiv_F H_2 \text{ ?}}$$

The first step to establishing feature based equivalence between two Hybrid Automata is to define features relevant to an equivalence comparison. Each feature of interest will correspond to a system behaviour associated with a quantity, a signature of the feature, which can be expressed as a linear combination of control variables of the Hybrid Systems. The values of these control variables represent the internal state of the Hybrid System. Internal states of a Hybrid System that are visited during its operation can be identified using standard reachability analysis techniques.

Reachability analysis for Linear Hybrid Systems has been well studied [4][9][13] and many tools exist that perform reachability analysis [14]. Reachability analysis answers questions about states that are reachable in each mode of operation of the system and in the system as a whole. However, current techniques for reachability analysis do not answer questions such as, *In the discrete state l, when variable x crosses 10, what is the value of variable y?*, cannot be answered due to the level of abstraction at which the systems behaviour is represented.

Features might consider cross events that occur within a given location of the Hybrid Automaton, or might require variable values to be captured across different time points within a location and across locations. A reachability analysis will provide the entire set of reachable states within each location of the automaton, independent of the events and predicates specified in the features. The feature might also require the introduction of auxiliary variables such as clocks, or variables derived from existing control variables of the system. How can current reachability analysis algorithms be extended to handle features? *In what way can existing tools be leveraged to answer questions about equivalence?* In this dissertaion a technique for performing feature based abstract interpretation is introduced, that piggybacks feature computation over standard flow-pipe analysis, to extract feature values and determine equivalence.

The major objectives of this research are summarized as follows:

1. Developing a formal language for specifying features for Hybrid Automata and for specifying the equivalence criterion that dictates whether or not two Hybrid Automata are equivalent.

2. Algoirthms for creating feature driven abstractions of the given Hybrid Automaton, with the aim of driving the reachability analysis implicitly using the feature specification.

1.3. Summary of Contributions

This thesis presents the findings of our research on creating a formal framework for determining feature based equivalence between Hybrid Automata. We formalize the notion of features and feature based equivalence, discuss various categories of features that are relevant to the definition of equivalence, present feature driven non-trivial transformations that prepare the Hybrid Automaton for feature based reachability analysis, establish the use of reachability results in determining equivalence and present case studies on two power management circuits.

In this work, categories of features that can be used for feature based equivalence checking are identified. Two broad categories of features are explored:

1. **Features in the value domain:** Features in the value domain consist of those features whose measurement is solely based on aspects of the system

that do not directly measure time. An example would be a feature whose value is the *dropout voltage* of an LDO(Smallest difference between input and output voltages required to maintain regulation). Here, *dropout voltage* is a value that is independent of time.

2. **Features in the time domain:** Features in the time domain consist of features whose signatures are computed based on time. An example would be a feature that measures the *time* it takes to charge a completely drained battery to full capacity. Here, the feature signature solely consists of a time measurement.

We formalize the syntax used to specify a feature and the classes of feature assertions that can be used in the context of determining Feature based equivalence between Hybrid Systems.

We show that, if the feature signature is expressed linearly in terms of control variables of the Hybrid Automaton, then the range of values that define the feature signature, can be calculated from a flow pipe analysis of the Hybrid Automaton.

The *flow pipe analysis* provides an over approximation of the reachable region for each location, consisting of all possible trajectories and state valuations. In order to focus the flow pipe analysis on the trajectories of interest, the given Hybrid Automaton is modified on the basis of the feature specification.

A Hybrid Automaton is modified, by adding new control variables, special transitions, splitting and duplicating hybrid locations, and finally marking locations for analysis, on the basis of the assertion that captures the behaviour of interest, as specified by the feature.

Running the flow pipe analysis on the modified Hybrid Automaton then ensures that the feature implicitly drives the flow pipe analysis.

Once the flow pipe analysis is complete, the feature signature is computed using the polyhedra computed from the flow pipe. This gives us the maximum and minimum values of the feature signature. We obtain the extremal points for the feature signature defined for each Hybrid Automaton. Using these extremal values, in conjunction with the goal predicate, feature based equivalence between the two Hybrid Automata can be determined.

Automated evaluation of ranges for formally specified features is a new challenge addressed for the first time in this dissertation. We introduce a set of transformations corresponding to a given feature, which when applied on the given Hybrid Automaton, reduces the feature range computation problem to an equivalent reachability analysis problem in the modified Hybrid Automaton. The reduced problem is then solved using standard abstract interpretation techniques, and interpreting the result yields the desired feature range.

1.4. Organization of the Report

This report demonstrates how features can be incorporated into the given Hybrid Automata, allowing features to drive the reachability analysis for Hybrid Systems, and thus how this helps us in formally checking for feature based equivalence between two Hybrid Systems.

The thesis is organized as follows:

- ▷ Chapter 2 discusses the theory required to understanding how Hybrid Systems are modeled as Hybrid Automata, how reachability analysis is performed and tools for modeling and analyzing Hybrid Automata. It also describes assertion languages relevant to developing a framework for formally specifying feature and equivalence definitions.
- ▷ Chapter 3 introduces the notion of features and feature based equivalence. This chapter also shows how the result of the reachability analysis can be used to determine the equivalence between two Hybrid Automata.
- ▷ Chapter 4 enumerates various classes of features and focuses on algorithms that use the feature definition to transform the Hybrid Automaton so that the feature implicitly drives the reachability analysis of the automaton.
- ▷ Chapter 5 extends the notion of features, by formally defining equivalence between two Hybrid Automata. This chapter introduces the syntax used for specifying the equivalence criterion, and demonstrates the use of the syntax using examples over AMS circuits in the Power Management Domain.
- ▷ Chapter 6 describes case studies of two AMS circuits in the Power Management Domain, namely Low Dropout (LDO) Voltage Regulators and Battery Chargers. Each circuit is modeled as a Hybrid Automaton, relevant features are enumerated for each, and the methods introduced in Chapter 4 are used to compute feature value ranges for each feature.
- ▷ Chapter 7 concludes the thesis and presents outlines for future problems.

Chapter 2
Background and Related Work

“An investment in knowledge pays the best interest."
-Benjamin Franklin

Much work in the analysis and verification of systems, has been in writing very specific assertions, whose truth or falsity makes or breaks the idea of correctness for that system. In testing for correctness, for complex Hybrid Systems, a list of scenarios are enumerated, and the system is made to execute according to each scenario, the outcome of which determines the safety and correctness of these systems.

In this chapter, we describe well known Hybrid Automata based abstractions for modeling Hybrid Systems, and present a technique for performing reachability analysis of Linear Hybrid Automata. To add a flavour of verification, we introduce assertion languages that are relevant to our work in formally specifying behavioural features for Hybrid Systems. We also list various tools that are widely used in the verification of Hybrid Systems.

2.1. Hybrid Systems and Hybrid Automata

A Hybrid System consists of a discrete program working with an analog environment which consists of a set of real variable entities. The way these real entities evolve describes the behaviour of the environment. We view each run of a Hybrid System as a sequence of discrete steps. Between each step, the system state evolves continuously according to a dynamical law until a transition occurs and the next discrete step is taken. Transitions are instantaneous state changes that separate continuous state evolutions.

2.1.1. Definition of a Hybrid Automaton

We model a Hybrid System as a finite automaton that is equipped with a set of variables. The automaton consists of a set of locations that are labeled with evolution laws. Each location represents the part of Hybrid System that is discrete. Within the location the evolution of the variables is governed by the evolutionary laws labeling that state. Edges between the locations represent discrete transitions and are labeled with a guarded set of assignments. When in a location, if the guard condition for a transition becomes true, the transition is enabled. If an enabled transition is taken, then this translates into executing the guarded assignments which modifies the values of the variables accordingly. Each location is also labeled with an invariant condition that must hold when the control resides at the location. This model for Hybrid Systems can be viewed as a generalization of timed automata [5][25][27].

Definition 2.1. ***Hybrid Automaton***
A Hybrid Automaton H is a collection H = (Q, X, f, Init, Dom, E, G, R), where:

- Q = $q_1, q_2, ...$ is the set of ***discrete states*** *also known as* ***locations;***
- X = $\mathbb{R}^n$ is the set of ***continuous states;***
- f(q,x) : Q $\times X \rightarrow \mathbb{R}^n$ is a ***vector field***. *The vector field (denoted as $\dot{x}$) describes how the continuous state $x \in X$ evolves over time, while in location q;*
- Init $\subseteq$ Q $\times$ X is a set of ***initial states;***
- Dom(q) : $Q \rightarrow P(X)$ is a ***domain***. *This function assigns a set of continuous states,* Dom(q) $\subseteq \mathbb{R}^n$*, to each discrete state $q \in Q$.*
- E $\subseteq$ Q $\times$ Q is a set of ***edges***
- $G(q_i, q_j) : E \rightarrow P(X)$ is a ***guard condition;***
- $R(q_i, q_j, x) : E \times X \rightarrow P(X)$ is a ***reset map***

A state of H *is given as* $(q, x) \in Q \times X$.

□

Initially the system is in one of the initial states (q_{i0}, x_{j0}). The overall composite state then evolves according to the vector field $\dot{x}$ = *f(q,x)*. A vector field is basically a differential equation, wherein $\dot{x}$ describes how variable x evolves. While in a discrete state q, the system remains in state q so long as x remains in *Dom(q)*. When the continuous state x satisfies the guard condition given by *G*$(q, q') \in E$, the discrete state may change its value to q'. At this time when the discrete state change takes place, the continuous state gets reset based on the reset function $R(q, q', x) \in \mathbb{R}^n$. And this process continues.

2.1.2. Modeling systems using Hybrid Automata

2.1.2.1. Nuclear Reactor Example

This example has been taken from [4]. We have a nuclear reactor tank, whose temperature we need to maintain between θ_m and θ_M. We do this, using a system that controls the coolant temperature in the reactor, by moving two independent control rods. When the temperature reaches its maximum value θ_M, the tank must be refrigerated with one of the rods. The temperature rises at a rate v_r and decreases at at rate v_1 and v_2 when rod 1 and rod 2 are respectively inserted. A rod can be moved again, only if T time units have elapsed since the end of its previous movement. The insertion and removal of a rod constitutes a single movement. If the temperature of the coolant cannot decrease because there is no available rod, a complete shutdown is required. Figure 2.1 graphically describes the Hybrid Automaton for this system.

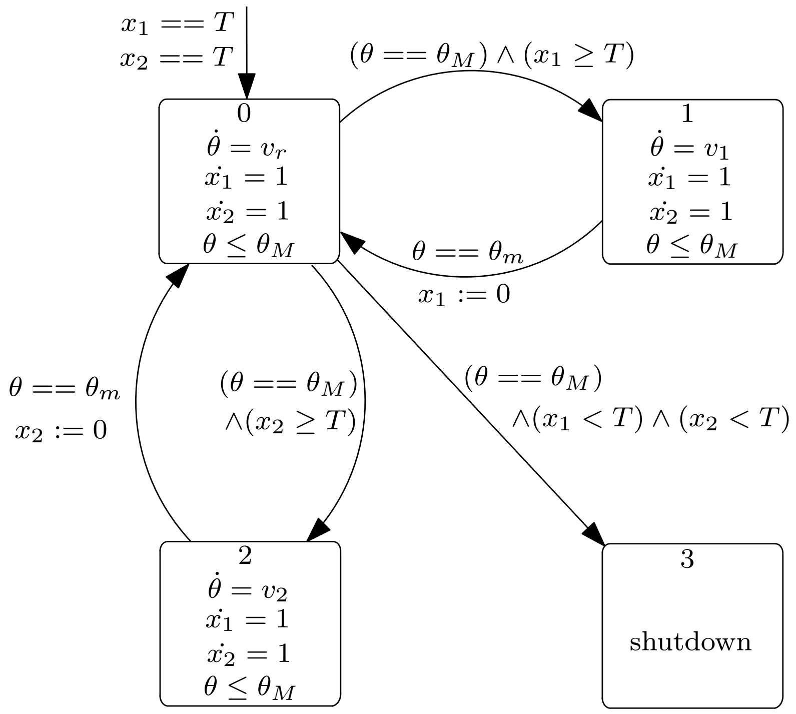


Figure 2.1: Hybrid Automaton for the Nuclear Reactor

The variables are as follows:

- θ indicated the present temperature of the reactor tank.
- x_1 represents the time that elapsed since the last use of rod 1.
- x_2 represents the time that elapsed since the last use of rod 2.
- θ_m represents the minimum temperature constraint

- θ_M represents the maximum temperature constraint

Let us analyze the automaton given in Figure 2.1. Initially, we begin with both rods out, and the timers for both rods set to T. This allows us to move the rods if required. The temperature in the reactor, without either rods inserted, rises at a rate v_r. If the temperature reaches θ_M, and $x_1 \geq T$, i.e. we can insert rod 1, we move to state 1 where the temperature now starts to decrease at a rate of v_1. When the minimum temperature is reached, we reset the timer for rod 1 and move back to state 0, effectively removing the rod from the reactor. Thus T is the time measured since you last pulled the rod out of the reactor. A similar execution is followed for moving into state 2, i.e. we move to state 2, if $x_2 \geq T$ and if the temperature exceeds its maximum. If we can't move into either state 1 or 2, then the reactor must be shutdown.

2.2. Reachability Analysis of Hybrid Systems

If σ and σ' are two states of a Hybrid System H, the state σ' is *reachable* from the state σ, written as $\sigma \overset{*}{\rightarrow} \sigma'$, if there is a run of H that starts in σ and ends in σ'. The *reachability question* is: Is $\sigma \overset{*}{\rightarrow} \sigma'$ true for two given states σ and σ' of a Hybrid System H.

We can approximate the set of states reachable from an initial state σ or from a set of initial states.

According to [13], it is possible to compute the outer boundaries for the set of all trajectories of a dynamic system starting from a given set of initial states. The reachable set over an interval of time $[0,t_f]$ is called a *flow pipe*.

We make the assumption that in every location of the Hybrid System, the variables evolve linearly. Under this assumption it becomes clear that the set of states consisting of n variables, reachable from some initial set of states, can be constrained using n-dimensional hyperplanes. [13] suggests a method for approximating this reachable set. This method is described in Section 2.2.

2.2.1. Basic Flow Pipe Computation Procedure

Consider an autonomous dynamical system with state equation,

$$\dot{x}(t) = f(x(t)) \tag{2.1}$$

in the bounded and connected domain $W \subset R^n$. We assume that the vector field f is *Lipschitz*, that is, there exists a constant L such that,

$$||f(x_1) - f(x_2)|| \leq L \times ||x_1 - x_2|| \tag{2.2}$$

for all $x_1,\ x_2 \in W$. The constant L is called the Lipschitz constant. The Lipschitz condition implies that every initial state x_0 there is a unique solution $x(t, x_0)$ to the state equation.

Given a set of initial states X_0, the set of states reachable at time t is formally given as,

$$R_t(X_0) = \{x_f | x_f = x(t, x_0),\ for\ some\ x_0 \in X_0\} \tag{2.3}$$

Given Equation 2.3, the flow pipe, or the set of states reachable from X_0 during the time interval $[t_1, t_2]$ is defined as,

$$R_{[t_1,t_2]}(X_0) = \bigcup_{t \in [t_1,t_2]} R_t(X_0) \tag{2.4}$$

Given a polytope X_0 of initial state, and a final time t_f, the procedure [13] aims to compute a polyhedral approximation $\hat{R}_{[0,t_f]}(X_0)$ to the exact flow pipe $R_{[0,t_f]}(X_0)$ such that,

$$R_{[0,t_f]}(X_0) \subseteq \hat{R}_{[0,t_f]}(X_0) \tag{2.5}$$

A series of convex polytopes is used to capture the set $\hat{R}_{[0,t_f]}(X_0)$.

The convex polytope $POLY(C, d)$ is defined the pair $(C, d) \in R^{m \times n} \times R^m$ according to,

$$POLY(C, d) = \{x | Cx \leq d\} \tag{2.6}$$

Each row ${c_i}^T$, i = 1,...,m of C represents the normal vector to the i^{th} face of the polytope. For each polytope P, no vertex of P can be expressed as a convex combination of any other two points in P. Let the set of vertices of P be denoted as V(P).

Give a finite set of points, τ ,the convex hull of τ, denoted as $CH(\tau)$, is the smallest convex set (a polytope) that contains τ.

A non-convex flow pipe is approximated by segmenting the flow pipe such that each segment relates to a specific time interval. The k^{th} flow pipe segment corresponding to the time interval $[t_{k-1}, t_k]$ is the set $R_{[t_{k-1},t_k]}(X_0)$.

The approximation must fully contain given flow pipe. Hence the k^{th} segment must be an outer approximation to the k^{th} flow pipe segment. If the approximating polytope corresponds to a matrix-vector pair (C, d), then we want,

$$R_{[t_{k-1},t_k]}(X_0) \subseteq POLY(C, d) \tag{2.7}$$

Given C, to obtain minimal approximation error for a fixed C, d is computed by solving the following optimization problem:

$$\begin{aligned} \min_{d} \ & volume[POLY(C,d)] \\ s.t. \ & R_{[t_{k-1},t_k]}(X_0) \subseteq POLY(C,d) \end{aligned} \tag{2.8}$$

The solution $POLY(C,d^*)$, to the equations 2.8 represents the minimal set denoted by, $S_C^{min}(R_{[t_{k-1},t_k]}(X_0))$. The components of d^* which would be obtained solving 2.8, can be found by solving the following constrained optimization problems for i=1,...,m.

$$\begin{aligned} \max_{x} \ & {c_i}^T x \\ s.t. \ & x \in R_{[t_{k-1},t_k]}(X_0) \end{aligned} \tag{2.9}$$

The constrained problem shown above tries to determine the components of d^* by ensuring that every valuation of variable x in the flow pipe segment $R_{[t_{k-1},t_k]}(X_0)$ in contained in the polytope.

From the definitions given above, the optimization problem 2.9 can be rewritten as,

$$\begin{aligned} \max_{x_0,t} \ & {c_i}^T x(t,x_0) \\ s.t. \ & x_0 \in X_0 \\ & t \in [t_{k-1}, t_k] \end{aligned} \tag{2.10}$$

Proposition 2.1. *Let ($x^*_{0,i}$,t^*_i) be the solution to 2.10 for i=1,...,m. The solution to 2.8 is given by $d^*_i = c_i^T x(t^*_i, x^*_{0,i})$, for i=1,...,m.*

A proof is provided in [13].

To solve 2.10, one needs to solve the state equation to find $x(t,x_0)$ for each t and x_0. The solution $x(t,x_0)$ can be computed numerically using an ordinary differential equation (ODE) solver.

Choosing a Set of Direction Vectors Previously, the assumption was that we had the set of normal vectors C. To compute the set of normal vectors, that is, the rows of the matrix C, for each flow pipe segment, the heuristic that is used is described below.

Let $V_{t_{k-1}}(X_0)$ and $V_{t_k}(X_0)$ be the sample points at times t_{k-1} and t_k along the flow pipe trajectory starting with the vertices of X_0; that is:

$$V_t(X_0) = \{x(t,v) | v \in V(X_0)\}. \tag{2.11}$$

Starting at each vertex of X_0, using simulation, each point in the above set can be obtained. Using these points a convex hull is formed, which serves as an initial approximation to the flow pipe segment $R_{[t_{k-1},t_k]}(X_0)$, denoted by

$$\phi_{[t_{k-1},t_k]}(X_0) = CH(V_{t_{k-1}}(X_0) \cup V_{t_{k-1}}(X_0)). \tag{2.12}$$

This step is illustrated in the figure 2.2.

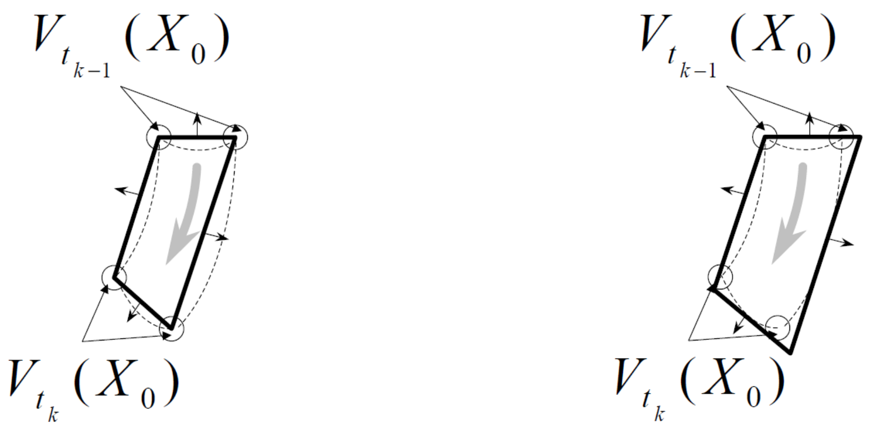


Figure 2.2: Flow Pipe Approximation

The dashed lines indicate the boundary of the flow pipe segment. For simplicity purposes, there are two vertices in X_0. Hence the initial set X_0 is a line segment. Simulation begins with the vertices of X_0, and sets $V_{t_{k-1}}(X_0)$ and $V_{t_k}(X_0)$ are computed. The convex hull $\phi_{[t_{k-1},t_k]}(X_0)$, drawn with thick lines, is then constructed from these points. As illustrated in Figure 2.2, $\phi_{[t_{k-1},t_k]}(X_0)$ may not contain the flow pipe segment $R_{[t_{k-1},t_k]}(X_0)$. Let (C_ϕ, d_ϕ) be the matrix pair defining $\phi_{[t_{k-1},t_k]}(X_0)$, i.e.

$$\phi_{[t_{k-1},t_k]}(X_0) = POLY(C_\phi, d_\phi). \tag{2.13}$$

The normal vectors to the faces of this convex hull obtained and then used as the set of direction vectors to compute the minimal convex set containing the flow pipe segment.

2.3. Tools for Reachability Analysis

Over the past decades, a number of efforts have been made to develop tools that implement verification techniques for Hybrid Systems. These tools [9, 11, 10, 19, 37] work on Linear or Affine Hybrid Automata and produce a flow pipe output as a set of linear constraints or vertex corners of polyhedra that contain the reachable region.

The tool *UPPAAL* [11] can be used to verify properties of Linear Hybrid Automata. One needs to manually re-write the Hybrid Automaton as a set of concurrent Timed Automata [5], which are then fed to UPPAAL via its graphical user interface.

A MATLAB plug-in, *CheckMate* [37], was developed for modeling, prototyping and simulating specific situations and formally verifying Hybrid Systems that have constant, linear or non-linear dynamics. Dynamics of locations are modeled using Switched Continuous System blocks of MATLAB. CheckMate, was tested and found to be very unstable in our work and was hence discarded.

A different group of scientist published a tool *HyTech* [24]. HyTech has a very easy to use text based input language which can be used to express linear Hybrid Automata. It also allows users to ask reachability questions, and can print linear constraints representing regions that are reachable. HyTech however can work on only Linear Hybrid Automata. There have been many challenges faced while using HyTech, one of the major ones includes the overflow problem. A detailed discussion of the issues facing its use are detailed in [22].

We finally settled on the *Polyhedral Hybrid Automaton Verifier (PHAVer)* [19]. PHAVer addresses some of the problem faced by HyTech. It uses more accurate mathematical libraries to prevent the overflow problem, and supports the use of affine dynamics. The tool has a syntax that is similar to that of HyTech and allows us to ask reachability questions. PHAVer also support a variety of set operations and various configuration parameters to manage the over-approximation during flow-pipe computation. The output of PHAVer can be either a set of location names and linear formulas, or a sequence of linear constraints in floating point form, or linear formulas as a sequence of vertices in floating point form. We prefer the use of the third output alternative. The syntax and usage of PHAVer can be found in [20].

We demonstrate the tool PHAVer on an example Hybrid Automaton for an Oscillator [18]. PHAVer performs a reachability analysis on the Hybrid Automaton. The analysis produces an approximation to the flow pipe. A graphical representation of the flow pipe approximation, as a result of running PHAVer is shown in Figure 2.3.

2.4. Assertion Languages for Hybrid Systems

Systems are designed to meet the designers specifications. Verification engineers use assertion languages to formally specify properties that express the designers intent. Assertion languages exist to alleviate the problems that result from incorrect interpretation of specifications written in natural languages such as English. The verification process then tries to validate whether the system meets the given specifications. Expressing properties using formal languages also allows for construction of automated property checker. This section reviews some well known formal property specification languages that form the backbone of the feature

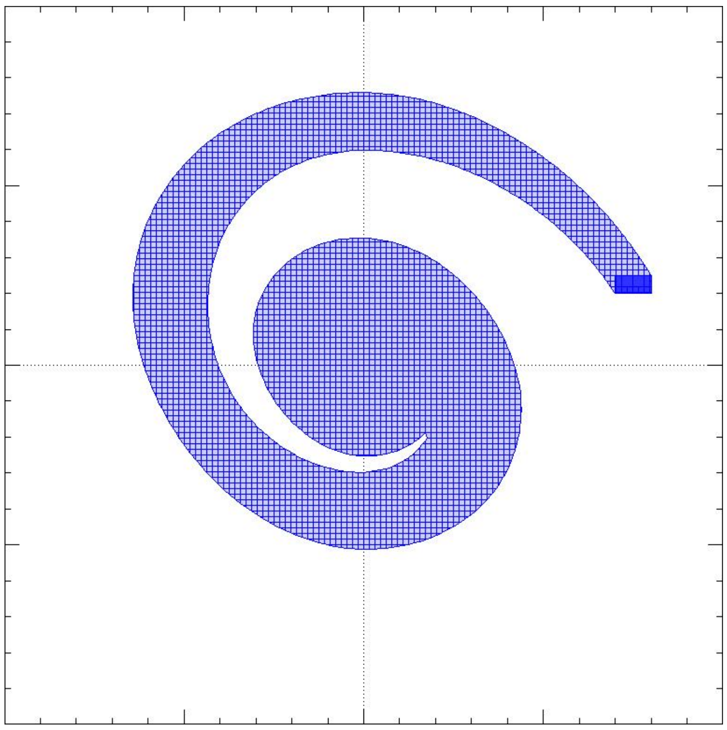


Figure 2.3: Flow pipe approximation for the PHAVer Oscillator Demo

specification language presented in Chapter 3.

Systems that have a basis in temporal domain require languages that allow us to relate the truth of propositions over different time instances. Languages that assert properties over time have their basis in Linear Temporal Logic (LTL) [35]. LTL which is also referred to as Propositional Temporal Logic (PTL) [21], can be used to assert properties over a sequence of states; a path. In order to express properties for systems in which a state can have more than one possible next state, LTL was extended to create branching-time logics such as Computational Time Logic (CTL) [15]. Branching-time logics are based on the notion that at a state, multiple next states are possible, representing multiple possible computations that begin at that state. These logics were then included in industry standards such as SystemVerilog Assertions (SVA) [1] and Property Specification Language (PSL) [2].

LTL syntax is defined over a set of *atomic propositions*, $\mathcal{AP}$, as follows:

$$\varphi := true \mid a \mid \neg\ \phi \mid \varphi_1\ \wedge\ \varphi_2 \mid \varphi_1\ \vee\ \varphi_2 \mid \varphi_1\,\mathcal{U}\,\varphi_2 \mid\ \bigcirc\varphi$$

The operators $\bigcirc$ and $\mathcal{U}$ represents the *next-state* and *until* operators respectively. $\bigcirc\varphi$ is true iff φ is true in the next state. $\varphi_1\,\mathcal{U}\,\varphi_2$ is true iff φ_1 is true from the current state upto the state where φ_2 is true. These operators can be extended by the use of the $\Box$ (globally true) and $\Diamond$ (true sometime in the future) operators. $\Box\varphi$ is true iff φ is true in all states, and $\Diamond\varphi$ is true iff φ is true at some state in the future. We can express them in terms of the more basic $\mathcal{U}$ operator as follows:

$$\begin{aligned}\Diamond\varphi &\equiv \mathit{true}\,\mathcal{U}\,\varphi \\ \Box\varphi &\equiv \neg(\Diamond\,\neg\varphi)\end{aligned}$$

Assertion languages like LTL are based on a clocked model of a system, and can be used to write properties for clocked digital systems that have boolean signals. Efforts have been made to relax the semantics of linear temporal logic operators in Metric Temporal Logic (MTL) [7], allowing temporal operators to have time bounds. Further, in order to write properties over dense time, LTL was extended to Metric Interval Temporal Logic [6]. This allows us to express properties such as:

$$\Box(\Diamond_{[0,5]}x \Rightarrow \Diamond_{[6,7.5]}y)$$

Temporal logics were made more expressive in Timed Propositional Temporal Logic (TPTL) [8] by including temporal variables that could be bound to a time instant in the local temporal context of the instant and used in a later part of the temporal formula. In [29], an extension of MITL [6], called Signal Temporal Logic (STL), was proposed which allows PORVs along with the dense timed temporal operators.

The formal definition of PORVs is given below:

Definition 2.2. ***Predicate over Real Variables***: *If the set of continuous variable is denoted by X=$\{x_1,\ x_2,\ ...,\ x_n\}$, then a Predicate over Real Variables, p_a, may be defined as, $p_a ::= f(x_1,\ x_2,\ ...,\ x_n)\ \sim 0$, where f is a mapping, $f : \mathbb{R}^n \rightarrow \mathbb{R}$, and $\sim$ is a relational operator such that $\sim\ \in \{>, \geq\}$. Throughout this thesis, we consider f to be a linear map.*

$\Box$

Other relational operators like $<, \leq$, and $=$ are derived from $\sim$ and the propositional connectives.

The interested reader may refer to [29] for a comprehensive discussion of the syntax and semantics of STL formulae.

Example properties that are expressed using STL are presented below:

Example 2.1. *If enable is low, V_{out} will be less than $0.2V$ within 10μs, is expressed by the following STL safety property:*

$$\neg \mathit{enable} \Rightarrow \Diamond_{[0,10^{-6}]}(V_{out} < 0.2)$$

While STL was being developed, the research group at IIT-Kharagpur developed a similar logic, AMS-LTL [30], which like STL extended MITL with PORVs and cross-events. The inclusion of a formalism to specify cross-events in AMS-LTL is what differentiates it from its sibling STL. A property written in AMS-LTL is shown in the example below:

Example 2.2. *The property If enable is low, V_{out} will cross $0.2V$ within 10μs, is expressed by the following AMS-LTL safety property:*

$$\neg enable \Rightarrow \Diamond_{[0,10-6]} @^{+}(V_{out} < 0.2)$$

In the example the $@^{+}(V_{out} < 0.2)$ indicates the point at which V_{out} becomes less that $0.2V$, and is termed as a cross-event, recognizing that the event when signal V_{out} crosses $0.2V$.

Assertion languages like SystemVerilog [1] work with discrete time semantics, wherein events are checked at clock boundaries. SystemVerilog adds the notion of *local variables* to the logic. Local variables allow binding of uninterpreted variables across time domains. For example, consider the following property:

```
property LOOPBACK;
    int loopData;
    (posedge clk)
    (RTS && CTS, loopData=TX) |-> ##[1,100] (RX == loopData);
endproperty
```

The property says that, at each posedge of the clock when the Request to send(`RTS`) and Clear to send (`CTS`) signals are asserted, capture the data on the transmission line (`TX`) in the local variable `loopData` and asserts that the data on the receive line(`RX`) within the next 1 and 100 cycles is what was transmitted. Here, `loopData` is an uninterpreted variable. To verify the property, the simulator binds the data transmitted to `loopData` when `(RTS && CTS)` evaluates to `true`, so that it can be compared with the received data during the next 99 clock cycles. In SVA, a boolean expression over propositional variables are used trigger assertion matching. However, in the domain of Hybrid Systems, we have continuously evolving variables, that need to be monitored when certain thresholds are crossed. The specification of cross events, is not supported by SVA.

Extensions to SVA and the Open Verification Library (OVL) have been suggested in [34], that incorporate PORVs into existing syntax for applications for simulation environment that support SVA/PSL and those that don't. We first look at the syntax of AMS-LTL, for specifying properties over PORVs and cross-events.

Definition 2.3. ***Syntax of AMS-LTL***: *The syntax of an AMS-LTL formula φ is defined recursively by the following grammar rules.*

$$\varphi ::= true \mid p \mid E \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1\, \mathcal{U}_{\mathcal{I}}\, \varphi_2$$

E is defined as follows:

$$E ::= @^+(B) \mid @^-(B) \mid @(B)$$
$$B ::= p \mid \neg B \mid B_1 \wedge B_2$$

where, $p \in \mathcal{AP} \cup \mathcal{AP}_A$.

□

Definition 2.4. ***Time Interval***: *A Time Interval I is a non-empty convex subset of $\mathbb{R}_{\geq 0}$ and is expressed in the forms (a,b), [a,b], (a,b], [a,b), where $a, b \in \mathbb{R}_{\geq 0}$ and $b > a$. The left and right end points of I are denoted by l(I) = a and r(I) = b respectively.*

In this interval notation, for $t, t' \in \mathbb{R}_{\geq 0}$, (I+t) denotes the interval $\{t' + t | t' \in I\}$. Similarly (I-t) denotes the interval $\{t' - t | t' \in I \text{ and } t' - t \geq 0\}$.

□

Definition 2.5. ***Timed State Sequence***: *A state sequence is an infinite sequence of states, $\mathcal{S} = s_0, s_1, \ldots$ such that $s_i \subset \mathcal{AP} \cup \mathcal{AP}_A$. An interval sequence $\mathcal{I}$ is an infinite sequence of intervals $\mathcal{I} = I_0, I_1, \ldots$, where I_i is left closed and right open $\forall i \in \mathbb{N}_{\geq 0}$, and $\mathcal{I}$ has the following three properties.*

- ***Initiality:*** *For I_0, $l(I_0) = 0$.*
- ***Progress:*** *For every $t > 0$, there exists and interval I_i such that $t \in I_i$.*
- ***Adjacency:*** *For every $i \geq 0$, I_i and I_{i+1} are adjacent to each other.*

A timed state sequence is a pair $\tau = (\mathcal{S}, \mathcal{I})$ such that for each time, $t \in I_i$, the state is s_i.

□

Definition 2.6. ***Semantics of AMS-LTL***: *For an AMS-LTL φ and a timed sequence $\tau = (\mathcal{S}, \mathcal{I})$, the satisfaction relation $\tau \models \varphi$ is identical to that of STL with the following additional semantics of events.*

- $\tau^t \models @^+(b)$, *iff* $\exists I_i$ *such that* $l(I_i) = t$, $\forall t' \in I_i$, $\tau^{t'} \models b$, *and* $\forall t'' \in I_{i-1}, \tau^{t''} \not\models b$

- $\tau^t \models @^-(b)$, *iff* $\exists I_i$ *such that* $l(I_i) = t$, $\forall t' \in I_i$, $\tau^{t'} \not\models b$, *and* $\forall t'' \in I_{i-1}, \tau^{t''} \models b$
- $\tau^t \models @(b)$, *iff* $\tau^t \models @^+(b)$ *or* $\tau^t \models @^-(b)$

□

Inspired by the use of boolean variables in the assertion syntax of SVA [1], AMS-LTL was extended to AMS-LTL^L with real-valued local variables. Based on AMS-LTL^L, further extensions have been proposed in [31] that augment AMS-SVA [34] with local variables that can capture real valued signals, thus extending SVA into the analog domain.

Definition 2.7. ***Syntax of AMS-LTL^L:*** *An $AMS-LTL^L$ subformula φ_l involving a single local variable follows the grammar:*

$$\varphi_l ::= (E, x_l \leftarrow z) \wedge \varphi(\mathcal{AP} \cup \mathcal{AP}_A \cup \mathcal{AP}_L) \mid (E, x_l \leftarrow z) \Rightarrow \varphi(\mathcal{AP} \cup \mathcal{AP}_A \cup \mathcal{AP}_L)$$

Here E is an event on trace S, $z \in X$, $\mathcal{AP}$ is the set of Boolean variables, and $\mathcal{AP}_A$ is a set of PORVs over X, where X is the set of continuous variables. $x_l \leftarrow z$ denotes assignment of the current value of z to the local variable x_l. $\mathcal{AP}_L$ is a set of PORVs over $X \cup \{x_l\}$. φ is an AMS-LTL^L formula over $\mathcal{AP} \cup \mathcal{AP}_A \cup \mathcal{AP}_L$.

□

We have already seen in Section 2.1, that a Hybrid System can be described using a Hybrid Automaton. The state of a Hybrid Automaton is written as:

$$< q_t, [\eta(x_1),\ \eta(x_2), ...,\ \eta(x_n)]_t >$$

where $x_t = [\eta(x_1),\ \eta(x_2), ...,\ \eta(x_n)]_t$ is the valuation of the control variables of the Hybrid Automaton while in location q_t at time t. A *run* of the Hybrid Automaton is a timed sequence of states $< q_t, x_t >$.

The discrete location is an important factor while writing assertions and forms part of the state of the Hybrid Automaton. Languages such as AMS-LTL^L only express a match of a sequences of states between different variable valuation combinations. However, they do not take Hybrid System locations into consideration. [33] extends the AMS-LTL^L specification, by augmenting the system over which properties are written using Auxiliary State Machines. An *Auxiliary State Machine* captures the design intent in the form of abstract states with guarded transitions allowing movement between states. The assertions written in this new language, capture specific behaviours of the AMS circuit and form the basis of features that were introduced in [17].

[17] suggests formalisms for specifying feature based equivalence for AMS circuits. The specification language suggested is based on AMS-LTL^L that is extended with the use of Auxillary State Machines [33][32]. In this research, the language described in [17] is adapted to our requirement of establishing feature based equivalence between Hybrid Automata.

2.5. Automated Hybrid Automaton Parameter Extraction

Most systems exist as variations of similar topologies. For example: multiple battery charger controllers may exist, however most follow the same constant-current, constant-voltage cycle, varying only in specific application related parameters [39]. A few other topologies may exist, for instance, a pulse-mode battery charger [12].

[3] proposes the use of a tool called CHASIS for Power Management Units that can extract exactly those parameters that vary across the circuit family. Given a circuit netlist, and a test-bench that drives the circuit into desired regions of operation wherein required parameters can be extracted, CHASIS is able to extract distinguishing parameters. CHASIS is designed to work with Linear Dropout Voltage Regulators and Buck Converters. It generates the behavioral model of a PMU component by creating a parameterized skeleton model of the component from a pre-written template. CHASIS supports the estimation of two broad categories of parameters:

1. *Specified parameters*: are those given in the design specification. Examples include bias currents, reference voltages etc.

2. *Unspecified parameters*: are those parameters that are not readily available with the designer. Examples include the shutdown resistance of the LDO.

We propose a tool-flow that can use a tool like CHASIS to extract parameters from AMS circuits. We simultaneously maintain a repository of Hybrid Automaton skeletons fully fitted with location activities, but having parameters that are free. The parameters obtained from the CHASIS-like tool can then be fitted into a Hybrid Automaton skeleton, giving us a fully functional Hybrid Automaton model of the circuit. This model can then be used for further feature based analysis.

Chapter 3
Feature based Abstract Interpretation

"If we knew what it was we were doing,
it would not be called research, would it?"
-Albert Einstein

It is most natural for the human mind to think of systems in terms of the behaviours it portrays. It is also intuitive to make statements about what to expect from complex systems in terms of their behaviours. In this chapter, system behaviours are presented as features; described qualitatively as a pattern that matches when the system exhibits a specific behaviour, and quantitatively as a range of values that represent the feature signature. We also study how features affect standard reachability analysis in Hybrid Systems, and ask the question, can we piggy-back feature computation over flow-pipe analysis of Hybrid Automata.

3.1. Feature Specification: Syntax and Semantics

This Section presents a summary of the syntax and semantics of the proposed language formalism for specifying features. A significant portion of the syntax has been borrowed from SystemVerilog Assertions (SVA), for which the detailed syntax and semantics can be found in [1].

The basic syntax of a feature definition is as follows:

```
feature <feature-name> ( <list-of-parameters> );
begin
    var <list-of-local-variables>;
    <sequence-expr>
          |-> <feature-name> = <feature-expr> ;
end
```

The definition of `sequence-expr` is very similar to sequence expressions of SVA with the following restrictions and annotations:

1. We do not support the repetition operator of SVA in the present version.

2. In SVA all signals are Boolean. AMS extensions of assertion languages [34] [29] support the notion of a *predicate over real variable* (PORV). For example `(M.Vout > 0.1*Vrated)` is a PORV used in the example presented earlier.

3. PORVs can be prefixed with the `@+` and `@-` operators to indicate the positive and negative crossings. These are similar to cross events of Verilog-AMS, and the `$rose` and `$fell` constructs of SVA. For example, `@+(M.Vout > 0.1*Vrated)` specifies the positive crossing of the PORV, `(M.Vout > 0.1*Vrated)`.

4. Local variables can be assigned in `sequence-expr` in the same way as in SVA, but the values assigned can be used only in `feature-expr`. In `sequence-expr`, local variables can only be assigned the values of real variables used in the PORVs. Therefore we use local variables only for the purpose for feature computation, not for the purpose for which they are used in SVA.

5. The feature value is computed using `feature-expr` which is a linear arithmetic expression over the local variables assigned in `sequence-expr` and other constants.

6. Parameter values are treated as constants everywhere.

The semantics of `sequence-expr` is similar to the AMS extension of SVA presented in [34], except that time intervals are treated as dense. Each match of the sequence described by `sequence-expr` yields a valuation of the feature obtained by computing `feature-expr`.

The formal interpretation of features over Hybrid Automata models gleaned from AMS models/designs is the main subject of this thesis.

A match of a given `sequence-expr` on a run of the Hybrid Automaton is defined as the timed sequence of states with valuations of discrete and continuous variables that together satisfy the chain of PORVs and events specified in the `sequence-expr`.

A run of the Hybrid Automaton, H, may have one or more matches of `sequence-expr` defined in a feature, F. Each match yields a feature valuation, which is a real number obtained by computing `feature-expr` corresponding to that match. The range of valuations of F on H is the interval $[F_{min}, F_{max}]$, where F_{min} and F_{max} are the minimum and maximum valuations of F among all matches of `sequence-expr` on all runs of H.

Given a Hybrid Automaton, H, and a feature, F, our goal is to determine the range of values of F over all runs of H. Abstract interpretation and reachability analysis techniques have been extensively used [23, 4, 13] to find the (possibly overapproximated) set of reachable states of a Hybrid Automata. For simple features, it is possible to define the feature as a function of the variables of H, and then add them as additional variables into H, so that abstract interpretation yields the feature range. For more complex features, we can reduce the feature range computation problem to the abstract interpretation problem through a non-trivial set of transformations. Chapter 4 presents an outline of the proposed transformations, along with their correctness proofs; and Chapter 6 demonstrates the use of these transformations with examples. Appendix A describes the feature specification syntax.

3.2. Evaluating Feature Values over Flow Pipe Approximations

We express a feature expression, `feature-expr`, as given in Section 3.1, as a linear combination of the system's variables. We obtain the flow pipe for the variables that define the feature using a method for appropriately over-approximating the flow pipe, for example, using the method described in [13]. Currently the tool *PHAver* is being used to compute the flow pipe for the Hybrid Automaton [19]. This flow pipe helps constrain the range of values the variables can take.

We express the equivalence relation between two Hybrid Automata as a predicate over real variables [29].

Lemma 3.1. *Given, a linear function $f(x_1, x_2, x_3, ..., x_n)$ defined over n variables, and m linearly independent constraints that form a convex region over the same n variables, then, f takes values (within the constrained convex region), that have a maximum and minimum defined at two corner points of the convex region defined by the constraints.*

Proof. Let us begin with the assumption that there are two variables x and y. m linear constraints exist defined in terms of x and y. The i^{th} constraint is of the form,

$$a_i.x + b_i.y \leq c_i \tag{3.1}$$

where $a_i, b_i, c_i \in \mathbb{R}$.

We also have a function $f(x, y) \subseteq \mathbb{R}$. We need to maximize or minimize the function over the constrained domain. The function f has the following form.

$$f(x, y) = a.x + b.y \tag{3.2}$$

where $a, b \in \mathbb{R}$.

We observe that the constraints represent lines. Since the constraints are pairwise linearly independent, each pair of lines intersect. The convex region defined by the constraints has a boundary that consists of a set of line segments connected to each other at corners.

If we observe the function f, we can determine the slope of the line. The slope of the line is given as $\frac{-a}{b}$.

Let us write the equation of this line as,

$$y = \frac{-a}{b} \times x + c \tag{3.3}$$

which can be re-written as,

$$y + \frac{a}{b} \times x = c \tag{3.4}$$

Here c represents the y intercept of the line.

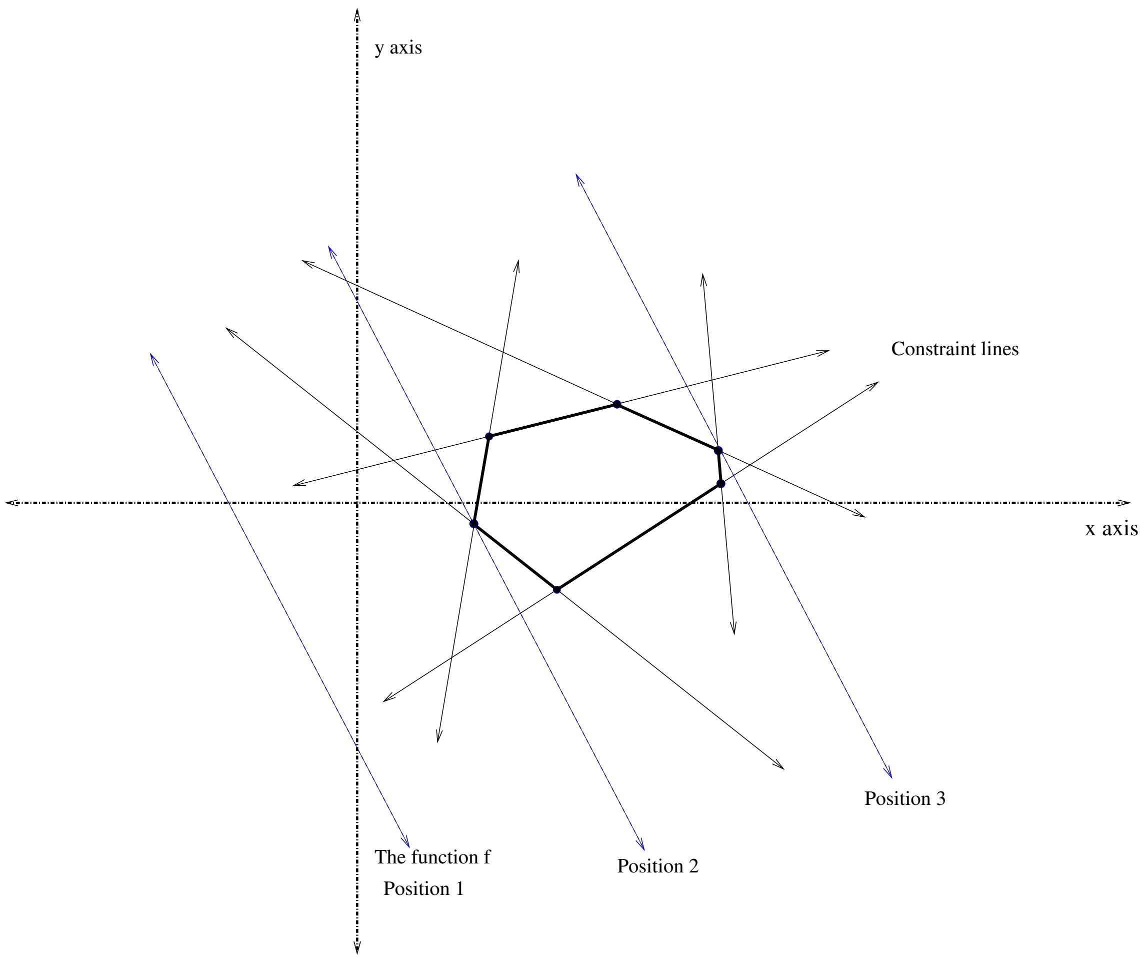


Figure 3.1: Graphical visualization of the function f, and the set of constraints.

In Figure 3.1, the highlighted boundary represents the constrained domain, having corner points clearly marked with heavy circles as the intersections of the constraints. In the figure, the slope $\frac{-a}{b}$ is shown to be negative. An identical argument can be made when the slope is positive or zero. The example used in the figure is presented for illustration purposes only.

As can be seen in Figure 3.1, the line with slope $\frac{-a}{b}$ can be arbitrarily positioned. Initially we assume that the line is at Position 1 as shown in the figure. To maximize $f(x, y)$ in Equation 3.2, we are effectively maximizing the y-intercept, c, in Equation 3.4. If we sweep the line corresponding to the slope $\frac{-a}{b}$, from left to right, across the space, we observe that the line enters the constrained region at a corner point, i.e at Position 2. As the line is moved towards the right, it exits the constrained region at another corner point, at Position 3. Observe that when the line is at Position 2, the y-intercept is minimum. As the line moves right, the y-intercept increases until the line reaches Position 3. At this position the y-intercept is maximum among y-intercepts for all alternative positions of the line within the constrained region. Thus the minimum and maximum are seen at the corner points of the constrained space.

If any constraint has the same slope as the line corresponding to f, then the line corresponding to the constraint is parallel to that corresponding to f. If one of the corner points corresponds to either one extreme (maximum or minimum), then every point along the line segment that is part of that constraint line, will also define the same extreme value.

When the constraints are defined over three variables, the constraints correspond to planes in three dimensional space. The function f defined over the three variable, $f(x_1, x_2, x_3)$, also corresponds to a plane. The normal to this plane is known from the equation of the function f. We maximize or minimize the perpendicular distance of the plane from the origin of the co-ordinate space of x_1, x_2, x_3, to maximize or minimize f respectively. Similar to the above case, as the plane sweeps across the space, the plane enters the constrained convex region at some corner point or along some corner edge of the region and similarly leaves at some corner point or edge of the same region.

We can extend this idea to higher dimensions. In higher dimensions, the constraints will determine the position of hyperplanes, and the function f will define a hyperplane H, whose exact position will depend on the variable independent constant.

In general it is known [26] that geometrically, a hyperplane in $\mathbb{R}^n$ is a set of the form

$$H = \{x : a^T x = b\} \tag{3.5}$$

where $a \in \mathbb{R}^n, a \neq 0$, and $b \in \mathbb{R}$. The position of the hyperplane in space is given by the direction of the vector a, and the amount of the displacement along the direction a, is given by b. Precisely, $|b|$ is the length of the closest point, x_0, on H from the origin, and the sign of b determines if H is away from the origin, along the direction a or $-a$. As we increase the magnitude of b, the hyperplane is shifting further away along $\pm a$, depending on the sign of b. Hence, similar analogies to planes can thus be extended to hyperplanes as well, wherein the factor being maximized or minimized is b.

In conclusion, given a set of m linear constraints in n-variables, and a function $f(x_1, x_2, x_3, ...)$, we see that the maximum and minimum of f, in the constrained region, is always found to lie at two corner points of the constrained region. □

Lemma 3.2. *Given a Hybrid Automaton $H = (Q, X, f, Init, Dom, E, G, R)$ with linear dynamics, and a feature whose signature is defined as a linear equation over the real variables of the Hybrid Automaton, the maximum and minimum of the range of values corresponding to the signature of the feature, can be correctly computed from the flow pipe for the system variables that define the feature.*

Proof. Given a set of initial states, flow pipe analysis of Hybrid Automata can be performed using the method suggested in [13]. The flow pipe consists of a set of constraint equations. These constraints are expressed in terms of the system variables, thus describing the part of the state space that is reachable from the set of initial states. These constraints are linear constraints.

The linear equation describing the feature signature is given. From Lemma 3.1, the linear constraints can be used to obtain the minimum and maximum of the range of this signature.

Thus we can correctly compute the maximum and minimum range of values corresponding to the signature of the feature directly from the flow pipe of the system variables that define the feature. □

Theorem 3.1. *Given two Hybrid Automata $H_1 = (Q_1, X_1, f_1, Init_1, Dom_1, E_1, G_1, R_1)$ and $H_2 = (Q_2, X_2, f_2, Init_2, Dom_2, E_2, G_2, R_2)$, feature equivalence can be determined using the traditional flow pipe analysis for Hybrid Automata.*

Proof. Let f_1 and f_2 be the feature signatures corresponding to feature F for the Hybrid Automata H_1 and H_2 respectively.

Using the method specified in Lemma 3.2, we can obtain the range of the feature signatures f_1 and f_2. The equivalence between the two Hybrid Automata is defined based on the feature definition. We express the equivalence between the two automata using the feature signatures, as the predicate,

$$|f_1 - f_2| \le threshold \tag{3.6}$$

$$-threshold \le f_1 - f_2 \le threshold \tag{3.7}$$

where $threshold \in \mathbb{R}$ is a threshold value.

Since f_1 and f_2 are given as ranges, we have the minimum and maximum of f_1 and f_2 given as $f_{1_{min}}$, $f_{1_{max}}$ and $f_{2_{min}}$, $f_{2_{max}}$ respectively. Hence the equivalence relation can be expressed more succinctly as the conjunction of the predicates,

$$(f_{1_{max}} - f_{2_{min}}) \leq threshold \tag{3.8}$$

$$(f_{2_{max}} - f_{1_{min}}) \leq threshold \tag{3.9}$$

If the conjunction of the above predicates evaluates to true, then it can safely be ascertained that the two Hybrid Automata H_1 and H_2 are equivalence with respect to the feature F. $\square$

3.3. Concluding Remarks

The syntax used for specifying features is described in Section 3.1. The feature syntax is an extension of AMS-LTL^L [31] and SVA [1], and introduces the notion of a feature signature, evaluated over a match of an AMS-LTL property. Section 3.2 brings feature computation in the context of flow-pipe based reachability analysis, and shows why it is possible to compute the feature signature over a flow pipe represented as a convex polyhedron.

Chapter 4
Feature Driven Modifications to the Hybrid Automaton

"Science is a way of thinking
much more than it is a body of knowledge."
-Carl Sagan

Given an abstraction of an AMS circuit model as a Hybrid Automaton, a reachability analysis of the model is easy to perform using readily available tools such as Checkmate [37] and PHAver [19]. During this research, the tool PHAver has been used to express the Hybrid Automaton and to extract the flow pipe.

In order to focus our analysis on specific features of the system, we need to make certain modifications to the Automaton to take advantage of the algorithms used to extract the flow pipe. The modifications, do not in any way modify the behaviour of the original abstraction. The modifications augment the Hybrid Automaton with additional locations and transitions that help focus the reachability search on the behavioural aspects concerning the features of interest.

We broadly consider the following attributes of features:

1. **Value attributes**. We consider the relation between the propositions/PORVs in the property and the invariants/guards in the Hybrid Automaton. We further consider two possible types of such attributes, namely:

   Type-1 : PORVs and propositions which are either true at all states in a location or false at all states in the location.

   Type-2 : PORVs that are not location invariants. Overlaying the computation of such features over abstract interpretation can be achieved by splitting of locations.

2. **Temporal attributes**. We consider the nature of time dependency of the feature. Specifically we consider two possible types under this head, namely:

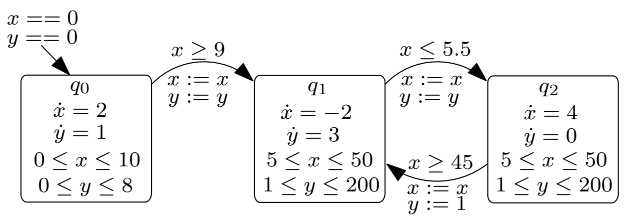


Figure 4.1: Hybrid Automaton H1

Type-3 : Features that are concerned with computing the time between two events.

Type-4 : Features that are defined over the valuations of state variables at two or more points in time.

A feature may contain any combination of the above attributes. The presence of each attribute is handled with a specific type of transformation, which is presented in the remainder of this chapter. For the sake of simplicity we demonstrate the handing of each attribute type in isolation.

4.1. Handling Type-1 Attributes

Consider the Hybrid Automaton, $H1$, in Figure 4.1 and the following feature:

```
feature valueX();
begin
    H1.state == q1 || H1.state == q2
          |-> valueX = H1.x ;
end
```

The feature aims to find the range of the variable x when $H1$ is in the location $q1$ or $q2$. Given such a feature, we use the standard flow pipe analysis for Hybrid Automata [4, 13, 19], to derive the reachable region for each location of $H1$. Given the convex nature of the polyhedra computed by this analysis, it suffices to compute the range of x over the corner points of the polyhedra at the locations $q1$ and $q2$. In our tool flow, we use PHAver [19] to compute the polyhedra of reachable states at $q1$ and $q2$, and then compute the range of x over the corner points of the polyhedra. It is easy to see that the same approach works well for Type-1 attributes where `feature-expr` is linear.

4.2. Handling Type-2 Attributes

If a feature contains a PORV which is not a location invariant, then we shall split the location into states that satisfy the PORV and states that do not satisfy the PORV. For example, consider the following feature over the same hybrid automaton, $H1$, of Figure 4.1:

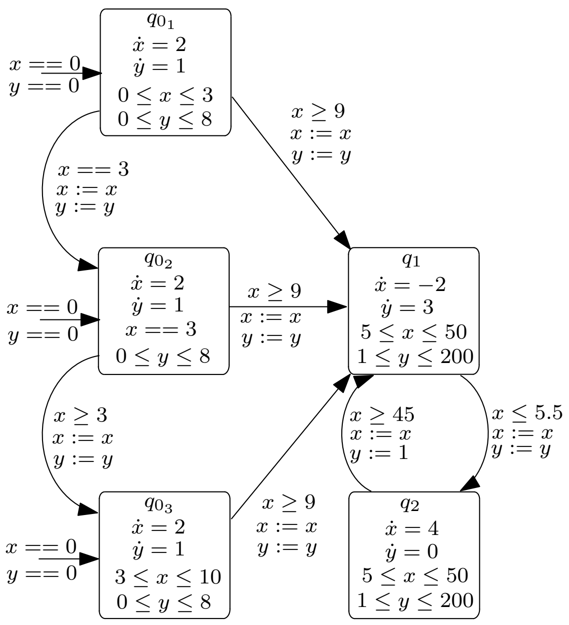


Figure 4.2: Splitting Hybrid Automaton H1

```
feature yValue(c);
begin
    ((H1.state == q0) && (H1.x >= 3))
          |-> yVvalue = H1.y ;
end
```

The feature computes the value of y when x crosses 3 while the automaton is in location $q0$. The expression: `((H1.state == q0) && @`$^+$`(H1.x >= 3))` is false at all states of all locations other than $q0$, and therefore we consider splitting the location $q0$ only. Since we are concerned with those states at which the crossing of `(H1.x >= 3)` takes place, we split location $q0$ into three locations, namely a location $q0_1$ where $x < 3$, a location $q0_2$ where $x = 3$, and a location $q0_3$ where $x > 3$. The resulting Hybrid Automaton is shown in Figure 4.2, and we seek the reachable states under the location $q0_2$.

We apply algorithms for the feature patterns given below:

1. $HA_i.state = loc1,\ condition_1,\ var\_assignment_1\ \#\#[0,\$]$
   $HA_i.state = loc2,\ condition_2,\ var\_assignment_2$
   $\mapsto feature\_value = expr$

2. $HA_i.state = loc1,\ condition_1,\ var\_assignment_1\ \#\#W$
   $HA_i.state = loc2,\ condition_2,\ var\_assignment_2$
   $\mapsto feature\_value = expr$

In the algorithms and definitions used to describe the transformations, we use the following notations for convenience:

1. loc_1 and loc_2 refer to the locations in the sequence expressions.

2. *condition* refers to either $condition_1$ or $condition_2$.

3. $PORV$ refers to a relational expression of the form "$x\ op\ c$", where $op \in \{=, \geq, \leq\}$. The condition is a convex combination of such PORVs. If a PORV is part of a cross-event, we denote it as $PORV_E$.

4. L and L' refer to the set of variables that form destination operands for local variable assignments preceding and succeeding respectively, the '$\mapsto$' operator. L_H/L'_H refers to the variables L/L' that are used in the assertion for Hybrid Automaton H.

5. R_{expr} and R'_{expr} refer to the set of expression on the right of local variable assignments preceding and succeeding respectively, the '$\mapsto$' operator. $R_{expr}(v)/R'_{expr}(v)$ corresponds to the expression assigned to variable v.

Given a Hybrid Automaton defined as H = *(Q, X, f, Init, Dom, E, G, R)*, where *Q, X, f, Init, Dom, E, G, R* are as defined in Definition 2.1, and a feature specification F, the Hybrid Automaton H is split on the basis of F as follows.

Definition 4.1. ***Hybrid Automaton Modification-Type 2***

The Hybrid Automaton H' is a collection $H' = (Q', X, f', Init', Dom', E', G', R')$, where

- $$Q' = \begin{cases} q & \text{for } Dom(q) \cap condition = \phi,\ q \in Q \\ \{q_1, q_2, q_3\} & \text{for } q \in < sequence - expr > \quad Dom(q) \cap PORV_E \neq \phi,\ q \in Q \\ \{q_1, q_3\} & \text{for } q \in < sequence - expr > \quad Dom(q) \cap PORV \neq \phi,\ q \in Q \end{cases}$$

  is the set of **discrete states**;

- $$f'(q, x) = \begin{cases} f(q, x) & \text{for } x \in X \quad q \in Q \cap Q' \\ f(p, x) & \text{for } x \in X \quad q \in \{p_1, p_2, p_3\} \end{cases}$$

  is a **vector field**. The vector field (denoted as $\dot{x}$) describes how the continuous state $x \in X$ evolves over time, while in state q;

- $$Init' = \begin{cases} q & \text{for } q \in Init,\quad q \in Q \cap Q' \\ \{q_1, q_2, q_3\} & \text{for } q \in Init,\quad q \notin Q \cap Q,\ Dom(q) \cap PORV_E \neq \phi \\ \{q_1, q_3\} & \text{for } q \in Init,\quad q \notin Q \cap Q,\ Dom(q) \cap PORV \neq \phi \end{cases}$$

  is a set of **initial states**;

- $$Dom'(q) = \begin{cases} Dom(q) & \text{for } \; q \in Q \cap Q' \\ Dom(p) \cap \overline{PORV} & \text{for } \; p \in Q, \quad q \equiv p_1 \in Q',\ p \models PORV. \\ x == c & \text{for } \; p \in Q, \quad q \equiv p_2,\ p_2 \in Q', \\ & \qquad PORV \equiv x \ op \ c, \\ & \qquad p \models PORV. \\ Dom(p) \cap PORV & \text{for } \; p \in Q, \quad q \equiv p_3 \in Q', \\ & \qquad p \models PORV. \end{cases}$$

  is a **domain** function. This function assigns a set of continuous states, $Dom'(q) \subseteq \mathbb{R}^n$, to each discrete state $q \in Q'$.

- $$E' = \begin{cases} (p,q) & \text{for } \; (p,q) \in E, \quad p,q \in Q \cap Q' \\ p \times \{q_1,q_2,q_3\} & \text{for } \; (p,q) \in E, p \in Q', \quad q \notin Q' \\ \{p_1,p_2,p_3\} \times q & \text{for } \; (p,q) \in E, q \in Q', \quad p \notin Q' \\ (q_1,q_2) & \text{for } \; q \in Q, \quad q \notin Q', \quad q_1,q_2 \in Q' \\ (q_2,q_3) & \text{for } \; q \in Q, \quad q \notin Q', \quad q_2,q_3 \in Q' \\ (q_1,q_3) & \text{for } \; q \in Q, \quad q,q_2 \notin Q', \quad q_1,q_3 \in Q' \end{cases}$$

  is a set of **edges**

- $$G'(p,q) = \begin{cases} G(p,q) & \text{for } \; (p,q) \in E, \quad p,q \in Q \cap Q' \\ G(p,r) & \text{for } \; (p,r) \in E, \quad p \in Q \cap Q', \quad q \in \{r_1,r_2,r_3\} \\ G(r,q) & \text{for } \; (r,q) \in E, \quad q \in Q \cap Q', \quad p \in \{r_1,r_2,r_3\} \\ x == c & \text{for } \; p,q \notin Q, \quad p \equiv r_1,\ q \equiv r_2, \quad (p,q) \in E',\ r \in Q \\ PORV & \text{for } \; p,q \notin Q, \quad p \equiv r_2,\ q \equiv r_3, \quad (p,q) \in E',\ r \in Q \end{cases}$$

  is a **guard condition**;

- $$R'(p,q,x) = \begin{cases} R(p,q,x) & \text{for } \; p,q \in Q \cap Q' \\ R(p,r,x) & \text{for } \; (p,r) \in E, \quad p \in Q \cap Q', \quad q \in \{r_1,r_2,r_3\} \\ R(r,q,x) & \text{for } \; (r,q) \in E, \quad q \in Q \cap Q', \quad p \in \{r_1,r_2,r_3\} \\ x & \text{for } \; p,q \in \{r_1,r_2,r_3\}, \quad r \in Q, \quad (p,q) \in E' \end{cases}$$

  is a **reset map**.

□

State splitting with respect to PORVs is a known technique, and was used for model checking abstractions of Hybrid Systems [36]. However the requirement to capture a feature valuation at a cross event (as in the above example) necessitates a three way split as opposed to a two way split between states satisfying or refuting the PORV.

To prove that the modification introduced in Definition 4.1 doesn't change the behaviour of the original Hybrid Automaton, we introduce the definition for paths of a Hybrid Automaton.

Definition 4.2. ***Path in a Hybrid Automaton***
A Path in a Hybrid Automaton H, is defined as a timed sequence of states

$$\langle l_0, x_0\rangle \xrightarrow{\tau_1} \langle l_1, x_1\rangle \xrightarrow{\tau_2} \langle l_2, x_2\rangle \xrightarrow{\tau_3} \dots \dots \xrightarrow{\tau_i} \langle l_i, x_i\rangle \xrightarrow{\tau_{i+1}} \dots$$

such that state $\langle l_i, x_i\rangle$ is reachable from state $\langle l_{i-1}, x_{i-1}\rangle$ in τ_i time, $\forall i \geq 1$, according to the activities and transition rules of H, where $\langle l_0, x_0\rangle \in Init$.

□

$Paths(H)$ refers to the set of all paths permitted by Hybrid Automaton H. A *run* in a Hybrid Automaton is a continuous timed-sequence of states permitted by some path in the Hybrid Automaton. A *sub-run* of a run ρ, is a timed sequence of states that form a sub-sequence of states from the run ρ.

Definition 4.3. ***Difference between Runs***
The difference between runs ρ_1 and ρ_2, $\rho_1 - \rho_2$, where $\rho_1 = \rho_a.\rho_b$ and $\rho_2 = \rho_b.\rho_c$ is defined as the timed-sequence of states, ρ_a.

$$\rho_1 - \rho_2 = \rho_a \text{ , where } \rho_1 = \rho_a.\rho_b \text{ and } \rho_2 = \rho_b.\rho_c$$

□

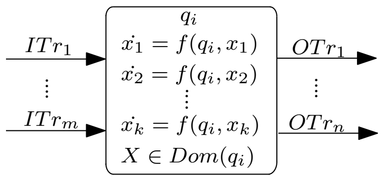

q_i
$\dot{x_1} = f(q_i, x_1)$
$\dot{x_2} = f(q_i, x_2)$
$\dot{x_k} = f(q_i, x_k)$
$X \in Dom(q_i)$
ITr_1
ITr_m
OTr_1
OTr_n

Figure 4.3: Location q_i before splitting on $PORV_j \equiv x_j \sim c$

Theorem 4.1. *Definition 4.1 transforms Hybrid Automaton H into H' such that $Paths(H') \equiv Paths(H)$.*

Proof. The transformation of a Hybrid Automaton location in H that is affected by Definition 4.1, is described in Figures 4.3, 4.4 and 4.5. ITr and OTr refer to incoming and outgoing transitions respectively. Essentially a location q_i is split on account of a $PORV_j \equiv x_j \sim c$, where $\sim \in \{\geq, \leq, ==\}$ as follows:

1. Split q_i into three parts q_{i_1}, q_{i_2}, q_{i_3}, if the PORV is part of an event; else we require a two way split into q_{i_1} and q_{i_3}.

2. All incoming and outgoing transitions (guards and resets included), location invariants, and location activities of part locations q_{i_1}, q_{i_3}, q_{i_3} are identical to those of the parent location q_i.

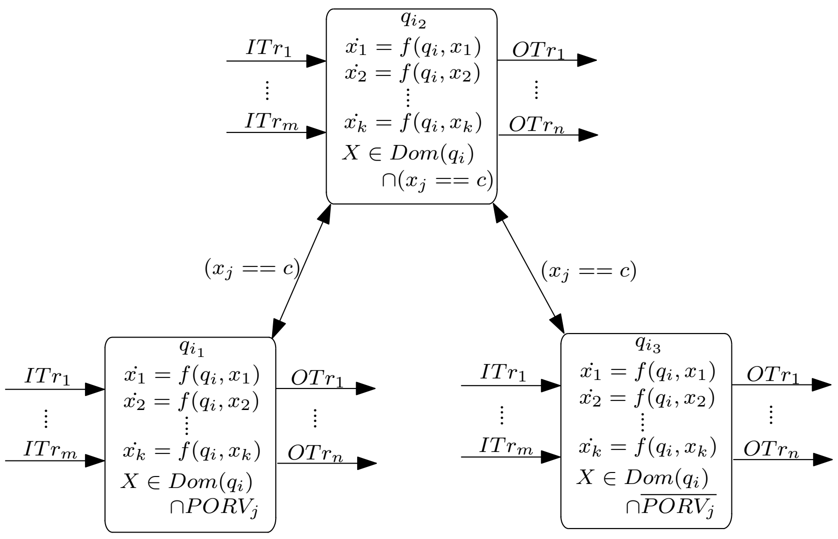


Figure 4.4: q_i after splitting on $@^+$ or $@^-$ events on $(PORV_j \equiv x_j \sim c)$

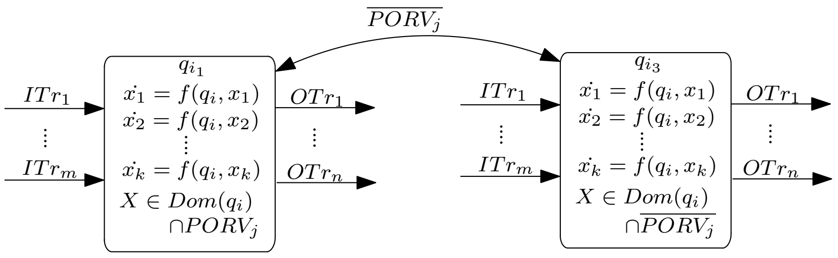


Figure 4.5: q_i after splitting on $PORV_j \equiv x_j \sim c$

3. We add invariants $PORV_j$, $x_j == c$, and $\overline{PORV_j}$ respectively to locations q_{i_1}, q_{i_3} and q_{i_3}.

4. We also allow the following transitions:

   (a) On guard: $x_j == c$ between q_{i_1} and q_{i_2} in both directions (equivalent to two transitions, one in either direction), when the PORV is associated with an event.

   (b) On guard: $\overline{PORV_j}$ between q_{i_2} and q_{i_3} in both directions (equivalent to two transitions, one in either direction), when the PORV is associated with an event.

   (c) On guard: $\overline{PORV_j}$ between q_{i_1} and q_{i_3} in both directions (equivalent to two transitions, one in either direction), when the PORV is NOT associated with an event.

Steps 1 and 2 create identical copies of the original location q_i. Locations q_{i_1}, q_{i_2} and q_{i_3} each are identical to q_i, in terms of the transitions, invariants and activities in the locations. As of this point, addition of these locations introduces three redundant sets of states.

Step 3 modifies the new locations by adding invariants. The union of the new domains of q_{i_1}, q_{i_2}, q_{i_3}, is equal to the domain of the single location q_i.

$$\begin{aligned} Dom(q_{i_1}) \cup Dom(q_{i_2}) \cup Dom(q_{i_3}) = \quad & \{Dom(q_i) \cap PORV_j\} \cup \\ & \{Dom(q_i) \cap x_j == c\} \cup \\ & \{Dom(q_i) \cap \overline{PORV_j}\} \\ = \quad & Dom(q_i) \end{aligned}$$

Let ρ be a run of H. Let ρ_i be a timed sequence of states $\langle q_i, X_t \rangle$, a sub-sequence of ρ, where $X_t \in Dom(q_i)$ is a valuation of variables in H at time t in the run, ρ. We can break ρ_i into:

- $\rho_{i_1} \subseteq \rho_i \,|\langle q_i, X_t\rangle \in \rho_{i_1}$ iff $X_t \in \{Dom(q_i) \cap PORV_j\}$
- $\rho_{i_2} \subseteq \rho_i \,|\langle q_i, X_t\rangle \in \rho_{i_2}$ iff $X_t \in \{Dom(q_i) \cap x_j == c\}$
- $\rho_{i_3} \subseteq \rho_i \,|\langle q_i, X_t\rangle \in \rho_{i_3}$ iff $X_t \in \{Dom(q_i) \cap \overline{PORV_j}\}$

Since $PORV_j \equiv x_j \sim c$, where $\sim \in \{\geq, \leq, ==\}$; $\rho_{i_2} \subseteq \rho_{i_1}$ and $\rho_{i_2} \subseteq \rho_{i_3}$. We can re-construct ρ_i as the concatenation of, ρ_{i_1} with $(\rho_{i_3} - \rho_{i_2})$, or $(\rho_{i_1} - \rho_{i_2})$ with ρ_{i_3}.

Let $l_r(\rho)$ refer to the initial state of the run ρ. Since ρ_i is a run through q_i, ρ_{i_1} is a run through q_i, with its domain restricted by $PORV_j$. Similarly ρ_{i_2} is a run through q_i, with its domain restricted with $x_j == c$, and ρ_{i_3} is a run through q_i with its domain restricted with $\overline{PORV_j}$. Since the activities in q_{i_1}, q_{i_2} and q_{i_3} are identical to q_i, with initial states $l_r(\rho_{i_1})$, $l_r(\rho_{i_2})$ and $l_r(\rho_{i_3})$ in q_{i_1}, q_{i_2} and q_{i_3} respectively, will result in exactly the runs ρ_{i_1}, ρ_{i_2} and ρ_{i_3}.

The discrete transitions added by Step 4, allow discrete moves between locations q_{i_1}, q_{i_2} and q_{i_3}. If we project the run ρ of H in H', for the sub-sequence ρ_i of the run ρ, when state $\langle q_i, X_t\rangle$, $X_t \in x_j == c$ is reached, state $\langle q_{i_1}, X_t\rangle$, $X_t \in x_j == c$ is reached in H', and ρ_{i_1} completes, moving forward into q_{i_2} producing the sub-run ρ_{i_2}. For the case when the PORV is not associated with an event, when state $\langle q_i, X_t\rangle$, $X_t \in x_j == c$ is reached, state $\langle q_{i_1}, X_t\rangle$, $X_t \in x_j == c$ is reached in H', and ρ_{i_1} completes, moving forward into q_{i_3} producing the sub-run ρ_{i_3}. Since $x_j == c$ is an invariant of q_{i_2}, it is important to observe that no time is spent in q_{i_2}. Therefore, every run ρ of H is a run ρ in H'.

The reverse can be easily proven by recreating ρ_i in H from ρ_{i_1} and $(\rho_{i_3} - \rho_{i_2})$, or $(\rho_{i_1} - \rho_{i_2})$ and ρ_{i_3}. $\square$

It is clear that for events of the form $@^+(PORV_j)$, $@^-(PORV_j)$, or $@(PORV_j)$, the states reachable in q_{i_2} captures information at the cross-event.

Theorem 4.2. *The Transformation of Definition 4.1, correctly captures the intent in the feature specification, that is:*

1. *q_2 captures exactly the value of variable $x \in X$ at the cross-event involving x.*

2. *The location labeled with the PORV, captures exactly the values of the variable $x \in X$, that satisfy the PORV.*

Proof. Sequence expressions are of two types,

1. φ `|-> <feature-computation>`

2. φ_1 `##[<time-window>]` φ_2 `|-> <feature-computation>`

Due to the semantics of the Hybrid Automaton, any sub-run ρ' that matches `<sequence-expr>`, will have a timed-sequence of states that either matches φ in its entirety, or if the sequence expression is of the alternate form, ρ' will have a prefix that matches φ_1 and a suffix that matches φ_2.

For any run ρ which has a sub-run ρ' that matches the sequence expression, we assert that, the transformation specified captures the matched sub-run ρ'. For sequence expressions of the first type, φ causes the automaton to be transformed by location splitting, so that exactly the location for which φ is true, is labeled by φ. A match of ρ' implies that the timed sequence of states of ρ' satisfy φ. These states are thus states of a location labeled with φ, and a reachability analysis focused on locations labeled with φ will produce exactly the states of ρ'. For sequence expressions of the second type, the prefix of ρ' will be contained in the states of some location labeled by φ_1. The suffix will be contained in the states of some location labeled by φ_2. A reachability analysis of the locations labeled φ_1, and those labeled φ_2, respectively contain the states of the prefix and suffix of ρ'.

We also need to prove that although the reachability analysis techniques used include possibly unreachable states in the result, these variations do not produce false positives. The feature signature is interpreted over the range of values of variables in locations labeled by PORVs. Unreachable states can only add to the size of the state space discovered. These states could introduce false negatives, as it so happens with all abstract interpretation techniques used in the formal analysis of systems. [16] False negatives are handled by refining the set of runs captured in the reachable regions, using smaller time steps to more accurately capture the evolution of the state variables.

Any results obtained due to permitted runs are never lost as proved above, hence ensuring that feature signatures contain all runs that are permitted.

□

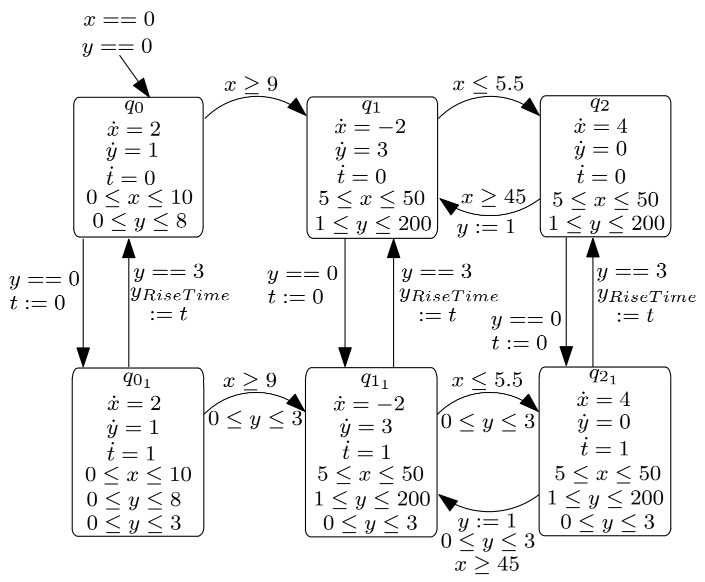


Figure 4.6: Hybrid Automaton H1 with time

4.3. Handling Type-3 Attributes

In the proposed language, the time of occurrence of an event can be recorded in a local variable using the following syntax:

```
<event>, <local-var> = $time
```

For example, consider the following feature:

```
feature yRiseTime();
begin
    var t1, t2;
    (@+(H1.y >= 0), t1=$time)
        ##[0,$]
            (@+(H1.y>=3), t2=$time)
              |-> yRiseTime = t2 - t1;
end
```

In this feature, the time of occurrence of cross events, `@+(H1.y >= 0)` and `@+(H1.y >= 3)` are recorded in the local variables $t1$ and $t2$ respectively. The feature value is defined as the difference between these values, and therefore reflects the time between these events. In order to overlay the computation of such features over abstract interpretation of a Hybrid Automaton, we add a new clock variable, t, into the Hybrid Automaton. We also add a new variable, $y_{RiseTime}$, to capture the feature value. The transformation of $H1$ is shown in Figure 4.6.

As shown in the figure, we first create a copy of the Hybrid Automaton. In the figure the subgraph induced by $q0, q1, q2$ represents the original automaton, and the subgraph induced by $q0_1, q1_1, q2_1$ represents the copy. The clock t advances only in the states of the copy. Initially we are in the original automaton. When the first cross event, namely `@+(H1.y >= 0)`, occurs we may reset t and move to the copy. Once we move to the copy, we continue in the copy (advancing t with time spent) until the second cross event, namely `@+(H1.y >= 3`, occurs. When this happens, we enforce the return to the original automaton by adding the location invariant $0 \leq y \leq 3$ in all locations of the copy. A reachability analysis of the Hybrid Automaton for variable $y_{RiseTime}$ yields the required feature value.

Note that the transition to the copy is a non-deterministic choice when the event `@+(H1.y >= 0)` occurs. This takes care of overlapping matches of of `<sequence-expr>` and ensures that the feature range is computed over all possible matches.

Given a Hybrid Automaton defined as *H = (Q, X, f, Init, Dom, E, G, R)*, where *Q, X, f, Init, Dom, E, G, R* are as defined in Definition 2.1, and a feature specification F, the Hybrid Automaton H is split on the basis of F as follows:

Definition 4.4. ***Hybrid Automaton Modification-Type 3***

The Hybrid Automaton H' is a collection $H' = (Q', X, f', Init, Dom', E', G', R')$, where

- $Q' = \begin{cases} q & \text{for} \quad q \in Q \\ q_1 & \text{for} \quad q \in Q \end{cases}$

  is the set of **discrete states**;

- $f'(q,x) = \begin{cases} f(q,x) & \text{for} \quad x \in X \quad q \in Q \\ f(p,x) & \text{for} \quad x \in X \quad q \equiv p_1,\ p \in Q \end{cases}$

  is a **vector field**. The vector field (denoted as $\dot{x}$) describes how the continuous state $x \in X$ evolves over time, while in state q;

- $Dom'(q) = \begin{cases} Dom(q) & \text{for} \quad q \in Q \\ Dom(p) & \text{for} \quad q \equiv p_1,\ p \in Q \end{cases}$

  is a **domain** function. This function assigns a set of continuous states, $Dom'(q) \subseteq \mathbb{R}^n$, to each discrete state $q \in Q'$;

- $E' = \begin{cases} (p,q) & \text{for} \quad (p,q) \in E \\ (p_1,q_1) & \text{for} \quad (p,q) \in E \\ (p,p_1) & \text{for} \quad < p, Dom(p) > \in condition_1 \\ (p_1,p) & \text{for} \quad < p, Dom(p) > \in condition_2 \end{cases}$

  is a set of **edges**;

- $$G'(p,q) = \begin{cases} G(p,q) & \text{for} \quad (p,q) \in E \\ G(r,s) & \text{for} \quad p \equiv r_1,\ q \equiv s_1,\ (r,s) \in E \\ \mathit{condition}_1 & \text{for} \quad q \equiv p_1, < p, \mathit{Dom}(p) > \in \mathit{condition}_1 \\ \mathit{condition}_2 & \text{for} \quad p \equiv q_1, < q, \mathit{Dom}(q) > \in \mathit{condition}_2 \end{cases}$$

  is a **guard condition**;

- $$R'(p,q,x) = \begin{cases} R(p,q,x) & \text{for} \quad (p,q) \in E \\ R(r,s,x) & \text{for} \quad p \equiv r_1, \quad q \equiv s_1, \quad (r,s) \in E \\ 0 & \text{for} \quad x \equiv t, \quad q \equiv p_1, \quad < p, \mathit{Dom}(p) > \in \mathit{condition}_1 \\ t & \text{for} \quad x \equiv F, \quad p \equiv q_1, \quad < q, \mathit{Dom}(q) > \in \mathit{condition}_2 \end{cases}$$

  is a **reset map**.

□

Theorem 4.3. *Definition 4.4 transforms Hybrid Automaton H into H' such that $Paths(H) \in Paths(H')$.*

Proof. The transformation creates two identical copies of H in H'. Let us refer to the two copies as H_1 and H_2. Initial states of H are projected as initial states of H_1. Transitions between H_1 and H_2 are merely suggestions for movements that are voiced whenever the guard predicates are satisfied by the state. Since a run begins in H_1, all runs of H are contained in runs of H_1. The paths that are added to $Paths(H)$ are those that have cumulative time, t, captured as part of H_2. If we delete t from all states in H_2, $Paths(H) \equiv Paths(H')$ by overlapping identical runs through H_1 and H_2 at equivalent time points. □

Theorem 4.4. *Definition 4.4 correctly captures the temporal attribute of 'time' when* `sequence -expr` *matches in a run of H'.*

Proof. Definition 4.4 duplicates H into H_1 and H_2, which together form H'. If a state $\langle p, X_{t_1} \rangle \models \mathit{condition}_1$ in run ρ in H_1, then Definition 4.4 allows a transition to p_1 in H_2. On the transition, t is reset to zero. While ρ proceeds in H_2, t counts from the instance t_1 at which the transition to H_2 was made. When a state $\langle q, X_{t_2} \rangle \models \mathit{condition}_2$ is reached in run ρ in H_2, then Definition 4.4 allows a transition to q in H_1. At the instant when the transition is taken, t has counted from 0, when H_2 was first entered, upto time instant t_2, when H_2 is first left, effectively computing $t = t_2 - t_1$. On the transition, the current value of t is captured in the feature variable F. Time sub-sequence $\rho' \subset \rho$ while in H_2, represents a match of `sequence-expr` on ρ, and F contains the elapsed-time corresponding to the match.

□

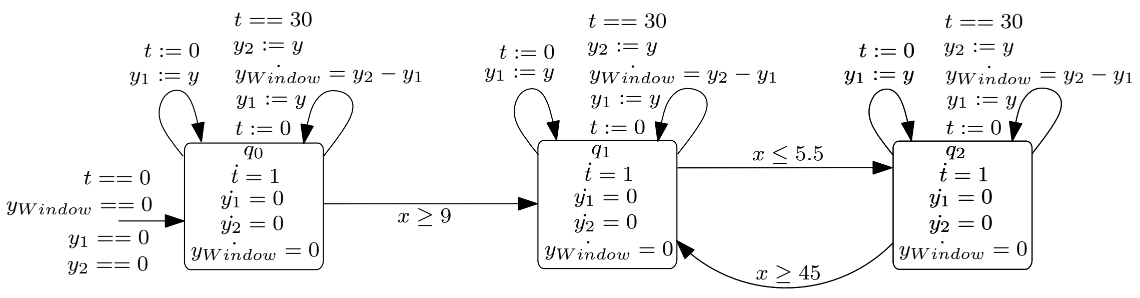


Figure 4.7: Hybrid Automaton H1 with Rabbit Ears

4.4. Handling Type-4 Attributes

We now consider features that are functions of values of variables at different time points. As an example, consider the following feature:

```
feature YWindow();
begin
    var y1, y2;
    y1 = H1.y ##30 y2 = H1.y
           |-> YWindows = abs(y2 - y1);
end
```

This feature returns the change in the value of y in any time window of 30. Since time is dense, we have to consider time windows starting at *every time point*.

We leverage the notion of non-determinism to overlay the computation of such features over abstract interpretation of Hybrid Automata. Intuitively, we add a clock variable t, and add the following transformations at every location of the Hybrid Automaton:

1. We add a self-loop at every location which resets t at any non-deterministically selected point of time and records the value of y in the variable $y1$ at this time.

2. We add a self-loop at every location which can be taken when t is equal to 30. The self loop also assigns to some feature variable, the value of `abs(y2 – y1)`, where $y2$ is the value of y recorded at this time.

Since abstract interpretation of Hybrid Automata automatically considers all possible computations, the above transformations are sufficient to evaluate `abs(y2 – y1)` over all time windows.

Adding the two self loops as indicated above renders a visual similarity of the locations with *rabbit ears*, hence we shall refer to this transformation as *rabbit ear transformations.*

The rabbit ear transformation for the Hybrid Automaton, $H1$, with respect to the above feature is shown in Figure 4.7.

Theorem 4.5. *The Rabbit Ear Transformation applied to locations of paths of interest, captures the required feature exactly for the window specified.*

Proof. The proof idea is as follows:

The generic rabbit ear method, adds rabbit ears along locations on paths of interest. This is abstracted out as a graph problem, that finds all locations along paths from the location that matches the sequence expression first, upto and including the location that matches last. If any PORV causes locations to be split, then the transformation in Section 4.2 must be applied first. We proceed to prove that the rabbit ear transformation applied is sufficient to capture the feature signature over runs of the Hybrid Automaton.

Local variables of the feature and the feature variable F, are added as control variables in the Hybrid Automaton. The activity acting upon them causes no change in their value while in a location. A variable t, that is a representative of time, is also added as a control variable to the automaton. t is a clock variable, and as such has an activity $\dot{t} = 1$ working on it in all locations. In locations not on paths of interest $\dot{t} = 0$. The value of t is carried over across transitions along paths of interest and is reset to 0 other wise. Incoming transitions into locations on paths of interest have resets for local variables, that set them using assignment expressions as given in the feature specification. On locations on paths of interest, we take advantage of non-determinism to add two transitions:

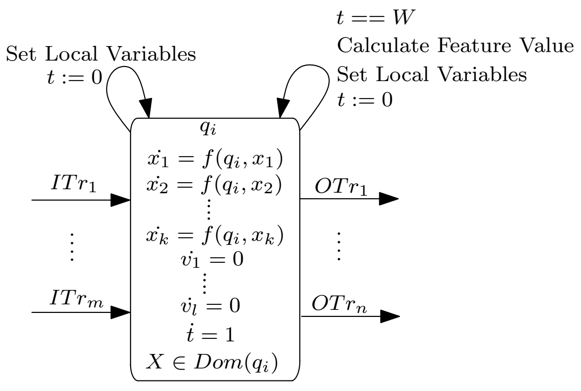


Figure 4.8: Location with Rabbit Ears

1. A Transition that sets $t := 0$, restarting the window; and sets assigns to each local variable a value computed from the corresponding assignment expression in the feature specification. The guard on the transition is always *true*. The semantics of *Non-determinism*, thus allow the transition to be taken at any instant of time, i.e. from every state in the location, thus

allowing us to restart the window computation non-deterministically at any point in time.

2. A Transition that evaluated the feature F when the window width is breached. If W is the window width, then the transition is guarded by the expression $t == W$.

Type 4 features discuss features evaluated over finite-windows, over paths specified by the sequence expression. These paths are identified by abstract matches of the sequence expression, considering only location names and PORVs labeling locations; which follows from the semantics of the sequence expression. The rabbit ears are added only to locations on these paths. Taking any one of the transitions (rabbit ears) can be thought of as, tapping on a rabbit ear.

Of the rabbit ears added, Rabbit Ear 1, allows us to non-deterministically start the feature computation window at any state in the location. As time progresses, non-determinism allows the computation of many parallel windows. For each run, starting with tapping on Rabbit Ear 1, time progresses until the the time window has elapsed. Since the feature must be computed only when a sequence expression matches over a time equal to the window size, exactly when the window bound is reached, tapping on Rabbit Ear 2, achieves exactly what we desire. Tapping on the second ear, allows us to calculate the feature value and restart the window. If the window crosses over locations, there are two possibilities:

- *The window crosses over locations on paths of interest:* In this case transitions along the path carry over all variable values assigned at the beginning of the window by using unit resets (Of the form $x := x$). When the window boundary is reached, tapping on the second rabbit ear in the location where the window boundary is reached, will cause the feature to be computed using the values that were carried over from when the first ear was tapped.

- *The window crosses over some locations not on paths of interest:* In this case when the first transition is taken from a location on a path of interest, to one that is not on a path of interest, the variable t is reset, with $t := 0$ effectively starting a new window. In such locations that are not of interest $\dot{t} = 0$, and hence the window can never begin while in such a location. When a location of interest is first entered, the window can begin a fresh because all transitions coming into such a location set all local variables with expressions specified in the feature specification.

We compute the feature by extracting the values taken on by F in all locations. And since F is assigned a value only when the second rabbit ear is tapped, that is when the window boundary is reached, the feature range computed is exactly what is required. □

4.5. Reachability Output: A Graphical Perspective

We demonstrate the output of PHAVer on H1 after processing the Type-3 feature detailed in Section 4.3.

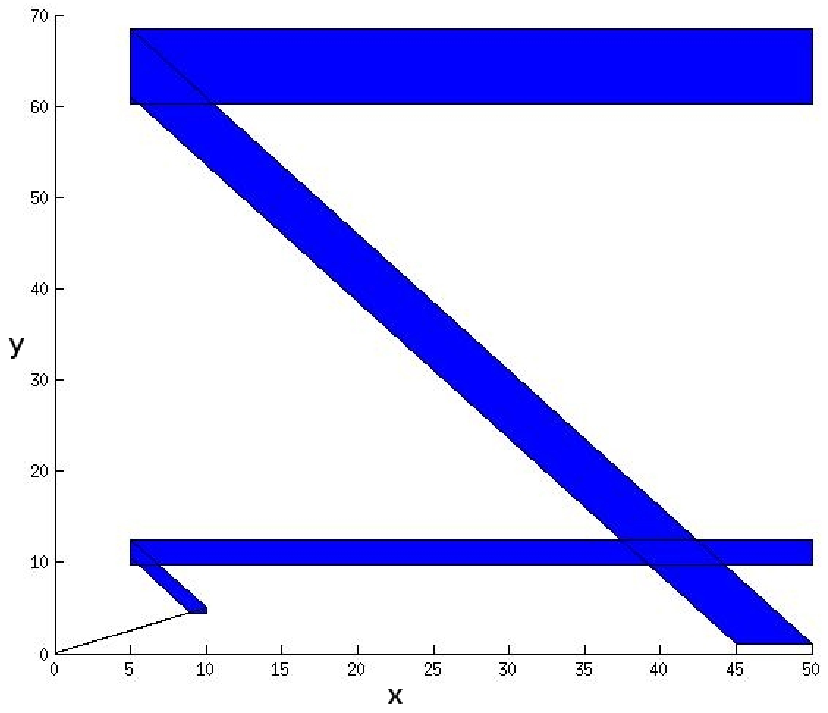


Figure 4.9: Reachability Analysis of H1. Variables x and y are plotted on the x and y axes respectively

Figure 4.9 depicts the reachable regions for variables `x` and `y` of Hybrid Automaton H1 shown in Figure 4.1. In the figure, polygons are plotted describing the regions that are reachable in H1, given the initial condition $x == 0 \wedge y == 0$.

Referring to the feature `yRiseTime` introduced in Section 4.3, modify H1 as shown in Figure 4.6. Running PHAVer on the resulting transformed Hybrid Automaton produces the same reachable region for variables x and y as shown in Figure 4.9. Analyzing for the feature variable `yRiseTime`, we obtain the feature values depicted in Figure 4.10. The feature signature `yRiseTime` is plotted on the y-axis, while variable `y` is plotted on the x-axis. The figure shows two parallel lines indicating the runs in the identical locations in the new automaton. `yRiseTime` is zero (initial value) when in *Level* 2 of the automaton. `yRiseTime` is set when moving from *Level* 2 to *Level* 1. Thus the non-zero value of `yRiseTime` in the plot indicates values of the variable captured for runs in the *Level* 1 of the automaton.

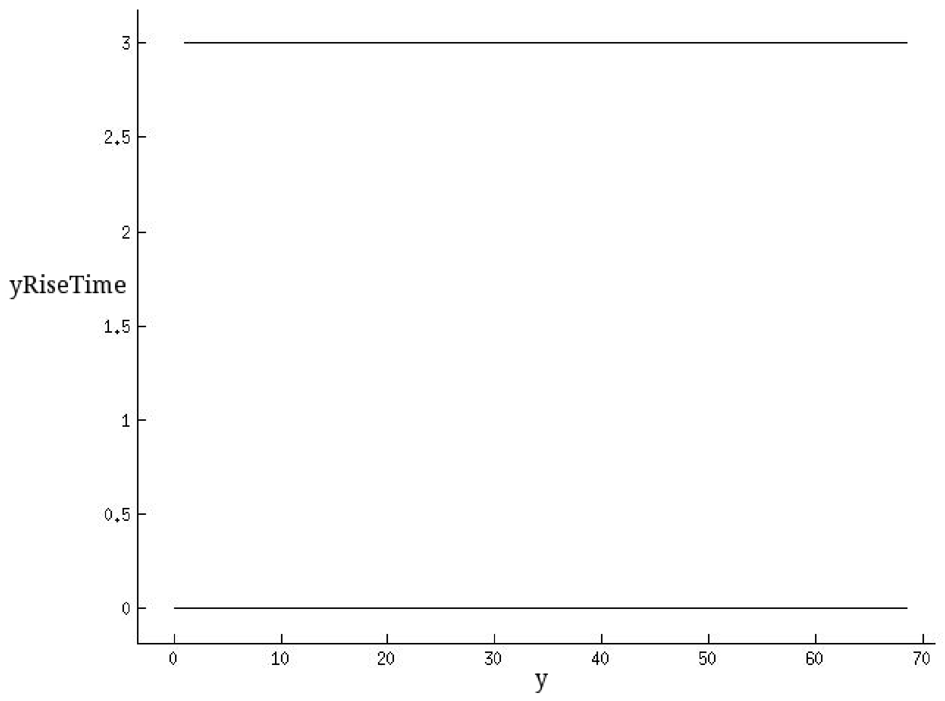


Figure 4.10: Feature Driven Reachability Analysis of H1, with variables y and yRiseTime on the x and y axes respectively

4.6. Concluding Remarks

This chapter expresses the notion of using the feature specification to drive modifications made to the Hybrid Automaton. These modifications essentially overlay the assertions to be matched, along with feature computation over the standard reachability analysis. It is shown that features can consist of properties specified over control variable valuations or valuations of time. For each type of feature an appropriate modification is made to allow a flow-pipe analysis to embed the feature intent.

Chapter 5
Formal Feature Based Equivalence Checking

"It's only 'single steps' that make a journey of 1000 miles.
It means the combined effect of many steps is the equivalent of
a great journey. Go, take many little steps."
- Israelmore Ayivor

Feature ranges may be used to define the equivalence between AMS designs. In the first report submitted under the SRC project [17], a language formalism for specifying feature based equivalence was introduced. The notion of feature based equivalence is recapitulated and some new thoughts in the light of the new research is presented in this chapter.

5.1. Features and Equivalence Definitions

Intuitively, the definition of feature based equivalence has the following components:

1. The definition of the feature.

2. A predicate that defines equivalence in terms of the feature values.

Example 5.1. The *dropout voltage* of a LDO is defined as the *smallest difference between the input and output voltages required to maintain regulation.* The dropout voltage can be evaluated in the *dropout mode of operation* (which is a location of the Hybrid Automaton). In this mode of operation, the output voltage of the LDO falls with the input voltage while maintaining this difference. Suppose our definition of equivalence between two LDO circuits/models includes the requirement that the dropout voltage must be within 5 mV of each other. In other words, if $ldo1.V_{out}$ and $ldo2.V_{out}$ are respectively the output voltages of LDO1 and

LDO2 for input voltages $ldo1.V_{in}$ and $ldo2.V_{in}$ respectively, then the equivalence requirement is formally specified as follows:

```
feature vdrop1;
begin
    (ldo1.state == DROPOUT) |-> vdrop1 = (ldo1.Vin - ldo1.Vout) ;
end

feature vdrop2;
begin
    (ldo2.state == DROPOUT) |-> vdrop2 = (ldo2.Vin - ldo2.Vout) ;
end

equiv DropoutRange;
begin
    feature vdrop1, vdrop2;
    DropoutRange := abs(vdrop1 - vdrop2) <= 0.005;
end
```

Formally, an equivalence specification will consist of one or more formally specified features, and an equivalence predicate as shown below:

```
feature <feature-name> ( <list-of-parameters> );
begin
  var <list-of-local-variables>;
  <sequence-expr>
   |-> <feature-name> = <feature-expr> ;
end
----
----
equiv <equivalence-name>;
begin
  feature <list-of-feature-names>;
  <equivalence-name> := <equiv-predicate>;
end
```

The syntax is elaborated as follows. The definition of features has been explained in Chapter 3.

1. Each equivalence criterion has a name by which it is referred. The name is also the placeholder for its truth (which is Boolean), as explained below.

2. The equivalence predicate is a linear constraint (or a conjunction of linear constraints) over one or more features, which are defined independently outside the equivalence definition, but declared inside the equivalence definition.

Appendix B describes the equivalence specification syntax.

5.2. Semantics of Feature based Equivalence

We now explain the notion of feature based equivalence. Without loss of generality, suppose $P(x_1, \ldots, x_k)$ denotes the equivalence predicate over the features, $x_1, \ldots, x_k$. Suppose $R(x_1)$ denotes the range of values that x_i can take on the design for which it is defined. For example, R(vdrop1), denotes the range of dropout values for ldo1 in its dropout mode. Then the validity of the following expression captures the specified equivalence criterion:

$$\forall z_1 \in R(x_1) \ldots \forall z_k \in R(x_k) P(z1, \ldots, z_k)$$

Once the feature ranges are computed (say, using the methods presented in this report), the task of determining the validity of the equivalence predicate is straight forward. Since the equivalence predicate is linear, it suffices to check the truth of the equivalence predicate on the corner points of the convex region defined by the intersection of the range constraints for each feature.

It is interesting to note that feature based equivalence is not an equivalence relation in general, because transitivity does not hold always. For example, the following situation is possible with respect to the dropout criterion above:

- The dropout ranges of LDO1, LDO2, and LDO3 are respectively, [2.002, 2.005], [2.003, 2.006], and [2.004, 2.008] respectively.
- The equivalence predicate, `abs(vdrop1 - vdrop2) <= 0.005`, is valid for LDO1 and LDO2, because it holds for all valuations of `vdrop1` and `vdrop2` within their ranges.
- Likewise, the equivalence predicate, `abs(vdrop2 - vdrop3) <= 0.005`, is valid for LDO2 and LDO3, because it holds for all valuations of `vdrop2` and `vdrop3` within their ranges.
- The equivalence predicate, `abs(vdrop1 - vdrop3) <= 0.005`, is *not* valid for LDO1 and LDO3, because the predicate is false for `vdrop1 = 2.002` and `2.007 < vdrop3 <= 2.008`.

It is interesting to observe that feature based equivalence is not an equivalence relation in general.

5.3. Concluding Remarks

Chapters 3 and 4 explain how features can implicitly drive a reachability analysis to consider the feature being computed. This chapter effectively describes how features influence the definition of equivalence between two Hybrid Automata. The equivalence predicate is defined as a function of feature valuations of two Hybrid Automata. Equivalence between the two Automata then requires the equivalence predicate to be true for all possible feature valuation combinations. These valuation combinations are abstracted using an interval of values. Checking for equivalence thus relies on evaluating the equivalence predicate at the boundaries of the intervals.

Chapter 6
Case Studies

"We are apt to forget that children watch examples better than they listen to preaching."
-Roy L. Smith

The proposed methodology has been worked out on two important test cases from the power management domain, namely a Low Dropout Regulator (LDO) and a Battery Charger. For both of these circuit families, there exists well defined Hybrid Automata models. In past work, a tool flow had been developed for generating Hybrid Automata models for PMU components that match the given circuit netlists with high level of accuracy [3]. In this chapter we outline the structure of the Hybrid Automata models for these families, and then report our results on the proposed method for formal evaluation of feature ranges.

In the past, researchers have attempted to generate abstract models of circuits using techniques such as model order reduction. Such techniques have not been successful in raising the level of abstraction to a form where formal techniques such as abstract interpretation are feasible, particularly for complex circuits such as LDOs and battery chargers. At the other extreme, researchers have suggested manual development of behavioral models that are amenable for formal analysis, as the first step in the design flow. Our approach lies in between these extremes, that is:

1. It has been shown in the past [3] that accurate Hybrid Automata models can be automated by starting with a parameterized model skeleton for the family and learning the model parameters automatically. We believe that this is a practical way to make best use of domain knowledge without having to manually create models for every instance of a circuit family.

2. The Hybrid Automata models are amenable for formal evaluation of circuit feature ranges using the techniques outlined in this thesis.

We believe that the combination of the above two steps yields a powerful as well as practical direction for formal analysis of AMS designs. This chapter substantiates this claim by outlining two case studies and by reporting our results on these test cases.

6.1. CASE STUDY-1: Low Dropout Regulator

A Low Dropout regulator (LDO) is a linear DC voltage regulator that can operate with a very small input-output differential voltage. An LDO has the following operational modes [38]:

- **Start-up Mode**: When the bias current, reference voltage, and input supply voltage are within specified range, and the necessary enable signals are asserted, an LDO enters its start-up mode of operation.
- **Steady (ST) Mode**: When the output voltage rises above a threshold in the start-up mode, the LDO transits to the steady mode. In this mode of operation, steady output voltage is maintained.
- **Shutdown Mode**: If the enable signal is de-asserted, or if the bias current, the reference voltage, or the input supply voltage falls outside the specified range, then the LDO moves to the shutdown mode of operation.
- **Dropout Mode**: For proper functioning of the LDO, the relation:
  $$v_{in} > v_{out} + vdrop_{min}$$
  should hold. If the input voltage starts falling, then the output voltage will remain constant as long as the above relation holds. But if the input-output differential voltage falls below the dropout value, then the output voltage starts falling below its rated value to keep the input-output difference above dropout value. This mode of LDO operation, when the output voltage falls with the fall of input voltage is called the dropout mode of operation.
- **Short Circuit (SC) Mode**: When the load current crosses a certain current limit (called the short-circuit current), the LDO enters the short circuit mode, where it lowers its output voltage to protect its own circuitry from burnout.

Some features that would be of interest are:

1. *Time Constant in Start-up Mode* is the time taken by the output voltage to rise from 0% to 63.2% of the steady state voltage.
2. *Time Constant in the Shutdown Mode* is the time taken by the output voltage to fall from 100% to 32.8% of steady state voltage.
3. *Dropout Voltage* is the input-output voltage difference in the dropout mode.

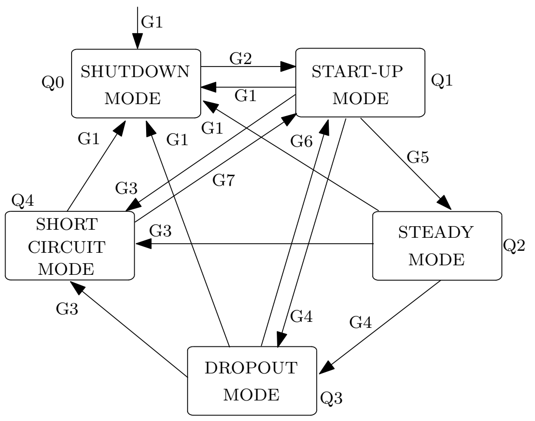


Figure 6.1: Hybrid Automaton for the Low Dropout Regulator: `ldo`

4. *Short Circuit Current* is the current value for which the LDO transits to the short circuit mode.

5. *Rise Time* is the time for the output voltage to rise from 10% to 90% of steady state voltage.

6. *Line Regulation* is the ratio of change in output voltage to change in input voltage during regulation.

7. *Load Regulation* is the change in output voltage to change in output current under varying load conditions.

Due to the complexity of the Hybrid Automaton for the LDO, we cannot diagrammatically describe it in this report. We refer to [3] to create the Hybrid Automaton model of the LDO. Figure 6.1 and Table 6.1 depict an abstraction of the Hybrid Automaton that we use, without the dynamics listed.

These features are coded in the proposed specification language as follows:

1. Time Constant in the Start-up Mode

```
feature StartupTimeConstant(Vss);
begin
  var t1, t2;
  ((ldo.state == STARTUP) && @+(ldo.Vout >= 0*Vss)), t1= $time
    ##[0:$]
      ((ldo.state == STARTUP) && @+(ldo.Vout >= 0.632*Vss)), t2= $time
           |-> StartupTimeConstant = t2 - t1;
end
```

2. Time Constant in the Shutdown Mode

Table 6.1: Predicates and Guard Conditions of an LDO Regulator

(a) Predicates used for an LDO Regulator

Predicates	**Explanations**
$p_1 :: bias_{min} \leq Ibias \leq bias_{max}$	The bias current is within tolerance limits, where $bias_{min}, bias_{max}$ are lower and upper tolerance values for bias current ($Ibias$).
$p_2 :: Vin \geq tol$	Input voltage is above *tol*, the tolerance value for supply voltage.
$p_3 :: Ven > fullscale/2$	Enable is asserted when the enable(Ven) voltage crosses $fullscale/2$.
$p_4 :: ref1 \leq Vref \leq ref2$	$ref2, ref1$ are upper and lower tolerance values for reference voltage ($Vref$).
$p_5 :: Iout \geq I_{shrt}$	The output current $Iout$ is above the short circuit limit (I_{shrt}).
$p_6 :: Vin - Vout \leq dropout$	Vin is the input voltage, $Vout$ is the output voltage. The difference between the voltages should be below the dropout rating.
$p_7 :: \|Vout - V_{ss}\| < \epsilon$	V_{ss} is the rated steady state output voltage and derived from *select* and *trim* input variables; ϵ is a very small value.

(b) Guard Conditions of an LDO Regulator

Guard Conditions (G)	**Transition Relation (δ)**
$G_1 = \neg p_1 \vee \neg p_2 \vee \neg p_3 \vee \neg p_4$	From any mode to *shutdown* mode i.e., ($Q_i \rightarrow Q_1$)
$G_2 = p_1 \wedge p_2 \wedge p_3 \wedge p_4$	From *shutdown* to *start-up* mode i.e., ($Q_0 \rightarrow Q_1$)
$G_3 = p_1 \wedge p_2 \wedge p_3 \wedge p_4 \wedge p_5$	From any mode except *shutdown* to *short circuit* mode i.e., ($Q_1/Q_2/Q_3 \rightarrow Q_4$)
$G_4 = p_1 \wedge p_2 \wedge p_3 \wedge p_4 \wedge \neg p_5 \wedge p_6$	From *start-up* or *regulatory* to *dropout* mode i.e., ($Q_1/Q_2 \rightarrow Q_3$)
$G_5 = p_1 \wedge p_2 \wedge p_3 \wedge p_4 \wedge \neg p_5 \wedge \neg p_6 \wedge p_7$	From *start-up* to *regulatory* mode i.e., ($Q_1 \rightarrow Q_2$)
$G_6 = p_1 \wedge p_2 \wedge p_3 \wedge p_4 \wedge \neg p_5 \wedge \neg p_6$	From *dropout* to *start-up* mode i.e., ($Q_3 \rightarrow Q_1$)
$G_7 = p_1 \wedge p_2 \wedge p_3 \wedge p_4 \wedge \neg p_5$	From *short circuit* to *start-up* mode i.e., ($Q_4 \rightarrow Q_1$)

```
feature ShutdownTimeConstant(Vss);
begin
  var t1, t2;
  ((ldo.state == SHUTDOWN) && @+(ldo.Vout <= Vss)), t1= $time
    ##[0:$]
      ((ldo.state == SHUTDOWN) && @+(ldo.Vout <= 0.328*Vss)), t2= $time
           |-> ShutdownTimeConstant = t2 - t1;
end
```

3. Dropout Voltage

```
feature VDrop;
begin
  (ldo.state == DROPOUT)
           |-> VDrop = (ldo.Vin - ldo.Vout);
end
```

4. Short Circuit Current

```
feature ShortCircuitCurrent;
begin
  (ldo.state == STARTUP || ldo.state == REGULATORY || ldo.state == DROPOUT)
      && ##E (ldo1.state == SHORTCIRCUIT)
```

```
                |-> ShortCircuitCurrent = ldo.Iout;
end
```

5. Rise Time

```
feature RiseTime(Vss);
begin
  var t1,t2;
  (ldo.state == STARTUP) && @+(ldo.Vout > 0.1 * ldo.Vss), t1=$time
    ##[0,$]
      (ldo.state == STARTUP) && @+(ldo.Vout > 0.9 * ldo.Vss), t2=$time
           |-> RiseTime = t2 - t1;
end
```

6. Line Regulation

```
feature LineRegulation(Vss);
begin
  var Vout1,Vout2,Vin1,Vin2;
  (ldo.state == REGULATORY), Vout1=ldo.Vout, Vin1 = ldo.Vin
    ##[E,$]
      (ldo.state == REGULATORY), Vout2=ldo.Vout, Vin2 = ldo.Vin
           |-> LineRegulation = (Vout2-Vout1)/(Vin2-Vin1);
end
```

7. Load Regulation

```
feature LoadRegulation(Vss);
begin
  var Vout1,Vout2,Iout1,Iout2;
  (ldo.state == REGULATORY), Vout1=ldo.Vout, Iout1 = ldo.Iout
    ##[E,$]
      (ldo.state == REGULATORY), Vout2=ldo.Vout, Iout2 = ldo.Iout
           |-> LoadRegulation = (Vout2-Vout1)/(Iout2-Iout1);
end
```

Note: The use of 'E' in the above properties refers to a very small constant, semantically equivalence to the use of ϵ.

The PORV `ldo.Vout >= 0.632*Vss` in the feature StartupTimeConstant requires a three was split of the location encapsulating the behaviour of the startup mode. The PORV `ldo.Vout <= 0.328*Vss` in the feature ShutdownTimeConstant requires a three was split of the location encapsulating the behavior of the shutdown mode. The measurement of time for both features requires location duplication as was explained previously.

On evaluating these feature on the LDO yields a startup time constant of 171.74μs and a shutdown time constant of 2.568μs. PHAver took about 4 seconds to extract the range of values for each feature.

The type of modifications required for the remaining features are described in Table 6.2, and the result of reachability analysis for the same are given in Table 6.3. As can be seen from the tables, some features can be evaluated only if the inputs to the LDO vary in certain ways. These variations can be exercised through test-benches written for simulating behavioural models. However, we are still to determine what constructs can be used to model inputs whose behaviours are essentially unknown. Hence, modeling inputs while constructing hybrid automaton models is considered as an important direction for future research in the area of verification.

Sr.No	**Feature Name**	**Modification Type**	**Location & Variable of Interest**
1.	StartupTimeConstant	Type 2, Type 3	STARTUP: ‘`StartupTimeConstant`’
2.	ShutdownTimeConstant	Type 2, Type 3	SHUTDOWN: ‘`ShutdownTimeConstant`’
3.	VDrop	Type 1	DROPOUT: ‘`VDrop`’
4.	ShortCircuitCurrent	Type 1	SHORTCIRCUIT: ‘`Iout`’
5.	RiseTime	Type 2, Type 3	‘`RiseTime`’
6.	LineRegulation	Type 4	REGULATION: ‘`LineRegulation`’
7.	LoadRegulation	Type 4	REGULATION: ‘`LoadRegulation`’

Table 6.2: Types of modifications required for each LDO feature

Sr.No	**Feature Name**	**Feature MIN**	**Feature MAX**	**Execution Time(s)**
1.	StartupTimeConstant	171.739 μs	171.739 μs	4.039
2.	ShutdownTimeConstant	2.568 μs	2.568 μs	3.878
3.	VDrop	Requires modeling variations of Input Signal(Vin)		
4.	ShortCircuitCurrent			
5.	RiseTime	88.88 μs	800 μs	3.699
6.	LineRegulation	Requires modeling variations of Input Signal(Vin)		
7.	LoadRegulation			

Table 6.3: LDO Feature Ranges and PHAVer Exectution Times

6.2. CASE STUDY-2: Battery Charger

A typical Li-ion battery has the following operational modes [39]:

- **Off-Mode**: In this mode, the charger is in the shut-down phase and no charging occurs.

- **Pre-Charge(PC) Mode**: When the input voltage(V_{in}) lies between 4.5V and 6V, the charger enters into the pre-charge mode. The charging starts with a very low current value, called the pre-charge current(I_{pre}), and continues till the battery voltage, V_{batt}, reaches 3V, which is called the fullrate voltage.

- **Constant Current(CC) Mode**: In this mode a constant current, called the constant charging current(I_{cc}) is used till the battery voltage rises to 4.2V (terminal voltage).

- **Constant Voltage (CV) Mode**: After reaching the terminal voltage, the charger enters into the CV mode to charge up the battery with a constant voltage. The charging current (I_{chg}) starts decreasing as V_{batt} remains constant.

- **Maintenance Mode**: When the charging current drops to 25 mA (end of charge current), the charger enters the maintenance mode. With time, the battery voltage starts falling from the terminal voltage value. The charging process is re-initiated when the battery voltage falls below 4V (restart voltage).

Some features that would be of interest are:

1. *Time to Charge* from ϵ volts (completely drained of charge) to the battery's rated voltage.

2. *Maximum rise in battery voltage* in any 10 minutes during charging.

3. *Pre-charge Current* in the Pre-charge mode.

4. *Constant Charge Current* in the Constant Current mode.

5. *Constant Charge Voltage* in the Constant Voltage mode.

6. *CC Time Constant of the Voltage Response* during the Constant Current mode.

7. *Maintenance Time Constant of the Voltage Response* during the Maintenance mode.

We use the Hybrid Automaton model of the battery charger `batt` shown in Figure 6.2.

These features are coded in the proposed specification language as follows:

1. Time to Charge:

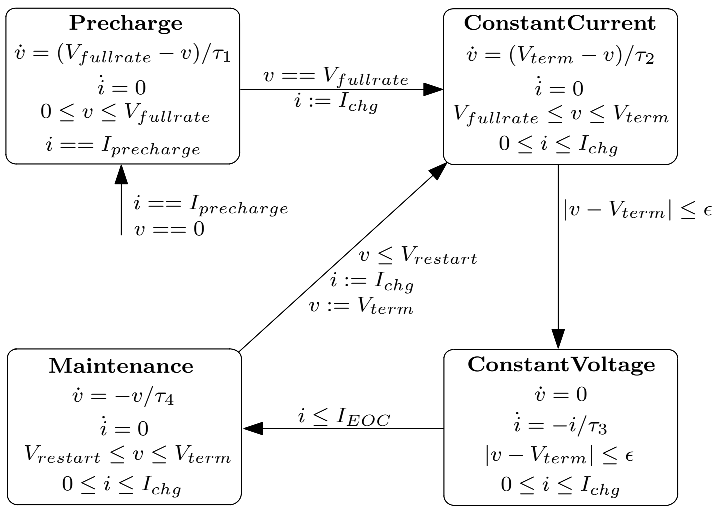


Figure 6.2: Hybrid Automaton for a Battery Charger: `batt`

```
feature ChargeTime(Vterm,epsilon);
begin
    var t1, t2;
    ((batt.state == PRECHARGE) && @+(batt.V >= epsilon), t1 = $time
      ##[0,$]
        ((batt.state == CV) && @+(batt.V == Vterm)  , t2 = $time
           |-> ChargeTime = t2-t1;
end
```

2. Maximum Rise in Battery Voltage:

```
feature MaxVoltageRise(W);
begin
var v1, v2;
v1 = batt.V ##[0,W] v2 = batt.V
           |-> MaxVoltageRise = v2-v1;
end
```

3. Pre-charge Current

```
feature preChargeCurrent;
begin
    (batt.state == PC)
     |-> preChargeCurrent = batt.Ichg;
end
```

4. Constant Charge Current

```
feature ConstantChargeCurrent;
begin
(batt.state == CC)
     |-> ConstantChargeCurrent = batt.Ichg;
end
```

5. Constant Charge Voltage

```
feature ConstantChargeVoltage;
begin
(batt.state == CV)
     |-> ConstantChargeVoltage = batt.Vchg;
end
```

6. CC Time Constant of the Voltage Response

```
feature CCTimeConstant(Vfullrate,Vterm);
begin
  var t1, t2;
  ((batt.state == CC) && @+(batt.Vout >= Vfullrate)), t1= $time
    ##[0:$]
      ((batt.state == CC) &&
          @+(batt.Vout >= Vfullrate + 0.632*(Vterm-Vfullrate), t2= $time
               |-> CCTimeConstant = t2 - t1;
end
```

7. Maintenance Time Constant of the Voltage Response

```
feature MaintenanceTC(Vterm,Vrestart);
begin
  var t1, t2;
  ((batt.state == Maintenance) && @+(batt.Vout <= Vterm)),
    t1= $time
       ##[0:$]
         ((batt.state == Maintenance) &&
            @+(batt.Vout <= Vrestart + 0.328*(Vterm-Vrestart)), t2= $time
                |-> MaintenanceTC = t2 - t1;
end
```

The modifications required for features *ChargeTime* and *MaxVoltageRise* are described below.

For the feature *ChargeTime*, since the PORV `batt.V >= epsilon` is not a location invariant, we split the pre-charge and constant voltage locations into three

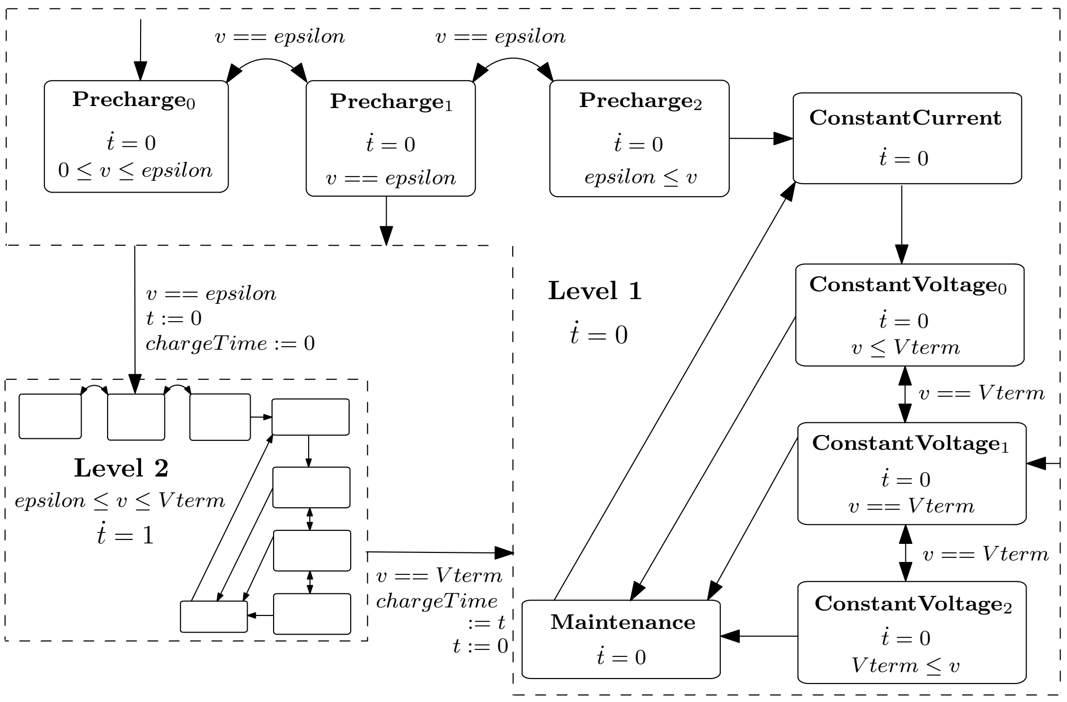


Figure 6.3: Battery Charger Automaton modified for the feature, "*ChargeTime*"

parts each. The sequence of pre-charge locations are labelled with the PORVs `(batt.V <= epsilon)`, `(batt.V == epsilon)` and `(batt.V >= epsilon)` respectively. The sequence of Constant voltage locations are labelled with the PORVs `(batt.V <=Vterm)`, `(batt.V == Vterm)` and `(batt.V >= Vterm)`. We then make a duplicate of every location of the automaton, resulting in two levels of locations. We transition from each Precharge location of level 1 to its counterpart precharge location of level 2 on the guard `(batt.V == epsilon)`. We transition back from Level 2 to Level 1 in each location for Constant Voltage on the guard `(batt.V == Vterm)`. We add a variable t that does not change while in locations in Level 1, and behaves as a clock variable when in locations of Level 2. These modifications are depicted in Figure 6.3.

For the feature *MaxVoltageRise* we add rabbit ears to all locations of the battery charger automaton. We add a clock variable t and variables $v1$, $v2$, $MaxVoltageRise$. On one ear, we reset $MaxVoltageRise$ and the clock t, and capture the current voltage in $v1$. On the other ear we capture the current voltage in $v2$ and calculate $MaxVolatageRise = v2 - v1$, and reset the clock t to indicate the end of the window.

Evaluating these features over the Hybrid Automaton for the battery charger yields a maximum voltage rise of $3V$ for a battery with rated voltage of $4.2V$, and a battery that takes between 54 minutes 48 seconds and 120 minutes to charge. PHAVer took under a second to extract the flow pipe for each feature.

The type of modifications required for the remaining features are described in Table 6.4, and the result of reachability analysis for the same are given in Table 6.5.

Sr.No	**Feature Name**	**Modification Type**	**Location & Variable of Interest**
1.	ChageTime	Type 2, Type 3	‘`chargeTime`’
2.	MaxVoltageRise	Type 4	‘`MaxVoltageRise`’
3.	preChargeCurrent	Type 1	PRECHARGE: ‘`Ichg`’
4.	ConstantChargeCurrent	Type 1	CC: ‘`Ichg`’
5.	ConstantChargeVoltage	Type 1	CV: ‘`Vchg`’
6.	CCTimeConstant	Type 3	CC: ‘`CCTimeConstant`’
7.	MaintenanceTC	Type 3	MAINTENANCE: ‘`MaintenanceTC`’

Table 6.4: Types of modifications required for each Battery Charger feature

Sr.No	**Feature Name**	**Feature MIN**	**Feature MAX**	**Execution Time(ms)**
1.	ChageTime	54m 48sec	120m	109.56
2.	MaxVoltageRise	0.24V	3V	294.90
3.	preChargeCurrent	0.05 μA	0.05 μA	18.97
4.	ConstantChargeCurrent	5.0 μA	5.0 μA	15.89
5.	ConstantChargeVoltage	4.2V	4.2V	14.34
6.	CCTimeConstant	1896 s	1896 s	127.03
7.	MaintenanceTC	38.38857 s	40.308 s	49.05

Table 6.5: Battery Charger Feature Ranges and PHAVer Exectution Times

6.3. Concluding Remarks

It is possible to extract feature signatures from Hybrid Automaton models of AMS circuits, such as LDOs and Battery Chargers, using standard reachability analysis. This analysis is performed after transforming the given abstract models using the transformations introduced in Chapter 4. Feature value ranges can be extracted for two comparable Hybrid Automata. Equivalence between these two automata is determined by comparing the feature ranges using the equivalence definition, as explained in Chapter 5.

Chapter 7
Conclusion and Future Work

"People mistakenly assume that their thinking is done by their head;
It is actually done by the heart which first dictates the conclusion,
then commands the head to provide the reasoning that will defend it."
-Anthony de Mello

The primary focus of this work has been to give an initial direction to determining equivalence between Hybrid Systems. We suggest that equivalence be defined on the basis of features, which are in turn described using, assertions over locations and PORVs; and a feature computation function. By carefully introducing transformations into the given Hybrid Automata, we are able to piggyback feature computation over standard reachability analysis techniques. After computing the feature ranges for two Hybrid Automata, computing feature based equivalence is a trivial step, accomplished by determining the truth of the equivalence predicate, evaluated over the feature ranges.

7.1. Conclusion

This research demonstrates that for certain classes of Hybrid Systems, for example in the case of AMS circuits, abstracted as Hybrid Automata(Linear or Affine), the traditional definition of equivalence needs to be adapted to incorporate the notion of features. A feature can be described using a signature which is expressed as a linear equation over the variables of the Hybrid Automaton. The feature dictates that the original Hybrid Automaton be modified by splitting or duplicating locations and by adding transitions such as rabbit ears. Flow pipe analysis is performed over the variables of the resulting Hybrid Automaton. The flow pipe that results constrains the values taken by the state variables. We can then use these constraints to obtain the range of values for the feature signature. Having extracted this information for two Hybrid Automata, we can then define the equivalence relation between the two automata, in terms of these feature signatures. The equivalence relation is defined using a predicate over real variables. The truth of the predicate gives us the truth of equivalence between the two automata, thus giving us the equivalence relationship between the Hybrid Systems corresponding to the automata.

We have primarily focussed our efforts and used the methods developed on the class of AMS circuits. We have found that assertion languages used to express properties over AMS circuits, can be extended and suitably adapted to express features over the general class of Hybrid Systems.

Finding the range of values for AMS features is an important requirement in practice, which is primarily done today through simulation. It is infeasible in practice to cover all possible behaviors in simulation, more so due to the complexity of analog simulation. Formal symbolic evaluation of feature ranges is therefore an attractive proposition, but it has several technical challenges.

This thesis presents the first attempt to declaratively overlay the definition of feature values on an AMS assertion language. The report also presents the first techniques for preparing a given Hybrid Automata description of an AMS circuit in a way such that abstract interpretation yields the desired feature range. We have demonstrated the utility of the proposed approach on two important circuit families. We have also outlined the utility of our approach in determining feature based equivalence between two AMS models / designs.

7.2. Future Work

There are many aspects of the problem of equivalence checking that are yet to be solved. Some of them are listed below.

7.2.1. Varying Inputs to the Hybrid Automata

To determine equivalence between two Hybrid Systems, such as AMS-circuits, we first need a Hybrid Automaton model of the circuit. Circuits receive input from their environment and produce output. Simulation is the typical method followed for testing the correctness of circuits. A testbench is the primary engine that drives the circuit through desired scenarios, by driving the inputs to the circuit and observing its output.

While designing the Hybrid Automaton for an AMS-circuit, one needs to answer the following questions:

1. What variables form the state of the circuit? Which variables are real-valued and which are discrete? The variables that are real will be termed control-variables. The discrete variables will contribute to determining the modes of the circuits operation.

2. Which are the modes of operation of the circuit? Each mode identified must have linear/affine activities associated with its control-variables.

3. What are the domains of the control-variables in each mode?
4. When does the circuit switch between its modes of operation? What happens when it switches between two modes?
5. Which are the initial states from where the circuit starts operating?
6. How do we define the admissible input patterns for which the circuit is expected to function?

We find that Questions 3 and 6 can pose issues to creating a Hybrid Automaton model of a closed system consisting of the circuit and its environment. Hybrid I/O Automata [28] re-defines Hybrid Automata to include formalisms that represent the interaction of the automaton with its environment. In cases where input behaviours are not obvious, we have begun exploring methods for modeling inputs using non-determinism and the ramifications on creating a closed system consisting of the circuit and its environment.

7.2.2. Further Optimizations

Other improvements to the methodology that need to be explored are as follows:

1. **Optimizing Location splitting**
   The number of locations that results in the splitting operation is very large. When more features are added, it is possible for redundant states to be created. This begs the question "Is it possible to reduce the number of locations by smartly considering the feature definitions?"

2. **Unconstrained Variables**
   It is often the case that variables in the original Hybrid Automaton are unconstrained at either or both ends. It is essential to have them constrained so that reachability analysis tools like PHAVer can perform an analysis of the system. Is there a general solution to this problem? An example of an unconstrained variable is *time*. Perhaps an extension to the tool used, that widens [4] regions when appropriate could help solve this problem. Another proposition is that, from feature definition, we extract constraints that add upper and lower bounds to the range of values a variable can take in locations where it has a non-zero activity. However, these constraints should not in any way hamper the original behaviour of the system.

7.2.3. Parameterized Systems

The general design and working of a Hybrid System can be modelled as a Hybrid Automaton. Every instance of the design is an implementation of the system in which the some system parameters are set. For different instances certain parameters differ. Is it possible to check for equivalence between two design classes that have design parameters whose ranges are known?

7.2.4. Developing a tool flow for automating the Equivalence Checking Process

Currently the process of equivalence checking, Hybrid Automaton location splitting and duplication is being done manually by studying the feature specification and the given Hybrid Automaton. Similarly, generation of reachability analysis code, feature value computation code and goal computation code is being performed manually. All these processes need to be automated.

Appendix A
Feature Specification Syntax

A.1. Tokens

```
RATIONAL          :   -?(([0-9]+|([0-9]*\.[0-9]+))(([eE][i+]?[0-9]+)?)
ARITHOP           :   \+|\-|\* |\/
LEFT_SQR_BRKT     :   [
RIGHT_SQR_BRKT    :   ]
RELOP             :   < | > | <= | >= | ==
EVENTTYPE         :   @\+ | @\- | @
BINLOGICOP        :   (&&) | (\|\|)
ATOM              :   (([_a-zA-Z]+)([_a-zA-Z0-9\.]*))
```

A.2. Grammar Production Rules

```
FEATURE_SPEC      :   feature ATOM ( PARAMETER_LIST ); FEATURE_DEFINITION

PARAMETER_LIST    :   PARAMETERS
                      | ϵ

FEATURE_DEFINITION:   begin FEATURE_BODY end

FEATURE_BODY      :   VARDECL FEATUREDECL

VARDECL           :   var PARAMETERS ;
                      |ϵ

PARAMETERS        :   PARAMETERS, ATOM
                      | ATOM

FEATUREDECL       :   SEQUENCE_EXPR |-> ASSIGNMENT ;

SEQUENCE_EXPR     :   ( SEQUENCE_EXPR DELAY SEQUENCE_EXPR )
                      | PREDICATE

PREDICATE         :   PORVExpr
```

```
                  | EVENT
                  | PORVExpr, ASSIGNMENT
                  | EVENT, ASSIGNMENT

PORVExpr     :    ( PORVExpr BINLOGICOP PORVExpr )
                  | PORV

PORV         :    ( ArithExpr RELOP ArithExpr )

EVENT        :    EVENTTYPE ( ArithExpr, ArithExpr )

ASSIGNMENT   :    ASSIGNMENT, ATOM = ArithExpr
                  ATOM = ArithExpr

DELAY        :    ## LEFT_SQR_BRKT RATIONAL:$ RIGHT_SQR_BRKT
                  | ## RATIONAL

ArithExpr    :    ArithExpr ARITHOP ArithExpr
                  | ATOM
                  | RATIONAL
                  | ( ArithExpr )
```

Appendix B
Equivalence Specification Syntax

The equivalence specification syntax uses the tokens and production rules of Appendix A.

B.1. Grammar Production Rules

```
EQUIVALENCE_SPEC :   equiv ATOM ( FEATURE_LIST ); EQUIV_DEFINITION

FEATURE_LIST     :   PARAMETERS

EQUIV_DEFINITION :   begin EQUIV_BODY end

EQUIV_BODY       :   FEATURE_LIST_DECL EQUIVDECL

FEATURE_LIST_DECL:   feature PARAMETERS ;

EQUIVDECL        :   ATOM := ArithExpr RELOP RATIONAL ;
```

References

[1] 1800-2012 - IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language (`http://standards.ieee.org/findstds/standard/1800-2012.html`).

[2] 1850-2010 - IEEE Standard for Property Specification Language (PSL) (`https://standards.ieee.org/findstds/standard/1850-2010.html`).

[3] Ain, A., Pal, D., Dasgupta, P., Mukhopadhyay, S., Mukhopadhyay, R., and Gough, J. Chassis: A platform for verifying pmu integration using autogenerated behavioral models. *ACM Transactions on Design Automation of Electronic Systems 16*, 3 (June 2011), 33:1–33:30.

[4] Alur, R., C.Courcoubetis, N.Halbwachs, Henzinger, T., Ho, P., Nicollin, X., Olivero, A., J.Sifakis, and S.Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science 138* (1995), 3–34.

[5] Alur, R., and Dill, D. L. A theory of timed automata. *Theoretical Computer Science 126* (1994), 183–235.

[6] Alur, R., Feder, T., and Henzinger, T. A. The benefits of relaxing punctuality. *J. ACM 43*, 1 (Jan. 1996), 116–146.

[7] Alur, R., and Henzinger, T. A. Real-time logics: Complexity and expressiveness. Tech. rep., Stanford University, Stanford, CA, USA, 1990.

[8] Alur, R., and Henzinger, T. A. A really temporal logic. *J. ACM 41*, 1 (Jan. 1994), 181–203.

[9] Asarin, E., Bournez, O., Dang, T., and Maler, O. Approximate reachability analysis of piecewise-linear dynamical systems. In *Hybrid Systems, Computation and Control* (2000), vol. 1790, Springer, pp. 20–31.

[10] Asarin, E., Dang, T., and Maler, O. d/dt: A tool for reachability analysis of continuous and hybrid systems. In *5th IFAC Symposium Nonlinear Control Systems (NOLCOS) , 2001. [ACH + 95* (2001), pp. 3–34.

[11] Bengtsson, J., Larsen, K., Larsson, F., Pettersson, P., Wang, Y., and Weise, C. New generation of uppaal, 1998.

[12] Chen, L.-R. A design of an optimal battery pulse charge system by frequency-varied technique. *IEEE Transactions on Industrial Electronics 54* (2007), 398–405.

[13] Chutinan, A., and Krogh, B. H. Computing polyhedral approximations to flow pipes for dynamic systems. In *Proceedings of the 37th IEEE Conference on Decision and Control* (1998), vol. 2, IEEE Press, pp. 2089–2094.

[14] Chutinan, A., and Krogh, B. H. Computational techniques for hybrid system verification. *IEEE Transactions of Automatic Control 48* (2003), 64–75.

[15] Clarke, E. M., Emerson, E. A., and Sistla, A. P. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst. 8*, 2 (Apr. 1986), 244–263.

[16] Cousot, P., and Cousot, R. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages* (New York, NY, USA, 1977), POPL '77, ACM, pp. 238–252.

[17] Dasgupta, P., Mukhopadhyay, S., Ain, A., Sinha, C., and R, M. Y. Towards a formal framework for specifying feature based equivalence between ams models. Tech. rep., Indian Institute of Technology, Kharagpur, 2013.

[18] Frehse, G. PHAVer - Polyhedral Hybrid Automaton Verifyer (`http://www-verimag.imag.fr/~frehse/phaver_web/`).

[19] Frehse, G. Phaver: Algorithmic verification of hybrid systems past hytech. In *Hybrid Systems: Computation and Control*, M. Morari and L. Thiele, Eds., vol. 3414 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2005, pp. 258–273.

[20] Frehse, G. *Language Overview for PHAVer version 0.35*, June 22 2006.

[21] Gabbay, D., Kurucz, A., F.Wolter, and M.Zakharyaschev. *Many-Dimensional Modal Logics: Theory and Applications.* Elsevier Science Publishers B. V., 2003.

[22] Henzinger., T., Preussig, J., and Wong-Toi, H. Some lessons from the hytech experience. In *Proceedings of the 40th Annual Conference on Decision and Control (CDC'01)* (2001), IEEE Press (2001), pp. 2887–2892.

[23] Henzinger, T. A., and Ho, P.-H. A note on abstract interpretation strategies for hybrid automata. In *Hybrid Systems II* (London, UK, UK, 1995), Springer-Verlag, pp. 252–264.

[24] Henzinger, T. A., Ho, P.-H., and Wong-toi, H. Hytech: A model checker for hybrid systems. *Software Tools for Technology Transfer 1* (1997), 460–463.

[25] Henzinger, T. A., Nicollin, X., Sifakis, J., and Yovine, S. Symbolic model checking for real-time systems. *Information and Computation 111* (1992), 394–406.

[26] L. El Ghaoui, EECS Department, U. B. *Hyper-Textbook: Optimization Models and Applications.* UC Berkeley, 2013.

[27] Lygeros, J. Lecture notes on hybrid systems. Tech. rep., University of Patras, 2004.

[28] Lynch, N., Segala, R., and Vaandrager, F. Hybrid i/o automata. *Information and Computation 185*, 1 (2003), 105 – 157.

[29] Maler, O., and Nickovic, D. Monitoring temporal properties of continuous signals. In *Proceedings of Formal Modeling and Analysis of Timed Systems (FORMATS-FTRTFT). Volume 3253 of LNCS* (2004), Springer, pp. 152–166.

[30] Mukherjee, S. *Assertions: From a Mixed-Signal Perspective.* PhD thesis, Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, August 2012.

[31] Mukherjee, S., and Dasgupta, P. Incorporating local variables in mixed-signal assertions. In *IEEE International Conference TENCON* (2009).

[32] Mukherjee, S., and Dasgupta, P. Auxiliary state machines and auxiliary functions: Constructs for extending ams assertions. In *VLSI Design* (2011).

[33] Mukherjee, S., Dasgupta, P., and Mukhopadhyay, S. Auxiliary specifications for context-sensitive monitoring of ams assertions. *IEEE Transactions on CAD (TCAD) 30(10)* (2011), 1446–1457.

[34] Mukhopadhyay, R., Panda, S. K., Dasgupta, P., and Gough, J. Instrumenting ams assertion verification on commercial platforms. *ACM Trans. Des. Autom. Electron. Syst. 14*, 2 (Apr. 2009), 21:1–21:47.

[35] Pnueli, A. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science* (Washington, DC, USA, 1977), SFCS '77, IEEE Computer Society, pp. 46–57.

[36] Prabhu, S. M., and Dasgupta, P. Model checking controllers with predicate inputs. In *Proceedings of the 26th International Conference on VLSI Design* (2013), VLSID '13, IEEE Computer Society, pp. 332–337.

[37] Silva, B. I., Richeson, K., Krogh, B., and Chutinan, A. Modeling and verifying hybrid dynamic systems using checkmate. In *Proceedings of 4th International Conference on Automation of Mixed Processes* (September 2000), pp. 323–328.

[38] TI. LM2940-N/LM2940C 1A Low Dropout Regulator (`http://www.ti.com/lit/ds/symlink/lm2940-n.pdf`).

[39] TI. LM3658 Dual Source USB/AC Li Chemistry Charger IC for Portable Applications (`http://www.ti.com/lit/ds/symlink/lm3658.pdf`).

[40] Zaki, M. H., Tahar, S., and Bois, G. Review: Formal verification of analog and mixed signal designs: A survey. *Microelectronics Journal 39*, 12 (Dec. 2008), 1395–1404.